

# SlickEdit® 2007

Code Quick | Think Slick™

slickedit.



# SlickEdit<sup>®</sup> 2007 User Guide

(Version 12.0.2)

Information in this documentation is subject to change without notice and does not represent a commitment on the part of SlickEdit Inc. The software described in this documentation is protected by U.S. and international copyright laws and by other applicable laws, and may be used or copied only in accordance with the terms of the license or nondisclosure agreement that accompanies the software. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. The licensee may make one copy of the software for backup purposes. No part of this documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the licensee's personal use, without the express written permission of SlickEdit Inc.

Copyright 1988-2007 SlickEdit Inc.  
Cover design copyright by SlickEdit Inc.  
Produced in the United States of America.

SlickEdit, Visual SlickEdit, Clipboard Inheritance, DIFFzilla, SmartPaste, Context Tagging, and Slick-C are registered trademarks of SlickEdit Inc. Code Quick | Think Slick is a trademark of SlickEdit Inc. All other products or company names are used for identification purposes only and may be trademarks of their respective owners. Protected by U.S. Patent 5,710,926.



# Table of Contents

---

Table of Contents 3

## Introduction 27

How to Get the Most out of SlickEdit® 29

Cool Features 29

Write More Code, Faster 29

Get Started 30

Features and Enhancements in this Version 31

Code Annotations 31

New Class Tool Window 31

XML/HTML Formatting 31

Java Project Improvements 31

Java Live Errors Using Java Compiler 32

New Find Symbol Tool Window 32

Dynamic Surround and Unsurround 32

Copy and Paste in Color 32

Enhanced Symbol Preview 32

Documentation Comments Preview 32

Revised Key Binding Dialog 33

New Files Tool Window 33

Drag-and-Drop Support for KDE and GNOME 33

Windows Vista Support 33

Line Ruler 33

New Color Picker 33

Additional Enhancements 33

Documentation Revisions 34

Documentation and Conventions 35

Accessing Documentation 35

Documentation Feedback 35

Documentation Conventions 35

Default Emulation/Key Binding Mode 35

Platform-Specific Notes 35

*Mac OS X Notes 35*

Menus and Dialogs 36

Code Syntax Conventions 37

Supported Languages and Environments 39

Supported Languages and File Types 39

Special Features for Mac OS X 40

Embedded Languages 40

*Embedded Languages in HTML 41*

*Embedded Languages in Perl and Other Scripting Languages 41*

Supported Editor Emulations 42

Supported Project Types 42

Supported Version Control Systems 42

## Install/Uninstall 43

### Installing SlickEdit® 43

Named User Installation 43

*Windows 43*

*Linux/UNIX 43*

*Mac 43*

Concurrent User Installation 44

Unattended Installation 44

### Uninstalling SlickEdit® 44

Windows 44

Linux/UNIX 45

Mac 45

## Starting and Exiting SlickEdit® 47

### Starting the Program 47

Running Multiple Instances 47

Running SlickEdit® for the First Time 47

### Exiting the Program 48

Exiting with Modified Buffers 48

Default Exit Options 48

## Help and Product Support 49

### Using the Help System 49

F1 Index Help 49

*UNIX, Linux, and Mac OS X F1 Help 49*

*Help Index Options 50*

Help Key Shortcuts 50

Supported Web Browsers 50

### Product Support 51

Product Registration and Updates 51

*Using the Update Manager 51*

Maintenance and Support Service 51

Hot Fixes 51

*Installing Hot Fixes 52*

*Listing Installed Hot Fixes 52*

*Unloading Hot Fixes 52*

Contacting Product Support 52

## Quick Start 53

### Set Your Preferences 53

General Options 53

Extension-Specific Options 54

### Set Up a Workspace and Project 55

Create a New Workspace 55

Create a New Project 55

Add Files to the Project 55

### Start Coding 56

## User Interface 57

### User Interface Overview 59

### Toolbars and Tool Windows 63

Displaying Toolbars and Tool Windows 63

Docking and Grouping Toolbars and Tool Windows	63
Customizing Toolbars	63
Changing Toolbar Button Command Properties	64
Available Toolbars and Tool Windows	64
Tool Windows	64
Backup History	64
Bookmarks	64
Breakpoints	64
Build	64
Class	65
Code Annotations	65
Current Context	65
Defs	65
Exceptions	65
File Tabs	65
Files	65
Find and Replace	66
Find Symbol	66
FTP Client	66
FTP	66
Open	66
Output	66
Preview	66
Projects	67
References	67
Regex Evaluator	67
Search Results	67
Slick-C® Stack	67
Symbols	67
Symbol Properties	67
Unit Testing	67
Toolbars	67
Debug	68
Edit	68
HTML	68
Project Tools	68
Selective Display	68
Standard	68
Context Tagging®	68
Tools	68
XML	68
Debug Toolbars and Tool Windows	69
Debug	69
Autos	69
Breakpoints	69
Call Stack	69
Classes (debug)	69
Debug Sessions	69
Exceptions	69
Locals	69
Members	69
Memory	70
Registers	70
Threads	70
Watch	70
Buffers and Editor Windows	71
Managing Windows	71

## TABLE OF CONTENTS

Changing the Window Left Margin Width	71
Splitting Windows	71
Duplicating Windows	72
Cascading and Tiling Windows	72
<i>Manipulating Tiled Windows</i>	72
Maximizing and Minimizing Windows	72
Switching Between Buffers or Windows	73
Next Window Style	73
Buffer and Window Switching Commands	73
Listing Open Files	74
Linking to a Window	75
Closing Buffers and Windows	76
Accessing Menus	77
Right-Click Context Menus	77
Context Menu Settings	77
Menu Hotkeys	77
Alt Menu	78
Alt Menu Hotkeys	78
Short Key Names in Menus	78
The SlickEdit® Command Line	79
Activating the Command Line	79
Command Line History	79
Command Line Completions	79
Disabling Command Line Completions	80
Using Shortcuts Inside the Command Line	80
Using the Command Line to View Key Binding Associations	80
Determining the Command of a Key Binding	80
Determining the Key Binding of a Command	80
Starting a Program from the Command Line (Shelling)	81
Command Line Prompting	81
Common SlickEdit® Commands	82
Screen Management	83
Full Screen Mode	83
Multiple Monitor Support	83
JAWS Screen Reader Software	83
Configuring JAWS	83
Using the Mouse and Keyboard	85
Key Shortcuts in Text Boxes	85
Text Box Editing Keys	85
Redefining Common Keys	86
Printing	89
Printing on Windows	89
Windows Printer Configuration	89
<i>Header/Footer Print Settings</i>	90
<i>Print Margin Settings</i>	90
<i>Print Schemes</i>	90
Printing on UNIX, Linux, and Mac OS X	91
UNIX, Linux, and Mac OS X Printer Configuration	91

Inserting a Formfeed	91
<b>User Preferences</b>	<b>93</b>
Introduction to User Preferences	95
Global Preferences	95
Extension-Specific Preferences	95
Emulations	97
Supported Emulations	97
Changing Emulations	98
Determining Keys/Functions	99
Key and Mouse Bindings	101
What is a Binding?	101
Managing Bindings	101
Viewing and Filtering Bindings	103
Creating Bindings	103
Editing Bindings	105
Removing Bindings	105
Exporting and Importing Bindings	105
<i>Exporting Bindings</i>	105
<i>Importing Bindings</i>	105
Saving a Bindings Chart	106
Running a Command/Macro using the Key Bindings Dialog	106
Resetting Default Bindings	106
Key Binding Settings	106
Key Message Delay	106
Using Shorter Key Names in Menus	106
Cursor, Mouse, and Scroll Settings	107
Setting the Cursor Style	107
Hiding the Mouse Pointer	107
Displaying Tool Tips	107
Scroll Bar and Scroll Style Settings	107
Setting Fonts and Colors	109
Fonts	109
Setting Fonts for Screen Elements	109
<i>Recommended Fonts for Elements</i>	110
Setting Editor Window Fonts	110
Colors	111
Setting Colors for Screen Elements	111
<i>Using Color Schemes</i>	112
<i>Setting an Embedded Language Color</i>	112
Restoring Settings on Startup	113
Setting File Associations	115
<b>Workspaces, Projects, and Files</b>	<b>117</b>
Workspaces and Projects	119
Overview of Workspaces and Projects	119
Organizing Files	120

## TABLE OF CONTENTS

Version Control	121
Add Wildcard	122
Working with Libraries	122
<b>Managing Workspaces</b>	<b>122</b>
Opening and Closing Workspaces	122
Creating Workspaces	123
Managing Projects within a Workspace	123
Sharing Projects between Workspaces	123
Working with Third-Party Workspaces	124
<b>Managing Projects</b>	<b>124</b>
Project Types	124
<i>GNU C/C++</i>	<i>124</i>
<i>Microsoft Visual Studio</i>	<i>125</i>
<i>Other C/C++ Compiler Compatible with GDB (UNIX only)</i>	<i>125</i>
<i>Other C/C++ Compiler</i>	<i>125</i>
<i>Java</i>	<i>125</i>
<i>Dynamic Languages: Perl, Python, PHP, Ruby</i>	<i>125</i>
Creating Projects	126
Creating Custom Project Types	126
Setting the Active Project	127
Defining Project Dependencies	127
Project Configurations	127
Configuring Project Directories	128
Configuring Project Tools	128
<i>Setting Extension Options</i>	<i>129</i>
<i>Command Line Execution</i>	<i>130</i>
<i>Specifying a Command Directory</i>	<i>130</i>
<i>Other Options</i>	<i>130</i>
Configuring Build Settings	130
<i>Build System Options</i>	<i>131</i>
Defining Extension-Specific Projects	132
<b>Managing Source Files</b>	<b>132</b>
Adding and Removing Files	132
Creating New Files	134
Importing Files	134
Loading Project Files for Editing	134
<b>Creating, Opening, and Saving Files</b>	<b>135</b>
<b>Creating Files</b>	<b>135</b>
Quick Create/Open	135
Using the New File Dialog	135
Creating a File from a Selection	135
<b>Opening Files</b>	<b>135</b>
Opening Files on Windows	136
Opening Files on UNIX or Mac OS X	136
Opening a URL	137
Finding a File to Open	137
Inserting Files into Buffers	137
<b>Saving Files</b>	<b>137</b>
Using Save As	137
Failed Saves	138
<b>Closing Files</b>	<b>138</b>
<b>Setting File Options</b>	<b>139</b>
General File Options	139
<i>Auto CAPS Mode</i>	<i>139</i>

<i>Auto-Change Directory</i>	139
<i>Automatically Close Visited Files</i>	139
<i>First File Opened is Active</i>	139
Load and Save Options	140
<i>Load Options</i>	140
<i>Save Options</i>	141
AutoSave Options	142
File Filter Options	143
Virtual Memory Options	143
<b>Using the File Manager</b>	<b>145</b>
Creating a New File List	145
Appending Files to the List	145
Selecting Files in the File Manager	145
Operating on Selected Files	146
<b>File History and Backups</b>	<b>149</b>
Using Backup History	149
Backing Up Network Files	149
Windows Backups	149
UNIX and Mac OS X Backups	149
Viewing the History of Opened Items	150
Setting Backup Options	150
<b>Code Templates</b>	<b>153</b>
Instantiating a Template	153
Creating Templates	155
<i>Create the Template Source Files</i>	155
<i>Insert Substitution Parameters into the Template Files</i>	155
<i>Use the Template Manager to Create a New Template</i>	155
<i>Add the Template Files to the Newly-Defined Template</i>	156
Substitution Parameters	156
Predefined Substitution Parameters	156
Organizing Templates	158
Template Manager Dialog	158
Creating a New Category	158
Creating a New Template	158
Editing an Existing Template	158
Deleting a Template	158
Categories	159
Templates	159
Template file	159
Details tab	159
<i>Name</i>	159
<i>Description</i>	159
<i>Default name</i>	159
<i>Sort order</i>	159
Files tab	159
Custom Parameters tab	159
Template Options Dialog	160
Global Substitution Parameters	160
Add File Dialog	160
Source file name	160
Copy source file to template directory	160

## TABLE OF CONTENTS

Target file name	160
Replace parameters in target file content	160
Preview	160
Add Parameter Dialog	160
Name	161
Value	161
Prompt for value	161
Prompt string	161
Add New Item Dialog	161
Categories	161
Templates	161
Name	161
Location	162
Add	162
Locating Templates	162
Installed Templates	162
User Templates	162
Manually Creating a Template	162
Example	163
Creating a Multi-file Template	163
Example	164
Code Template Metadata File Reference	164
Elements	165
DefaultName	165
Description	166
File	167
Files	168
Name	169
Parameter	170
Parameters	171
SETemplate	172
SortOrder	173
TemplateContent	174
TemplateDetail	175

## Context Tagging® 177

### Context Tagging® Overview 179

Tag-Driven Navigation	179
List Members	179
Parameter Information	180
Auto List Compatible Parameters	181
Completions	182
Symbol Browsing	182
Statement Level Tagging	182

### Building and Managing Tag Files 183

Building Tag Files	183
Creating Tag Files for Run-Time Libraries	183
Creating Extension-Specific Tag Files	184
Tagging Run-Time Libraries	185
Configuring Context Tagging® for COBOL	185
Managing Tag Files	186
Tag File Categories	186



Tag File Search Order	187
<i>Example: Java Tag File Search Order</i>	187
<i>Example: C/C++ Tag File Search Order</i>	187
Rebuilding Tag Files	187
Context Tagging® Options	188
General Context Tagging® Options	188
Extension-Specific Context Tagging® Options	188
<b>Building, Running, and Debugging</b>	<b>189</b>
Building and Compiling	191
Using Build and Compile Operations	191
Compiling a Project	191
<i>Using VSBUILD to Compile</i>	192
<i>Compiling a Visual C++ Project</i>	192
Specifying Build on Save	192
Specifying Open Commands	192
Language-Specific Build Methods	192
Build Methods for GNU C/C++	192
Build Methods for Xcode	193
Working with Build Errors	193
Viewing and Navigating Errors	193
<i>Listing Errors</i>	194
Parsing Errors with Regular Expressions	194
<i>Configuring Error Parsing</i>	194
<i>Enabling Expressions</i>	195
<i>Setting Priority</i>	195
<i>Adding New Categories</i>	195
<i>Adding New Expressions</i>	196
<i>Editing Expressions</i>	196
<i>Error Expression Groups</i>	196
<i>Sample: Creating a New Error Parsing Expression</i>	197
<i>Testing Expressions</i>	198
<b>Running and Debugging</b>	<b>199</b>
Running a Program	199
Debugging	199
Mixed Mode View in Debugger	199
Debug Key Bindings	199
Multiple Session Debugging	200
<i>Named Sessions</i>	200
<i>Attaching to a Running Process (GNU C++ only)</i>	201
<i>Attaching to a Remote Process (GNU C++ only)</i>	201
<i>Attaching to a Core File (GNU C++, UNIX only)</i>	201
<i>Attaching to a Remote VM (Java only)</i>	201
Setting Breakpoints	201
<i>Setting Conditional Breakpoints</i>	201
<i>Setting Java Exception Breakpoints</i>	202
Generate Debug	202
Setting Debug Options	202
<i>Viewing Debugger Info</i>	202
<i>Fine-Tuning Debugger Performance</i>	203
<i>Viewing Numbers in Multiple Bases</i>	204
<i>Setting Runtime Filters</i>	205
<i>Setting Debug Directories</i>	206
<i>Setting GDB Debugger Configurations</i>	207

Debugger Tool Windows 208

## Editing Features 209

### Navigation 211

#### Code Navigation 211

##### Symbol Navigation 211

*Navigating Between Multiple Instances 212*

*Using the Find Symbol Tool Window 212*

*More Symbol Navigation Methods 212*

##### Begin/End Structure Matching 212

*Viewing and Defining Begin/End Pairs 212*

*Setting the Paren Match Style 213*

#### Cursor Navigation 213

Navigating in Pages and Files 213

Navigating in Statements and Tags 214

Navigating between Words 214

Navigating to a Specific Line 215

Navigating to an Offset 215

### Symbol Browsing 217

#### Class Tool Window 217

Filtering in the Hierarchy Pane 219

*Class Exclusion Manager 219*

Filtering and Sorting in the Members Pane 219

#### Current Context Tool Window 220

#### Defs Tool Window 221

Defs Tool Window Options 221

#### Find Symbol Tool Window 222

#### Preview Tool Window 223

Information Displayed by the Preview Window 224

#### References Tool Window 225

References Tool Window Options 226

#### Symbols Tool Window 227

Filtering Symbols in the Symbols Tool Window 227

Symbols Tool Window Options 228

Viewing Symbol Uses with the Calling Tree 229

Viewing Base and Derived Classes 230

Symbol Browser Filter Options 231

#### Symbol Properties Tool Window 233

### Text Editing 235

#### Selections 235

Character Selections 235

Block Selections 235

*Editing a Block of Text: Block Insert Mode 236*

Line Selections 236

Selection Keys 237

Modifying Selected Text 237

*Adding Numbers to a Selection: Enumeration 241*

Counting Selected Lines and Characters 241

Setting Selection Options 242

#### Cutting, Copying, and Moving Text 242

Dragging and Dropping 242

Using Clipboards	242
<i>Setting the Maximum Number of Clipboards</i>	243
Working with Lines	243
Clicking Past the End of a Line	243
Highlighting the Current Line	243
Preserving the Column on Top/Bottom	243
Setting the Line Insert Style	243
Sorting Text	244
Sort Commands	244
Inserting Literal Characters	245
Color Coding	247
Resetting Modified Lines on Save	247
Adding Color-Coded Keywords to Supported Languages	247
Creating Color Coding for a New Language	247
Color Coding Configuration	248
Advanced Color Coding Configuration	249
Color Coding Settings	249
Syntax Indent and SmartPaste®	251
Syntax Indent	251
Indenting with Tabs	251
<i>Setting Tab Spacing</i>	251
<i>Setting Tab to Indent Selections</i>	252
<i>Setting Tabs for the Current File</i>	252
Setting the Backspace Unindent Style	252
SmartPaste®	252
Completions	253
Auto-Complete	253
Using Auto-Complete	253
Word Completion	254
Completion in Dialogs	255
Argument Completion	255
Configuring Completion Settings	256
Aliases	257
Directory Aliases	257
Defining a New Directory Alias	257
Using Directory Aliases	258
Embedding Environment Variables in Directory Aliases	258
Extension-Specific Aliases	258
Creating an Extension-Specific Alias	258
<i>Choosing the Alias File</i>	259
<i>Using the Alias Editor</i>	259
Alias Escape Sequences	260
<i>Escape Sequence Examples</i>	261
Parameter Prompting	261
<i>Creating an Alias for Parameter Prompting</i>	262
Creating an Extension-Specific Alias from a Selection	262
Syntax Expansion	265
Syntax Expansion Settings	265
Modifying Syntax Expansion Templates	266

Adding Syntax Expansion for Other Languages	266
Dynamic Surround and Surround With	267
Dynamic Surround	267
Surround With	270
Modifying Surround With Templates	271
Surround With Commands	272
Unsurround	273
Deleting Code Blocks	274
Bookmarks	275
Named Bookmarks	275
Naming Bookmarks	275
<i>Command Line Shortcut: sb</i>	275
Allowing Automatic Naming	275
Using a Key Binding for the Name	276
Navigating Named Bookmarks	276
<i>Command Line Shortcut: gb</i>	276
Deleting Named Bookmarks	276
Bookmarks Tool Window	276
Pushed Bookmarks	277
Pushing a Bookmark	277
Popping a Bookmark	277
Viewing Pushed Bookmarks	277
Setting Bookmark Options	278
Code Annotations	279
Code Annotations Overview	279
Annotation Types	279
Purpose-Based Locations	279
Private and Shared Annotations	280
Relocatable Code Marker	280
Annotations Tool Window and File Manager	280
Managing Annotations	280
Creating Annotations	281
Viewing and Filtering Annotations	282
Copying and Moving Annotations	282
<i>Copying</i>	282
<i>Moving</i>	282
Editing Annotations	283
Deleting Annotations	283
Managing Annotation Types	283
Handling Annotation Type Conflicts	284
Managing Annotation Files	285
<i>Personal Annotations</i>	285
<i>Workspace Annotations</i>	285
<i>Project Annotations</i>	285
<i>User-Defined Annotations</i>	286
<i>Annotation File Manager</i>	286
Using Code Annotations to Record Tasks	287
Using Code Annotations for Code Reviews	287
Commenting	289
Commenting Blocks and Lines	289
Comment Block and Line Settings	289

Creating Doc Comments	290
Doc Comment Examples	290
Javadoc Format	290
XMLdoc Format	291
Doxygen Format	291
String Editing	292
Comment Wrapping	292
Reflowing Comments	292
<b>Find and Replace</b>	<b>293</b>
Quick Search and Replace	293
Quick Search	293
Quick Replace	293
Incremental Searching	293
Find and Replace Commands	294
Find and Slash (/) Commands	294
Replace and c Commands	297
Replace Command Search Examples	298
Find and Replace Tool Window	299
Docking the Tool Window	299
Saving Search and Replace Values	299
Syntax-Driven Searching	300
Setting Options	300
Search Results Output	300
Find Symbol Tool Window	301
Find and Replace with Regular Expressions	301
Special Characters in Regular Expression Find/Replace	301
Using Expressions to Search for Binary Characters	303
Using Tagged Search Expressions	303
Minimal versus Maximal Matching	304
Undoing/Redoing Replacements	305
<b>Beautifying Code</b>	<b>307</b>
Code Beautifiers	307
Reflowing Text	307
<b>Refactoring</b>	<b>309</b>
Quick Refactoring	309
Available Quick Refactorings	309
Quick Rename	309
Quick Extract Method	309
Quick Modify Parameter List	310
Quick Replace Literal with Constant	310
C++ Refactoring	310
Available C++ Refactorings	311
Rename	311
Extract Method	311
Modify Parameter List	312
Push Down to Derived Class	312
Pull Up to Super Class	314
Encapsulate Field	315
Extract Class	315
Extract Super Class	316
Move Method	317
Move Static Field	318

	<i>Convert Static to Instance Method</i>	318
	<i>Convert Global to Static Field</i>	318
	<i>Convert Local to Field</i>	319
	<i>Replace Literal with Constant</i>	319
	<i>Create Standard Methods</i>	320
	Test Parsing Configuration	320
	Reviewing Refactoring Changes	323
<b>Viewing and Displaying</b>	<b>327</b>	
	Hexadecimal View and Edit Mode	327
	Hex Mode Key Bindings	327
	Viewing Special Characters	327
	Special Character Toggles	327
	Defining Special Characters	328
	Selective Display	328
	Expanding/Collapsing Code Blocks	329
	Selective Display Regions	330
	Other Display Options	330
	Inserting a Top of File Line	330
	Displaying a Vertical Line	331
	Viewing Line Numbers	331
<b>Language-Specific Editing</b>	<b>333</b>	
<b>Language-Specific Editing Overview</b>	<b>335</b>	
	Language Editing Modes	335
	Changing and Creating Modes	335
	Extension Options	335
	Referring to Extensions	336
	Creating a New Extension	336
	Deleting an Extension	337
<b>C and C++</b>	<b>339</b>	
	C/C++ Formatting Options	339
	Begin-End Style Tab	339
	Indentation Tab	340
	Other Tab	342
	C/C++ Beautifier	343
	Begin-End Style Tab	343
	Indenting Tab	344
	Comments Tab	346
	Other Tab	347
	Schemes Tab	348
	C/C++ Compiler Settings	349
	Creating New Configurations	350
	Building the Tag File	351
	C/C++ Preprocessing	351
<b>Java</b>	<b>353</b>	
	Java Formatting Options	353
	Java Beautifier	355
	Javadoc Beautifier	355
	Javadoc Editor	356

Organizing Java Imports	356
Adding Imports	356
Import Options	356
JUnit Test Support	357
JUnit Toolbar	358
Java Live Errors	358
Working with Apache Ant	360
Adding Ant XML Files to a Project	360
Opening an Ant XML File	360
Invoking Ant Targets	360
Setting Shortcuts for Build and Rebuild	360
<b>XML and HTML</b>	<b>363</b>
XML	363
XML Toolbar	363
XML Formatting Options	363
XMLdoc Editor	364
XML Beautifier	365
<i>Indent Tab</i>	365
<i>Tags Tab</i>	366
<i>Attributes/Values Tab</i>	369
<i>Comments Tab</i>	369
<i>Advanced Tab</i>	371
<i>Schemes Tab</i>	371
DTD Caching	372
<i>Opening DTD Files from XML</i>	372
URL Mappings	372
Toggling Between Begin and End XML Tags	373
HTML	373
HTML Toolbar	374
Exporting to HTML	374
Configuring the Web Browser	374
HTML Formatting Options	375
HTML Beautifier	377
<i>Indent Tab</i>	378
<i>Tags Tab</i>	379
<i>Attributes/Values Tab</i>	381
<i>Comments Tab</i>	382
<i>Advanced Tab</i>	383
<i>Schemes Tab</i>	384
<b>XML/HTML Formatting</b>	<b>387</b>
Enabling/Disabling XML/HTML Formatting	387
Enabling/Disabling Globally	387
Enabling/Disabling for the Current Document	388
Working with Schemes	388
Default Schemes	389
Specifying the Scheme to Use	390
<i>Specifying a Different Default Scheme</i>	390
Creating Schemes	390
Saving and Deleting Schemes	390
Working with Tags	391
Default Tags	391
Base Tags	391
Adding and Deleting Tags	392

## TABLE OF CONTENTS

Formatting Settings	392
General Settings	392
Content Wrap Settings	393
<i>Tag Content Width Settings</i>	<i>394</i>
Tag Layout Settings	395
More Settings	396
Ada	397
Ada Formatting Options	397
Ada Beautifier	397
Indent Tab	397
Statements/Declarations Tab	398
Horizontal Spacing Tab	399
Vertical Alignment Tab	399
Blank Lines Tab	399
Comments Tab	400
Advanced Tab	401
Schemes Tab	402
COBOL	403
COBOL Formatting Options	403
Pascal	405
Pascal Formatting Options	405
PL/I	407
PL/I Formatting Options	407
Python	409
Begin/End Structure Matching for Python	409
<b>Tools and Utilities</b>	<b>411</b>
Comparing and Merging	413
DIFFzilla®	413
Using the DIFFzilla® Dialog	413
Dynamic Difference Editing	413
Comparing Two Files	413
Comparing Symbols or Parts of Files	414
Comparing All Symbols of Two Files	415
Comparing Two Directories	415
Generating File Lists	416
Automatic Directory Mapping	417
Diffing File History	417
3-Way Merge	417
Performing a Three-Way Merge	418
3-Way Merge Settings	419
The compare Command	419
Setting Compare Options	420
Version Control	421
Overview of Version Control	421
Using Version Control	421
Configuring Version Control	422
Advanced Setup Options	423



Setting Up Command Line Version Control Systems	423
Specific Version Control Support	423
Source Code Control (SCC)	423
Configuring SCC	424
Opening an SCC Project	424
PVCS	424
CVS	425
Subversion	425
Spell Checking	427
Spell Check Operations	427
Running Spell Check	427
Spell Checking Multiple Files	428
Setting Spell Check Options	429
FTP	431
Working with FTP	431
FTP Tool Windows	431
Creating a New FTP Profile	431
Starting a Connection	432
Stopping a Connection	433
Opening FTP Files	433
Setting FTP Options	433
The Regex Evaluator	435
Using the Regex Evaluator	435
Entering Test Cases	435
Entering a Regular Expression	435
Regex Evaluator Options	436
Using the Calculator and Math Commands	437
The Calculator	437
Calculating Expressions with Mixed Bases	437
Math Commands	438
Math Command Examples	438
Overflow/Underflow	439
Document Math	439
Prime Numbers	439
OS File Browser	441
<b>Macros and Macro Programming</b>	<b>443</b>
Recorded Macros	445
Recorded Macro Operations	445
Recording a Macro	445
Binding Recorded Macros to Keys	446
Binding Macros Using the Key Bindings Dialog	446
Binding Macros Using <code>execute_last_macro_key</code>	447
Running a Recorded Macro	448
Saving and Editing Recorded Macros	448
Deleting Recorded Macros	449
Using Macros to Discover and Control Options	449
Programmable Macros	450
Loading Macros	450

Setting Macro Variables 450

## Menus, Dialogs, and Tool Windows 453

### File 455

#### File Menu 455

File FTP Menu 456

File Manager Menu 456

*File Manager Select Menu 457*

*File Manager Files Menu 458*

#### Dialogs and Tool Windows 458

New Dialog 458

*File Tab 458*

*Project Tab 459*

*Workspaces Tab 460*

Open Dialog - Windows 461

Open Dialog - Linux, UNIX, and Mac 463

Find File Dialog 465

Open URL Dialog 465

Save As Dialog 466

Save Failed Dialog 467

Exiting with Modified Buffers Dialog 467

Change Directory Dialog 468

Print Dialog - Windows 468

*Print Dialog - General Settings 469*

*General Tab 470*

*Header/Footer Tab 471*

*Margins Tab 471*

*Schemes Tab 472*

Print Dialog - Linux, UNIX, and Mac 472

*General Tab 473*

*Header/Footer Tab 474*

*Advanced Tab 474*

Add/Edit FTP Profile Dialog 475

*General Tab 475*

*Advanced Tab 476*

FTP Options Dialog 477

*General Tab 477*

*Advanced Tab 477*

*Firewall/Proxy Tab 478*

*SSH/SFTP Tab 479*

*Debug Tab 480*

### Edit 483

#### Edit Menu 483

Edit Select Menu 484

Edit Delete Menu 484

Edit Other Menu 485

*Copy Unicode As Menu 485*

#### Dialogs and Tool Windows 486

Select Text to Paste Dialog 486

Enumerate Dialog 487

Filter Selection: Command Dialog 487

### Search 489

#### Search Menu 489

Dialogs and Tool Windows	490
Find and Replace Tool Window	490
<i>Right-Click Context Menu</i>	491
<i>Find Tab</i>	492
<i>Find in Files Tab</i>	493
<i>Replace Tab</i>	495
<i>Replace in Files Tab</i>	495
Find Symbol Tool Window	495
Go to Definition Dialog	497
<b>View</b>	<b>499</b>
View Menu	499
Dialogs and Tool Windows	500
Selective Display Dialog	500
<i>Search Text</i>	501
<i>Function Definitions</i>	501
<i>Preprocessor Directives</i>	502
<i>Multi-Level</i>	502
<i>Paragraphs</i>	502
<i>Hide Selection</i>	502
<i>Expansion Options</i>	502
Toolbar Customization Dialog	503
<i>Toolbars Tab</i>	503
<i>Categories Tab</i>	504
<i>Options Tab</i>	505
Toolbar Control Properties Dialog	506
<b>Project</b>	<b>507</b>
Project Menu	507
Open Other Workspace Menu	508
Dialogs and Tool Windows	508
Workspace Properties Dialog	508
Project Properties Dialog	509
<i>Project Properties Dialog - General Options</i>	510
<i>Files Tab</i>	511
<i>Directories Tab</i>	512
<i>Tools Tab</i>	513
<i>Build Tab</i>	515
<i>Compile/Link Tab</i>	517
<i>Dependencies Tab</i>	518
<i>Open Tab</i>	519
Add Tree Dialog	519
<i>Exclusion Examples</i>	520
<b>Build</b>	<b>523</b>
Build Menu	523
<b>Debug</b>	<b>525</b>
Debug Menu	525
Debug Windows Menu	526
Attach Debugger Menu	526
Dialogs and Tool Windows	526
Debugger Options - Settings Tab	527
<b>Document</b>	<b>529</b>
Document Menu	529
XML/HTML Formatting Menu	530

## TABLE OF CONTENTS

Dialogs and Tool Windows	530
Files Tool Window	531
<i>Accessing the Tool Window</i>	531
<i>List Views</i>	531
<i>Working with the Files List</i>	532
<i>Opening Files for Editing</i>	532
<i>Saving Modified Files</i>	532
<i>Closing Files</i>	533
<i>Files Tool Window Interface</i>	533
Reflow Comment Dialog	534
Current Document Options Dialog	535
Macro	537
Macro Menu	537
Dialogs and Tool Windows	538
Save Macro Dialog	538
List Macros Dialog	538
Variable Editor Dialog	539
Grid Settings Dialog	541
The width and height parameters are in twips (1440 twips equal one inch on the display).	541
Menu Editor Dialog	541
Auto Enable Properties Dialog	542
Tools	545
Tools Menu	545
Version Control Menu	546
C++ Refactoring Menu	547
Quick Refactoring Menu	547
Imports Menu	547
Spell Check Menu	548
Dialogs and Tool Windows	548
Version Control Setup Dialog	548
Version Control Advanced Setup Dialog	550
Checkin/Checkout Files Dialog	552
Organize Imports Options Dialog	553
3-Way Merge Dialog	554
DIFFzilla® Dialog	555
<i>DIFFzilla® Files Tab</i>	555
<i>DIFFzilla Options Tab</i>	556
Multi-File Diff Output Dialog	559
Context Tagging® - Tag Files Dialog	560
Options	563
Options Menu	563
Dialogs and Tool Windows	564
General Options Dialog	564
<i>General Tab</i>	565
<i>Search Tab</i>	567
<i>Selections Tab</i>	570
<i>Special Characters Tab</i>	571
<i>More Tab</i>	572
<i>Exit Tab</i>	576
<i>Virtual Memory Tab</i>	576
Extension Options Dialog	578
<i>Extension Options - General Dialog Settings</i>	579
<i>Indent Tab</i>	579

Word Wrap Tab	581
General Tab	583
Comments Tab	584
Comment Wrap Tab	588
Advanced Tab	590
Auto-Complete Tab	591
Context Tagging® Tab	593
Select a Tag Dialog	596
File Options Dialog	597
Load Tab	597
Save Tab	599
Backup Tab	600
AutoSave Tab	602
File Filters Tab	603
Key Bindings Dialog	604
Bind Key Dialog	607
Redefine Common Keys Dialog	607
Redefinable Keys	608
More Options	609
Context Tagging® Options Dialog	610
Color Coding Setup Dialog	612
Color Coding Setup Options - General Dialog Settings	613
Tokens Tab	614
Numbers Tab	615
Strings Tab	616
Language Tab	618
Comments Tab	619
Tags Tab	621
Color Settings Dialog	622
Font Configuration Dialog	623
XML/HTML Formatting Dialog	625
URL Mappings Dialog	626
Proxy Settings Dialog	626
Network Options Dialog	627
Web Browser Setup Dialog	628
Visual C++ Setup Dialog	629
<b>Window</b>	<b>631</b>
Window Menu	631
Dialogs and Tool Windows	631
Window Font Dialog	631
Link Window Dialog	633
<b>Help</b>	<b>635</b>
Help Menu	635
Product Updates Menu	636
Dialogs and Tool Windows	637
Cool Features Dialog	637
Help Index Dialog	637
Configure Help Index File Dialog	638
Default Help Dialog	638
Update Manager Options Dialog	638
<b>Menu Editing</b>	<b>639</b>
Creating and Editing Menus	639
Creating a New Menu Resource	639
Editing Menus	639

Defining Menu Item Aliases 640  
Enabling/Disabling Menu Items 640

## Appendix 641

### Tutorials 643

Hello World Tutorial (C/C++) 643  
    Create a Project Using GNU C/C++ Wizard 643  
    Create a Project Without Using the GNU C/C++ Wizard 643  
    Build the Project 643  
    Run the Program 644  
    Comments 644  
Hello World Tutorial (Java) 644  
    Create the Project 644  
    Create the File 644  
    Edit the File 644  
    Build the Project 645  
    Run the Program 645

### Encoding 647

Using Unicode 647  
    Unicode File Recognition 648  
    Opening Unicode Files 648  
    Surrogate Support 649  
    Converting Unicode to UCN 649  
    Unicode Limitations 649  
    Unicode Implementation 650

### Invocation Options 651

### Environment Variables 655

Setting Environment Variables in vslick.ini 656  
Using the set Command 656

### Configuration Variables 659

Viewing Configuration Variables 659  
Setting/Changing Configuration Variables 659  
Table of Configuration Variables 659

### Configuration Directories and Files 663

User Configuration Directory 663  
    Configuration Directory Location 663  
    Backing Up the Configuration Directory 663  
    Table of User Configuration Files 663  
System Configuration Files 664  
    Table of System Configuration Files 664

### File Search Order 667

Search Order for Configuration Files 667  
Search Order for Executable Files 667

### Advanced Help Configuration 669

Accessing Multiple Help Files 669  
    Building the Help Index File vslick.idx 669  
    Changing the Default Word Help File(s) 669

Configuring F1 MSDN Help	670
<b>VLX File and Color Coding</b>	<b>671</b>
Modifying the VLX File to Change a Color Definition	671
Creating a Lexer Name and a New VLX File	671
Table of style Values	674
<b>Editing the Key Binding Source</b>	<b>677</b>
<b>Using the ISPF and XEDIT Emulations</b>	<b>679</b>
ISPF Options Dialog	679
ISPF Primary Commands	680
ISPF Line Commands	682
ISPF Line Labels .label	683
ISPF Shift Lines Left or Right	683
ISPF Insert After A	684
ISPF Insert Before B	684
ISPF Insert Bounds Ruler BNDS	684
ISPF Copy Lines C and CC for blocks	685
ISPF Insert Columns Ruler COLS or SCALE	685
ISPF Delete Lines D and DD for blocks	685
ISPF Expose First Lines F and FF	685
ISPF Insert Lines	686
ISPF Lowercase Lines LC, LCC and LCLC for blocks	686
ISPF Expose Last Lines L and LL	686
ISPF Move Lines M and MM for blocks	686
ISPF Insert Mask Line MASK	687
ISPF Make Data Lines MD, MDD and MDMD for blocks	687
ISPF Overlay Lines O and OO for blocks	687
ISPF Repeat Lines	687
ISPF Expose Next Level of Code S and SS	688
ISPF Insert Tabs Ruler TABS or TABL	688
ISPF Insert Text TE	688
ISPF Insert Lines TF	688
ISPF Join Lines TJ	688
ISPF Split Line TS	689
ISPF Uppercase Lines UC, UCC and UCUC for blocks	689
ISPF Exclude Lines X and XX for blocks	689
ISPF Select Lines Z and ZZ for blocks	689
XEDIT Line Commands	690
ISPF Unsupported Primary Commands	690
ISPF Emulation Commands	691
<b>Regular Expression Syntax</b>	<b>693</b>
UNIX Regular Expressions	693
UNIX Regular Expression Examples	697
SlickEdit® Regular Expressions	697
SlickEdit® Regular Expression Examples	702
Brief Regular Expressions	702
Brief Regular Expression Examples	706
Unicode Category Specifications for Regular Expressions	706
Unicode Character Blocks for Regular Expressions	708

## Glossary 713

## Index 717





# Introduction

This chapter contains the following topics:

- [How to Get the Most out of SlickEdit®](#)
- [Features and Enhancements in this Version](#)
- [Documentation and Conventions](#)
- [Supported Languages and Environments](#)
- [Install/Uninstall](#)
- [Starting and Exiting SlickEdit®](#)
- [Help and Product Support](#)



# How to Get the Most out of SlickEdit®

---

At SlickEdit, our belief is that it's the code that really matters. We are a company of power programmers working to develop the tools that power programmers demand—tools that provide the best editing capabilities to help you write your code the way you want.

We take great pride in delivering unparalleled power, speed, and flexibility to our customers. Our goal is to remove the tedious tasks involved with programming, allowing you to focus on the reason you first got into programming: the thrill of writing great code.

## Cool Features

SlickEdit® contains the most powerful and comprehensive set of features available in any editor. Many are unique to SlickEdit. To see a list of some we think are particularly cool, select **Help > Cool Features** from the SlickEdit main menu. Each feature is described and you can watch a short demo of the feature in action.

## Write More Code, Faster

The keys to programming efficiency are the same in SlickEdit® as any other editor. These strategies will help you write more code, faster than you ever have before:

- **Keep your hands on the keyboard** – Time is wasted each time you reach for the mouse. SlickEdit contains 13 editor emulations with predefined key bindings that are ready for use in performing common tasks. Define your own key bindings or invoke editor operations from the SlickEdit command line. For more information, see [Using the Mouse and Keyboard](#).
- **Type as little as possible** - SlickEdit contains many features that reduce the number of key-strokes you type, including: completions, syntax expansion, aliases, macros, code templates, and code generators. For information about these features, see the topics in the [Editing Features](#) chapter.
- **Rapidly navigate code** – Instantly jump from a symbol to its definition or view a list of references. Preview definitions for the current symbol without having to open the file. Use bookmarks to mark important locations in the code. SlickEdit includes powerful browsers and search capabilities, allowing you to quickly find the code you want. See [Navigation](#) for more information.
- **Access information quickly** - SlickEdit uses visual indicators to provide you with information about your code, including syntax highlighting and color coding. Special tool windows are also available for viewing information about files, classes, symbols, definitions, and more. To learn more, see [Toolbars and Tool Windows](#), [Symbol Browsing](#) and the [Editing Features](#) chapter.
- **Let SlickEdit do the formatting** - Syntax indenting, SmartPaste®, and code beautifiers are just a few of the automatic formatting features in SlickEdit. For more information, see the topics in the [Editing Features](#) chapter.
- **Utilize utilities** - SlickEdit provides many utilities for working with your code, such as DIFFzilla®, 3-Way Merge, spell check, FTP, a RegEx Evaluator, math commands, and even a calculator. See the topics in the [Tools and Utilities](#) chapter for more information.
- **Integrate to other tools** - SlickEdit integrates with other tools to make your world complete, including source control systems, compilers, debuggers, profilers, and analyzers. See [Tools and Utilities](#) for more information.

## **Get Started**

Be sure to check out the [Quick Start](#) guide. It contains a short list of common user preference settings and describes how to quickly create a workspace and project.

## Features and Enhancements in this Version

---

The following information describes the features and enhancements in SlickEdit® 2007.

### Code Annotations

Code Annotations allow you to store information about the code without actually modifying the code. You can use Code Annotations for recording various information, like comments about something that needs to be changed, tasks that need to be performed, or comments in preparation for a code review. Anything you can record in a code comment can be stored in a Code Annotation. Because code annotations are not stored in the source code, you can use them to record private information you don't want to share with the rest of the team. Or you can use code annotations to record information that is shared with the team but should never be visible in the source code.

To create a new Code Annotation, position the cursor on the desired line of code, then right-click and select **New Code Annotation** (or use the **new\_annotation** command). To view a list of annotations that you have created, use the Code Annotations tool window (**View > Toolbars > Code Annotations** or **annotations\_browser** command). For more information about recording and using annotations, see [Code Annotations](#).

### New Class Tool Window

The new Class tool window provides an outline view of both the members of the current class as well as any visible inherited members, and also shows the inheritance hierarchy of the current class. This is useful for object-oriented programming languages such as Java™. Note that this is a new tool window, not to be confused with the tool window formerly known as “Classes,” which has been renamed to “Symbols” in SlickEdit® 2007.

To use the Class tool window, select **View > Toolbars > Class**, or use the **activate\_tbclass** command. See [Class Tool Window](#) for more details.

### XML/HTML Formatting

Content in XML and HTML files may be set to automatically wrap and format as you edit according to user-defined formatting schemes. A formatting scheme is comprised of any number of XML or HTML tags, each of which can be configured individually for indent levels, wrapping, and tag structure. Multiple schemes can be defined—for example, you may want one scheme for HTML files and another for XML files, or perhaps you are required to code certain files to various standards. Schemes can be saved and imported, so they can be shared with your team. Tags for each scheme can be entered manually or you can import tags from the current file. For information about enabling/disabling this feature and configuring settings, see [XML/HTML Formatting](#).

### Java Project Improvements

Improvements were made to the Java Options dialog (**Build > Java Options**), and new menu options have been added to the Projects tool window (docked as a tab on the left side of the editor by default). Right-click on a Java package in the tool window to add a new enum, class, interface, or file. For more information about working with projects, see [Workspaces and Projects](#). For more information about Java options, see [Java](#).

## Java Live Errors Using Java Compiler

Java Live Errors identifies errors in your Java code as you type. Our previous implementation used Jikes to compile your code, but Jikes does not support JDK 5 and later. Java Live Errors now uses javac from Sun's JDK 6.

To activate Live Errors, JDK 6 must be installed on your system. The root directory of your JDK 6 installation must be specified on the **Live Error** tab of the Java Options dialog (**Build > Java Options**). If already installed, SlickEdit® will attempt to detect that location automatically the first time a Java project is opened. See [Java Live Errors](#) for more information.

## New Find Symbol Tool Window

This new tool window (**Search > Find Symbol**) is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match. See [Find Symbol Tool Window](#) for more information.

## Dynamic Surround and Unsurround

Dynamic Surround provides a convenient way to surround a group of statements with a block statement, indented to the correct levels according to your preferences. SlickEdit® enters Dynamic Surround mode automatically, immediately after you expand a block statement (for instance, by typing “if” then pressing Space). After expanding the statement, a box drawn around it as a visual guide, and you can pull the subsequent lines of code or whole statements into the block by using the Up, Down, PgUp, or PgDn keys. Pressing any other key will exit Dynamic Surround mode. To disable Dynamic Surround, select **Tools > Options > File Extension Setup**, select the [Indent Tab](#), and deselect the option **Use Dynamic Surround**.

The Unsurround feature allows you to remove structures from a code block. Right-click on a selected code block and select **Unsurround**, or use the **unsurround** command. For more information on these features, see [Dynamic Surround and Surround With](#).

## Copy and Paste in Color

SlickEdit® now provides HTML and RTF clipboard formats, allowing you to paste formatted and color-coded text into other applications (as well as plain text). Select **Tools > Options > General**, then click the [Selections Tab](#). Select one of the **Clipboard format** options. For more information on working with color-coding, see [Color Coding](#). For more information on using clipboards, see [Using Clipboards](#).

## Enhanced Symbol Preview

The tool window known as “Symbol” in previous versions has been enhanced and renamed to “Preview” (**View > Toolbars > Preview**) to be more consistent with its functionality. This window provides a portal for viewing information in other files without having to open them in the editor. It automatically shows this information when you are working with certain interface elements and features. The Preview window also features the “Documentation Comments Preview”—a pane that displays information and code comments for the symbol under the mouse. See [Preview Tool Window](#) for more information.

## Documentation Comments Preview

This feature is implemented as a pane in the [Preview Tool Window](#) (formerly called “Symbol tool window”). Documentation Comments Preview displays information and code comments for the symbol at the cursor.

This gives you a mouse-free way to see the same information provided by the option **Show info for symbol under mouse** (**Tools > Options > File Extension Setup**, [Context Tagging® Tab](#)), which shows the information in a screen tip when you hover over a symbol with the mouse.

## Revised Key Binding Dialog

The Bind to Key dialog has been revised and renamed as the Key Binding dialog (**Tools > Options > Key Bindings**). The dialog lets you create and view key and mouse bindings for commands and user-created macros, and gives you the ability to save your bindings into a chart in XML format for importing and exporting. See [Key and Mouse Bindings](#) for more information about working with these features.

## New Files Tool Window

A new Files tool window lets you view open buffers, project files, and workspace files, sortable by file name or path. Includes incremental search and file name filters. This feature replaces the Select a Buffer dialog found in previous versions. To use the Files tool window, select **Document > List Buffers**, or use the **list\_buffers** command. Alternatively, you can select **View > Toolbars > Files**, or use the **activate\_files** command. See [Listing Open Files](#) for more information.

## Drag-and-Drop Support for KDE and GNOME

For KDE® and GNOME™ users, files can be opened in SlickEdit® by dragging and dropping them from the file manager into the SlickEdit editor pane.

## Windows Vista Support

SlickEdit® now works with the latest Microsoft® Windows operating system, Vista™.

## Line Ruler

The **Draw box around current line** option (**Tools > Options > General**, [General Tab](#)) has been enhanced. You can now choose a box with tab stops, syntax indent levels, or decimal points shown. See [Highlighting the Current Line](#) for more information.

## New Color Picker

This is a more up-to-date color picker that appears when you change the Foreground or Background colors on the [Color Settings Dialog](#) (**Tools > Options > Color**).

## Additional Enhancements

Along with the new features listed above, we have made many other enhancements to SlickEdit® 2007. The following are particularly noteworthy:

- **Classes tool window renamed as Symbols tool window** - The tool window known as “Classes” in previous versions has been renamed to “Symbols” (**View > Toolbars > Symbols**) to be more consistent with its functionality. It contains the symbol browser, which lists the symbols from all of the tag files. See [Symbols Tool Window](#) for more information.

- **Context Tagging® enhancements** - These include the new Find Symbol and Preview tool windows, as well as bug fixes and other minor enhancements. See [Find Symbol Tool Window](#) and [Preview Tool Window](#) for more information.
- **Enhanced color coding for selections** - Font colors and styles are now preserved in selections and for current line coloring (**Tools > Options > File Extension Setup > Advanced tab**, select **Current line**). Predefined color schemes (**Tools > Options > Color**) now include less-obtrusive selection background colors. For more information, see [Setting Fonts and Colors](#) and [Selections](#).
- **Quick Replace** - This new feature gets the current word or selection at the cursor, prompts for replacement text on the command line, then highlights each occurrence of the word and prompts if you want to replace the text. Right-click on a word or selection and select **Quick Replace**, or use the **quick\_replace** or **qr** commands. See [Quick Search and Replace](#) for more information.
- **Line/character count indicator** - The number of lines or characters in the current selection is now displayed in the editor status area (located along the bottom right edge of the editor). If there is no selection, the indicator is dimmed, with the text "No Selection." See [Counting Selected Lines and Characters](#) for more information.
- **Slick-C® enhancements** - Several enhancements have been made to the Slick-C macro programming language. Please see this post on the SlickEdit Community Forums for details: <http://community.slickedit.com/index.php?topic=691.0>.
- **Buffer navigation enhancements** - Two buffer navigation commands have been enhanced: **top\_of\_buffer** and **bottom\_of\_buffer** can now push bookmarks so you can get back quickly. This is particularly useful when you just need to jump to the top of the file to look at your #includes. See [Setting Bookmark Options](#) for more information.
- **New bookmark commands** - Two new bookmark commands have been added: **alt\_bookmark**, for setting a bookmark, and **alt\_gtbookmark**, for navigating to the bookmark. The purpose of these commands is so that you can bind them to keys, providing a way for you to have one type of keyboard shortcut for setting the bookmarks, naming them in the process, and another for navigating to the bookmarks. See [Named Bookmarks](#) for more information.
- **IPv6 support** - IPv6 (Internet Protocol version 6) hosts are now supported. See [Network Options Dialog](#) for more information.

## Documentation Revisions

The following revisions have been made to the documentation for SlickEdit® 2007:

- The Help system (**Help > Contents**) and *SlickEdit® User Guide* have undergone significant reorganization, with improved and expanded content.
- Many more cross-referencing links have been added.
- Color-coded tips and notes have been added.
- Documentation has been improved and expanded in many areas, including [Workspaces and Projects](#) and [Aliases](#).
- The *Getting Started Guide* no longer ships with the product. Instead, a [Quick Start](#) is included in the Help system and *SlickEdit® User Guide* (**Introduction > Quick Start**).



# Documentation and Conventions

---

## Accessing Documentation

Documentation for SlickEdit® is located in the `docs` directory on the root of the product CD. After SlickEdit is installed on your computer, these documents are installed in the `docs` subdirectory of the installation directory. The `docs` directory contains a PDF file of the following documents:

- The *SlickEdit® User Guide* - This guide provides comprehensive information about using the editor.
- The *Slick-C® Macro Programming Guide* - This guide contains details about how to write macros using the Slick-C macro programming language.
- Emulation charts for the following editors: BBEdit, Brief, CodeWarrior™, CodeWright®, CUA (default), Epsilon, GNU Emacs, ISPF, SlickEdit (Text Mode), Vim, Visual C++® 6, Visual Studio®, and Xcode®.

In addition to the documentation, SlickEdit provides a fully-functional built-in Help system. The Help system contents match the contents of the PDF documents, and also include categorized lists of C and Slick-C macro functions.

## Documentation Feedback

We welcome your comments and suggestions regarding our documentation. Please send feedback to [docs@slickedit.com](mailto:docs@slickedit.com).

## Documentation Conventions

The conventions listed below are used in SlickEdit® documentation:

### Default Emulation/Key Binding Mode

CUA is the default editor emulation mode. Therefore, key bindings and shortcuts listed in the documentation follow the CUA emulation.

### Platform-Specific Notes

Platform-specific notes appear throughout the documentation and are included for Microsoft Windows, UNIX® (which includes Linux®), and Mac OS® X.

### Mac OS X Notes

SlickEdit® can be used on a Mac OS X operating system, displayed as an X11 application. The X11 application can be customized to accommodate your preferences.

Throughout the user documentation, information that is available for Linux and UNIX operating systems will be the same or similar when using SlickEdit on a Mac OS X operating system. The documentation contains specific information for the Mac OS X operating system where relevant. Mouse and keyboard short-

cuts in the documentation are written for Microsoft Windows, but can be adapted for Mac OS X using the information below.

**TIP** In SlickEdit, a key or key sequence that is bound to an operation is called a *key binding*. See [Using the Mouse and Keyboard](#) and [Key and Mouse Bindings](#) for more information.

Mouse and keyboard shortcuts on Windows and Mac OS X have the following similarities:

- The Command (Cmd) key on the Mac keyboard functions the same as the Windows Control (Ctrl) key.
- The Cmd key plus a mouse click on the Mac keyboard functions the same as right-clicking the mouse on Windows.
- Single-button mouse support (available through the preferences menu for the X11 application).

The following table shows some of the differences between mouse and keyboard shortcuts on a Windows operating system and the Mac OS X operating system:

Microsoft Windows	Mac OS X
Right-click with the mouse	Cmd+Click
Left-click with the mouse	Single mouse click
Ctrl+F (Find)	Cmd+F (Find)
Ctrl+G (Find again)	Cmd+G (Find again)
Print Screen	Cmd+P (Print)

**NOTE** Until they are disabled, X11 keyboard shortcuts override SlickEdit shortcuts. For example, in SlickEdit, Cmd+Q quits the SlickEdit application, but in X11, Cmd+Q quits the X11 application. To disable X11 keyboard shortcuts:

1. With SlickEdit in the foreground, go to the **X11** menu.
2. Select **Preferences** (Cmd+Comma).
3. On the Preferences dialog, under the **Input** category, uncheck the option **Enable keyboard shortcuts**.

## Menus and Dialogs

Instructions for navigating to items accessed from the main menu are written in the form **MainMenuitem > SubMenuitem**. For example, the text “choose **Search > Find**” indicates that you should first select **Search** from the main menu, then select **Find** from the Search submenu.

Our documentation structure is set up so that instructions for using the product make up the bulk of the content, while listings of dialog boxes and options can be found in the [Menus, Dialogs, and Tool Windows](#) chapter. Buttons on dialogs, such as **OK**, **Close**, and **Help**, are not usually documented since the meaning is obvious.

## Code Syntax Conventions

- Commands, switches, keywords, properties, operators, options, and text to be typed by the user are shown in **bold** type.
- User-input variables and placeholders are shown in *italic* type.
- Code samples and file names are displayed in a `monospaced` font.
- File extensions are written with an UPPERCASE font.
- SlickEdit® commands that contain two or more words are written with underscore separators: for example, **cursor\_down**. Note that in the user interface, however, these commands are displayed with hyphen separators: for example, **cursor-down**. Both of these forms work in SlickEdit, so you can use whichever style you prefer.



## Supported Languages and Environments

This section outlines the languages and file types supported by each SlickEdit® feature, including special features for the Mac®, as well as supported emulations, project types, and version control systems.

### Supported Languages and File Types

The table below indicates the languages and file types that support key SlickEdit® features. Features that are not language-specific, such as DIFFzilla®, are not listed here.

Feature	Languages
<b>Automatic Syntax Expansion</b>	ActionScript, Ada, AWK, C, C#, C++, CFML, CFScript, Ch, COBOL, DTD, Fortran, HTML, IDL, InstallScript, J#, Java, JavaScript™, JSP™, Objective-C, Pascal, Perl, PHP, PL/SQL, PV-WAVE®, Python™, REXX, Ruby, SAS®, Slick-C®, Tcl, Transact-SQL®, VBScript, Verilog®, VHDL, Visual Basic®, Visual Basic .NET™, XML, XSD
<b>Code Beautifier</b>	ActionScript, Ada, C, C#, C++, CFML, HTML, Java, JavaScript, JSP, Slick-C, XML, XSD
<b>Color Coding</b>	ActionScript, Ada, ANTLR, AppleScript®, Assembly Language, AWK, Bourne shell scripts, C, C Shell, C#, C++, CFML, CFScript, Ch, CICS®, COBOL, DB2®, DTD, Fortran, High Level Assembler, HTML, IDL, InstallScript, J#, Java, JavaScript, JCL, JSP, Lex, Modula-2, Objective-C, Pascal, Perl, PHP, PL/I, PL/SQL, PowerNP™ Assembler, Progress® 4GL, PV-WAVE, Python, REXX, Ruby, SAS, Slick-C, Tcl, Transact-SQL, VBScript, Verilog, VHDL, Visual Basic, Visual Basic .NET, Windows batch files, x86 Assembly, XML, XSD, Yacc
<b>Context Tagging®: Auto List Members, Auto Parameter Info</b>	ActionScript, Ada, C, C#, C++, CFML, CFScript, Ch, CICS, COBOL, DTD, High Level Assembler, HTML, IDL, InstallScript, J#, Java, JavaScript, JSP, Objective-C, Pascal, Perl, PHP, PL/I, PV-WAVE, Python, Ruby, Slick-C, VBScript, Verilog, VHDL, Visual Basic .NET, XML, XSD
<b>Context Tagging: Auto List Parameters</b>	ActionScript, C, C++, Ch, J#, Java, Slick-C
<b>Javadoc™ Editor</b>	ActionScript, C, C#, C++, J#, Java, JavaScript, Slick-C
<b>Select/Hide Code Block</b>	ActionScript, Ada, C, C#, C++, CFML, Ch, COBOL, DB2, Fortran, High Level Assembler, HTML, IDL, InstallScript, J#, Java, JavaScript, Modula-2, Objective-C, Pascal, Perl, PHP, PL/SQL, Slick-C, Tcl, Visual Basic, Visual Basic .NET, XML, XSD

Feature	Languages
<b>Selective Display</b> Collapsible code block and function bodies.	ActionScript, Ada, ANTLR, C, C#, C++, CFML, CFScript, Ch, CICS, COBOL, DB2, DTD, Fortran, High Level Assembler, HTML, IDL, InstallScript, J#, Java, JavaScript, JCL, JSP, Lex, Modula-2, Objective-C, Pascal, Perl, PHP, PL/I, PL/SQL, PowerNP Assembler, PV-WAVE, Python, REXX, Ruby, SAS, Slick-C, Tcl, Transact-SQL, VBScript, Verilog, VHDL, Visual Basic, Visual Basic .NET, x86 Assembly, XML, XSD, Yacc
<b>SmartPaste®</b> Pasted code re-indents to correct level.	ActionScript, AWK, C, C#, C++, IDL, InstallScript, J#, Java, JavaScript, JSP, Objective-C, Pascal, Perl, PHP, PL/I, PV-WAVE, Python, Ruby, Slick-C, Tcl
<b>Source Code Navigation and Lookup</b> Includes Class, Defs, Preview, References, and Symbols tool windows, as well as symbol navigation.	ActionScript, Ada, ANTLR, Assembly Language, AWK, Bourne shell scripts, C, C Shell, C#, C++, CFML, CFScript, Ch, CICS, COBOL, DB2, DTD, Fortran, High Level Assembler, HTML, IDL, InstallScript, J#, Java, JavaScript, JCL, JSP, Lex, Makefile, Modula-2, Objective-C, Pascal, Perl, PHP, PL/I, PL/SQL, PowerNP Assembler, Progress 4GL, PV-WAVE, Python, REXX, Ruby, SAS, Slick-C, Tcl, Transact-SQL, VBScript, Verilog, VHDL, Visual Basic, Visual Basic .NET, Windows batch files, x86 Assembly, XML, XSD, Yacc
<b>Syntax Indenting</b> Cursor is placed at correct indent level.	ActionScript, Ada, AWK, C, C#, C++, CFML, CFScript, Ch, COBOL, Fortran, HTML, IDL, InstallScript, J#, Java, JavaScript, JSP, Objective-C, Pascal, Perl, PHP, PL/SQL, PV-WAVE, Python, REXX, Ruby, SAS, Slick-C, Tcl, Transact-SQL, VBScript, Verilog, VHDL, Visual Basic, Visual Basic .NET, XML, XSD

## Special Features for Mac OS X

The following features are available for programmers using Mac OS X:

- Xcode project support
- Objective-C language support
- Emulations for CodeWarrior, BBEdit, and Xcode
- Mac OS X default line endings are the same as UNIX

**NOTE** C++ Refactoring is supported on Mac OS X, but not for Objective-C. While Objective-C is closely related to C++, the refactoring engine has not been tuned to handle all syntactic differences.

See [Platform-Specific Notes](#) for information on Mac OS X documentation conventions and keyboard and mouse commands.

## Embedded Languages

SlickEdit® recognizes languages embedded in HTML, COBOL, Perl scripts, and UNIX shell scripts. When editing embedded languages, all language-sensitive features are supported, including Context Tagging®, SmartPaste®, syntax expansion, syntax indenting, and color coding. In fact, Context Tagging picks up

embedded tags. For example, the Defs tool window displays function names if any exist. Embedded language colors are user-defined.

## Embedded Languages in HTML

SlickEdit® supports any embedded language in HTML. However, Web browsers usually only support VBScript, JavaScript, and/or Java, while Web servers typically support VBScript, Java, or PHP. The following screen is an example of VBScript, JavaScript, and Java embedded in HTML:

```
<% @ LANGUAGE="VBSCRIPT" %>
<%setupParams%><%if request.form("Add") = "Add" then addRecord%>
<%if request.form("Delete") = "Delete" then deleteRecord%></p>
<%
OPTION EXPLICIT
DIM L_Guestbook
L_Guestbook = "Guest Book"
' $Date: 10/20/97 4:17p $
' $ModTime: $
' $Revision: 17 $
' $Workfile: guestbk.asp $
IF request.QueryString("message") <> "" Then
    intMID= request.QueryString("message")
End If
%>
<SCRIPT LANGUAGE="javascript">
<!--
function turnRed() {
    what = window.event.srcElement;
    if (what.tagName == "IMG") {
        what.src= "red.gig";
        window.event.cancelBubble = true;
    }
}
//-->
</SCRIPT>

<java type="print">
public class junk3 {
    public static void main (String args[]) {
    }
}
</java>
```

## Embedded Languages in Perl and Other Scripting Languages

To allow SlickEdit® to recognize embedded source in a Perl script or UNIX shell, prefix the here-document terminator with the color coding lexer name. The following Perl example shows HTML embedded in a Perl script. Unknown languages are color coded in string color.

```
print <<HTMLEOF
<HTML><HEAD><TITLE>...</TITLE></HEAD>
<BODY>
...
</BODY>
</HTML>
HTMLEOF
```

## Supported Editor Emulations

SlickEdit® provides keyboard emulations for the following editors:

- BBEEdit
- Brief
- CodeWarrior
- CodeWright
- CUA (default)
- Epsilon
- GNU Emacs
- ISPF
- SlickEdit® (text mode edition)
- Vim
- Visual C++ 6
- Visual Studio
- Xcode

See [Emulations](#) for more information.

## Supported Project Types

SlickEdit® supports the creation of projects that use the Microsoft Visual C++ Toolkit and Microsoft .NET Framework Software Development Kit. Many other project types are supported, including many for Java. For a list of all supported types, see the list box on the **Project > New** dialog. For information about working with projects, see [Workspaces and Projects](#).

## Supported Version Control Systems

SlickEdit® provides support for the version control systems listed below. To learn more about working with version control in SlickEdit, see [Version Control](#).

- CCC/Harvest
- ClearCase®
- ComponentSoftware RCS
- CVS
- MKS Source Integrity®
- Perforce
- PVCS®
- RCS
- SCC
- StarTeam®
- Subversion
- TLIB
- Visual SourceSafe®



# Install/Uninstall

---

The information below describes how to correctly install and uninstall SlickEdit®.

## Installing SlickEdit®

Licensing options and system requirements can be found on the SlickEdit product Web site: [www.slickedit.com/slickedit](http://www.slickedit.com/slickedit). The type of licensing that you purchased (Named User or Concurrent User) determines the type of installation. Unattended installation is also available.

**NOTE** When you start SlickEdit for the first time after an installation, several dialog boxes automatically appear that require action. See [Running SlickEdit® for the First Time](#) for more information.

## Named User Installation

If you have purchased a Named User license, use the information below to install SlickEdit® on your Windows, Linux, UNIX, or Mac OS X platform.

### Windows

When you place the CD in the drive, the setup program is started automatically. Use the setup wizard to complete the installation. To start setup if it does not start automatically, or if you are installing a downloaded product, double-click the .msi file to start the setup.

**NOTE** The Windows installer will not allow you to install more than one copy of SlickEdit® with the same version number on the same machine. For example, you can have SlickEdit v11.0.2 and SlickEdit 2007 installed, but you cannot install two copies of SlickEdit 2007. To create a second installation, copy the installation directory to another location.

### Linux/UNIX

Complete the following steps to install SlickEdit® on a computer running Linux or UNIX:

1. Mount the CD-ROM block device to a local directory. To prevent file name casing issues on HP-UX®, use the `-o cdcase` option when mounting.
2. At an xterm or full-screen prompt, change to the applicable UNIX platform directory on the CD-ROM (where `<cdrom>` is your CD-ROM mount point):
  - o AIX - `<cdrom>/aix/rs6000`
  - o HP-UX - `<cdrom>/hpux/hp9000`
  - o IRIX - `<cdrom>/irix`
  - o Linux x86 - `<cdrom>/linux/x86`
  - o Solaris™ Sparc® - `<cdrom>/solaris/sparc`
  - o Solaris x86 - `<cdrom>/solaris/x86`
3. Start the installation at the prompt: `# ./vsinst`

### Mac

Complete the following steps to install SlickEdit® on a computer running Mac OS X:

1. Ensure that X11 is fully installed on your system (see the Note below).
2. Browse to the `mac` folder on the product CD.
3. Double-click `slickedit.dmg`. A disk image is mounted, with Finder™ displaying the contents.
4. Double-click `slickedit.pkg`, and the installation process begins.

The default installation path is `/Applications`.

After installing SlickEdit, if you want to use SlickEdit keyboard shortcuts instead of X11 shortcuts, you will need to disable X11 keyboard shortcuts. See [Mac OS X Notes](#) for instructions.

**NOTE** SlickEdit runs as an X11 application, so X11 must be installed prior to installing SlickEdit. To check if X11 is installed, select **Finder > Applications > Utilities**. A file named `x11.app` should exist at this location. Otherwise, you will need to install X11 from your OS X installation CD:

1. Using installation disk 1, run the `Optional Installs.mpkg` utility.
2. At the **Install Type** phase, expand the **Applications** tree and select **X11**.
3. Continue with the X11 installation.

## Concurrent User Installation

SlickEdit® utilizes the FLEXnet™ Publisher license manager for Concurrent User licenses. If you have purchased this type of license, you can find complete instructions at [www.slickedit.com/vsflex](http://www.slickedit.com/vsflex).

## Unattended Installation

Using command line switches and arguments, SlickEdit® can be installed in an unattended manner. This is useful for network administrators to deploy SlickEdit on multiple client machines with standardized settings. Instructions can be downloaded in PDF format from [www.slickedit.com/docs/unattended](http://www.slickedit.com/docs/unattended).

## Uninstalling SlickEdit®

To remove SlickEdit from your computer, use the information below specific to your platform.

**NOTE** Uninstalling SlickEdit does not automatically remove the user configuration directory. See [User Configuration Directory](#) for more information.

## Windows

If your computer is running Windows, complete the following steps to uninstall SlickEdit®:

1. From the Windows Control Panel, open **Add or Remove Programs**.
2. Click **Change or Remove Programs**.
3. Select the SlickEdit installation that you want to remove.
4. Click **Change/Remove** or **Remove**. This will delete the installation directory as well as any registry settings. You may be prompted to reboot for the changes to take effect.

Alternatively, double-click on the original `.msi` installation file, located on the product CD or in your product installation download, and select **Remove**.

## Linux/UNIX

To uninstall SlickEdit® on a computer running Linux or UNIX, simply delete the installation directory.

## Mac

To uninstall SlickEdit® on a computer running Mac OS X, open the `Application` folder, and drag the **SlickEdit** icon to the Trash, then empty the Trash.



## Starting and Exiting SlickEdit®

---

### Starting the Program

To manually run SlickEdit® on a computer that is using a Microsoft Windows or Mac OS X operating system, use the SlickEdit icon displayed on your desktop or launch it from the Start menu (or equivalent).

On Linux or UNIX, run SlickEdit using the **vs** binary located in `...slickedit/bin`. For a complete list of options that can be used with the **vs** command, see [Invocation Options](#).

#### TIPS

- You can make settings so that certain items such as files, clipboards, and Selective Display are restored each time SlickEdit is started. See [Restoring Settings on Startup](#) for more information.
- You can also set a macro to be run upon startup. See the section “Hooking Exit and Other Events” in the *Slick-C® Macro Programming Guide* for more information.

### Running Multiple Instances

You can have two instances of SlickEdit® running at the same time. On Windows or Mac OS X, right-click on the SlickEdit icon displayed on the desktop. In the **Target** field, append the text **+new** to the end of the existing text.

On UNIX or Linux, append **+new** to the invocation command (for example, **vs +new**).

**CAUTION** SlickEdit cannot save configuration changes when another instance is running. If you attempt to close an instance that contains configuration changes, while another instance is running, a save-failed message is displayed, then a prompt asks whether or not to exit anyway.

### Running SlickEdit® for the First Time

The first time SlickEdit is started after an installation, several dialog boxes are automatically launched that require action:

**NOTE** For Mac OS X, SlickEdit requires a one-time background caching of the font list. This will take a few minutes, after which SlickEdit will start normally.

1. **Release Notes** - The SlickEdit dialog displays the Release Notes for this version. The **Program Information tab** contains the serial number, version, and other details about your product. This information is used by Product Support should a problem arise (the **Copy to Clipboard** button is useful in this case). The **License Agreement tab** displays the “End User License Agreement” (EULA). Information about contacting SlickEdit is shown on the **Contact Information tab**. The SlickEdit dialog can be displayed at any time by selecting **Help > About SlickEdit**, or by using the **version** command. Click **OK** to proceed.
2. **Emulation** - The process of imitating another program is called *emulation*. SlickEdit provides emulations of key bindings for 13 editors, so that you can use the style to which you are accustomed. See [Emulations](#) for more information. Use the default (CUA) emulation or select another, then click **OK**.

3. **Product Registration** - For Maintenance and Support customers, registration of your product is required in order to receive priority notification of updates and upgrades, and access to product downloads. To register now, click **Register**. To skip the registration process for now, click **Close**. You can register at any time by selecting **Help > Register Product**, or by using the **online\_registration** command.
4. **Create Tag Files for Run-Time Libraries** - This dialog prompts for tag generation. Context Tagging® is required so that the most powerful features in SlickEdit will work, such as [Code Navigation](#) and [Symbol Browsing](#). To use the default code base directories and tag file destination paths for C, C++, Java, and .NET, click **Create tag file(s)**. To tag source files for other languages, some configuration is required. For more information, see [Building and Managing Tag Files](#).

**NOTE** In order for SlickEdit to help you work more quickly and efficiently (such as using [Code Navigation](#) and [Symbol Browsing](#) mentioned above), it is important to create these tag files. This may take several minutes, depending on the size of your code base. You can create tag files at any time by clicking the **Auto Tag** button on the Context Tagging® - Tag Files dialog (**Tools > Tag Files**).

5. **Cool Features** - The Cool Features dialog displays information about some of the best SlickEdit features and contains options for viewing feature settings, getting help on a feature, and viewing a demo of the feature. There is also an option labeled **Show on startup**: deselect this option if you do not want to see the Cool Features dialog each time SlickEdit is launched. See [Cool Features Dialog](#) for more details. Click **Close** to close the dialog.

## Exiting the Program

To safely exit SlickEdit®, from the menu choose **File > Exit** (Alt+F4). You can also use the SlickEdit command line to exit. Activate the command line by pressing the Escape key or by clicking on the message line with the mouse, then type **safe\_exit**. If files have not been saved or closed upon exit, you will be prompted with a dialog to save or discard any changes.

For more information, see [Creating, Opening, and Saving Files](#).

### TIPS

- You can make settings so that certain items such as files, clipboards, and Selective Display are restored each time SlickEdit is started. See [Restoring Settings on Startup](#) for more information.
- You can also set a macro to be run upon exit. See the section “Hooking Exit and Other Events” in the *Slick-C® Macro Programming Guide* for more information.

## Exiting with Modified Buffers

If files have not been saved or closed upon exit, you will be prompted with a dialog to save or discard any changes. The buffer names in the list box are buffers which have not been saved. See [Exiting with Modified Buffers Dialog](#) for option descriptions. See also [Saving Files](#).

## Default Exit Options

To access default options for saving configuration changes, choose **Tools > Options > General**, then select the **Exit tab**. See [Exit Tab](#) for descriptions of these options.

# Help and Product Support

---

There are several ways to get help when you are using SlickEdit®:

- **Use the Help system** - See [Using the Help System](#) below.
- **Search the FAQ** - A list of frequently asked questions and answers is available on our Web site. To invoke a browser that pops directly to this page, from the SlickEdit menu, choose **Help > Frequently Asked Questions** (or use the **goto\_faq** command). You can also visit the FAQ from our Product Support Web page at [www.slickedit.com/support](http://www.slickedit.com/support).
- **Use the Community Forums** - Search for or post your question on the SlickEdit Community Forums to seek help from other SlickEdit users. The forums are located at <http://community.slickedit.com>.
- **Contact Product Support** - See [Contacting Product Support](#) below.

## Using the Help System

When SlickEdit® is installed, the searchable Help system is installed with the product. The contents of the Help system are the same as the contents of the PDF documents located in the `/docs` installation subdirectory (see [Accessing Documentation](#)), and also include categorized lists of C and Slick-C® macro functions.

The Help system can be accessed in several ways:

- To view the Help Table of Contents, from the main menu choose **Help > Contents**.
- To search for an item in the Index, from the main menu choose **Help > Index**.
- To search for keywords in the Help system, from the main menu choose **Help > Search**.
- To invoke the Help entry for the item at the cursor in files, tool windows, menus, and dialog boxes, press F1 (or use the **wh** command). For C++ compiler packages, this includes help on the Windows API, Standard C library, and Microsoft C Foundation Classes (if you have MS VC++). In any emulation, you can click the Question Mark icon located on the [Standard](#) toolbar for help.

**TIP** You can configure F1 to access Microsoft's documentation library rather than SlickEdit's built-in help. You can also specify multiple word help files for the **wh** command. See [Advanced Help Configuration](#) for more information.

## F1 Index Help

F1 Help, also known as *word help* or *SDK help*, searches external help files for information about a symbol. To search these files for the keyword at the cursor, press F1, or from the main menu choose **Help > F1 Index Help** (or use the **help\_index** command). This will invoke the Help Index dialog box which scans a `help.idx` file for help files that contain a prefix match or exact match of the help keyword specified. You can specify other keywords by directly typing them into the **Help Keyword** field on this dialog.

To configure the help files used for this type of help, or to configure access to multiple help files (such as MSDN help), see [Advanced Help Configuration](#).

## UNIX, Linux, and Mac OS X F1 Help

UNIX, Linux, and Mac OS X users can get **man** page help on the word at the cursor by pressing F1 (executes the **wh** command). When you are editing a Slick-C® macro, pressing F1 brings up macro language

help for the word at the cursor. You will find this especially useful for getting help on macro source generated during macro recordings. F1 works for the default CUA emulation and most other emulations as well.

### Help Index Options

The options on the Help Index dialog are described in the topic [Help Index Dialog](#).

### Help Key Shortcuts

The table below lists keyboard shortcuts that can be used when working with the Help system.

Help Key	Description
Tab	Next hypertext item
Shift+Tab	Previous hypertext item
Enter	Go to hypertext item
Alt+I	Displays help index
Ctrl+S or Alt+S	Display Help Find dialog box
Alt+Dot	Next topic
Alt+Comma	Previous topic
Ctrl+C or Ctrl+Ins	Copy selection to clipboard
Home	Start of topic
End	End of topic
Up arrow	Scroll up
Down arrow	Scroll down
PgUp	Page up
PgDn	Page down
Click+Drag	Selects text
Shift+Click+Drag	Extends a selection

### Supported Web Browsers

The Help system runs on many browsers. If you are using Microsoft Windows, you must have Microsoft Internet Explorer® installed for the Help system to function. Based on our testing on other operating systems, we recommend the following:

- If you are using a UNIX, Linux, or another UNIX-based operating system, use the Firefox® browser.



- If you are using Macintosh®, use the Safari™ browser.

**NOTE** SlickEdit® provides a way to configure what Web browser to use when SlickEdit needs to launch a browser (such as when you select **Help > SlickEdit Support Web Site**). This configuration does not apply to the Help system. For more information, see [Web Browser Setup Dialog](#).

## Product Support

Patches, macros, FAQs, and more are available on SlickEdit's Product Support Web site. You can launch the Web site in a browser through the SlickEdit® menu item **Help > SlickEdit Support Web Site**. You can access the FAQs Web page directly by using the menu item **Help > Frequently Asked Questions**.

## Product Registration and Updates

Registering your product allows SlickEdit to provide you with automatic notification of free updates and new releases, and enters your name into a weekly drawing for a SlickEdit gift pack. To register your product from within SlickEdit®, from the menu choose **Help > Register Product**, then follow the steps indicated.

### Using the Update Manager

The Update Manager is used to indicate when you need to update the version of SlickEdit® that is installed on your system. A dialog box is displayed only if there are new updates. To manually check for updates, from the main menu, select **Help > Product Updates > New Updates**. If you need an updated version of SlickEdit, visit our Web site at [www.slickedit.com](http://www.slickedit.com). To set the frequency of automatic checking by the Update Manager and to change proxy settings, choose **Help > Product Updates > Options**.

## Maintenance and Support Service

Maintenance and Support Service offers the following benefits:

- 12 months of unlimited technical support via telephone or e-mail.
- Access to new releases, upgrades, patches, and fixes at no additional charge.
- The ability to participate in SlickEdit® beta programs and receive a free copy of SlickEdit beta software as a preview of the next release.

To check the status of your Maintenance and Support Service from within SlickEdit, choose the menu item **Help > Check Maintenance**. This will launch the SlickEdit Maintenance and Support Web page in a browser, showing the status of your service.

To subscribe to Maintenance and Support Service or learn more, contact SlickEdit Sales ([sales@slickedit.com](mailto:sales@slickedit.com)).

## Hot Fixes

Hot fixes are small, localized changes to address a specific problem with the previous release. They can consist of Slick-C® modules, configuration files, installation files, or DLL files. Hot fixes are distributed as ZIP files and made available on the SlickEdit Support Web site at [www.slickedit.com/support](http://www.slickedit.com/support). For convenience, a number of hot fixes may be aggregated into a single ZIP file.

**NOTE** If a user loads a hot fix on a multi-user installation of SlickEdit®, only that user will be updated. To apply a fix to a multi-user installation, please contact Product Support for assistance.

## Installing Hot Fixes

To install a hot fix, complete the following steps:

1. Save the ZIP file to any location on your computer.
2. From the SlickEdit® menu, choose **Help > Product Updates > Load Hot Fix** (or use the command **load\_hotfix**). The Apply Hot Fix dialog appears.
3. Browse to and select the hot fix ZIP file, then click **OK**.
4. A confirmation prompt appears describing the hot fix. Click **Yes**. The installation starts.

Details about the installed fix will be sent to the [Output](#) tool window.

## Listing Installed Hot Fixes

To see the list of hot fixes installed, from the SlickEdit® menu choose **Help > Product Updates > List Installed Fixes** (or use the command **list\_hotfixes**). A summary sheet appears with the location of the hot fix ZIP file, its revision number, the date it was published, and its description.

## Unloading Hot Fixes

To unload a hot fix, use the **unload\_hotfix** command from the SlickEdit® command line. At the prompt, select the hot fix to unload and click **OK**.

**CAUTION** Unloading a hot fix will reload the original files distributed with the previously installed release of SlickEdit. If other hot fixes include the same file or are dependent on the unloaded files, SlickEdit may behave unpredictably. If more than one hot fix has been installed, you may need to reinstall the other hot fixes after removing one of them.

## Contacting Product Support

To contact Product Support, use the menu item **Help > Contact Product Support**. This will automatically gather your program information, such as the current version and serial number, that helps us to better answer your questions.

You can also send e-mail directly to [support@slickedit.com](mailto:support@slickedit.com). Be sure to include the program information, which includes the serial number, version, and other details about your product. You can access this information from the menu item **Help > About SlickEdit**.

To speak to a member of our Product Support team, call the Support line at 1.919.473.0100. Telephone support is only available during business hours for customers with a valid Maintenance and Support Service Agreement.

# Quick Start

SlickEdit® is one of the most powerful programming editors available today, and one of the most flexible. SlickEdit contains hundreds of options to let you work *your* way. Most people don't have time to read the whole user guide. Take a few minutes to go through the Quick Start, and you'll be up and running with SlickEdit in no time.

The information is divided into the following steps:

1. [Set Your Preferences](#)
2. [Set Up a Workspace and Project](#)
3. [Start Coding](#)

## Set Your Preferences

After installation, you may want to change some default options to match your own coding preferences. This section contains a list of settings that are common in many code editors. This list is not comprehensive, so we encourage you to browse all of the options screens to set preferences for something that may not be mentioned here. Most options can be accessed through the **Tools > Options** menu item.

Option settings are divided into two categories: [General Options](#) and [Extension-Specific Options](#).

### General Options

General options affect all language extensions. You may want to look through all of the dialogs mentioned to see if there are any other settings you want to make. To see a listing of all of the option dialogs and their descriptions, see the appropriate topics in the chapter [Menus, Dialogs, and Tool Windows](#).

- **Changing the emulation** - During the product installation, you are prompted to choose the editor emulation. The default is CUA. To change the emulation at any time, choose **Tools > Options > Emulation**.
- **Maximizing the first window** - By default, when an editor window is opened, it is displayed as floating. To have editor windows maximized when they are first opened, choose **Tools > Options > General**, select the **General tab**, then select the option **Maximize first window**.
- **Expanding/collapsing with a single click** - Selective Display plus and minus bitmaps can be expanded or collapsed with a single click rather than a double-click. To specify this option, choose **Tools > Options > General**, select the **General tab**, then select the option **Expand/collapse single click**.
- **Clicking past the end of a line** - To have the ability to place the cursor past the end of a line, choose **Tools > Options > General**, select the **General tab**, then select the option **Click past end of line**.
- **Specifying cursor up/down behavior** - By default, **cursor\_up** and **cursor\_down** commands go to the same column of the next or previous line, unless that line is shorter than the current column, in which case the cursor is placed at the end of the line. To have the cursor placed in virtual space at the end of the line, choose **Tools > Options > Redefine Common Keys**, then uncheck the option **Up/Down on text**.
- **Changing the line insert style** - In code, a line of text is a meaningful unit of functionality. So, SlickEdit® treats line selections differently than character selections. Line selections are pasted either above or below the current line, saving you from tediously positioning the cursor at the beginning or end of a line prior to pasting. To specify where line selections are pasted, choose

**Tools > Options > General**, select the **More** tab, then set the **Line insert style** option to **Before** or **After**. The default is **After**.

- **Setting color schemes and fonts** - Predefined color schemes, as well as individual settings, are available for changing the colors of screen elements. To use a different color scheme, choose **Tools > Options > Color**, then click the Schemes button. Select a scheme that you like from the **Color scheme** drop-down list. To change the fonts used for screen elements, choose **Tools > Options > Font**.
- **Associating file types** - Files with certain extensions can be associated with SlickEdit, so that when those files are opened from Windows Explorer or the file manager, they run in the SlickEdit application. To set up these associations, choose **Tools > Options > Associate File Types**.

## Extension-Specific Options

These options are specific to language file extensions, and are available on the Extension Options dialog (**Tools > Options > File Extension Setup**). When the Extension Options dialog is displayed, before setting the options, select the extension you wish to affect from the **Extension** drop-down list.

In addition to the options described below, more settings for the selected language extension are available by pressing the **Options** button on the Extension Options dialog. Because each of these dialogs is different based on the language extension, we recommend that you look through these dialogs for any settings that you may want to make.

To see a listing of all of the option dialogs and their descriptions, see topics in the chapter [Menus, Dialogs, and Tool Windows](#).

- **Changing the brace style** - To change the brace style used for C, C++, C#, Java, and other languages that use braces, click the **Options** button on the Extension Options dialog, then specify the **Begin-End Style** that you want to use.
- **Changing the tab and indent styles**
  - **Indenting with spaces** - By default, when you press the Tab key to indent, literal spaces are inserted. This is a feature called *Syntax Indent*. To change the amount of spaces, select the **Indent** tab, make sure the **Indent style** is set to **Auto**, then specify the amount of spaces in the **Syntax indent** text box.
  - **Indenting with tabs** - If you plan to indent your code using tabs, or if you will be editing files that already contain tabs, specify your tab preferences on the **Indent** tab. Select the option **Indent with tabs**, then specify the amount of spaces tab characters should have in the **Tabs** text box.

**NOTE** For C, C++, Java, and similar languages, you can find more indenting options by clicking the **Options** button on the Extension Options dialog.

- **Enabling/disabling Syntax Expansion** - When you type a keyword, such as "if" or "for", press the spacebar to expand that syntax element, inserting the rest of the **if** or **for** statement. This feature is called *Syntax Expansion*. To disable it, select the **Indent** tab, then deselect the option **Syntax expansion**.
- **Setting symbol navigation** - For C and C++, by default, with each attempt to navigate to a definition (Ctrl+Dot or **Search > Go to Definition**), you will be prompted for whether you wish to navigate to the definition (proc) or the declaration (proto). To specify that **Go to Definition** always navigates to one or the other, select the **Context Tagging** tab, then select one of the **Go to Definition** options.

- **Showing the info for a symbol under the mouse** - By default, as the mouse cursor floats over a symbol, the information and comments for that symbol are displayed. To turn this behavior off, select the **Context Tagging® tab**, then deselect the option **Show info for symbol under mouse**.
- **Configuring C/C++ preprocessing** - For C and C++, your source code base will typically include preprocessor macros that you use in your code for portability or convenience. For performance considerations, Context Tagging® does not do full preprocessing, so preprocessing that interferes with normal C++ syntax can cause the parser to miss certain symbols. To configure your preprocessing to avoid these omissions, see [C/C++ Preprocessing](#).

## Set Up a Workspace and Project

After configuring settings, you are ready to set up your workspaces and projects, unleashing the powerful file-leveraging capabilities of SlickEdit®.

A *workspace* defines a set of projects and retains the settings for an editing session. A *project* defines a set of related files that build and execute as a unit. For each project you can specify the set of files it contains, a working directory, a set of commands to build and execute the project, compiler options, and dependencies between other projects. A tag file for each project's source files is automatically created and maintained, enabling SlickEdit's advanced navigation and unique Context Tagging® lookup features.

For more detailed information than is provided here, see the following sections:

- [Workspaces and Projects](#)
- [Building and Compiling](#)
- [Running and Debugging](#)

### Create a New Workspace

To create a new workspace, complete the following steps:

1. From the menu, choose **Project > New**.
2. Select the **Workspaces tab**.
3. In the **Workspace name** text box, give a name to your workspace.
4. In the **Location** text box, type a path or use the **Browse** button to pick a location.

### Create a New Project

To create a new project, complete the following steps:

1. From the menu, select **Project > New**.
2. It is important that you select the correct project type. From the list box on the left side of the dialog, select the type of project that you want to use.
3. In the **Project name** text box, give the project a name.
4. In the **Location** text box, type a path or use the **Browse** button to pick a location. If the directory does not exist, a prompt appears to create it when you click **OK**.
5. In the **Executable name** text box, type the name of the executable file or output file.
6. Select **Add to current workspace**.
7. Specify whether this project depends on another project in this workspace by checking the **Dependency of** check box and selecting the depended on project from the drop-down list.
8. Click **OK**.

### Add Files to the Project

To add files to your new project, complete the following steps:

1. From the menu, choose **Project > Project Properties**.

2. Select the **Files tab**.
3. To add individual files, click **Add Files**, and select the files that you want to add.
4. To add the source files in a directory, click **Add Tree**. Then select the directory and the file filter that you want to use.
5. When you are finished adding files, click **OK**.

## Start Coding

After settings have been configured and a workspace and project are set up, you are ready to start coding. See the [Editing Features](#) chapter to learn more about how SlickEdit® can help in your everyday work.

Tutorials are available for C/C++ and Java that describe how to create, build, and run a sample Hello World program. See [Hello World Tutorial \(C/C++\)](#) or [Hello World Tutorial \(Java\)](#).

If you're not ready to get to work just yet, you may want to configure even more preference options. For information, see the [User Preferences](#) chapter.

# User Interface

This chapter contains the following topics:

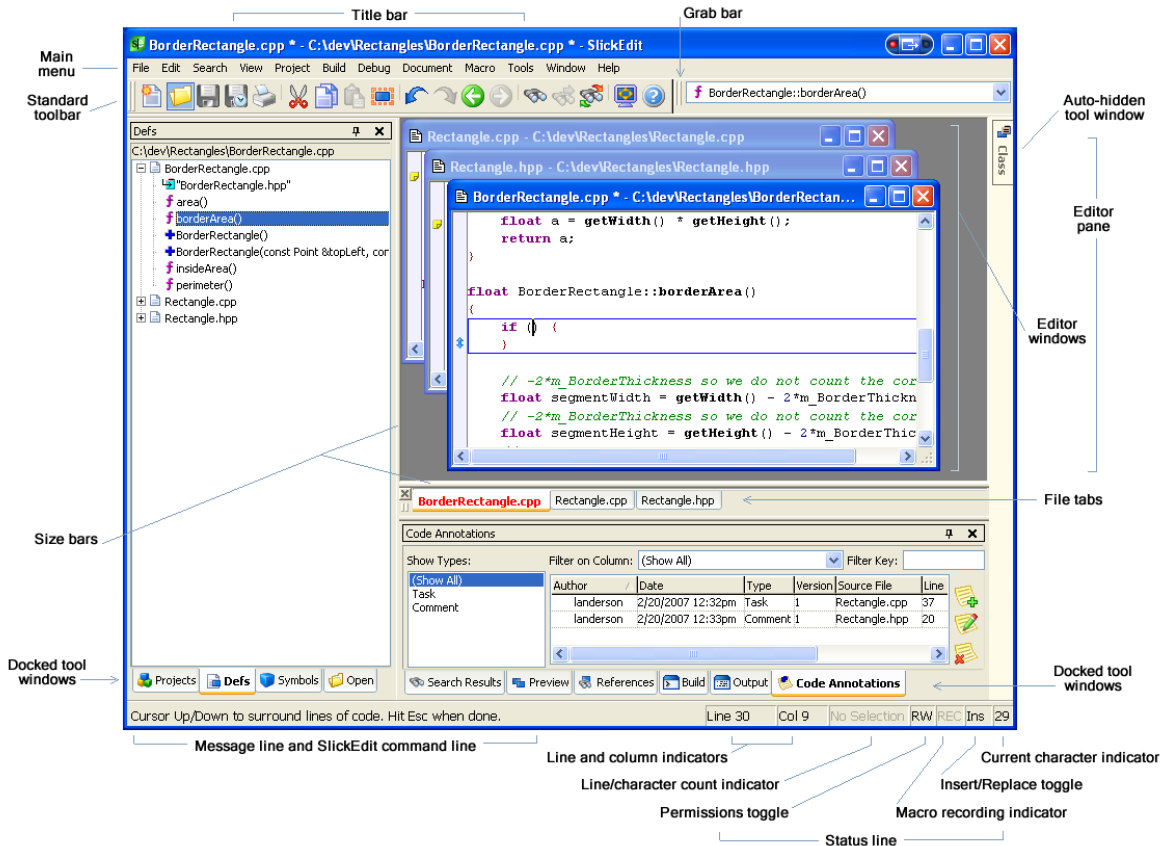
- [User Interface Overview](#)
- [Toolbars and Tool Windows](#)
- [Buffers and Editor Windows](#)
- [Accessing Menus](#)
- [The SlickEdit® Command Line](#)
- [Screen Management](#)
- [Using the Mouse and Keyboard](#)
- [Printing](#)





# User Interface Overview

SlickEdit® uses a Multiple Document Interface (MDI), common to Windows applications that support editing multiple documents. The outer window is called the MDI frame or window. For a non-clipped MDI child window, start a new instance of SlickEdit.



SlickEdit contains the following screen elements:

- **Title bar** - The title bar shows the product name and the name and path of the file currently in focus.
- **Main menu** - The main menu is displayed under the title bar (File, Edit, Search, etc.).
- **Standard toolbar** - Toolbars are groups of icons or buttons (called toolbar controls) that allow you to perform specific operations. The Standard toolbar is displayed and docked under the main menu by default. Toolbars can be moved by clicking and dragging the grab bars (or title bar if floating). See [Toolbars and Tool Windows](#) for more information.
- **Editor pane** - The editor pane is the viewing area within SlickEdit inside of which editor windows (files or buffers that are being editing) are floating or docked.
- **Editor windows** - Editor windows are files or buffers that are open for editing and are docked or floating inside of the editor pane. See [Buffers and Editor Windows](#) for more information.

- **File tabs** - File tabs are displayed for open buffers and files. Right-click on file tabs to display a menu of save, close, and window splitting options. Display of file tabs can be toggled from the main menu by selecting **View > Toolbars > File Tabs**.
- **Tool windows** - Tool windows are similar to toolbars except they may also contain settings and/or allow the viewing of information, and they can be docked. You can auto-hide a tool window by clicking on the Pin icon in the top right corner. See [Toolbars and Tool Windows](#) for more information.
- **Size bars** - Size bars indicate the parts of SlickEdit that can be resized. When the pointer becomes a double-headed arrow, click and drag to adjust the size in the direction indicated.
- **Message line and SlickEdit command line** - The message line appears at the bottom of the SlickEdit application window. It displays a single line of information, providing feedback from various operations in SlickEdit. The message line and the command line share the same screen space. As a result, when clicking the message line or invoking an editor command that requires the command line, the message line is hidden and the command line is displayed. See [The SlickEdit® Command Line](#) for more information.
- **Status line** - The status line holds the following indicators:
  - **Line and column indicators** - To the right of the message/command line are the line and column indicators for the current cursor position. Click on these indicators to move the cursor position.
  - **Line/character count indicator** - This displays the number of lines or characters in the current selection. This is useful to measure the length of a word or string, or the number of lines in a function. Click on the indicator or use the **select\_toggle** command to create successively larger common selections. For example, if you have a character selection, you can click on the indicator or use **select\_toggle** to extend the selection to include the entire word. See [Counting Selected Lines and Characters](#) for more information.
  - **Permissions toggle** - This area indicates the read/write permission setting of the file or buffer in focus. The letters **RW** indicate that the current file or buffer is read-write. The letters **RO** indicate that the file or buffer is read-only. Click on the letters to toggle between modes. You can configure the editor to prevent modification of read-only files. To access this setting, choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Protect read-only mode**. When this option is selected, the editor will not let you modify a file that is in read-only mode. The **save** command will always prompt for a different output file name if the file is in read-only mode.
  - **Macro recording indicator** - When a macro is being recorded, the recording indicator **REC** is active (not dimmed). Click on the indicator or use the **record\_macro\_toggle** command to toggle recording on and off. See [Recorded Macros](#) for more information.
  - **Insert or Replace toggle** - The Insert/Replace toggle is located to the right of the recording indicator. The letters **Ins** indicate that the editor is in Insert mode (default). In Insert mode, typing a character pushes characters at and after the cursor to the right. The letters **Rep** indicate that the editor is in Replace mode. In Replace mode, typing a character replaces the character under the cursor. Click on the letters to toggle between modes. To start in Replace mode instead of the default Insert mode each time the editor is invoked, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Deselect the option **Start in insert mode**.
  - **Current character indicator** - When editing an SBCS/DBCS mode file or non-Unicode file, the current character is displayed in hexadecimal format. If the current character is a double byte character (DBCS), then two bytes and its Unicode equivalent are displayed in hexadecimal (95 74 U+4ED8). When editing a Unicode file, the current composite character is dis-

played in hexadecimal. The indicator field is blank when the cursor is past the end of the line. See [Encoding](#) for more information about Unicode.



## Toolbars and Tool Windows

---

Toolbars are groups of icons or buttons (called toolbar controls) that allow you to perform specific operations. Tool windows are similar to toolbars except they may also contain settings and/or allow the viewing of information. Both toolbars and tool windows can be docked. For documentation purposes, the terms “toolbars” and “tool windows” are often used interchangeably.

## Displaying Toolbars and Tool Windows

By default, the [Build](#), [Output](#), [Preview](#), [References](#), and [Search Results](#) tool windows are docked into a tab group on the bottom of the editor, while the [Defs](#), [FTP](#), [Open](#), [Projects](#), and [Symbols](#) tool windows are docked into a tab group on the left side of the editor.

There are many more toolbars and tool windows that are not displayed by default. You can view and toggle the display of these by choosing **View > Toolbars**, then selecting the toolbar or tool window to display or hide.

**NOTE** Some tool windows that are used for debugging are only available when the editor is in debug mode. See [Debug Toolbars and Tool Windows](#) for a list.

## Docking and Grouping Toolbars and Tool Windows

Toolbars and tool windows have display and docking options, which are accessed by right-clicking on the tool window's title bar on Windows or the tool window's background on UNIX/Mac:

- **Dockable** - Toolbars and tool windows can be docked, or locked into, any edge within the editor pane. When this setting is on, you can click on the toolbar or tool window's title bar and drag and drop the window into position. When multiple tool windows are docked to the same location, they are automatically organized into tab groups.
- **Floating** - This is the default setting when a toolbar or tool window is first displayed. It allows you to click on the toolbar or tool window's title bar and drag it around within the editor pane. When this option is selected, the toolbar or tool window always appears on top of any open editor windows.
- **Auto-hide** - This setting is only available for tool windows when they are docked. When the tool window is not being used, it “slides” out of view and is replaced with a tab showing the name of the tool window. Click on the tab to “slide” the tool window back into view. Click the **Pin icon** on the right of a tool window's title bar to pin a tool window in place or unpin it to let it slide.
- **Hide** - This setting hides the toolbar or tool window. To view it again, from the main menu, choose **View > Toolbars**.

**CAUTION** If you un-dock a tool window that has been docked, the tool window may be destroyed and re-created, so you may lose any data that has been entered.

## Customizing Toolbars

Toolbars can be customized by using the [Toolbar Customization Dialog](#). To access this dialog, from the main menu choose **View > Toolbars > Customize**, or right-click on any toolbar's title bar and select **Customize**.

## Changing Toolbar Button Command Properties

To change a toolbar button's command binding, on the actual toolbar, right-click on any control and select **Properties**. This will display the [Toolbar Control Properties Dialog](#). You can change the command, the description, and the bitmap, as well as the command key binding and auto-enable properties.

## Available Toolbars and Tool Windows

The display of toolbars and tool windows can be toggled on and off by selecting the items in the **View > Toolbars** menu, which are grouped as follows:

- [Tool Windows](#) - grouped at the top of the menu
- [Toolbars](#) - grouped at the bottom of the menu
- [Debug Toolbars and Tool Windows](#) - grouped at the top of the menu when in debug mode

## Tool Windows

The tool windows listed below are used for various operations, allow the viewing of information, and/or provide option settings. They are grouped together at the top of the **View > Toolbars** menu.

### Backup History

Creates a backup version of a file each time it is saved, creating a detailed version history for this file. This is useful to track changes between check-ins. Use this tool window to compare previous versions to the current version or restore a previous version of the file. For more information, see [Using Backup History](#).

### Bookmarks

Displays a list of bookmarks and provides operations to add, delete, and navigate to a selected bookmark. This window can also be accessed by choosing **Search > Bookmarks**. For more details, see [Bookmarks](#).

### Breakpoints

Lists breakpoints (and exception breakpoints for Java) and allows you to modify them. You must use this tool window to set breakpoint properties. It can be used when you are not in debug mode. Right-click within the tool window to display a context menu which allows you to jump to the location of a breakpoint or modify breakpoints. The Breakpoints tool window can also be accessed from the **Debug > Windows** menu. For more details on this topic, see [Setting Breakpoints](#).

### Build

Docked as a tab along the bottom of the editor by default, the Build tool window, sometimes called the "concurrent process buffer," is a shell window that allows you to type operating system commands and see the results. It displays output from a build, compile, or any other Build menu command which sends output to the concurrent process buffer. Double-click on an error message to navigate to the error.

Right-click in the Build window to access build, search, and clipboard options. There are two options settings that apply to the Build tool window:

- **Auto exit process** - To have the Build tool window automatically exited when the buffer is closed or when exiting the editor, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Auto exit process**.
- **CR w/o LF erases line in build window** - It is possible that output sent to the Build window may contain carriage return (CR) characters without subsequent line feed (LF) characters. This causes

the line to be erased in the Build window. To prevent SlickEdit from erasing lines in this situation, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Deselect the option **CR w/o LF erases line in build window**.

For more details on this topic, see [Building and Compiling](#).

## Class

**NOTE** The Class tool window is new in SlickEdit® 2007 and not to be confused with the tool window named “Classes” in previous versions. The formerly named “Classes tool window” has been renamed to “Symbols tool window”.

Docked as a tab on the left side of the editor by default, the Class tool window provides an outline view of both the members of the current class as well as any visible inherited members. This tool window also shows the inheritance hierarchy of the current class. This is useful for object-oriented programming languages such as Java. In addition to the menu item, you can activate or toggle this window by using the **activate\_tbclass** or **toggle\_tbclass** commands. See [Class Tool Window](#) for more information.

## Code Annotations

Code Annotations allow you to store information about code—such as task notes, personal comments, and review comments—without actually modifying the code. This tool window provides a detailed view of annotations that you have created as well as operations for adding, modifying, and removing annotations. You can also use the tool window to create your own annotation types. See [Code Annotations](#) for more information.

## Current Context

Docked in the top upper-right section of the editor by default, Current Context displays the logical location of the cursor within your code. If it is within a class, it displays the class name. If it is within a function, it displays the function name. If the function is within a class, it displays the class and the function name. See [Current Context Tool Window](#) for more information.

## Defs

Docked as a tab on the left side of the editor by default, the Defs (Definitions) tool window contains the defs browser, which provides an outline view of symbols in the current workspace. In addition to the menu item, you can activate or toggle this window by using the **activate\_defs** or **toggle\_defs** commands. See [Defs Tool Window](#) for more information.

## Exceptions

Allows you to add, edit, disable, and delete breakpoints. For more information, see [Setting Breakpoints](#).

## File Tabs

Toggles visibility of the tabs for the open buffers in the editor. By default, these tabs are visible and the maximum number that can be displayed is 255. Right-click on the file tabs to display a menu of save, close, and window splitting options. For more information, see [Buffers and Editor Windows](#).

## Files

Allows you to view open files, project files, and workspace files, sortable by file name or path. Contains a filter to narrow the list of files incrementally, as well as shortcuts for basic file operations (Open, Save,

etc.). This tool window can also be displayed by selecting **Document > List Open Files** (Ctrl+Shift+B), or by using the **list\_buffers** command. For more information, see [Files Tool Window](#).

**NOTE** This tool window replaces the Select a Buffer dialog found in previous versions.

### Find and Replace

Used to perform search and replace operations. This tool window can also be displayed by using the key binding Ctrl+F or by selecting **Search > Find**. See [Find and Replace](#) and [Find and Replace Tool Window](#) for more information.

### Find Symbol

Used to locate symbols which are declared or defined in your code. It allows you to search for symbols by name using a regular expression, substring, or fast prefix match. See [Symbol Browsing](#) and [Find Symbol Tool Window](#) for more information.

### FTP Client

Used to connect to FTP servers and transfer files. As with most FTP clients, local directories and files are displayed in the left section of the tool window and the FTP server directories and files are on the right. Right-click on files to display a menu of FTP operations. See [Working with FTP](#) for more information.

### FTP

Used to connect to FTP servers and open files. Right-click on files to display a menu of FTP operations. See [Working with FTP](#) for more information.

### Open

Docked as a tab on the left side of the editor by default, the Open tool window can be used to browse directories and open files on disk. Right-click in the **Files** list to display a menu of options. For more information about opening files, see [Creating, Opening, and Saving Files](#).

### Output

Docked as a tab on the bottom of the editor by default, the Output tool window displays output from various operations within the editor, such as errors.

### Preview

**NOTE** In SlickEdit® 2007, the Preview tool window replaces the Symbol tool window found in previous versions. The Preview tool window does not have a search capability, so you should use the new [Find Symbol Tool Window](#) to search for symbols. This provides you with more power and more control over your symbol searching.

Docked as a tab on the bottom of the editor by default, the Preview tool window provides a portal for viewing information in other files without having to open them in the editor. It automatically shows this information when you are working with certain features. In addition to the menu item, you can activate or toggle this window by using the **activate\_preview** or **toggle\_preview** commands. See [Preview Tool Window](#) for more information.



## Projects

Contains the project browser, which allows you to browse the files in your open workspaces. It is docked as a tab on the left side of the editor by default. See [Workspaces and Projects](#) for more information.

## References

Docked as a tab on the bottom of the editor by default, the References tool window displays the list of symbol references (uses) found the last time that you used the Go to Reference feature (**Ctrl+I** or **push\_ref** command—see [Symbol Navigation](#) for more information). In addition to the menu item, you can activate or toggle this window by using the **activate\_refs** or **toggle\_refs** commands. See [References Tool Window](#) for more information.

## Regex Evaluator

Provides the capability to interactively create and test regular expressions. You can also access this window by choosing **Tools > Regex Evaluator**. See [The Regex Evaluator](#) for more details.

## Search Results

Docked as a tab on the bottom of the editor by default, this window displays the results of multi-file searches, or when the option **List all occurrences** is selected on the [Find and Replace Tool Window](#). See [Find and Replace](#) for more information about searching and replacing.

## Slick-C® Stack

Displays errors that occur within the editor. If errors occur during normal use, you can send this information to Product Support as a reference (see [Contacting Product Support](#)). If an error occurs in one of your macros, you can use this information to help debug it. Double-clicking on a line of code in this window will open the file and go to the line in the file that contains the error.

## Symbols

**NOTE** In SlickEdit® 2007, the Symbols tool window replaces the Classes tool window found in previous versions. A new [Class Tool Window](#) is available. Also note that the former Symbol tool window has been replaced with a new [Preview Tool Window](#).

Docked as a tab on the left side of the editor by default, the Symbols tool window contains the symbol browser, which lists the symbols from all of the tag files. In addition to the menu item, you can activate or toggle this window by using the **activate\_symbols** or **toggle\_symbols** commands. See [Symbols Tool Window](#) for more information.

## Symbol Properties

Displays detailed information about the symbol at the cursor location. Note that you cannot use this window to change the properties. See [Symbol Properties Tool Window](#) for more information.

## Unit Testing

Provides an interface to run JUnit unit tests and view the results. See [JUnit Test Support](#) for more details.

## Toolbars

The toolbars containing icons are listed below, and are grouped together at the bottom of the **View > Toolbars** menu.

### Debug

Provides buttons for commonly used debugger commands including start, restart, step, toggle breakpoint, and add watch. It is also very useful when you are not in debug mode. For more information about debugging within SlickEdit®, see [Debugging](#).

### Edit

Contains operations to edit text, such as convert to uppercase, indent lines, etc. These are the same options found under the main menu item **Edit**. For more information, see [Text Editing](#).

### HTML

Used to insert tags and values into an HTML file, spell check from the cursor or on selected text, beautify the document, open an FTP connection for transferring files, and more. For more information about these operations, see the topics [HTML](#), [Spell Checking](#), and [FTP](#).

### Project Tools

Contains options for the current project to process compiler error messages, run the build and compile commands, and check files in or out of version control. See [Building and Compiling](#) and [Version Control](#) for more details on these operations.

### Selective Display

Allows you to collapse unwanted lines of code so you can better see the structure of your code. Buttons allow you to outline a file with function headings, hide selected lines or lines inside code blocks, display the Selective Display dialog box, and end selective display. See [Selective Display](#) for more information about this feature.

### Standard

Toggles display of the Standard toolbar, which contains icons common to most applications, including Open File, Save, Cut, Copy, Paste, Undo, Redo, etc. By default, this toolbar is docked along the top of the editor just under the main menu.

### Context Tagging®

Contains several icons for manipulating tags and navigating. For information about these operations, see the following topics: [Building and Managing Tag Files](#), [Find and Replace](#), [Bookmarks](#), and [Navigation](#).

### Tools

Contains shortcut buttons for commonly used tools and operations within SlickEdit®, such as beautification, DIFFzilla®, merging, spell checking, and more. For more information about these operations, see the following topics: [Beautifying Code](#), [Comparing and Merging](#), [Find File Dialog](#), [Using the Calculator and Math Commands](#), [Spell Checking](#), and [Hexadecimal View and Edit Mode](#).

### XML

Contains options to beautify and validate XML documents. See [XML](#) for more information about these operations.

## Debug Toolbars and Tool Windows

The toolbars and tool windows listed below are used for debugging. In debugging mode, the **View > Toolbars** menu displays a group of these items at the top, and they are also accessible from the **Debug > Windows** menu. See [Debugging](#) for more information about working with these features.

### Debug

This toolbar is available when not in debug mode. See [Debug](#) tool window described previously.

### Autos

Displays the contents of auto, local, and member variables used before and after the current execution line. Right-click within the tool window to display a context menu which allows you to jump to the definition of a variable or add a variable to the watches.

### Breakpoints

This tool window is also available when not in debug mode. See [Breakpoints](#) tool window described previously.

### Call Stack

Displays the stack for the thread selected in the **Thread** combo box. Double-click on a method to navigate to any stack execution point.

### Classes (debug)

(Java only) Displays the currently loaded classes. Double-click on a class to display class properties. Double-click on a member to go to the definition of the member. Right-click on a method and select **Set breakpoint** to add a breakpoint to a method. Right-click on a member variable and select **Add Watch** to add a watch on a static class member. Use the **Show system classes** check box to display classes outside the scope of your workspace, like classes in the JFC.

### Debug Sessions

Lists the open debugging sessions.

### Exceptions

The Exceptions tool window is also available when not in debug mode. See [Exceptions](#) tool window described previously.

### Locals

Displays the locals variables for the method selected in the **Stack** combo box. You can modify the values of variables by double-clicking in the **Value** column of simple types.

### Members

Displays the value of static and non-static members for any method context. You can modify the contents of a member variable by specifying a valid Java expression. Right-click within the tool window to display a context menu which allows you to jump to the definition of a variable or add a variable to the watches. You cannot add or remove entries from an array.

### Memory

(GNU C/C++ only) Displays the contents of the specified memory address. Enter the memory location in a hexadecimal, decimal, or any valid C expression. In addition, you may specify the number of bytes to display.

### Registers

(GNU C/C++ only) Displays the contents of hardware registers.

### Threads

Allows you to view the threads currently running and choose a thread context, so you can view the stack for a particular thread. It displays the thread, group, status, and state.

### Watch

Contains watch tabs that are used to display the value variables or expressions you specify for the context method. There are several ways to add a new watch variable or expression:

- Double-click on the **<add>** text in the **Name** column of the tool window.
- Right-click on a variable or selected expression and select **Add Watch**.
- Select a variable or expression, then from the main menu choose **Debug > Add Watch**.

To display a context menu which allows you to jump to the definition of a variable or delete a watch expression, right-click within the Watch tool window.

## Buffers and Editor Windows

---

When you edit a file, it is loaded into a *buffer*. The buffer is the in-memory representation of the file. Your edits are made to the buffer, but the file is not updated until you save it. Buffers are displayed in *editor windows* in the *editor pane*. An editor window can be maximized to fill the entire pane, or you can organize the windows within the editor pane. Each buffer has an associated tab in the [File Tabs](#) tool window, which displays the name of the file. In general, you will have a one-to-one relationship between files and buffers. If you open a file that is already open, the existing buffer is displayed. Therefore, the terms "file" and "buffer" are sometimes used interchangeably.

When a buffer is modified (changed and not yet saved), there are two visual indicators:

- An asterisk will be displayed to the right of the file name in the title bar.
- The file name on the file tab turns red.

**TIP** For a list of Slick-C® buffer and window functions and commands, see "Macro Functions by Category" in the Help system.

SlickEdit® provides two main approaches to managing buffers and editor windows, controlled through the **One file per window** option (**Tools > Options > General**, [General Tab](#)). The value for this option affects the behavior of the window and buffer switching commands, described later in this section.

1. **One file per window checked** - This maintains a one-to-one correspondence between a buffer and an editor window. Each buffer is displayed in its own editor window. This is the default behavior for SlickEdit. This approach is preferred by many for its simplicity.
2. **One file per window unchecked (multiple buffers per window)** - With this approach, you determine how many editor windows you want and you select the buffer to display in each. You have to manually create a new editor window (typically by splitting or duplicating an existing window). All buffers are available to all windows. You can use the file tabs to select the buffer to edit, which will place the buffer in the currently active editor window. You can also use **Document > List Buffers** (`list_buffers` command) to view a list of the buffers and select one. This approach appeals to users who like a particular arrangement of windows and use editor windows for specific tasks.

## Managing Windows

### Changing the Window Left Margin Width

The left margin of an editor window is used to give visual indicators for certain operations (such as when diffing files). Increasing the size of the window left margin can make it easier to create line selections with the mouse. It also prevents window contents from jumping to the right when a bookmark, breakpoint, or error is first displayed.

To specify the space between the left edge of the window and the editor text, choose **Tools > Options > General**, then select the [More Tab](#). In the **Window left margin** text box, enter the amount of space desired in inches. This value has no effect when there are bitmaps displayed in the left margin, since more space is necessary.

### Splitting Windows

To split the current editor window into two parts so you can view/edit different parts of the same buffer at the same time, choose **Window > Split Horizontally** (`Ctrl+H` or `hsplit_window` command) or **Window >**

**Split Vertically** (**vsplit\_window** command). You can also right-click on the file tab of the current window to perform the same operations.

To view two different editor windows side-by-side, while the focus is on the current editor window, right-click on the file tab of the window you want to compare it with and choose one of the **Split...with** operations.

## Duplicating Windows

To create a duplicate of the current editor window, use the **duplicate\_window** command (**Window > Duplicate**). This will create a new window linked to the current buffer.

## Cascading and Tiling Windows

To resize and arrange the open editor windows in a cascading display, select **Window > Cascade** (**cascade\_windows** command).

To resize and rearrange the open editor windows so they don't overlap, select **Window > Tile** (**tile\_windows** command). If there are three or fewer open editor windows, they will be tiled vertically. To tile three or fewer windows horizontally, select **Window > Tile Horizontal** (**tile\_windows h**).

## Manipulating Tiled Windows

There are several SlickEdit® commands that can be used when the windows are tiled:

- **window\_below** - Switches to the window tile below the current window, if one exists.
- **window\_left** - Switches to the window tile to the left of the current window, if one exists.
- **window\_right** - Switches to the window tile to the right of the current window, if one exists.
- **move\_edge** - Moves the adjoining edge of a tiled window. This command is bound to **Alt+F2**. Press **ALT+F2**, then use the arrow keys to move the cursor to point to the window edge and move it to the new edge position, then press **Enter**.
- **delete\_tile** - Deletes an adjacent tiled window. Use the arrow keys to point to the edge of the window you wish to delete.

## Maximizing and Minimizing Windows

By default, editor windows are floating when they are first opened. If you change the size of the editor window, the size is remembered thereafter for new editor windows that are opened. To specify that all editor windows should be maximized when they are first opened, from the main menu, choose **Tools > Options > General**, then select the [General Tab](#). Select the option **Maximize first window**.

To maximize an editor window, click the **Maximize** button icon on the window's title bar, or use the **maximize\_window** command. You can also toggle the existing state with a maximized state by using the **zoom\_window** command (**Window > Zoom Toggle** or **Ctrl+Shift+Z**).

To minimize an editor window to an iconized state, click the **Minimize** button icon on the window's title bar, or use the **iconize\_window** command. To iconize all open editor windows, use the **iconize\_all** command.

When a window is iconized, you can click on the title bar for a menu of operations such as Restore, Move, Close, and Next. Use the mouse to drag the iconized windows around the editor pane. Use the **arrange\_icons** command (**Window > Arrange Icons**) to re-dock the iconized windows along the bottom edge of the editor pane.

To restore an iconized window to its previous state, click the **Restore Up** button icon on the window's title bar, or use the **restore\_window** command. To restore all iconized windows, use the **restore\_all** command.

## Switching Between Buffers or Windows

The method for switching the buffer or window with which you are working depends on whether you are using a one-to-one relationship between buffers and files (see the introduction to [Buffers and Editor Windows](#)). If you have selected **One file per window**, switching buffers and switching windows is the same thing. If that option is unchecked, then these are two very different operations and you manipulate windows and buffers separately.

There are many styles and commands for switching between buffers and windows within the editor, and we encourage you to try them out and pick the method that works best for you.

Using the mouse, you can switch between editor windows by clicking on the file tabs or by using the Window menu items. By default, the active window will change when you switch to a specific file or buffer, unless the active window is already displaying the buffer you select (see [Linking to a Window](#) below).

**TIP** For information about navigating within files, see [Cursor Navigation](#).

### Next Window Style

The default next window style is Smart Next Window. This style allows you to press **Ctrl+Tab** (**next\_window** command) to switch the focus between the two most frequently used open editor windows, rather than always going to the next window. Press **Ctrl+Shift+Tab** (**prev\_window** command) to switch between all open editor windows. This style is similar to how Ctrl+Tab and Ctrl+Shift+Tab work in other Windows MDI applications, like Visual Studio.

Smart Next Window is on by default (**Tools > Options > General, [More Tab](#)**). There are two alternatives to this behavior (note that all three options are mutually exclusive):

- **Reorder windows** - If this option is selected, activating an existing window reinserts the window after the current window. Neither Ctrl+Tab nor Ctrl+Shift+Tab reorders the windows. This option is very good for switching between more than two files, but it is not the Windows standard (which means you're probably not used to it). It's similar to the way SlickEdit® reorders buffers.
- **No window reordering** - If this option is selected, newly opened windows are inserted after the current window. Activating an existing window, pressing Ctrl+Tab, or pressing Ctrl+Shift+Tab does not reorder windows. This option is best if you like to memorize the hot key numbers on the Window menu (for example, Alt+W 1) because it attempts to keep the hot key numbers the same.

### Buffer and Window Switching Commands

The following is a list of SlickEdit® commands that you can use to switch between buffers and windows:

- **next\_buff\_tab/prev\_buff\_tab** - Navigates through the buffer tabs (also called file tabs) in the order they are displayed. **next\_buff\_tab** moves to the right and **prev\_buff\_tab** moves to the left. Both circle around to the other end when you are on the last item. These commands are not bound to keys by default.
- **next\_buffer/prev\_buffer** - Navigates through the buffers in the order they were last used. If you are using multiple files per window, then you need to use this to cycle through the buffers associated with a particular window. These commands are bound respectively to the menu items **Document > Next Buffer** (Ctrl+N) and **Document > Previous Buffer** (Ctrl+P).

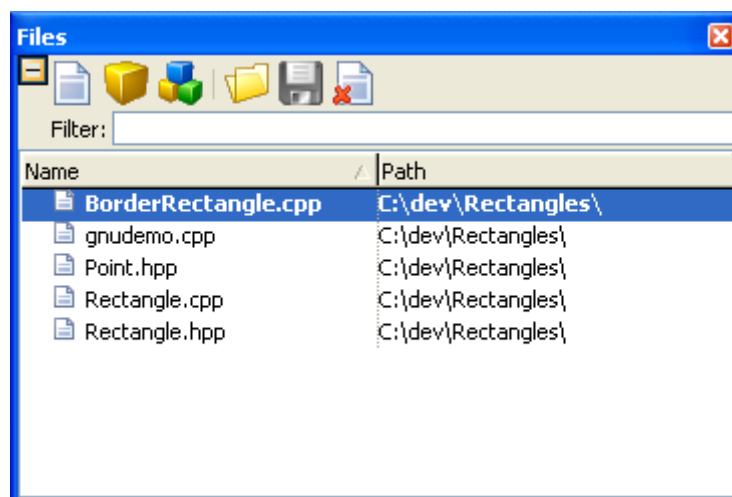
- **forward/back** - Same as **next\_buffer/prev\_buffer**. These commands are bound to the green arrows on the Standard toolbar and the forward/back mouse buttons.
- **next\_window/prev\_window** - Moves between editor windows. If you are using the option **One file per window** (on by default), then this is the same as **next\_buffer/prev\_buffer**. If you are using multiple files per window, you can use this to navigate between editor windows, but you will need to use **next\_buffer/prev\_buffer** to cycle through the buffers within a particular window. The **next\_window** and **prev\_window** commands are bound respectively to the menu items **Window > Next** (Ctrl+Tab or Ctrl+F6) and **Window > Previous** (Ctrl+Shift+Tab or Ctrl+Shift+F6).

## Listing Open Files

To view and work with a list of open files in SlickEdit®, select **Document > List Open Files** (Ctrl+Shift+B), or use the **list\_buffers** command. This will display the Files tool window, which contains an alphabetized list of the files and buffers currently open in the editor.

### NOTES

- The Files tool window replaces the Select a Buffer dialog found in previous versions of SlickEdit.
- For documentation purposes, the word “files” generally includes both files and buffers.



### TIPS

- By docking this tool window, you have quick access for switching between files or opening other files. Right-click on the title bar and select **Dockable**, then drag and drop the window to your desired location.
- When the Files tool window is not docked, it can be dismissed by opening a file for editing or by pressing Esc. To make this dialog behave like other tool windows, right-click inside the Files list area and uncheck **Dismiss on select**.
- The Files tool window can also show a list of files in the active project or workspace. Use the View icons or the view settings on the right-click menu to change the display.



Using the icons on the Files tool window (or the right-click menu), you can perform the following operations:

- **Open** - Opens and brings that file into focus, ready for editing. The active file is listed in the tool window in a bold font. You can also open files by double-clicking on them, or by pressing **Enter** or **Alt+E**.
- **Save** - Saves the selected file(s). Modified files have a Disk icon in the left margin of the tool window, and are listed in a red font; when selected, they have a red highlight. You can also click the **Disk icon** to quickly save a modified file, or press **Ctrl+S**, **Alt+S**, or **Alt+W**.
- **Close** - Closes the selected file(s) in the editor. You can also press **Delete**, **Alt+C**, or **Alt+D** to close a selected file. Upon close, you are prompted to save modified buffers. If you are using the option **One file per window** (**Tools > Options > General > General Tab**), which is on by default, all windows displaying the buffer are closed as well.

Sort any column by clicking on the column header. When you click to sort, an arrow on the right side of the column header shows the ascending or descending order.

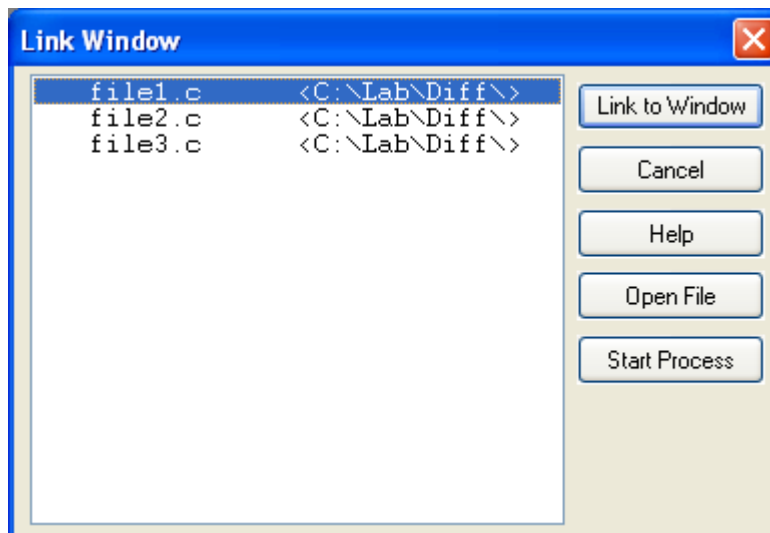
Use the **Filter** text box to display matching file names. Right-click inside the Files list area to enable **Prefix match** inside the Filter text box. When the focus is not in the Filter text box, you can incrementally search the list of file names by typing the first few characters of the name. See [Working with the Files List](#) for more details about filtering and searching the Files list.

**TIP** You can collapse display of the icons and the Filter text box by clicking the **Minus** icon in the top-left corner of the tool window. The collapsed area is replaced with a message that states your current view and filter settings. To view the icons and Filter text box again, click the **Plus** icon. Collapsing the icons and filter gives your window a cleaner look and more room for file listings.

For more detailed information about using the Files tool window, see [Files Tool Window](#).

## Linking to a Window

You can change the buffer that is displayed in the current window by using the Link Window dialog (**Window > Link Window** or **link\_window** command), pictured below. This is especially useful if you like to work with split windows, for quickly bringing the buffers you want to view into focus.



Select the buffer that you want, then click **Link to Window**. To start a process buffer in the current editor window, click **Start Process**. If a process buffer has already been started, it is linked to in the current window. See also [Link Window Dialog](#) for more descriptions of the available options.

## Closing Buffers and Windows

To close an editor window, select **Window > Close** (Ctrl+F4 or `close_window` command), or **File > Close All** (`close_all` command) to close all open editor windows. You will be prompted to save any modified buffers.

See [Closing Files](#) for more information about closing buffers and files.

## Accessing Menus

---

By default, the SlickEdit® main menu appears under the title bar of the editor (File, Edit, Search, etc.) and provides a way to access most of the editing features within SlickEdit. For listings of all the menus and associated dialogs and tool windows, see the [Menus, Dialogs, and Tool Windows](#) chapter.

Menu items are mapped to commands, so all of the items that you can access through a menu have command line counterparts. For example, to cut selected text, you could use the menu item **Edit > Cut**, or you could use the **cut** command. You can edit and customize menus with the use of macros. See [Creating and Editing Menus](#) for more information.

## Right-Click Context Menu

Click the right mouse button to access submenus or context-sensitive menus that list commonly used editing functions. Context-sensitive menus are supported in edit windows, all toolbars, and in many dialog boxes, including the [DIFFzilla® Dialog](#), the [Multi-File Diff Output Dialog](#), and the [Context Tagging® - Tag Files Dialog](#).

## Context Menu Settings

Extension-specific settings are available for specifying which context menu to display in the editor window, based on whether text is selected.

To access these options, choose **Tools > Options > File Extension Setup**. When the Extension Options dialog appears, select the extension that you wish to affect from the **Extension** drop-down list. Then select the [Advanced Tab](#), and select from the following options in the **Context menus** group box:

- **Menu if no selection** - This specifies that the context menu is displayed when right-clicking in an editor window that does not have a selection.
- **Menu if selection** - This specifies that the context menu is displayed when right-clicking in an editor window that does have a selection.
- **Select first (affects all extensions)** - When checked (default), a selection can be made with the right mouse button instead of displaying the extension-specific menu. When this is not checked, select menu items by clicking and dragging the mouse.

## Menu Hotkeys

A menu hotkey allows you to choose items from the main menu using the keyboard. This is done using the Alt key and a letter in the menu item. This works independently from key bindings, which bind arbitrary key sequences to commands. If a menu item has a key binding, it is shown to the right of the menu entry. See [Key and Mouse Bindings](#) for more information about key bindings.

SlickEdit provides two options for menu hotkeys:

- [Alt Menu](#) - When selected, pressing the Alt key switches focus to the menu bar and underlines the hotkeys. Now, when you type a letter, it selects the corresponding item on the menu.
- [Alt Menu Hotkeys](#) - For emulations other than CUA, selecting this option gives priority to the menu hotkeys, overriding any keybindings using the same Alt key combination.

### Alt Menu

When you press the Alt key in SlickEdit without following it with another key, focus shifts to the menu bar, and the hotkeys in the menu names are underlined. Pressing one of the underlined letters will activate the corresponding menu or menu item. For example, press **Alt, V, A** to invoke the **View > Show All** command. Pressing Alt again toggles the focus back to the cursor location. This is controlled by the **Alt menu** option on the [General Tab](#) of the General Options dialog (**Tools > Options > General**), which is on by default.

**TIP** On Microsoft Windows, to force menu names to be underlined all the time instead of just when you press Alt:

1. Right-click on the desktop and select **Properties**.
2. Select the **Appearance tab**.
3. Click on **Effects**.
4. Uncheck **Hide underlined letters for keyboard navigation until I press the Alt key**.
5. Click **OK**.

### Alt Menu Hotkeys

This option applies only to non-CUA emulations. You can make Alt-prefixed key bindings display the corresponding drop-down menu. For example, when you press Alt+F (where F corresponds to the underlined letter on the File menu), the File menu drop-down is displayed. Thus, pressing Alt+F, D will invoke the **File > Change Directory** command. To set this option, check the Alt menu hotkeys option on the [General Tab](#) of the General Options dialog (**Tools > Options > General**).

**CAUTION** When this option is selected, the Alt menu hotkeys may override your normal key bindings.

## Short Key Names in Menus

The SlickEdit® main menu displays the key bindings for commands associated with each menu entry. These bindings can be condensed for non-CUA emulations. For example, Ctrl+O becomes C-O. To set this behavior, select the option **Short key names** on the [More Tab](#) of the General Options dialog (**Tools > Options > General**).

See [Key and Mouse Bindings](#) for more information about working with bindings in SlickEdit.

## The SlickEdit® Command Line

---

SlickEdit provides a command line as a means to execute most SlickEdit operations without taking your hands off of the keyboard. This is useful for less frequently used operations that may not warrant a key binding, or complex commands that require arguments.

**TIP** SlickEdit® commands that contain two or more words are written throughout our documentation with underscore separators: for example, **cursor\_down**. Note that in the user interface, however, these commands are displayed with hyphen separators: for example, **cursor-down**. Both of these forms work in SlickEdit, so you can use whichever style you prefer.

## Activating the Command Line

To activate or toggle the command line in any emulation, click on the SlickEdit® message line with the mouse. Key bindings are also provided for toggling the cursor to the command line, based on your emulation:

- BBEEdit - **Esc**
- Brief - **Esc**
- CodeWarrior - **Esc**
- CodeWright - **F9**
- Epsilon - **Alt+X** or **F2**
- GNU Emacs - **Alt+X** or **F2**
- ISPF - **Esc**
- SlickEdit (Text Mode edition) - **Esc**
- Vim - **Ctrl+A**
- Visual C++ - **Esc**
- Visual Studio - **Esc**
- Xcode - **Esc**

See [Emulations](#) for more information.

## Command Line History

The command line maintains a command history, allowing you to quickly reuse previously entered commands. Once the command line is open, use the arrow keys to scroll up and down in the command history.

## Command Line Completions

As you type a command, a list of matching completions is displayed, including any command line arguments used in a previous command. Use **Tab** or the **Down** arrow to move to the next command in the list, and **Shift+Tab** or the **Up** arrow to move to the previous command. Press the **Enter** key to select the current command.

Some commands, like **set\_var**, prompt for arguments. SlickEdit® maintains a history of arguments used for each command. Use the same completion and history mechanism as described above for commands to complete arguments. Typically, the most recent argument you typed is automatically displayed.

**TIP** Command completions are useful for discovering operations in SlickEdit. For instance, to find all operations that begin with “find”, type **find** in the command line, and SlickEdit will display a list of those commands. Some search commands do not begin with “find”, like **gui\_find**, so you may not discover all related commands this way. To find all commands containing the word “find,” use the Key Bindings dialog (**Tools > Options > Key Bindings** or **gui\_keybindings** command). See [Key and Mouse Bindings](#) for more information.

For information about other items that can be automatically completed, see [Completions](#).

## Disabling Command Line Completions

To disable command line completions, from the main menu, choose **Tools > Options > General** and then select the [General Tab](#). Uncheck the option **List command line completions**. Note that this option does not apply to the Vim command line.

## Using Shortcuts Inside the Command Line

The command line is a text box control just like the text boxes that appear in various dialog boxes. For a list of key shortcuts that can be used inside the command line and other text boxes within SlickEdit®, see [Key Shortcuts in Text Boxes](#).

## Using the Command Line to View Key Binding Associations

You can use the SlickEdit® command line to determine what keys are associated with what commands, and vice-versa.

**TIP** Alternatively, you can use the Key Bindings dialog (**Tools > Options > Key Bindings** or **gui\_keybindings** command) to see a list of command/key binding associations. See [Key Bindings Dialog](#) for more information.

## Determining the Command of a Key Binding

To determine the function of a key or key binding, use the **what\_is** command (**Help > What Is Key**). For example:

1. Select **Help > What Is Key**, or activate the SlickEdit® command line (by pressing **Esc**) and type **what\_is** (or type **what** and press the spacebar for auto-completion), then press **Enter**.
2. The command line will prompt with the text “What is key.” Enter the key sequence in question. A message box will be displayed with the information. If the key or key sequence is not bound to a command, no message will appear.

## Determining the Key Binding of a Command

To determine the key to which a command is bound, use the **where\_is** command (**Help > Where Is Command**). For example:

1. Select **Help > Where Is Command**, or activate the command line and type **where\_is**, then press **Enter**.
2. The command line will prompt with the text "Where is command." Enter the command in question. The status line will display the key binding or state that the command is not bound to a key.

## Starting a Program from the Command Line (Shelling)

You can use the command line to start a program. Click on the command line or press **Esc** to toggle the cursor to the command line. Type the program name and arguments and press **Enter**. When entering a command that the editor does not recognize as an internal command, a path search is performed to find an external program to execute. To use a program whose name contains space characters, enclose the name in double quotes. For example, "this is" will start a program named `this is.exe` if it exists.

Executing the command **dir** (or **ls**) from the command line will invoke the SlickEdit® File Manager. To bypass an internal command, prefix the command with "dos". To execute the **dos dir** command, type **dos -w dir** and press **Enter**.

To get an operating system prompt, type the command **dos** with no arguments or from the main menu, choose **Tools > OS Shell**.

## Command Line Prompting

Many commands that display dialog boxes have equivalent commands that prompt for arguments on the command line. For faster prompting than the dialog boxes allow, you can choose to be prompted for arguments on the command line instead. To set this option, from the main menu, choose **Tools > Options > General**, then select the [General Tab](#). Select the option **Command line prompting**. To be more selective than this option permits, change the key bindings. For example, to be prompted only on the command line when opening files, bind the **edit** command to Ctrl+O, which is bound to the **gui\_open** command by default.

The following table contains a partial list of user interface commands and their command line counterparts.

Graphical Command	Command Line Version
<b>gui_open</b>	<b>edit</b>
<b>gui_find</b>	<b>find</b>
<b>gui_replace</b>	<b>replace</b>
<b>gui_write_selection</b>	<b>put</b>
<b>gui_append_selection</b>	<b>append</b>
<b>gui_margins</b>	<b>margins</b>
<b>gui_tabs</b>	<b>tabs</b>
<b>gui_find_proc</b>	<b>find_proc</b>

## Common SlickEdit® Commands

Commands are essentially the names of functions. The Help system contains a list of Macro Functions by Category (see **Help > Contents**). The following is a list of commands that we use frequently in our own work, which you may also find useful.

- |                                    |   |
|------------------------------------|---|
| 1. <b>e file</b>                   | Edit a file                             |
| 2. <b>sa file</b>                  | Save file as                            |
| 3. <b>number</b>                   | Go to line number                       |
| 4. <b>f symbol</b>                 | Find a symbol                           |
| 5. <b>/search_string/options</b>   | Search for a string                     |
| 6. <b>c/search/replace/options</b> | Replace a string                        |
| 7. <b>gt/search/options</b>        | Substring search for a symbol           |
| 8. <b>sb name</b>                  | Set a bookmark                          |
| 9. <b>gb name</b>                  | Jump to a bookmark                      |
| 10. <b>help topic</b>              | View help on topic                      |
| 11. <b>man command</b>             | Show UNIX man page                      |
| 12. <b>cd directory</b>            | Change directory                        |
| 13. <b>dir directory</b>           | Show directory in the file manager      |
| 14. <b>list wildcards</b>          | Show directory tree in the file manager |
| 15. <b>del filename</b>            | Delete file                             |
| 16. <b>pushd directory</b>         | Push directory                          |
| 17. <b>popd</b>                    | Pop directory                           |
| 18. <b>set env=value</b>           | Set environment variable                |
| 19. <b>dos command</b>             | Execute command outside of editor       |
| 20. <b>math expr</b>               | Evaluate expression                     |



## Screen Management

---

There are several features regarding the handling of the monitor screen, as described below.

### Full Screen Mode

Full screen editing provides a mode to expand the editor window to the full size of the screen. This is very useful when using large files while other views and toolbars are not needed. To enable this feature from the main menu, select **View > Full Screen** (or use the **fullscreen** command). Or, when you are working in the buffer and there is no code selected, right-click and select **Full Screen**.

### Multiple Monitor Support

SlickEdit® supports the use of multiple monitors on Windows, UNIX, and Mac OS X platforms. When the application, or any dialog or tool window, is moved to a particular monitor, the location is remembered. If you are running UNIX or Mac OS X and using multiple monitors that have different width and height values, you will need to set the invocation option **-summ**. See [Invocation Options](#) for more information.

### JAWS Screen Reader Software

JAWS®, the widely used Windows-based screen reading software developed by Freedom Scientific, allows applications to be accessible by vision-impaired software developers. All toolbars, menus, tab controls, and tree controls are accessible, along with all labels, controls, and contents of dialog boxes. Additionally, JAWS accurately reads the contents of the editor pane, [Output](#) tool window, and [Projects](#) tool window. Key bindings allow fast and accurate code navigation and control of other functionality.

The Context Tagging® features Color Coding, Go to Definition, and Go to Reference are not supported. Access is provided to third-party integrated development environments and version control systems. However, each third party product is responsible for its own accessibility functionality.

### Configuring JAWS

The installation program auto-detects if JAWS is installed. If detected, you are prompted to configure JAWS. To manually configure JAWS, type the following text in the environment section of the `win\vslick.ini` file (located in your SlickEdit® installation directory):

```
VSLICKJAWS=1
```

This is the only configuration needed.



## Using the Mouse and Keyboard

SlickEdit® provides four ways to launch operations: commands, menu items, key bindings, and icons. For example, to launch the Open dialog box in order to open a file, you could use any of the following methods:

- Type the **gui\_open** command on the SlickEdit command line.
- Select **File > Open**.
- Press the key binding **F7** or **Ctrl+O**.
- Click the **Open** icon on the Standard toolbar.

The command forms the basis of each method. As you can see, commands are often bound to more than one key sequence. They can also be bound to mouse events, including the spin wheel. Key bindings are the fastest and most efficient means of executing operations.

See [The SlickEdit® Command Line](#) for more information about commands, and [Key and Mouse Bindings](#) for more information about bindings.

## Key Shortcuts in Text Boxes

Key shortcuts for text operations (such as cut, copy, and paste) can be used inside of all text boxes within SlickEdit® (including the command line).

**TIP** The CUA emulation contains the shortcuts Ctrl+X, Ctrl+C, and Ctrl+V for cut, copy, and paste, respectively. If you are not using the CUA emulation, by default, these key bindings still work inside of text boxes. To disable this feature, from the main menu choose **Tools > Options > General**, then select the [More Tab](#). Deselect the option **CUA Text Box**.

## Text Box Editing Keys

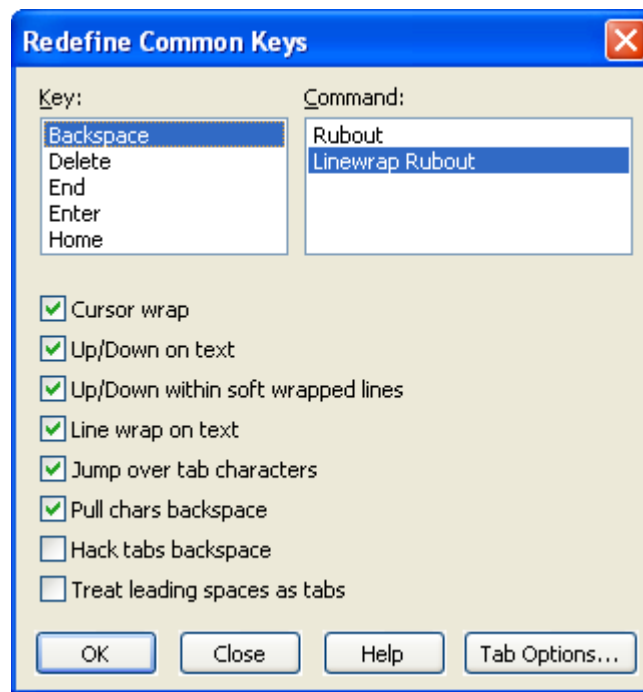
The table below contains a list of the key shortcuts (based on the CUA emulation) that can be used inside the command line and other text boxes within SlickEdit®.

Keys	Operation
Insert	Insert mode toggle
Spacebar	Expand partially-typed parameter or insert a space
?	List matches to partially-typed parameter
Ctrl+Shift+O	Expand alias
Ctrl+E	Cut to end of line
Ctrl+Backspace	Cut line
Ctrl+K	Copy word to clipboard
Ctrl+Shift+K	Cut word
Ctrl+Shift+L	Lowercase word

Keys	Operation
Ctrl+left arrow	Previous word
Ctrl+right arrow	Next word
Ctrl+V	Paste
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+Shift+X	Append cut
Ctrl+Shift+C	Append to clipboard
Ctrl+Shift+V	List clipboards
Shift+Home	Select text between cursor and beginning of line
Shift+End	Select text between cursor and end of line
Shift+Click	Extend selection to mouse position
Backspace	Delete previous character or selection
Delete	Delete character under cursor or selection
Left arrow	Move cursor left
Right arrow	Move cursor right
End	Move cursor to end of line
Home	Move cursor to beginning of line
Double-click	Select word
Triple-click	Select line

## Redefining Common Keys

Many users have a preference for the functions of the keys Backspace, Delete, Enter, Tab, and Home. The Redefine Common Keys dialog is designed for changing the function of these keys. To access this dialog, from the main menu, select **Tools > Options > Redefine Common Keys**.



In the **Key** list box, select the name of the key that you want to configure. The commands available for that key are then displayed in the **Command** list box. Additional options can be set using the check boxes.

Click the **Tab Options** button to change the function of the Tab key. The Indent tab of the Extension Options dialog box is displayed. For more information on changing Tab key functions, see [Indenting with Tabs](#).

For descriptions of all the elements on the Redefine Common Keys dialog, see [Redefine Common Keys Dialog](#).



# Printing

Printing within SlickEdit® varies slightly between Windows and UNIX (which includes Mac OS X) platforms.

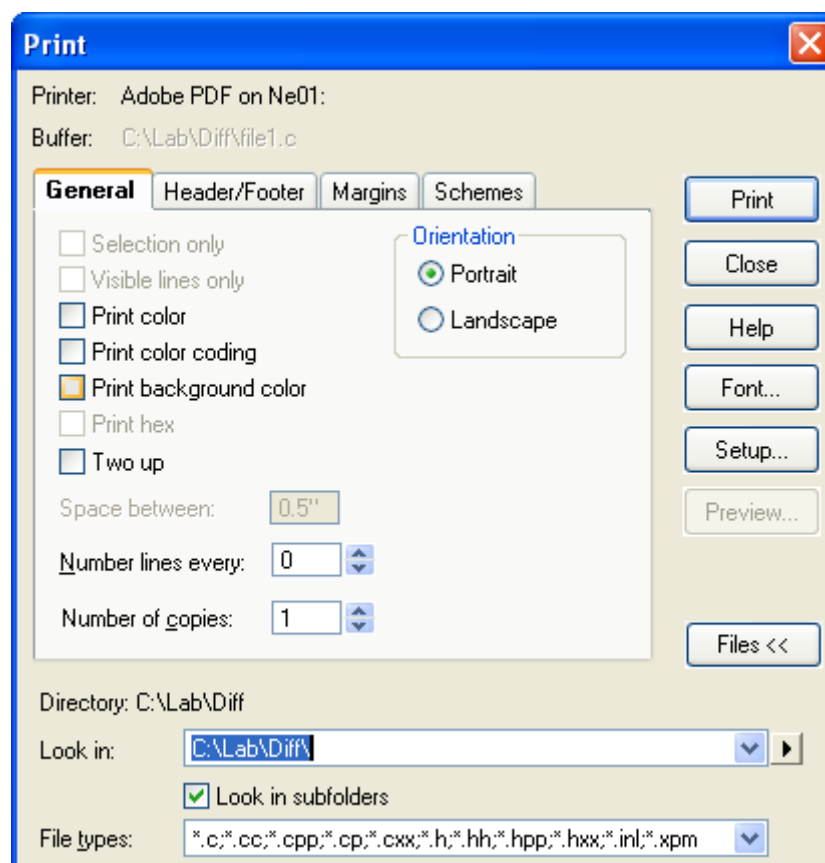
## Printing on Windows

To print, from the main menu, select **File > Print** (or use the **gui\_print** command). The Print dialog is displayed, which allows you to specify the printer to use and set configuration options. After the printer is specified and options configured, click **Print** to print the selection or active buffer.

Alternately, you can use the **print** command to immediately print the active buffer, or the **print\_selection** command to immediately print the selection, both based on the current configuration.

## Windows Printer Configuration

To configure printing options, from the main menu, select **File > Print** (or use the **gui\_print** command). The Print dialog is displayed, as pictured below.



To print immediately using the current settings, click **Print**. To select the font used for printing, click **Font**. For printer setup, click **Setup**.

For a complete list of the options on this dialog, see [Print Dialog - Windows](#). The sections below describe a few of the features.

### Header/Footer Print Settings

To define the contents of printed headers and footers, use the [Header/Footer Tab](#) of the Print dialog. Type directly into the text boxes to specify text for the top left, center, and right headers and bottom left, center, and right footers. Click the arrow to the right of each text box pick from a list of escape sequences to be inserted. Escape sequences are values that are replaced with real data, such as **%f** (which will be replaced with the file name) and **%d** (which will be replaced with the date).

### Print Margin Settings

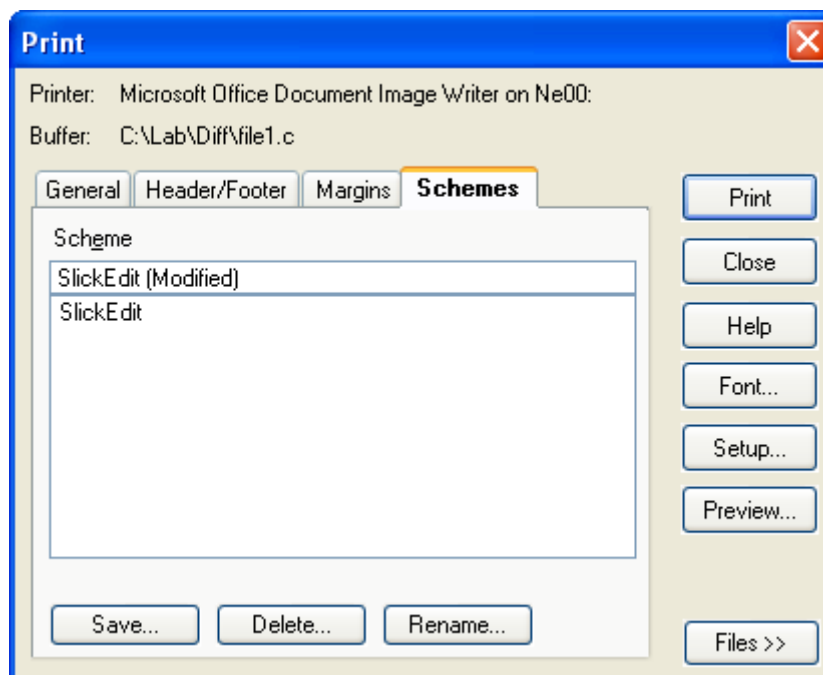
To define margin spacing, use the [Margins Tab](#) of the Print dialog. In the **After header** and **Before footer** fields, enter the amount of spacing to come between the header and the first line on a page, and the amount of spacing to come between the last line on a page and the footer.

In the top, bottom, left, and right margin fields, enter the amount of spacing in inches to come between the outer edge of the paper and the printed text.

To print the maximum amount of text, specify “0” for all margins.

### Print Schemes

Print schemes are a way to manage different printing configurations. For example, you might want to define one scheme that prints two up, landscape, and uses a small font, and another scheme that prints the same configuration with a different font. To define or select a print schemes, use the [Schemes Tab](#) of the Print dialog, which is pictured below.



The default scheme is named “SlickEdit”. When you change the configuration options on the other tabs in the Print dialog, the scheme name changes to append the text “(Modified)” to the end. Click **Save** to give a name to the modified scheme. Click **Delete** and **Rename** to delete and rename schemes, respectively.



System schemes are stored in the `print.ini` file located in the product installation directory. User-defined schemes or modified system schemes are stored in the `uprint.ini` file.

## Printing on UNIX, Linux, and Mac OS X

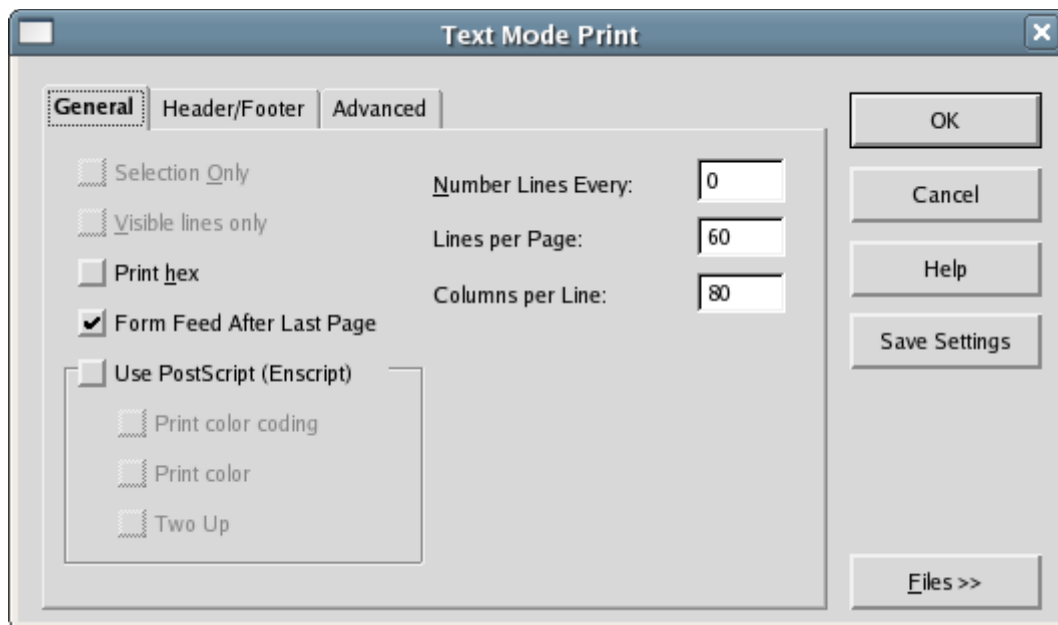
To print, from the main menu, select **File > Print** (or use the command `gui_print`). The Text Mode Print dialog is displayed, which allows you to specify the device to use and set configuration options. After the device is specified and options configured, click **OK** to print the active file or buffer.

Alternately, you can use the `print` command to immediately print the active buffer, or the `print_selection` command to immediately print the selection, both based on the current configuration. Mac OS X users can use the keyboard shortcut `Cmd+P` to immediately print the active buffer.

**NOTE** Graphical printing options (such as print preview) are not supported for UNIX.

## UNIX, Linux, and Mac OS X Printer Configuration

To configure printing options, from the main menu, select **File > Print** (or use the command `gui_print`). The Text Mode Print dialog, pictured below, is displayed.



This dialog contains some of the same options as the Windows Print dialog. See [Print Dialog - Linux, UNIX, and Mac](#) for a complete list of the available options.

## Inserting a Formfeed

The printing facility supports embedded formfeed characters. The formfeed character must be the only character on the line. To insert a formfeed into the current buffer, press `Ctrl+Q` (**quote\_key** command), and `Ctrl+L`. Alternatively, you can use the Insert Literal dialog box (**Edit > Insert Literal** or `insert_literal` command) to insert a formfeed or any other character (see [Inserting Literal Characters](#)).



# User Preferences

This chapter contains the following topics:

- [Introduction to User Preferences](#)
- [Emulations](#)
- [Key and Mouse Bindings](#)
- [Cursor, Mouse, and Scroll Settings](#)
- [Setting Fonts and Colors](#)
- [Restoring Settings on Startup](#)
- [Setting File Associations](#)



# Introduction to User Preferences

---

SlickEdit® can be customized to accommodate your own individual preferences. Most of the user preference information is available from the main menu when you select **Tools > Options**.

User preferences are broken into two categories: preferences that apply to all languages (global preferences), and preferences that apply to specific language extensions.

**TIP** If you are using SlickEdit in a multiple user environment, each user must define a VSLICKCONFIG environment variable that refers to a local directory. This allows each user to have their own configuration. If making modifications to `vslick.ini`, make a local copy of this file and place it in the VSLICKCONFIG directory file. See [Environment Variables](#) for more information.

## Global Preferences

Global preferences that can be set include the following:

- Emulation modes (see [Emulations](#))
- Fonts and colors (see [Setting Fonts and Colors](#))
- Auto Restore settings (see [Restoring Settings on Startup](#))
- File associations (see [Setting File Associations](#))

Other global preferences, such as search settings, selection styles, etc., can be configured by using the General Options dialog (**Tools > Options > General**). These options are described in the documentation on a contextual basis. For a flat listing of the options on the General Options dialog, see [General Options Dialog](#).

## Extension-Specific Preferences

The behavior of the editor can be customized for files based on specific language extensions. Indent, word wrap, comment, auto-complete, Context Tagging®, and other code-style settings are all extension-specific. These settings are located on the Extension Options dialog (**Tools > Options > File Extension Setup**). The options are described in the documentation on a contextual basis. For a flat listing of the options on the Extension Options dialog, see [Extension Options Dialog](#).

For more information about working with language extensions, see [Language-Specific Editing Overview](#).



# Emulations

---

*Emulation* is the process of imitating another program. SlickEdit® provides emulations of key bindings for 13 editors so that you can use the style to which you are accustomed, making your coding experience as efficient as possible.

The Key Bindings dialog allows you see what keys or key sequences are bound to what commands. Emulation charts are also available in the Help system and as printable PDF documents in the `docs` subdirectory of your SlickEdit installation directory. See [Key and Mouse Bindings](#) for more information.

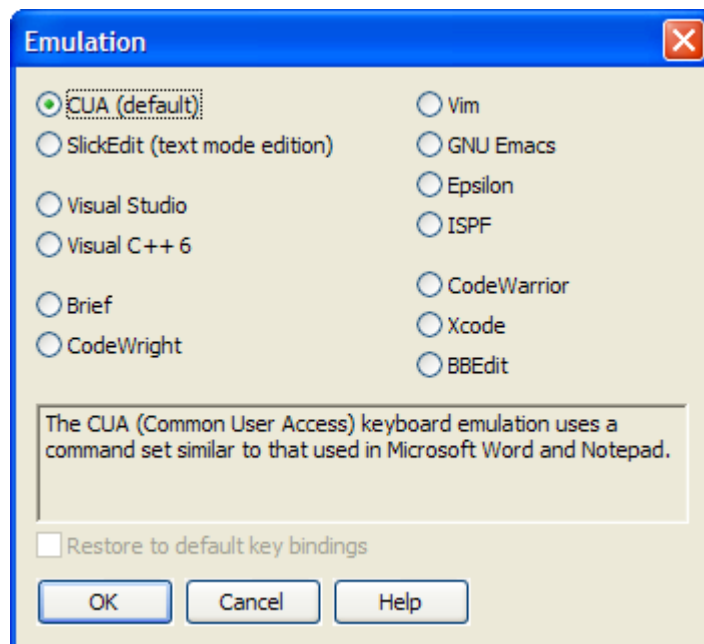
## Supported Emulations

This section describes each emulation mode and any special notes.

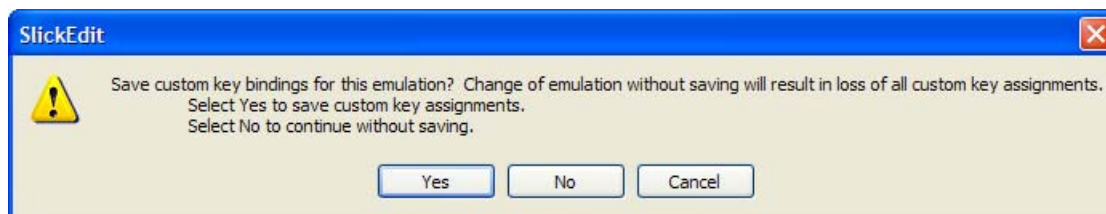
- **BEdit**
- **Brief** - This emulation relies heavily on Alt key bindings. In addition to Brief emulation support, SlickEdit® also supports Brief regular expressions. See [Regular Expression Syntax](#) for more information.
- **CodeWarrior**
- **CodeWright**
- **CUA** - CUA is an acronym for Common User Interface, a standard set of user interface guidelines similar to those used in Microsoft products. **This is the default emulation mode used by SlickEdit.**
- **Epsilon** - This emulation relies heavily on Ctrl+X and Escape (meta) key bindings.
- **GNU Emacs** - This emulation relies heavily on Ctrl+X and Escape (meta) key bindings. It does not include an Emacs Lisp emulator.
- **ISPF** - Support is included for ISPF prefix line commands, the ISPF command line, rulers, line numbering, and some XEDIT extensions. In addition to the ISPF emulation charts, additional documentation about using this emulation is available—see [Using the ISPF and XEDIT Emulations](#).
- **SlickEdit® (Text Mode Edition)**
- **Vim** - The Vim emulation contains special keys and key sequences that are case-sensitive. A plus (+) sign separates the simultaneous key presses. For example, the key binding **Ctrl+w W**, which moves the cursor to the window above, indicates to press *at the same time*, the Ctrl key and lower-case “w”, release, then press Shift plus “w” to insert the uppercase “W”. Another example is the key binding **gP**, which pastes the text before the cursor. Press the “G” key (which inserts a lower-case “g”), release, then press Shift plus “p” at the same time (which inserts the uppercase “P”).
- **Visual C++ 6**
- **Visual Studio** - The key bindings provided for the Visual Studio emulation are not the same as the key bindings used in Visual C++, but there might be some overlap. If Microsoft Visual Studio does not provide a default key binding for a particular SlickEdit command, the corresponding Visual C++ key binding is used.
- **Xcode**

## Changing Emulations

After SlickEdit® is installed, you are prompted to choose an emulation. CUA is the default emulation mode for SlickEdit. Key bindings and shortcuts mentioned in our documentation are based on this emulation. You can change emulation modes at any time by choosing **Tools > Options > Emulation**.



You should save custom key/mouse bindings for the current emulation before switching emulations. You can do this by exporting your custom bindings using the Key Bindings dialog (see [Exporting and Importing Bindings](#)), or you can save using the prompt that appears when you click **OK** on the Emulations dialog when switching emulations.



By saving your custom key bindings when you switch emulations, when you return to the original emulation those bindings are automatically available. For example, if you have created and saved custom bindings in the CUA emulation, and then switch to Vim, switching back to CUA will make your custom bindings for CUA available again.

To remove custom key bindings for an emulation, resetting to the defaults, select the **Restore to default key bindings** option on the Emulation dialog.

See [Managing Bindings](#) for more information on working with custom bindings.



## Determining Keys/Functions

When/if you switch emulations, the key bindings that are assigned to commands change according to the emulation chosen. You can use the Key Bindings dialog to look up what command is bound to what key or key sequence (or vice-versa), or you can use the SlickEdit® menu and command line to determine these items. See [Key and Mouse Bindings](#) and [Using the Command Line to View Key Binding Associations](#) for more information.



# Key and Mouse Bindings

---

Key and mouse bindings are quick ways to execute operations in SlickEdit®. Key bindings are the most efficient. Time is wasted each time you lift your hand from the keyboard to grab the mouse, and more time is wasted when you move your hand back to the keyboard in preparation for more typing. Therefore, if you learn the key bindings associated with operations that you use most frequently, you will save time coding. If an operation you use frequently isn't already bound by default, create your own easy-to-remember binding.

## What is a Binding?

A key or mouse binding is a key sequence or mouse event associated with a command. Key terms are defined as follows:

- **Mouse event** - The clicking of any button or motion of the mouse wheel.
- **Key sequence** - A series of one or more keys or key combinations. For example, Ctrl+X,R.
- **Key combination** - Two or more keys pressed simultaneously. For example, Ctrl+O.
- **Key** - Any single key on the keyboard.

An example of a key binding with one key combination is **Ctrl+O** (in CUA emulation, associated with the **gui\_open** command, **File > Open**, and the **Open** icon on the Standard toolbar). The plus (+) sign between the keys indicates that these keys must be pressed simultaneously: press the **Ctrl** and **O** keys at the same time. Note that the last key is case-insensitive. You do not need to press Shift.

An example of a key binding with a key sequence is **Ctrl+X,R** (in Vim emulation, this binding is associated with the **redo** command, **Edit > Redo**, and the **Redo** icon on the Standard toolbar). The comma (,) indicates that each key must be pressed consecutively: press **Ctrl** and **X** at the same time, release, then press the **R** key.

To view or change bindings, create new bindings, and export/import custom bindings, see [Key and Mouse Bindings](#).

The available key bindings change depending on the selected emulation. While SlickEdit provides emulations for 13 editors, CUA is the default emulation, so key bindings listed throughout the documentation will be for the CUA emulation. To change the emulation mode, select **Tools > Options > Emulation**. For more information, see [Emulations](#).

### NOTE

- For documentation purposes, both mouse events and keys that are bound to commands are often referred to collectively as *key bindings*.
- The SlickEdit® main menu displays the key bindings for commands associated with each menu entry. See [Accessing Menus](#) and [Creating and Editing Menus](#) for more information.
- A *menu hotkey* is a key sequence that corresponds to an underlined letter on a menu name. See [Menu Hotkeys](#) for more information about these items.

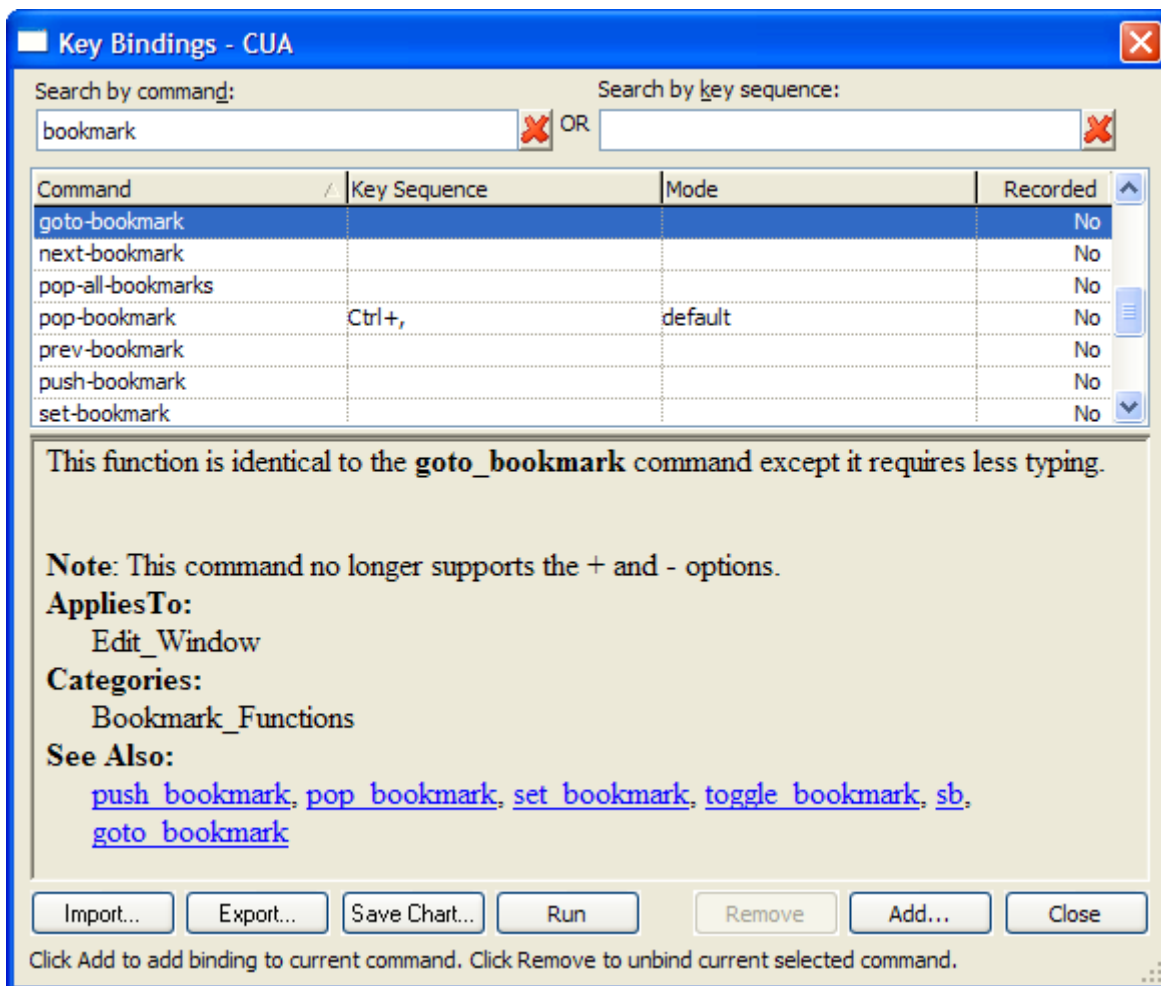
## Managing Bindings

Create and manage key bindings using the Key Bindings dialog. The dialog displays a list of all SlickEdit commands, including macros that you have recorded, their associated key sequences, and the language

editing mode in which the key binding can be used. Documentation for the selected command, if available, is also displayed. The Key Bindings dialog provides capabilities to incrementally search by command or by key sequence, export and import custom bindings, save an HTML chart of your bindings, and run a selected command or user-recorded macro.

To access the Key Bindings dialog, from the main menu select **Tools > Options > Key Bindings**, or use the **gui\_keybindings** command.

The first time the Key Bindings dialog is invoked, the Building Tag File progress bar may be displayed while Slick-C® macro code is tagged.



Bindings are based on the editor emulation mode (CUA is the default). The title bar of the Key Bindings dialog shows the current emulation. (To change the emulation mode, select **Tools > Options > Emulation**. For more information, see [Emulations](#).)

The **Search by command** and **Search by key sequence** boxes are used to filter the data. See [Viewing and Filtering Bindings](#).

The **Command** column shows all of the SlickEdit commands including macros that you have recorded. The **Key Sequence** column shows the key sequence or mouse event to which the command/macro is bound. If there is no binding, this field is empty. The **Mode** column shows the language editing mode to

which the binding is assigned. The **Recorded** column indicates if the item is a command (**No**) or user-recorded macro (**Yes**).

**TIP** What is a *language editing mode*? SlickEdit uses the extension of the current file to determine what language you are using, thereby only making available the options and features that are possible or useful in that language. You can also manually set the language editing mode. See [Language Editing Modes](#) for more information.

The bottom of the dialog contains documentation (if available) for the selected command.

Columns can be sorted by clicking on the column headers. An up or down arrow in the column header indicates ascending or descending sort order. All of the columns as well as the documentation pane can be resized by dragging the separator bars.

The sections below describe different ways to use the Key Bindings dialog. For a listing and descriptions of elements on this dialog, see [Key Bindings Dialog](#).

## Viewing and Filtering Bindings

You can filter the data in the Key Bindings dialog by using the **Search by command** and **Search by key sequence** boxes at the top. This is useful for finding a command/macro for creating, editing, or removing a binding, and for determining what key sequences are associated with a command/macro and vice-versa.

- To find a command/macro, search for it by entering a string in the **Search by command** box. The column of commands is filtered incrementally as you type, to show only commands that contain the specified string. Commands that have more than one key sequence associated with them are listed on separate rows. For example, in CUA emulation, the command **gui\_open** is bound to F7, Command+O (on the Mac), and Ctrl+O. Therefore, **gui\_open** appears in the Command column three times—one row per key sequence.
- To find a key sequence, place the focus in the **Search by key sequence** box (by tabbing or using the mouse) and then press the actual key or key sequence. The column of key sequences is filtered to show only bound sequences that contain the specified key(s). For example, to see all commands/macros that are bound to Ctrl+O, with the focus in the search box, simply press Ctrl+O.

To clear either field, click the red **X** button to the right of each box. This is especially handy for the key sequence search, due to the fact that the field recognizes any keyboard/mouse input including Backspace.

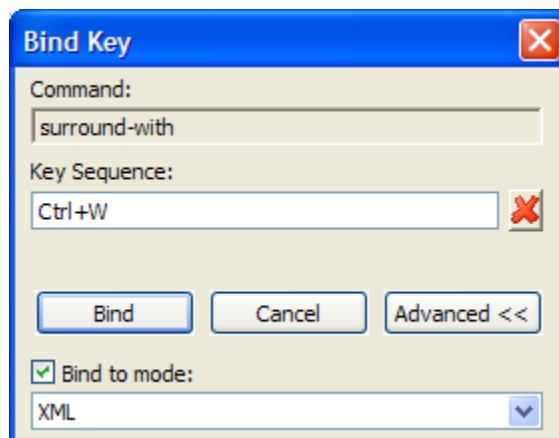
Alternatively, you can use the **what\_is** and **where\_is** commands (**Help > What Is Key** and **Help > Where Is Command**) on the SlickEdit command line to determine binding associations. See [Using the Command Line to View Key Binding Associations](#) for more information.

## Creating Bindings

You can work more efficiently if you create key/mouse bindings for commands or user-recorded macros that you use frequently. To create a new key or mouse binding:

1. Using the Key Bindings dialog, find the command or user macro you want to bind. You can search for a command/macro by entering a string in the **Search by command** box (see [Viewing and Filtering Bindings](#)).
2. Initiate the binding by using one of the following methods:
  - Select the row, then click the **Add** button.
  - Select the row, then press **Enter**.
  - Double-click on the row.

- When you initiate a binding, the Bind Key dialog is displayed with focus in the **Key Sequence** box.



- For a key binding, press the key sequence just as you would to use it. For example, to bind **surround\_with** to Ctrl+W, simply press Ctrl+W. The key sequence you pressed is displayed in the box.
- For a mouse binding, click the mouse button you want to use. For example, to bind **surround\_with** to the right-click mouse event, simply right-click with the mouse, and RButtonDown is displayed in the box.

Use the red **X** button to clear the input field if you make a mistake. If you enter a key sequence or mouse event that is already assigned to another command/macro, a warning prompt is displayed. If you continue, the previous binding is unbound and reassigned.

#### TIPS

- SlickEdit allows key sequences that are very long, but shorter sequences are easier to remember and more practical to use.
- Do not begin key sequences with keys that are normally used in typing. Otherwise, these keys will launch the operation and not appear when you type. For example, binding a command to the A key will prevent you from using that letter in your code. It is best to always begin your key sequences with a Ctrl or Alt key combination.

- The **default** language editing mode is the default language editing mode for new bindings, which means the binding will work in all language editing modes. If you want the binding to work only in a specific language editing mode, you can change it now by clicking the **Advanced** button on the Bind Key dialog. Click **Bind to mode**, then from the drop-down list, select the mode for which the binding should apply. Bindings assigned to a specific language editing mode override those assigned to **default**.

**TIP** You can create multiple bindings for the same command/macro and have one binding set to default and the others set to specific modes. In this case, when you are editing in a specified mode, that binding is in effect, and when editing in any other language editing mode not specified, the default binding will be in effect. For example, in CUA emulation, Ctrl+L is bound to **select\_line** by default, but when in HTML mode, you may want to use Ctrl+L to insert an HTML link instead (**insert\_html\_link** command). Therefore, you can bind Ctrl+L to **insert\_html\_link** and specify the HTML mode for use only when editing HTML files.

5. When finished, click **Bind**. The key sequence or mouse event now appears in the Key Sequence column on the Key Bindings dialog.

## Editing Bindings

To change the binding or language editing mode for a command/macro that is already bound, you will need to first unbind the command/macro, then recreate it. See [Removing Bindings](#) and [Creating Bindings](#). If you have advanced knowledge of SlickEdit®, you can edit the Slick-C® key binding source directly. See [Editing the Key Binding Source](#) for more information.

## Removing Bindings

To remove a binding:

1. Using the Key Bindings dialog, find the command/user macro or key sequence that you want to unbind. You can search by using the search boxes at the top (see [Viewing and Filtering Bindings](#)).
2. With the command/macro row selected, click **Remove**, or press **Delete**. You are prompted to confirm the unbind operation.

## Exporting and Importing Bindings

Key and mouse bindings can be exported out of SlickEdit® and imported in, useful for creating backups, sharing with other team members, or taking with you should you switch computers.

### Exporting Bindings

When you export bindings using the Key Bindings dialog, custom bindings for all language editing modes in the current emulation are exported into an XML file with a name and location that you can specify.

To export your bindings:

1. Click the **Export** button on the Key Bindings dialog. The Save As dialog is displayed.
2. If you want, change the directory location and change the file name to something more meaningful to you, such as `myname_cua.xml`.
3. Click **Save**.

### Importing Bindings

Imported bindings override any existing bindings for the selected emulation. For example, if you have the **surround\_with** command bound to Ctrl+W, and import **surround\_with** bound to Ctrl+Q, then Ctrl+Q is now the binding for that command in the selected emulation. When you import for the selected emulation, SlickEdit resets the key bindings to the default, then loads the user key bindings.

If you import a key bindings file from a different emulation than the one currently selected, SlickEdit displays a warning and prompts whether or not you want to continue. If you continue, the emulation mode is changed and the key bindings are loaded for that emulation.

To import bindings into SlickEdit:

1. Click the **Import** button on the Key Bindings dialog. The Open dialog is displayed.
2. Find and select a bindings file that was previously exported, then click **Open**.

## Saving a Bindings Chart

Click the **Save Chart** button on the Key Bindings dialog to save an HTML reference chart of all current bindings for all language editing modes in the selected emulation. Commands and user macros that are not bound are not included.

## Running a Command/Macro using the Key Bindings Dialog

If you have the Key Bindings dialog open, you can conveniently run a selected command or user-recorded macro by clicking the **Run** button.

## Resetting Default Bindings

To reset bindings for the selected emulation to the SlickEdit® defaults, from the main menu, select **Tools > Options > Emulation**, then select the **Restore to default key bindings** option on the Emulation dialog. See [Emulations](#) for more information.

## Key Binding Settings

The following are settings that you can make pertaining to key bindings.

### Key Message Delay

For key bindings that contain multiple key combinations, like Ctrl+X, Ctrl+C, you can specify the maximum delay between the two combinations. If that time limit is exceeded, this key sequence will be interpreted as two separate bindings, executing the command bound to Ctrl+X followed by the command bound to Ctrl+C, rather than the command bound to Ctrl+X, Ctrl+C.

To change this option, choose **Tools > Options > General**, then select the [More Tab](#). In the **Key message delay** spin box, enter the amount to delay before a prefix key in tenths of a second. The prefix key is not displayed if the next key is pressed before the delay specified in this text box.

### Using Shorter Key Names in Menus

The SlickEdit® main menu displays the key bindings for commands associated with each menu entry. These bindings can be condensed for non-CUA emulations. See [Short Key Names in Menus](#) for more information.



## Cursor, Mouse, and Scroll Settings

---

This section describes settings for the cursor, mouse, and scroll style. For cursor navigation information, see [Cursor Navigation](#).

### Setting the Cursor Style

You can use a text mode style cursor instead of a vertical cursor. To set this option, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Use block cursor**.

### Hiding the Mouse Pointer

To hide the mouse pointer when typing, from the main menu choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Hide mouse pointer**. The mouse pointer is then only displayed when moving the mouse or when a dialog box is displayed.

### Displaying Tool Tips

By default, hovering the mouse pointer over a button displays a tool tip about the item. To turn tool tips off or to change the amount of time before tool tips are displayed, from the main menu choose **Tools > Options > General**, then select the [More Tab](#). Deselect the option **Show tool tips**, or change the value in the **Delay** spin box. The Delay value is in tenths of a second.

### Scroll Bar and Scroll Style Settings

The scroll bars on the right and bottom edges of the editor windows are optional in SlickEdit. To turn these on or off, from the main menu choose **Tools > Options > General**, then choose the [General Tab](#). Select or deselect the options **Horizontal scroll bar** and/or **Vertical scroll bar**. When these options are selected, the scroll bars are displayed. These options do not affect edit window controls on dialog boxes.

To set the scroll style, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Select the **Scroll style** setting that you wish to use. Commands that move the cursor more than one page of text, such as searching, always center scroll text into view. The following scroll settings are available:

- **Center** - When center scrolling is on and the cursor moves out of view the cursor will be centered and the text will move by half the height or width of the window.
- **Smooth** - Smooth scrolling is a line by line scroll of the screen that occurs when the cursor moves out of view. Smooth scrolling is the default configuration.
- **Scroll when** - Specifies how close (in number of lines) the cursor may get to the top or bottom of the window before scrolling occurs. Does not affect horizontal scrolling.



## Setting Fonts and Colors

This section describes how to set fonts and colors for screen elements and editor windows. For information about changing the colors of code, such as colors used for keywords, see [Color Coding](#).

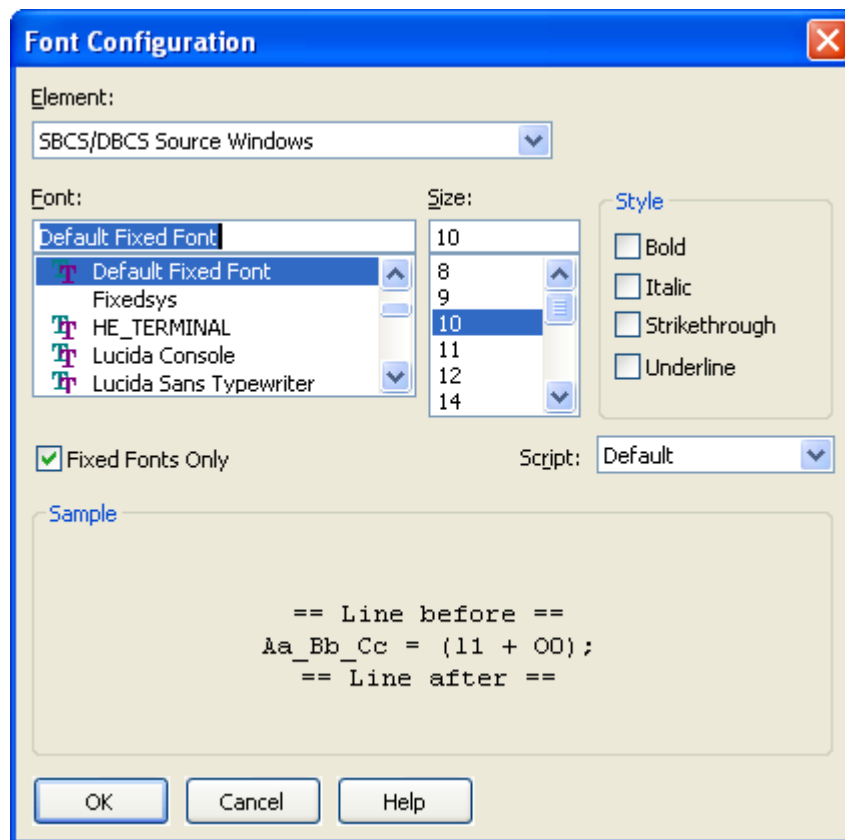
### Fonts

SlickEdit® provides the capability to change the fonts used by edit windows, the command line, status text, and other screen elements. Recommended fonts are listed. You can also set fonts for editor windows.

**TIP** Xft fonts are supported on Linux.

### Setting Fonts for Screen Elements

To configure font settings for screen elements, use the Font Configuration dialog, pictured below. To access this dialog, from the main menu, choose **Tools > Options > Font**.



For descriptions of the options on this dialog, see [Font Configuration Dialog](#).

### Recommended Fonts for Elements

Font recommendations are given for the best screen display. The information below contains recommended fonts for some of the screen elements.

**NOTE** Some font names are portable font names which are translated into other fonts. This allows Slick-C® macros and dialog boxes to be portable across Windows and UNIX.

### Command Line Fonts

The following table contains recommendations, based on the operating system, for the Command Line font element:

Platform	Font Recommendation
Windows	Choose Courier, Courier New, OEM Fixed Font, or Terminal fonts for the most visually appealing character displays.
Linux	Choose Courier, Lucida Sans Typewriter or a console font for the most visually appealing character displays. If these fonts are not visible, look for the UNIX fonts below.
UNIX	Choose Adobe Courier, B&H Lucida Typewriter, or Width x Height family fonts for the most visually appealing fixed fonts.

### Selection List Fonts

Choose **Courier** for best display of selection lists that need a fixed font.

### Dialog Box Fonts

Choose **MS Sans Serif** as an attractive font for dialogs.

### Text Box Fonts

Choose **System** or **MS Sans Serif** for fonts used in text boxes.

### SBCS/DBCS Source Window Fonts

This is the element used for all non-Unicode source windows. Choose **Terminal** for the most attractive visual display.

### Unicode Source Window Fonts

**Default Unicode Font** is the default font for the Unicode Source Windows element. When this font is selected, the **Arial Unicode MS** font is used if it is installed. Otherwise, the **ANSI Fixed Font** is used, which only supports the English character set. Arial Unicode MS is a fairly complete font which is included with Microsoft Office. Currently, no version of Windows ships with a complete Unicode font. For more information on Unicode support, see [Using Unicode](#).

### Setting Editor Window Fonts

In addition to setting the fonts for screen elements, you can specify the fonts and font styles for editor windows. To change editor window fonts, from the main menu choose **Window > Font**, or use the **wfont** command. The Window Font dialog will be displayed. Font, size, and style options are the same as for the Font Configuration dialog, described above.

From the Window Font dialog, choose the **Scope** that you wish to affect. Select the **Current window** option if you only want to change the current window's font. Select the **All windows and Default** option to set the font for all editor windows that are open as well as newly-created windows.

For a complete list of the options on the Window Font dialog, see [Window Font Dialog](#).

## Colors

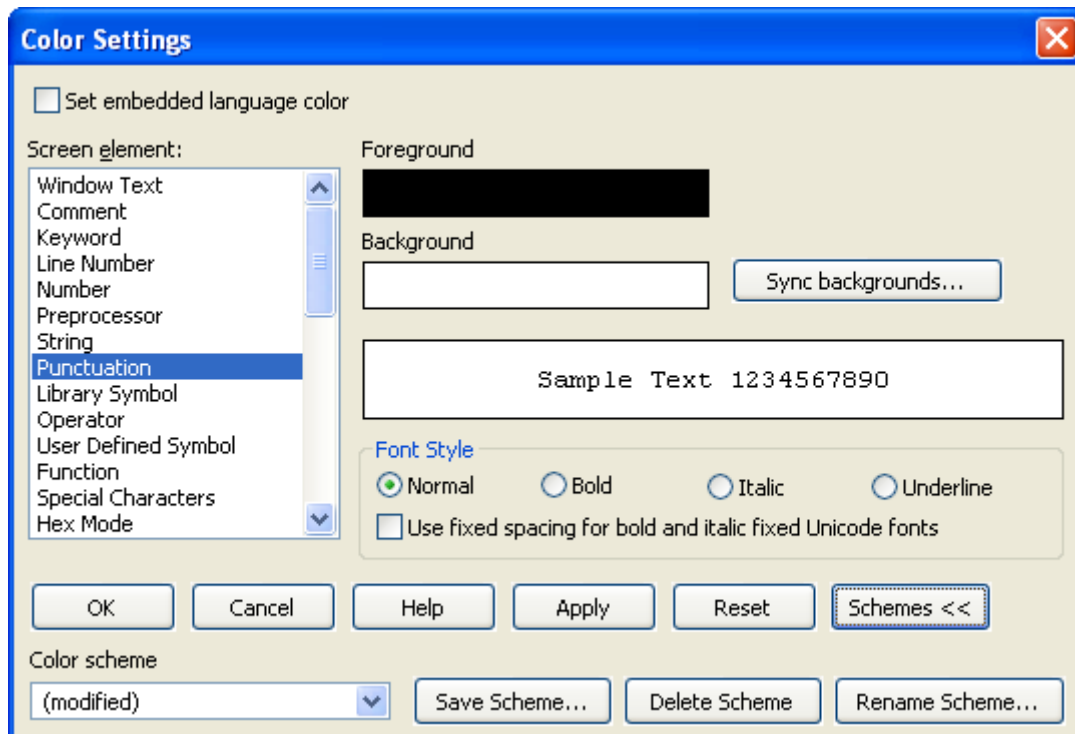
Use the Color Settings dialog (**Tools > Options > Colors** or **color** command) to set the color for different screen elements in SlickEdit®. This includes syntactic elements in the editor window, like keywords, comments, strings, etc. as well as other user interface elements like the message area or the status line. Window colors and backgrounds are set using the facilities provided by the operating system.

Color and Color Coding are different. Color Coding is a feature that combines current line coloring, modified line coloring, and language-specific coloring. SlickEdit recognizes and automatically displays color support for many languages. See [Color Coding](#) for more information.

## Setting Colors for Screen Elements

Colors can be customized in the user interface. Colors can be set either individually or by editing a scheme. To change the default colors, complete the following steps:

1. From the main menu, select **Tools > Options > Color** (or use the **color** command). The Color Settings dialog box is displayed.



2. Select the element you want to change from the **Screen element** list box (for descriptions of some of these elements, see [Color Settings Dialog](#)).

3. Set the **Foreground** and **Background** colors by clicking on the color squares. The Color Picker dialog is displayed, allowing you to pick a color from the palette, or set your own custom color using RGB values.

**NOTE** If you have chosen the **Selection** screen element, note that SlickEdit will attempt to render selections using your normal color settings for the Foreground color. The selected foreground color will only be used if there is not enough contrast between the font colors to be readable. It is best to specify a Background color for selections that is as close as possible to your normal background color, ensuring that the color-coded fonts are still easy to read.

4. If you change the background color for an element in the editor window, you can use the **Sync Backgrounds** button to propagate the background color for the currently selected element to other related elements. For example, if you change the background color for Keywords you will probably want that same color used for Strings, Comments, Numbers, etc. The **Sync Backgrounds** button prevents you from having to manually make all these changes.
5. If you want, choose a **Font Style** for the text.
6. Click **Apply** to update the colors that you have modified without closing the dialog box, or click **OK** to apply the changes and close the dialog.

For a complete list of all of the options available on the Color Settings dialog, see [Color Settings Dialog](#).

### Using Color Schemes

Color schemes store the settings for all screen elements, allowing you to quickly change the look of SlickEdit®. Several predefined color schemes are provided, and you can create your own.

To use color schemes, click the **Schemes** button on the Color Settings dialog (see the screen capture above). To try a different color scheme, from the **Color scheme** drop down text box, select a color scheme and click **Apply**. A sample of the color scheme is displayed in the **Sample** text box. To use this color scheme, click **OK**.

To define a new color scheme, set your colors for the various screen elements and click **Save Scheme**. User-defined color schemes are stored in the `uscheme.ini` file located in your configuration directory. You can change the name of a scheme by clicking **Rename Scheme**.

### Setting an Embedded Language Color

The option **Set embedded language color** allows you to specify the colors used for embedded languages. These occur when a file of one type embeds a language of another type within it, like HTML files containing JavaScript. For HTML, the syntax color coding recognizes the `<script language="???">` tag and uses embedded language colors for the new language. In addition, for Perl and UNIX shell scripts, you can prefix your **HERE** document terminator with one of the color coding lexer names to get embedded language color coding. The following is an example for Perl:

```
print <<HTMLEOF
<HTML><HEAD><TITLE>...</TITLE></HEAD>
<BODY>
...
</BODY>
</HTML>
HTMLEOF
```

## Restoring Settings on Startup

---

By default, the files, current directory, and more from the previous edit session are automatically restored when you switch workspaces or close and re-open SlickEdit®.

To access auto restore settings, from the main menu, choose **Tools > Options > General**. The [General Options Dialog](#) is displayed with the [General Tab](#) in view. The **Auto restore** options, listed below, control which elements of your SlickEdit environment that are restored.

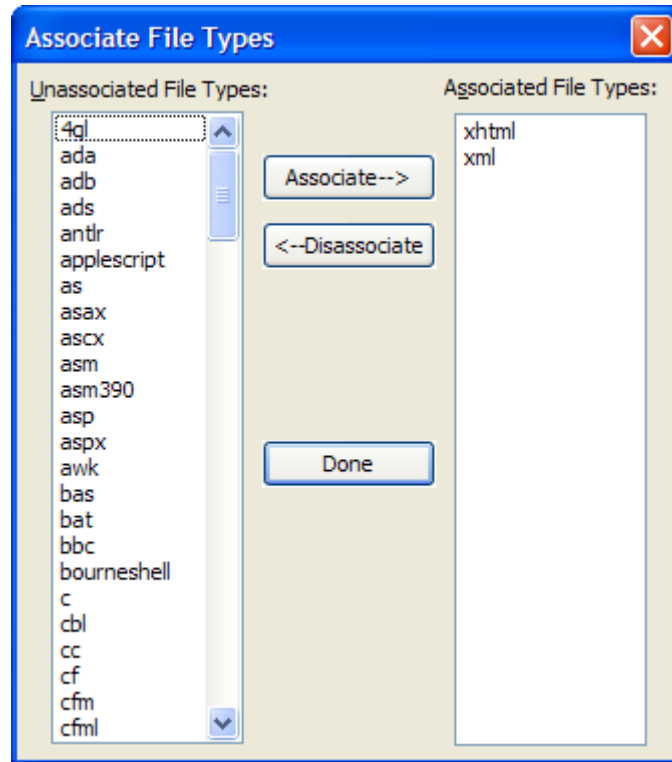
- **Files** - If checked, the files and windows that were opened in your last edit session are restored and re-opened when you start the editor.
- **Clipboards** - If checked, clipboards are saved across edit sessions.
- **Working directory** - If checked, the working directory is saved across edit sessions.
- **Build window** - If checked, the concurrent process buffer is saved across edit sessions.
- **Workspace files** - If checked, when switching workspaces, the files and windows that were opened for a workspace when it was last closed will be restored. See [Workspaces and Projects](#) for more information.
- **Line modify** - If checked, the line modification flags are saved and restored when you save and open files, respectively. Line modification information for the last 200 files is saved. SlickEdit stores line modification information in temporary files placed in the `selDisp` directory. This option works best when the **Modified Lines** color coding option is selected (**Tools > Options > File Extension Setup**, select the [Advanced Tab](#)).
- **Selective display** - If checked, Selective Display is saved and restored when saving and opening files, respectively. Selective Display for the last 200 files is saved. See [Selective Display](#) for more information.
- **Symbol browser tree** - If checked, the symbol browser tree (see [Symbols Tool Window](#)) is restored across edit sessions. The current position (displayed selected) is always restored regardless of this setting.





## Setting File Associations

Files of certain types can be associated with SlickEdit®, so that when those files are opened from Windows Explorer or the file manager, they run in the SlickEdit application. To set up these associations, use the Associate File Types dialog, which can be accessed by selecting **Tools > Options > Associate File Types** (or by using the **assocft** command).



Select the file types you wish to associate in the **Unassociated File Types** list box. Click the **Associate** button to move them to the **Associated File Types** list box. Use the **Disassociate** button to remove associations from the list. Click **Done** when you are finished.

**NOTE** If you are associating workspace files ( `.vpw` extension) to SlickEdit, SlickEdit restores the edit session and the project when opening the `.vpw` file. See [Workspaces and Projects](#) for more information about working with projects and workspaces.



# Workspaces, Projects, and Files

This chapter contains the following topics:

- [Workspaces and Projects](#)
- [Creating, Opening, and Saving Files](#)
- [Using the File Manager](#)
- [File History and Backups](#)
- [Code Templates](#)



# Workspaces and Projects

---

## Overview of Workspaces and Projects

Workspaces and projects provide a way to organize your work. Much of the power provided by SlickEdit® derives from the information in your projects. So, it's important to set them up correctly.

A **workspace** defines a set of projects and retains the settings for an editing session. Opening a workspace returns a session to the same state as when you last worked on it, including which files are open, the working directory, and more. To see the auto restore options, select **Tools > Options > General** and then select the [General Tab](#). The data for each workspace is stored in a text file with the extension `.vpw`.

A **project** defines a set of related files that build and execute as a unit. For each project you can specify the set of files it contains, a working directory, a set of commands to build and execute the project, compiler options, and dependencies between other projects. Files can only be added to projects, not directly to a workspace. A file may belong to multiple projects, and a project may belong to multiple workspaces. The data for each project is stored in a text file with the extension `.vpj`.

When you create a project, you select the project type based on the language and compiler you are using. Selecting the right project type is essential to configure SlickEdit to build and run your program. Once a project type is selected, it is not possible to change it. For more information on this topic, see the section on [Project Types](#).

The number of projects you create in a given workspace depends on the type of program you are creating. Typically, you create a separate project for each build target in your program. In C/C++ you would create a separate project for each DLL or SO and one for each executable. In Java, you might only create a single project.

If you have a workspace with multiple projects, you can use project dependencies to ensure that projects are built in the correct order (see [Defining Project Dependencies](#)). You may find it useful to define an *umbrella project* that depends on all other projects. This provides an easy way to rebuild all of your projects. Even if you have no project that meets this criterion, you can create an empty project for that purpose.

Files in a workspace are processed by the Context Tagging® feature, building a database of the symbols they contain. This information is used for completions, providing parameter information, navigating from a symbol to its definition or references, and more. The Context Tagging database provides near-instantaneous access to information for which you would otherwise have to search, saving you a great deal of time.

You can define as many workspaces as you like. For large systems that decompose into multiple subsystems and programs, you can create a separate workspace for each program or subsystem. This helps you manage the complexity by limiting the number of files in your workspace. It also prevents irrelevant information from being presented by Context Tagging when doing symbol lookups.

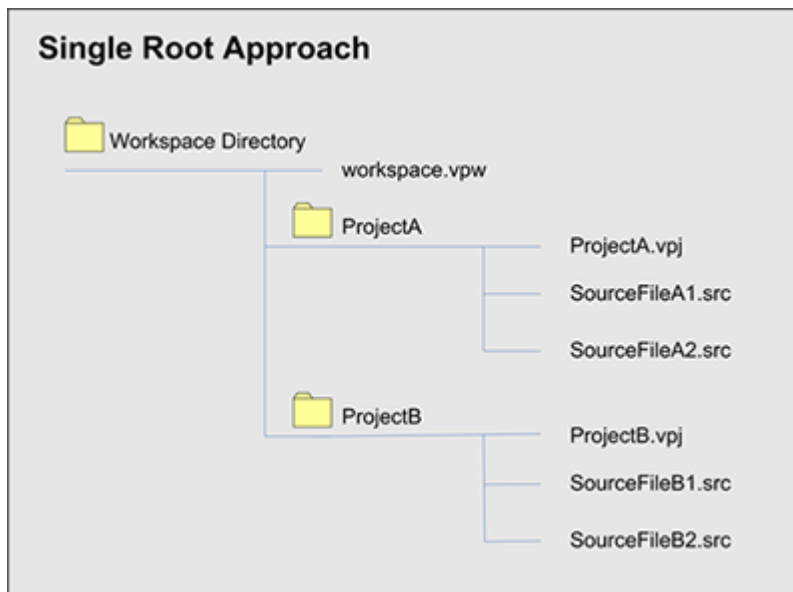
SlickEdit has a default workspace that is active before you define a workspace or after you close a workspace. However you can not add projects or files to this workspace. You can open files and edit them, and state will be saved, but using SlickEdit in this way is like using a basic editor and will not provide the full benefit of SlickEdit's symbol analysis. For more information on these features, see [Context Tagging® Overview](#).

## Organizing Files

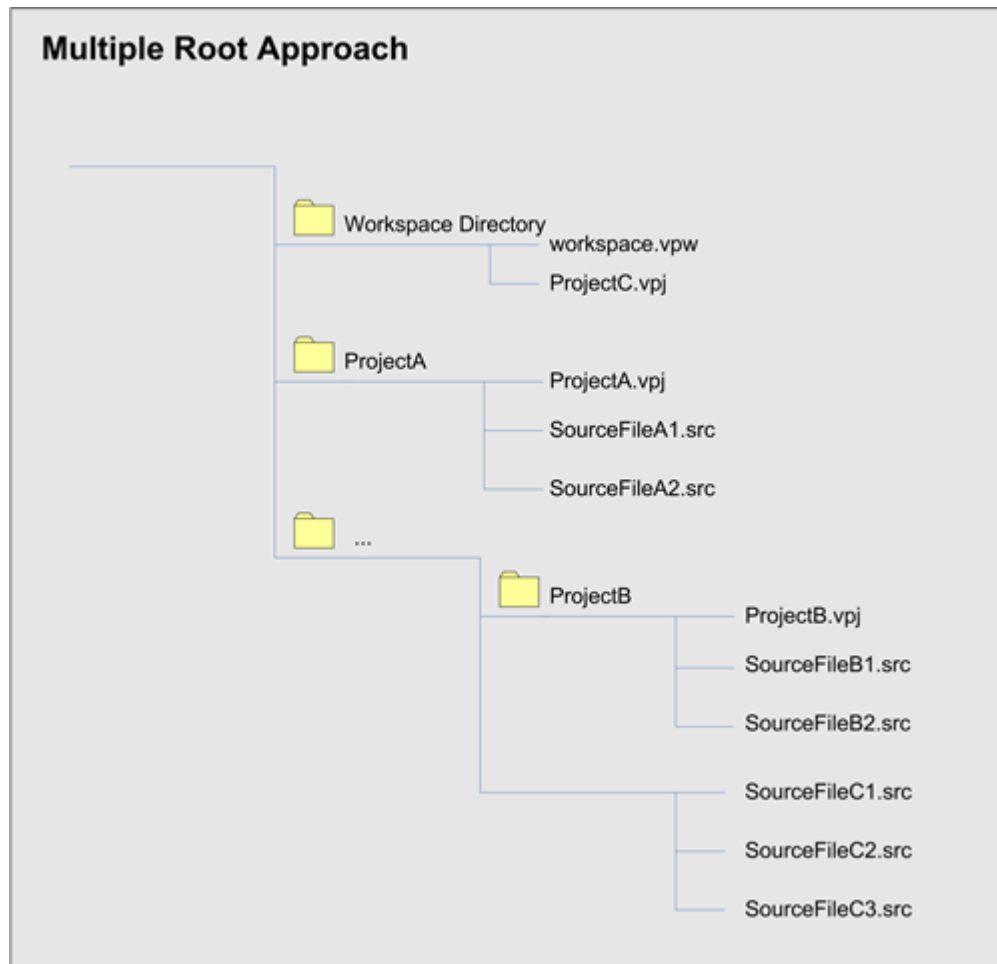
SlickEdit® places no restrictions on the location of your files. Your source files do not have to be located in the same directory as your project (`.vpj`) or workspace (`.vpw`) files. Adding files to a project does not copy the files to a new location. It simply associates those files with the project. Likewise, adding an existing project to a workspace does not copy the project. This gives you a great deal of flexibility to organize your files.

In general, there are two approaches to organizing your files:

- **Single Root Approach** - All files are stored under a single root directory. In this approach, the workspace `.vpw` file is typically located in the root folder and there is a subfolder for each project. Each project folder contains the project `.vpj` file and the source files.



- **Multiple Root Approach** - No single folder contains all of the workspace files exclusively. Each project may be created in a different directory unrelated to each other, and the workspace `.vpw` file may be placed in yet another directory.



The single root approach is common when all team members are working with SlickEdit. This organization provides a simple approach to storing your files and facilitates interaction with source control systems. The multiple root approach may be used on complex programs that share framework code with other programs. In this case, you may not want to duplicate the shared code to place it under the root directory for this program.

A hybrid approach may be used if you have a single root source hierarchy but the whole team is not using SlickEdit. In that situation, they may not want to have the SlickEdit project (.vpj) and workspace (.vpw) files checked into the same location as the source files. You can still check out the source tree into a single directory. Then define a separate subdirectory for your workspace and project files. If other team members are using SlickEdit, you can check these files into a different area in source control, allowing you to share them with other SlickEdit users but not interfere with non-SlickEdit users.

Storing files remotely will have a definite impact on performance since network latency is added to disk latency. If your standards require you to work with remote files, you should still try to set up your workspace locally. SlickEdit reads and writes workspace files frequently, so storing them remotely will reduce performance. Workspaces are just a list of projects, so setting up a local workspace is quick and easy.

## Version Control

If the whole team is using SlickEdit®, then project (.vpj) and workspace (.vpw) files should be checked in any time they are updated by SlickEdit. That way, all team members will see any new files or projects you

add to the workspace. Even if you are the only person using SlickEdit, it's a good idea to check in your project and workspace files. This protects you from loss and allows you to fall back to earlier versions of the program.

Do not check in the `.vpwhist` file. It contains breakpoints, bookmarks, file positions, list of open files—information that is unique to an editing session.

For more information about using version control with SlickEdit, see [Version Control](#).

### Add Wildcard

If all team members are using SlickEdit® for their development, you will pick up newly added files by getting the latest version of the `.vpj` files. If some of your teammates are not using SlickEdit, then you need to use **Add Wildcard** when adding files to your projects in order to pick up added files. Each time you start SlickEdit, the wildcards are evaluated and the file list is updated (see [Managing Source Files](#) for more information).

### Working with Libraries

A typical program also makes calls to library routines. A library is a pre-built unit of code providing application-independent functionality. Standard libraries are provided by the compiler, and many programs use third-party libraries. Some development projects have their own libraries.

Libraries should not be added to your workspace as a project. The key distinction is that libraries are pre-built and will not be edited as part of the development effort. If you have library routines that you plan to edit and build as part of your development, these should be added to your workspace as a project.

SlickEdit® automatically tags the standard libraries for C/C++, Java, .NET, and COBOL as part of normal installation. This adds the same type of symbolic information for these libraries to the symbol database that is created for your source code. If you skipped auto-tagging or you switch compilers and need to tag those libraries, you can re-run auto-tagging by selecting **Tools > Tag Files** and then click the **Auto Tag** button.

If you use third-party libraries or your own internal libraries, you will want to tag them as well. See [Creating Extension-Specific Tag Files](#) for instructions on how to tag libraries.

## Managing Workspaces

Workspaces are just a means to aggregate projects and store values from an editing session. They are easy to create, and you can quickly switch from one workspace to another. The Projects tool window allows you to browse the projects within a workspace and the files contained in those projects. It is docked as a tab on the left side of the editor by default and display can be toggled by selecting **View > Toolbars > Projects**.

### Opening and Closing Workspaces

To open an existing workspace, select **Project > Open Workspace** (`Ctrl+Shift+O` or `workspace_open` command). Locate the workspace file (`.vpw`) and click **Open**.

To close a workspace, select **Project > Close Workspace** (`workspace_close` command). Only one workspace can be open at any given time. If you open another workspace when one is already open, the existing workspace will be automatically closed first.



## Creating Workspaces

Workspaces are typically created by creating a new project. At that time you have the option to create a new workspace for this project or add it to the current workspace. If no workspace is open, you can only elect to create a new workspace. If you choose to create a new workspace, your current workspace will be closed and the new one opened. For more information on creating projects, see [Creating Projects](#).

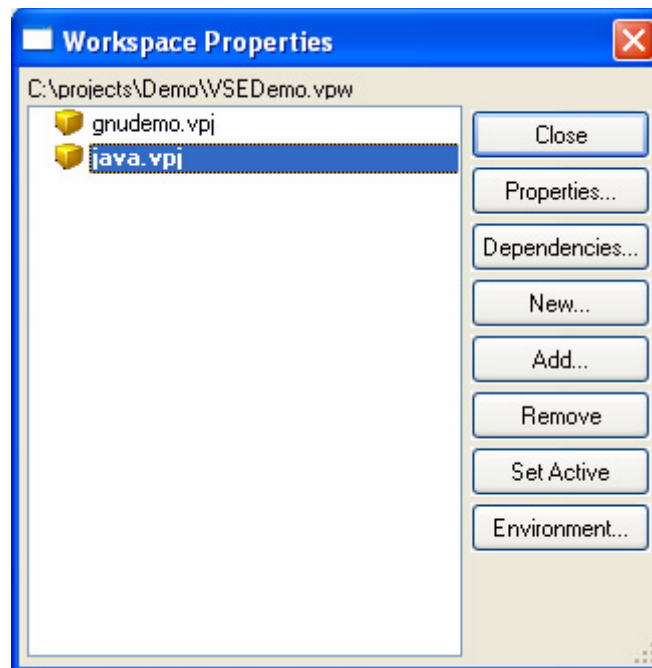
Creating a workspace without creating a project is useful when you plan to import existing projects or if you want a workspace for editing files that aren't part of a project, like shell script files.

To create a workspace without creating a project, complete the following steps:

1. From the main menu, select **Project > New** and click the [Workspaces Tab](#).
2. Type a value in the **Workspace name** field.
3. Type the location (or use the **Browse** button to the right of this field to pick a location) for the new workspace.

## Managing Projects within a Workspace

To list projects in the current workspace, add or remove projects from the current workspace, or to set the active project, use the Workspace Properties dialog box. The dialog, pictured below, can be accessed from the main menu by choosing **Project > Workspace Properties**.



For more information on using this dialog, see [Workspace Properties Dialog](#).

## Sharing Projects between Workspaces

A project can be used in more than one workspace. Adding an existing project to a workspace does not copy the project or its source files—it simply creates an association between the two. If you want a local copy of the project, you will need to copy it before you add it to the workspace.

To share an existing workspace, complete the following steps:

1. From the main menu, select **Project > Workspace Properties**.
2. Click **Add**.
3. Locate the file and click **Open**.

## Working with Third-Party Workspaces

SlickEdit® provides compatibility with the following third-party workspaces. Use **Project > Open Workspace** to open these project types.

- Visual Studio .NET - SlickEdit can directly open `.sln` and `.vpw` files. Visual Studio 2005 has added many new project types, and not all of these are supported. If you have difficulty opening a Visual Studio 2005 solution or workspace, please contact Product Support. SlickEdit cannot create Visual Studio workspaces. You need to create the workspace in Visual Studio and define the project structure there. Once created, SlickEdit can add files to existing projects.
- Tornado
- Xcode

## Managing Projects

The most important thing to remember when working with SlickEdit® projects is to use the correct project type. When you create a project, you are presented with a list of project types. These determine the build options and other behaviors for the project. Once created, the project type cannot be changed.

Once a project has been created, you need to add the source files to it. See [Managing Source Files](#) for more information.

## Project Types

SlickEdit® provides a variety of project types that match commonly available languages, compilers, and types of programs. Some project types create main programs, starting you off with a fully compilable program.

The **(Generic)** project type is provided for use when a specific project type does not match the language or compiler you are using. When you use this project type, you are responsible for configuring all build, run, and debug commands.

You can create a custom project type if none of the existing project types match your development. By doing this you avoid having to redefine the build, run, and debug commands each time you set up a new project. You can customize the **(Generic)** project type to create a completely new project type or customize one of the language-specific project types, like Java, to tailor it to your needs. See [Creating Custom Project Types](#) for more information.

The sections below describe the most commonly used project types.

### GNU C/C++

SlickEdit® provides a GNU C/C++ Wizard that leads you through the configuration options for setting up a new GNU C/C++ project. Using this wizard, you can quickly configure a new project that will build, run, and debug.

SlickEdit prompts you whether this project will build an executable, a shared library, or a static library. You can specify whether this project will use C++, C, or ANSI C. Further, you can select whether to create an

empty project, an application with a **main()** function, or a “Hello World” application. Finally, you are prompted whether to use SlickEdit’s build system or to use a makefile.

SlickEdit detects the presence of GNU tools on your system and configures the new project correspondingly. You can make changes to these settings by selecting **Project > Project Properties**.

See [C and C++](#) for more information about SlickEdit’s C/C++ features.

## Microsoft Visual Studio

SlickEdit® cannot create Visual Studio workspaces or projects. Visual Studio users should create workspaces, called *solutions* in Visual Studio, and define the projects it contains using Visual Studio. Once created, you can open the new solution in SlickEdit by selecting **Project > Open Workspace**. Navigate to the directory containing the solution and select the `.sln` file to be opened. SlickEdit reads the `.sln` file and configures the build, run, and debug operations to be performed just as they would in Visual Studio.

**NOTE** You can add files to projects using SlickEdit, but any modifications to the workspace or project settings must be performed using Visual Studio. This includes adding any new projects to the workspace.

SlickEdit lists a number of Visual Studio project types on the New Project dialog, but most of them will simply warn you that SlickEdit cannot create a project of that type and that you need to do that in Visual Studio. The project types for the Microsoft SDKs, however, can be used. These create SlickEdit projects that are not compatible with Visual Studio. This provides a low-cost way to use the Microsoft SDK compilers and still have a rich editing environment.

## Other C/C++ Compiler Compatible with GDB (UNIX only)

Some compilers, like the Sun™ compiler, are compatible with the GNU tool chain. For these, you should start with the GNU C/C++ project and customize it to use the compiler, debugger, and do builds the way you want. Doing this allows you to launch the integrated debugger using **Debug > Step Into** rather than **Debug > Attach > Debug Other Executable**. (See [C and C++](#) for more information about C/C++ features in SlickEdit®.)

## Other C/C++ Compiler

SlickEdit® provides a project type for Borland® C++, both 16- and 32-bit for Windows and for Symantec™ C++. These were created for older versions of these products and may not work with the most recent versions. In that case, or when using any other C/C++ compiler, you should select the **Generic C/C++** project type. You will then have to configure the build, compile, link, run, and debug commands for both the Release and the Debug configurations. (See [C and C++](#) for more information about C/C++ features in SlickEdit.)

## Java

SlickEdit® provides a broad selection of Java project types. Select the appropriate choice based on whether you are creating an applet or application and the type of program. SlickEdit detects the installed JDK on your system and configures the build, run, and debug commands. (See [Java](#) for more information about SlickEdit’s Java features.)

## Dynamic Languages: Perl, Python, PHP, Ruby

Dynamic languages do not get compiled. Most of the settings in the project types provided in SlickEdit® are related to compiling and debugging your program. Therefore, SlickEdit has no project types that are specific to these languages. Use the **(Generic)** project type and add your files there. You can configure run

and debug commands by selecting **Project > Project Properties** and selecting the [Tools Tab](#). Create a custom project type for this language to avoid having to redundantly configure projects each time you create them (see [Creating Custom Project Types](#)).

### Creating Projects

To create a project, complete the steps below.

1. From the main menu, select **Project > New**.
2. Select the type of project that you want. It is critical that you select the correct project type. Please see [Project Types](#) above for a full discussion of project types.
3. Type the project name.
4. Select a directory location. If the directory does not exist, a prompt appears to create it when you click **OK**.
5. Type the name of the executable file or output file.
6. Select either **Create new workspace** or **Add to current workspace**. If adding this project to the existing workspace, specify whether this project depends on another project in this workspace by checking the **Dependency of** check box and selecting the depended-on project from the drop-down list.
7. Click **OK**.

### Creating Custom Project Types

If an existing project type does not meet your needs, you can define a new project type or customize an existing one.

To create a custom project type, complete the following steps:

1. Select **Project > New** and click the **Customize** button.
2. On the Customize Packages dialog, click the **New** button.
3. Enter a name for the new custom project type.
4. Select the project type to use as the starting point for your custom project. Use (**Generic**) project if you are defining a project type for a completely new language/compiler or select one of the existing project types to make modifications.
5. Click **OK** to bring up the Project Properties dialog.
6. Configure the project settings for this project type. This is similar to the process of configuring a single project, except that you cannot add files to a project type.
7. Click **OK** when done to save your changes and return to the Customize Packages dialog. Click **OK** to return to the New Project dialog. Click **Cancel** in these dialogs to discard your changes.

To customize an existing project type, complete the following steps:

1. Select **Project > New** and click the **Customize** button.
2. On the Customize Packages dialog, select the project type to customize and click the **Edit** button.
3. Make the changes needed for this project type.
4. Click **OK** when done to save your changes and return to the Customize Packages dialog. Click **OK** to return to the New Project dialog. Click **Cancel** in these dialogs to discard your changes.

Creating or customizing a project type creates a new project template stored in the `userprjtemplates.vpt` file located in your configuration directory. Other team members can use this template by copying the template file into their own configuration directories. If they have also created custom project types, they can use [DIFFzilla®](#) to compare and merge the two versions of the file.

## Setting the Active Project

Each workspace contains one project that is the active project. The active project is the one that is built when you select **Build > Build**. If the active project depends on other projects, those projects will be built first.

To make a project active, choose **Project > Set Active Project**, and pick the project to make active. Alternatively, you can use the Workspace Properties dialog box to set the active project (see [Managing Projects within a Workspace](#)).

## Defining Project Dependencies

Dependencies define a relationship between two projects, causing the dependent project to be built after the projects it depends on. This ensures that elements in a depended-on project are up-to-date prior to building the dependent project. Project dependencies can be defined when a project is created.

To specify dependencies, complete the following steps:

1. Select **Project > Workspace Properties**.
2. Select the project you want to have depend on other projects.
3. Click the **Dependencies** button. The Project Properties dialog box opens with the [Dependencies Tab](#) displayed.
4. Mark the check box next to the projects upon which the selected project should depend. These dependencies will be built before the project is built when a build or rebuild is performed.
5. Click **OK**.

## Project Configurations

Projects can have multiple configurations (such as for Debug and Release). The most common use of project configurations is for building a debug or release version of a project without having to define a new project. The Project Configuration Settings dialog box (**Build > Configurations**) is available for viewing, adding, and deleting project configurations. Changing the configuration will have no effect unless you either have different build commands, or use the **%b** (current configuration) or **%bd** (object directory) escape sequence in the build commands.

### NOTES

- **Visual C++** - If you open a Visual C++ v5.0 or later workspace, the configurations are automatically retrieved from the Visual C++ project. Some typical configurations for Visual C++ v5.0 or later are "CFG=MyApp - Win32 Debug" and "CFG=MyApp - Win32 Release." Use Visual C++ to change the configurations.
- **Mac OS X** - Opening an Xcode project imports styles that you cannot change using SlickEdit®. You will need to change the styles using Xcode. You can work with the project in SlickEdit, but you cannot change the project settings.

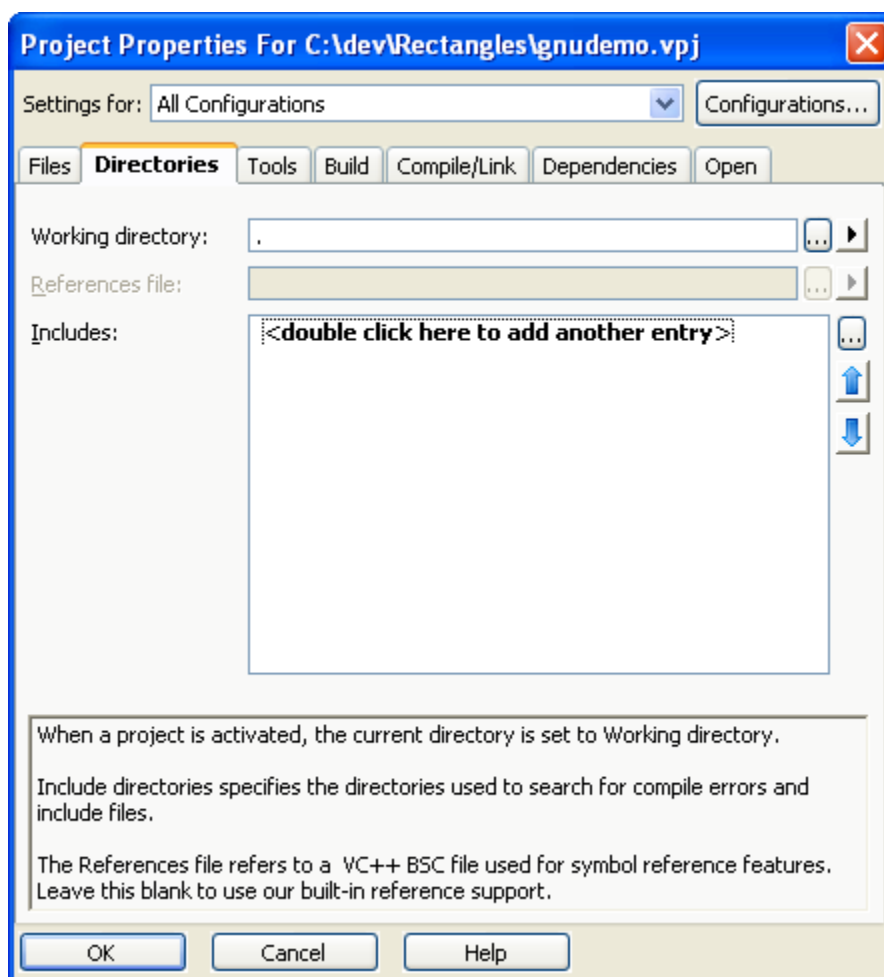
If you are using a **make** program, define a macro such as CFG, which is the configuration you want to build. Then add code to the makefile to check for this macro and perform some different statements like

choosing different compile options and a different directory for object files. The makefiles exported from Visual C++ already define a CFG macro. For a standard make program you will need to use the *name=value* syntax when passing a macro to the make program. For example:

```
make CFG=Debug
```

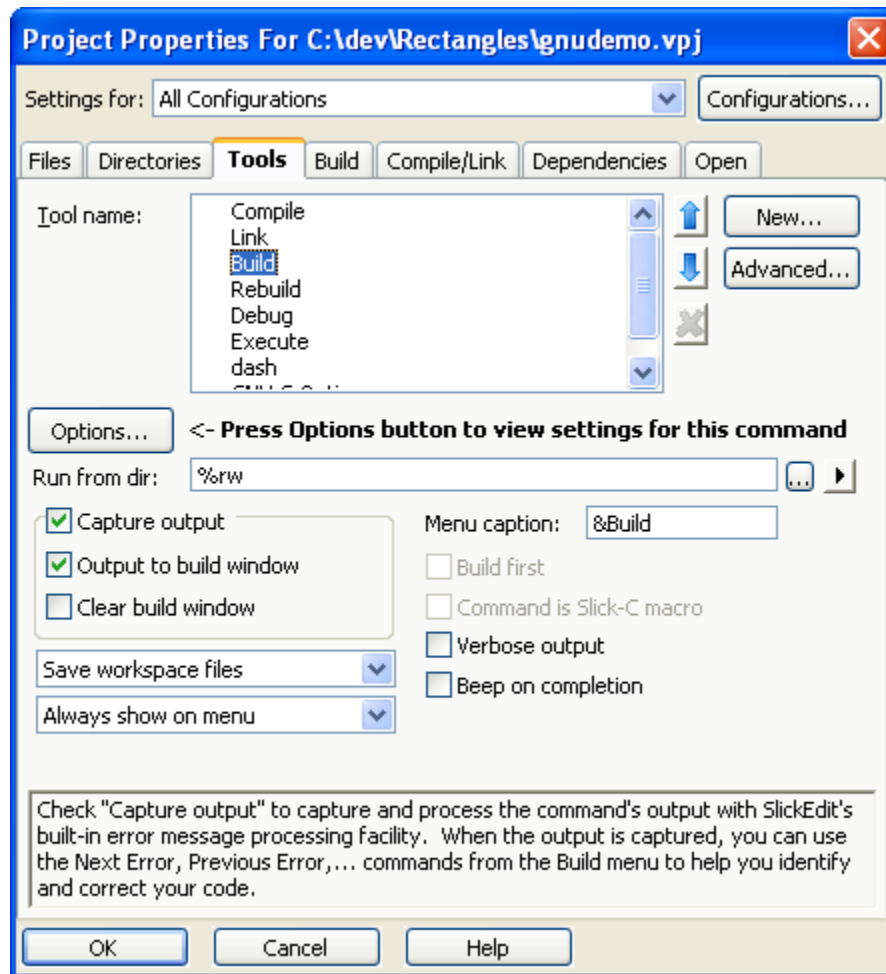
## Configuring Project Directories

The Directories tab of the Project Properties dialog box (**Project > Project Properties**) allows you to set the working directory, references file, and include file search directories for the current project. See [Directories Tab](#) for a list of the options.



## Configuring Project Tools

The Tools tab of the Project Properties dialog box (**Project > Project Properties**) is used to change project commands and their properties.



The options on the Tools tab vary, depending on the tool name that is selected in the **Tool name** text box. This text box contains a list of the tools/commands that can be used for projects. You can have different tools for different projects, and you can choose whether or not each tool should appear on the Build menu.

Use the up and down arrows to move the tools up and down in the list. This order corresponds to the order in which the tool appears on the Build menu. Click the red "X" icon to remove a user-defined tool (default tools cannot be deleted). Click the **New** button to add a tool. Click the **Advanced** button to change environment variables (see [Environment Variables](#)).

## Setting Extension Options

The **Options** button on the Project Properties Tools tab is only available for selected tools that support the extension-specific options. Click the **Options** button to display the Extension Options dialog box specific to the language extension with which you are currently working. From there you can set options for the com-

mand that gets executed for the tool specified in the **Tool name** combo box. For more information on changing extension options, see the topic for the language in the [Language-Specific Editing](#) chapter.

**TIP** (Java only) You can easily change Java tool options including the class path. Click the **Options** button here to display the Java Options dialog box, which allows you to customize options supported by Javac, Javadoc, and JAR. To change the compiler from Javac to another compiler (such as SJ or Jikes™), from the Java Options dialog, select the Compiler tab, then select the Other tab, and type the compiler name.

### Command Line Execution

The **Command line** text box on the Project Properties Tools tab is only available (and visible) for selected tools that support a command line execution. It defines the command line that is set to be executed for the selected tool in the Tool name combo box. This field is initially blank when you modify settings for “All Configurations”, and the settings differ for different configurations. Click the buttons to the right of this text box to insert files and escape sequences (such as **%f** which inserts the current buffer name) that you can use to build your command line.

### Specifying a Command Directory

For each tool listed on the Project Properties Tools tab, you can specify the directory from which to run the command in the **Run from dir** text box. By default, all of the tools are run from the working directory that is specified using the **%rw** or **%rp** escape sequences, which indicate the working directory or project directory, respectively. When running programs like **ant** or **make**, this is typically set to the directory containing the makefile.

### Other Options

The remaining options on the Project Properties Tools tab allow you to specify output, save, display, and other settings.

**NOTE** (UNIX only) Output of text mode programs that are executed using **xterm** cannot be captured. To see the output, uncheck the Output options **Capture output** and **Output to build window**, then prefix the program name in the **Command line** field with **xterm -e** or **dos -w** (this waits for a key press).

All of the options and settings on the Project Properties Tools tab are outlined in the section [Tools Tab](#).

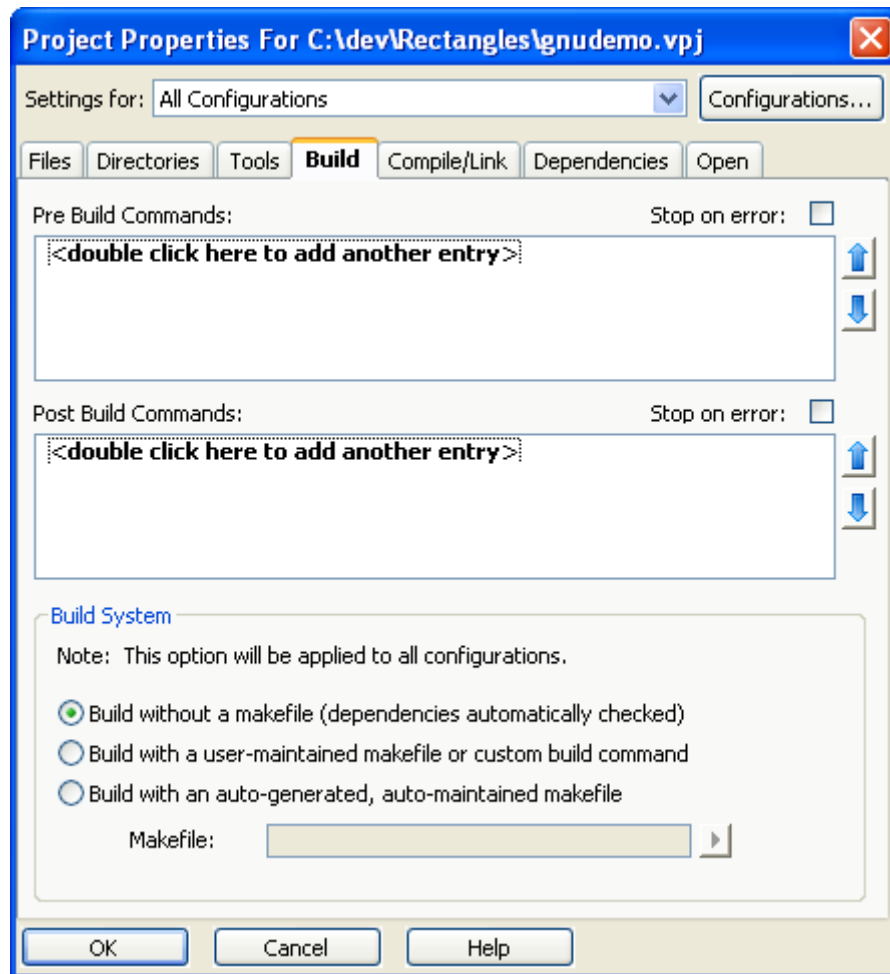
## Configuring Build Settings

The commands **project\_compile** (Shift+F10 or **Build > Compile**) and **project\_build** (Ctrl+M or **Build > Build**) start the compile and build commands respectively for the current project.

The commands **next\_error** (Ctrl+Shift+Down or **Build > Next Error**) and **prev\_error** (Ctrl+Shift+Up or **Build > Previous Error**) enable quick navigation of compiler errors. For information about building and compiling projects, see [Building and Compiling](#).

To change the build and compile commands for projects as well as other project options, use the Build tab of the Project Properties dialog (**Project > Project Properties**). The Build tab allows you to run programs and/or execute commands before or after a build. You can run different programs and commands for different projects as the information is stored per-configuration. The contents of this tab are disabled for extension-based projects.





Each line in the **Pre** and **Post Build Commands** text boxes can contain a program to execute a command. For example, the **set** command could be used to set environment variables. Double-click on the text as indicated in the text boxes to add commands. Use the **Up** and **Down** arrows to the right of the text boxes to move the commands up and down in the list. The order corresponds to the order in which the command will be run.

When the **Stop on error** option is checked and the current project depends on other projects, the **vsbuild** utility (see [Using VSBUILD to Compile](#)) will be used to build the projects and check for error codes. When the **vsbuild** program detects an error, it does not continue building other dependencies.

**NOTE** (Windows only) Under Windows 95 or later, **vsbuild** cannot detect error codes returned from a batch program.

## Build System Options

The build method options on the Build tab apply to GNU C/C++ projects only and affect all configurations. With these options, you will not need to convert the current build methods to use the GNU debugger; you can select one of these methods when you create a new GNU C/C++ Wizard project.

- **Build without a makefile (dependencies automatically checked)** - Automatically checks dependencies and does not generate a makefile. Instead, the **vsbuild** utility (see [Using VSBUILD](#)

[to Compile](#)) determines what should be compiled dynamically. This option is useful when you are not concerned with how the build gets done. Make sure the project include directories are set up correctly (**Project > Project Properties, Directories Tab**) so include files may be found (see [Configuring Project Directories](#)).

- **Build with a user-maintained makefile or custom build command** - Sets the build command to **make** and does not generate a makefile. The build command can be changed from the Tools tab of the Project Properties dialog box (see [Configuring Project Tools](#)). Select this option when you already have your own method for building the source.
- **Build with an auto-generated, auto-maintained makefile** - Automatically generates a makefile and updates when files are added to the project. This option is useful when you need a makefile and do not want to use the built-in **vsbuild** utility (see [Using VSBUILD to Compile](#)). Specify the path to the makefile in the **Makefile** field. Make sure the project include directories are set up correctly (**Project > Project Properties, Directories Tab**) so include files may be found (see [Configuring Project Directories](#)).

To start a build from outside the application, execute the following command where **make** is the name of the **make** program, **Makefile** is the name of the makefile, and **ConfigName** is the name of the configuration:

**`make -f Makefile CFG=ConfigName`**

## Defining Extension-Specific Projects

If you do not want to create a project and you are building something that only contains one source file, you can define an extension-specific project. Extension-specific projects are based on the extension (file type) of the current file. However, the working directory is ignored for these projects. All extension-specific projects are stored in the file `project.vpe` (UNIX: `uproject.vpe`).

To define an extension-specific project, complete the following steps:

1. From the main menu, select **Tools > Options > File Extension Setup**. The Extension Options dialog box is displayed.
2. Select the [Advanced Tab](#).
3. Select the extension that you want from the **Extension** drop-down list, then click **Extension Specific Project**.

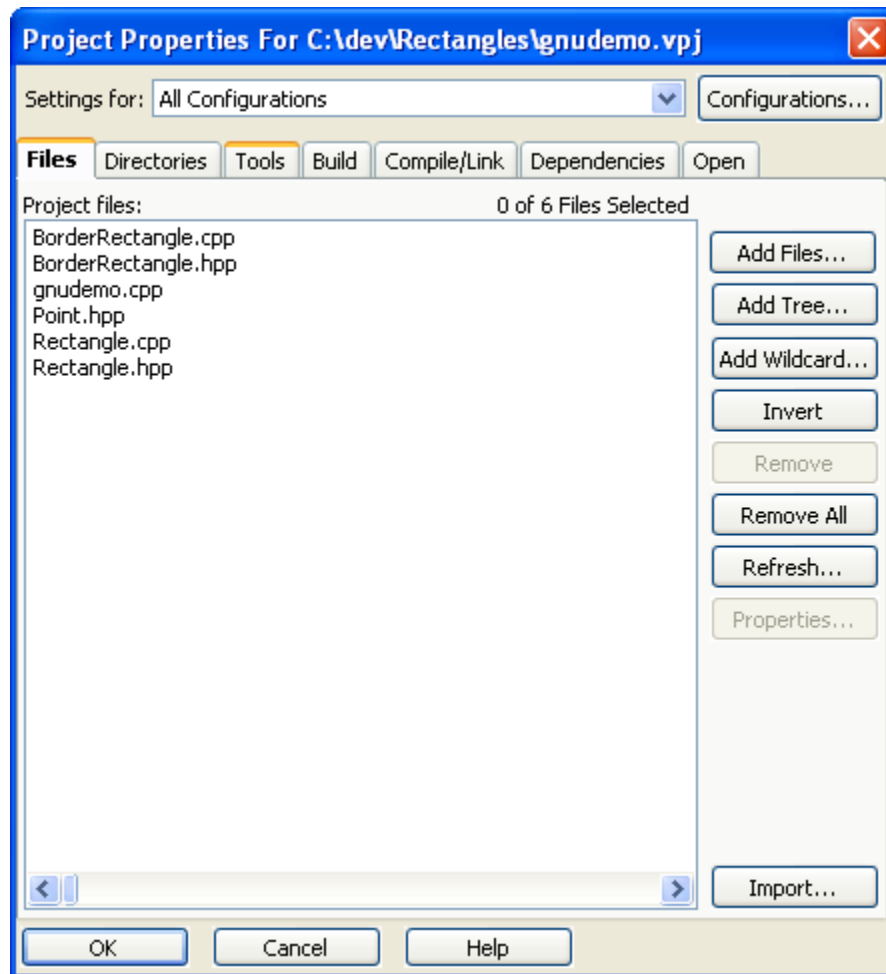
## Managing Source Files

### Adding and Removing Files

Files can be added to new and existing projects.

The Files tab of the Project Properties dialog (**Project > Project Properties**), shown below, displays a list of the files in the current project and also allows you to remove files from projects. See [Files Tab](#) for a list of the options.

SlickEdit® provides a number of ways to add files to a project. How you add files to the project depends on whether all team members are using SlickEdit and updating the project files. If so, you can use **Add Files**, **Add Tree**, or **Import** to perform the initial setup of existing files. All subsequent files will be added as they are created through SlickEdit. If some members of the team are not using SlickEdit, then you should use **Add Wildcard** to add files. The wildcard is evaluated each time SlickEdit is run and any new files are automatically added to your project.



To add files to a project, complete the following steps:

1. From the main menu, choose **Project > Project Properties**. The Project Properties dialog is displayed.
2. Click to display the **Files** tab.
3. Perform the file operation:
  - **Add Files** - Use for adding individual files.
  - **Add Tree** - Use for adding files in a directory or directory tree. Click **Add Tree**, then select the directory and the file filter to use.
  - **Add Wildcard** - Use for adding files from a directory or directory tree matching the specified wildcard. This expression is evaluated each time SlickEdit is loaded, adding new files from the specified directory.
  - **Remove** - To remove the selected files.
  - **Remove All** - To remove all files from this project.
  - **Refresh** - To update the list of files, re-evaluating any wildcards that have been specified.

- **Import** - Loads files and directories specified in an import file. See [Importing Files](#) for more details.

## Creating New Files

There are two different approaches to adding a new file to a project:

- Create a buffer with the correct name and then do a **File > Save As** to store the file in the correct location and, optionally, add it to the project.
- Use **File > New** or **Project > Add New Item from Template** to create the file and put it in the correct place in the project.

Many developers prefer the first approach because it is faster and allows you to keep your hands on the keyboard. A new buffer named `foo.cpp` can be created from the command line by typing **e foo.cpp**.

To create a new source file using the second approach, complete the following steps:

1. From the main menu, select **Project > New** and select the **File tab**.
2. Select the file type from the list on the left.
3. To add the new file to an existing project, mark the **Add to project** check box and select the project from the drop-down list.
4. Type a file name, including the extension, in the **Filename** field.
5. Verify that the **Location** is correct.
6. Change the **Encoding** as needed.
7. Click **OK**.

This same approach can be used with the SlickEdit® Code Templates by selecting **Project > New Item From Template**. See [Code Templates](#) for more information.

## Importing Files

The Import File List dialog (**Project > Project Properties**, select the **Files** tab, then click the **Import** button) allows you to load files and directories specified in an import file into your project.

## Loading Project Files for Editing

The Load Files dialog box is used to open one or more files from the current project. This dialog can be accessed from the main menu by choosing **Project > Load Files** or by using the **project\_load** command. Use Ctrl+Click to select more than one file. The following options are available:

- **Edit** - Switches to the selected buffer in the list box.
- **Save** - Saves the selected buffer in the list box.
- **Close** - Deletes the selected buffer in the list box. If the preference option **One File per window** is on (see [Buffers and Editor Windows](#)), all windows displaying the buffer are deleted as well. You are prompted whether you wish to save a modified buffer before the buffer is deleted.
- **Order** - This button toggles the display order between sorted and next or previous link order.

# Creating, Opening, and Saving Files

---

This section describes how to create, open, and save files, as well as how to set various file options. For additional information on working with files, see [Using the File Manager](#). For information about buffers and windows, see [Buffers and Editor Windows](#). For information about working with file history and backing up files, see [File History and Backups](#).

## Creating Files

### Quick Create/Open

The quickest way to create a new file or open an existing file is to use the **e file** command, where *file* is the name of the file you wish to create or open. This will create or open the file in a new buffer.

### Using the New File Dialog

To create a new file, from the main menu choose **File > New**. When the New dialog is displayed, select the [File Tab](#).

Select the language mode for the new file from the list. You can double-click on a language to create an untitled buffer set to that mode. Otherwise, enter the **Filename**, the **Location** to save it to, and select the **Encoding** type. If you wish to add the file to the current project, select **Add to Project**. See also [New Dialog](#).

### Creating a File from a Selection

The Write Selection dialog box can be used to write the selected text to a file you choose. This dialog box can be accessed from the main menu by choosing **File > Write Selection** or by using the **gui\_write\_selection** command.

## Opening Files

SlickEdit® provides support for file names up to 200 characters long. If you need to edit a file whose name contains space characters, place double quotation marks around the name.

When you specify the file to edit, the format is automatically recognized and necessary adjustments are made, whether the file type is DOS, UNIX or Mac\* (Classic Mac line endings are a single carriage return, ASCII 13). An EOF character will not be appended to a UNIX or Macintosh format text file when saved, regardless of the save options. When the text in one text file is copied to a buffer that has a different type of text file format, the line separation characters are reformatted to conform to the format of the buffer destination.

If you specify a workspace file name with extension (**.vpw**), the editor will Auto Restore the workspace. Extension (**.vpj**) Auto Restores the project.

The following table contains “invoke and edit” examples:

Example	Description
<b>vs project1.vpw</b>	Auto Restore from workspace file.

Example	Description
<b>vs project1.dsw</b>	Auto Restore from a VC++ workspace file.
<b>vs autoexec.bat config.sys</b>	Edit two files.
<b>vs this is.c</b>	Edit a file with a space character.
<b>vs test.c -#1000</b>	Edit test.c and go to line 1000.
<b>vs orders -#bottom-of-buffer -#/invoice/-</b>	Edit orders file, go to bottom of buffer, and search backward for invoice.
<b>vs +70 test.exe</b>	Edit binary file test.exe in record width 70.
<b>vs *.c</b>	Edit all C source files in the current directory.
<b>vs -r list \</b>	Start the file manager (Windows) and list entire drive contents.
<b>vs -x rich.sta autoexec.bat</b>	Specify different state file and edit the file <code>autoexec.bat</code> .

When SlickEdit is invoked, it concatenates the value of the environment variable `VSLICK` before the command line that you specify. When the file that you want to edit is not found, an empty buffer is created with that name.

As previously mentioned, the quickest way to open an existing file is to use the **e file** command (see [Quick Create/Open](#) above). Alternatively, you can use the Standard Open dialogs for Windows and UNIX.

## Opening Files on Windows

To open a file for editing, use the **gui\_open** command, or from the main menu choose **File > Open**. The standard Open dialog appears. This dialog is different from the dialog that appears on UNIX platforms.

**TIP** If you are running Windows, you can choose to use the dialog that is normally displayed for UNIX instead--it is generally faster to use and contains more options. To do this, from the main menu choose **Tools > Options > General**, select the [More Tab](#), then select the option **Windows 3.1 style open dialog**. See [Opening Files on UNIX or Mac OS X](#) for more information on using this dialog.

For a complete list of the fields and options that are available on the standard Open dialog, see [Open Dialog - Windows](#).

## Opening Files on UNIX or Mac OS X

To open a file for editing, use the **gui\_open** command, or from the main menu choose **File > Open**. The Windows 3.1 style Open dialog appears. This dialog is generally faster than the dialog that appears on Windows platforms. It contains some of the same options that are available on the Windows dialog, with a few differences and additional options. See [Open Dialog - Linux, UNIX, and Mac](#) for a complete description.

## Opening a URL

To open a file located at a particular Web address, from the main menu choose **File > Open URL**. The Open URL dialog is displayed. In the URL text box, enter a URL to open. You can use forward or backward slashes, and the prefix "http://" is not required.

For descriptions of all of the options on this dialog, see [Open URL Dialog](#).

## Finding a File to Open

Use the Find File dialog box to search for one or more files to open. To access this dialog, from the main menu choose **File > Open**, then click the **Find File** button. On the Find File dialog, specify the **File** or **File pattern** to search for. Subdirectories are always scanned. Click **Search** to start the search. See [Find File Dialog](#) for a complete list of options.

## Inserting Files into Buffers

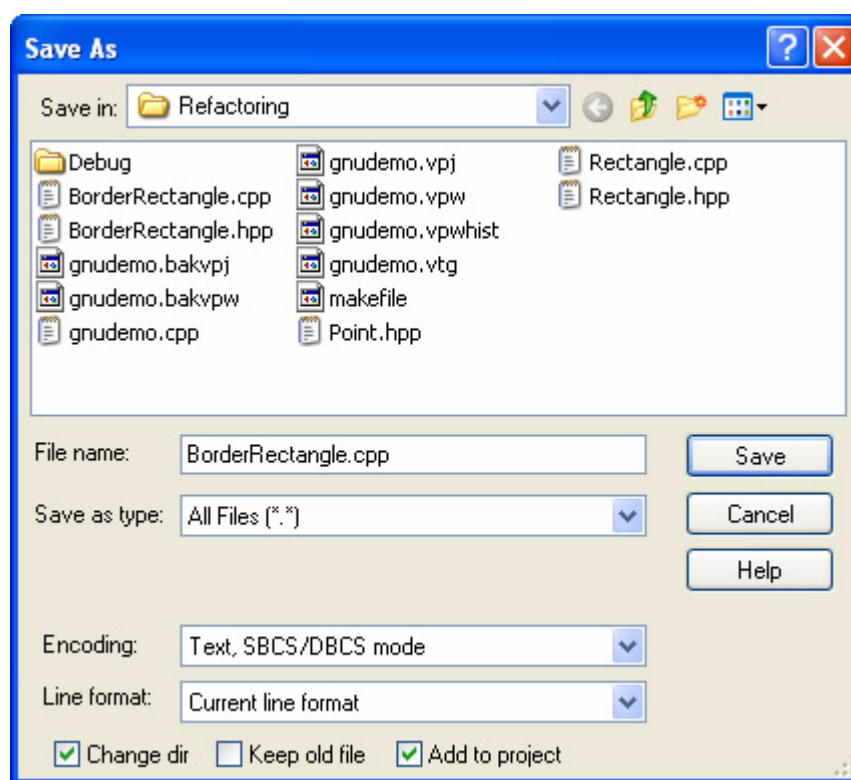
The Insert File dialog box can be used to insert a file at the cursor location in the current buffer. The Insert File dialog box can be accessed from the main menu by choosing **File > Insert a File** or by using the `gui_insert_file` command. The Open dialog box is displayed, prompting you for a file.

## Saving Files

To save the current file or buffer, from the main menu choose **File > Save** (Ctrl+S or `save` command), or **File > Save As** (`gui_save_as` command). To save all open buffers, select **File > Save All** (`save_all` command). You can also right-click on a file tab to access these same operations.

## Using Save As

Use the Save As dialog (**File > Save As** or `gui_save_as` command) to save the current buffer under a file name you choose. Check **Keep old file** if you do not want the buffer name to be changed.



The Save As dialog is similar to the Open dialogs. The options that are different are described in the section [Save As Dialog](#). See also [Open Dialog - Windows](#).

## Failed Saves

If a save operation fails, the Save Failed dialog box is displayed. See [Save Failed Dialog](#) for a list of options.

## Closing Files

Closing can be done by using one of the following methods. You will be prompted to save any changes.

- To close the current file, choose **File > Close** (**quit** command), press F3, or right-click on the file tab and select **Close**. You can also use Shift+LeftClick or the middle mouse button on a file tab to close it.

**TIP** To reduce keystrokes on the SlickEdit command line, use the **q** command, a shortcut for the **quit** command, in the syntax **q args**, where **args** is a list of files to close. Wildcards are permitted. For example, **q c:\temp\\*.\*** closes all files in the `temp` directory that you had open.

- To close all open files, choose **File > Close All** (**close\_all** command) or right-click on the file tab and select **Close All**.



- To close all open files except for the one currently in focus, right-click on the file tab and select **Close Others**.

#### NOTES

- You can click the middle mouse button on a file tab to close a file, even if the focus is not on the file to be closed. However, the middle mouse button must be configured to send a Middle Button event.
- File tabs are displayed by default in SlickEdit. Display can be toggled on/off by selecting **View > Toolbars > File Tabs** from the main menu.

## Setting File Options

This section describes the options that are available for loading, saving, and auto-saving files, as well as how to set file filters and options for virtual memory.

### General File Options

#### Auto CAPS Mode

When this feature is enabled, and a file is opened that does not contain any lowercase characters, caps mode is turned on (not the same as Caps Lock). When caps mode is on, all text is inserted in uppercase. This feature is intended to emulate ISPF.

To enable Auto CAPS, from the main menu, choose **Tools > Options > File Extension Setup**, then select the [General Tab](#). From the **Extension** drop-down list, choose the file extension you wish to affect. Then select the option **Auto CAPS**.

#### Auto-Change Directory

You can specify that the current directory is changed in the editor when the directory is changed in the Change Directory dialog (**File > Change Directory**) and the Open and Save As dialogs (**File > Open** and **File > Save As**). From the main menu, choose **Tools > Options > General**. Then select the [General Tab](#). Select the option **Change directory**.

#### Automatically Close Visited Files

SlickEdit® can automatically close a file which was briefly visited. A file is considered as *visited* if it is opened as a result of a symbol navigation or search operation, and not modified, and subsequently navigated away from. Some examples of operations that open files are **pop\_bookmark** (Ctrl+Comma) and **next\_error/prev\_error** (Ctrl+Shift+Down/Ctrl+Shift+Up). To enable this option, from the main menu, choose **Tools > Options > General**, then select the [General Tab](#). Select the option **Automatically close visited files**. When this option is selected, a visited file will be automatically closed when it is navigated away from. If not selected, the auto-close feature will be disabled. If left in the mixed state, you will be prompted whether or not you want to close files.

#### First File Opened is Active

By default, when opening multiple files, the last file in the list of files to open will be set as the active file ready for editing. You can reverse this behavior, so that the first file in the list is active instead. This is particularly useful when using the command line to open or edit files. To reverse the default behavior, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Edit “A B C” start on file A**.

## Load and Save Options

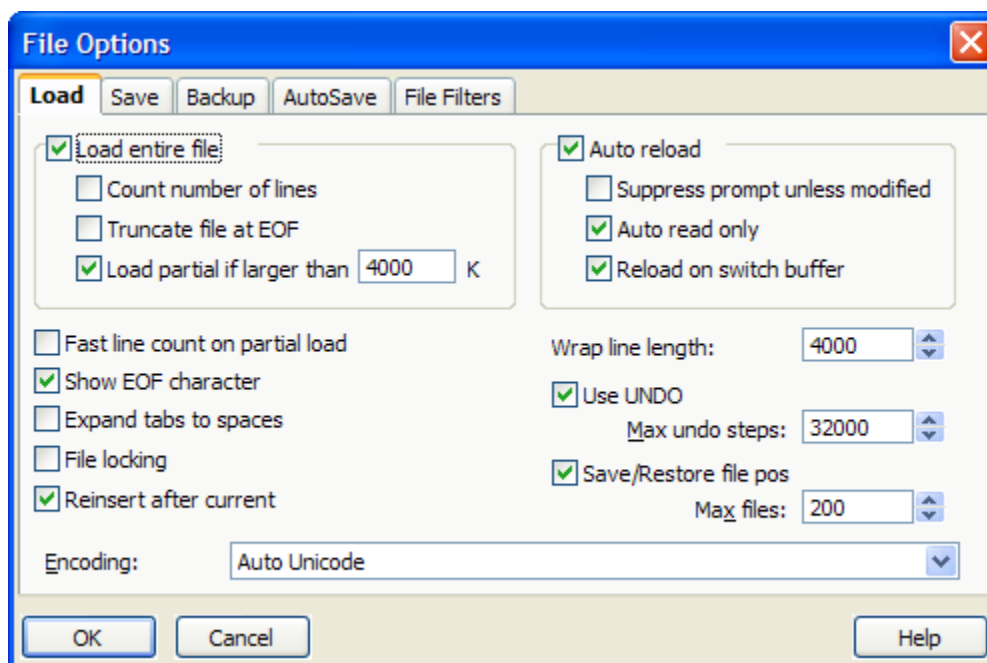
Options for loading and saving are available on the File Options dialog box. This can be accessed from the main menu by choosing **Tools > Options > File Options**. For descriptions of all of the options on the File Options dialog, see [File Options Dialog](#).

The default load and save options are safe for editing binary files since all reformatting is turned off. To ensure that no load or save reformatting takes place when opening and saving binary files, set the encoding to “Binary” when opening the file.

**CAUTION** Due to the risk of damaging or losing binary files, it is NOT SAFE to edit them with the following option settings on the Load and Save tabs of the [File Options Dialog](#): Show EOF Character unchecked, Expand Tabs to Spaces checked, Append EOF Character checked, Remove EOF Character checked, and Strip Trailing Spaces checked.

### Load Options

Load options affect the Open dialog box and any other command that uses the **e** or **edit** commands to open a file. You can override many of the load options by checking the appropriate box in the advanced section of the Open dialog box. To access load options, from the main menu, select **Tools > Options > File Options**, then select the [Load Tab](#).



**TIPS**

- **Working with Large Files** - The editor reads the entire file for files that are color-coded. However, the **Load partial** option enhances system performance because the original file is used as a read-only spill file. To determine when the entire file is read, go to the bottom of the file. If the line number is displayed at the bottom of the file, the entire file has been read, however this does not mean the entire file is in memory. When the entire file is not loaded, it will be locked, and other applications won't be able to write to it.

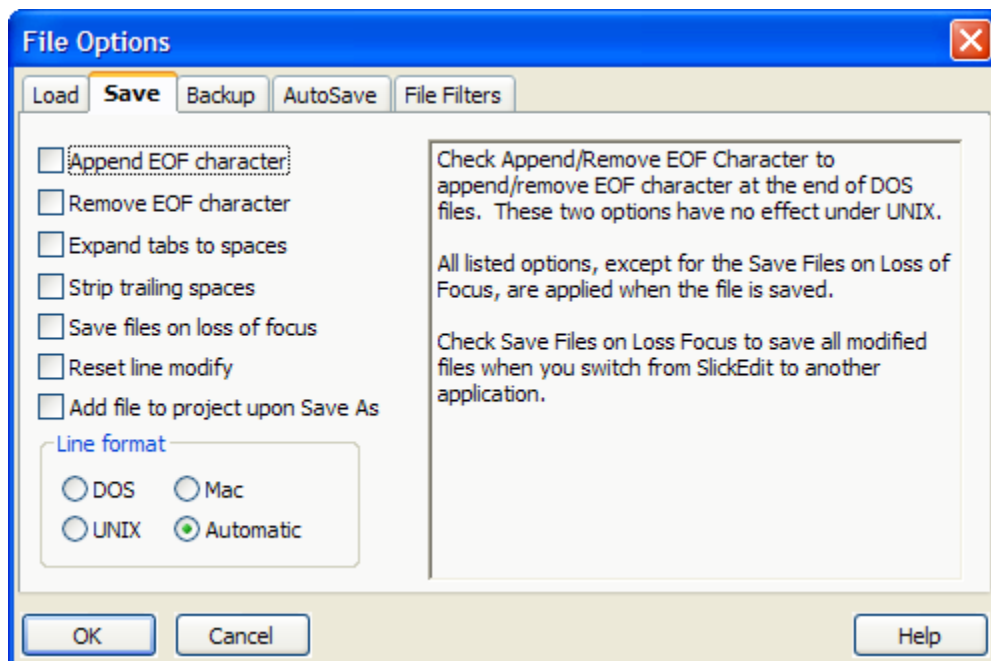
By default, 2 MB is used for the buffer cache. When the cache is full, modified blocks are written to the spill file. Blocks that are not modified and can be reread from the original file are discarded. Scrolling through a 2 GB file requires no more than 2 MB of memory (default). A search and replace operation that hits every block requires that almost all blocks be spilled. Saving a 2 GB file requires 4 GB of disk space. The file data must first be spilled before saving the file. Turn backups off before saving a large file since this requires additional disk space equal to the size of the file. The block size is 16 K.

- **Load Options for Different Disk Drives** - For different load options for different disk drives, define an environment variable called VSLICKLOAD and specify each drive followed by the appropriate switch. For example, set VSLICKLOAD= a: +L b: +L specifies preloading files from floppy drives A and B. When loading files using the command line (**e** and **edit** commands) the following options can be explicitly specified: '+' turns a switch on and '-' turns a switch off. See [Environment Variables](#) for more information

For more information about the options on the Load tab, see [Load Tab](#).

## Save Options

The Save tab of the File Options dialog (**Tools > Options > File Options, [Save Tab](#)**), shown below, provides options that can be selected to occur by default when save operations are invoked.



## CREATING, OPENING, AND SAVING FILES

Each of the options in this dialog can also be set using command line switches. The **save\_as** command switches and options are described below. To enable or disable a switch from the command line, type a plus (“+”) or minus (“-”) sign before the switch character.

For example, to expand tabs to spaces using **save\_as**, complete the following steps:

1. Press **Esc** to bring up the SlickEdit® command line.
2. Type **save\_as** and press **Enter**.
3. You will be prompted for the file to save, with the active file already entered for you. Modify the name of the file, then type “+E ” (including the space) before the complete file name.
4. Press **Enter**.

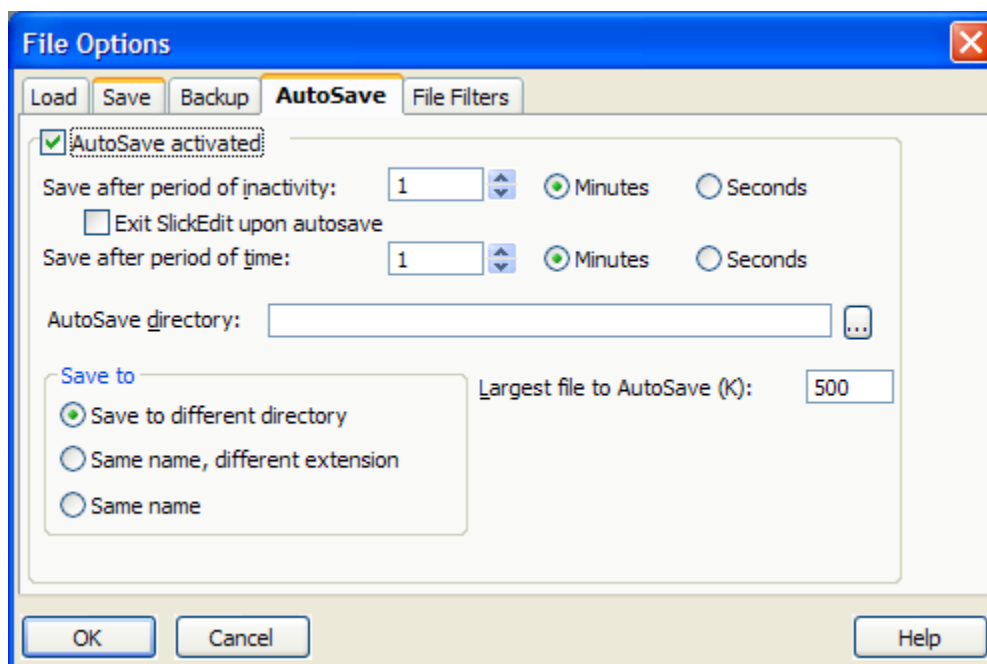
Alternately, if you know the path and file name, simply type the following in the command line then press Enter:

**save-as +E path/to/filename.cpp**

You can use the other switches in the same manner. You can also use them in combinations. For more information about each option on the Save tab, see [Save Tab](#).

## AutoSave Options

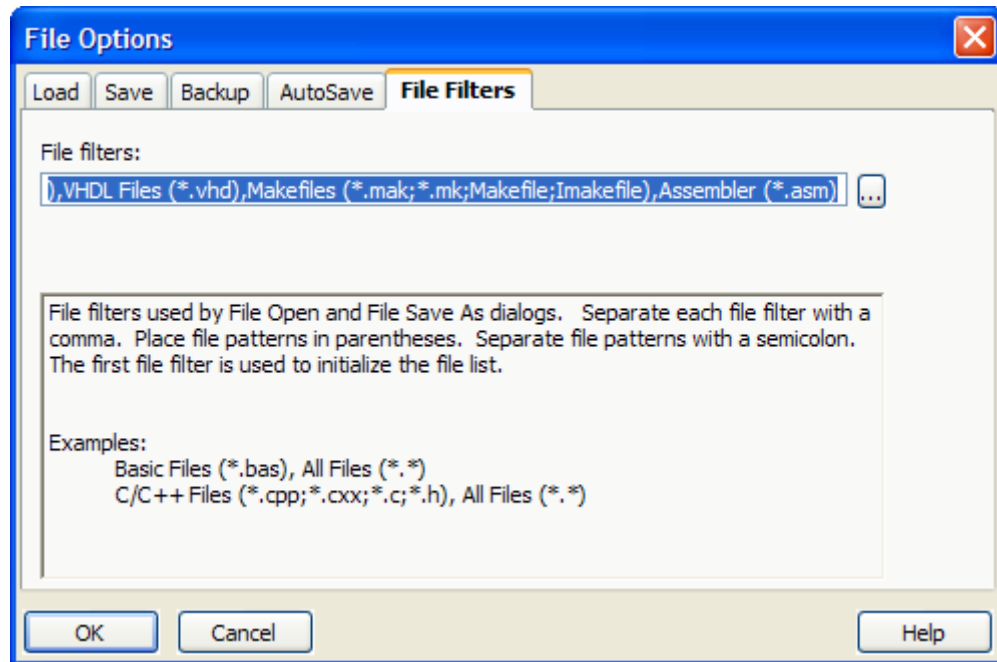
The AutoSave tab, shown below, provides parameters that you can set so you can automatically save files. To access the AutoSave tab, from the main menu, select **Tools > Options > File Options**, then select the [AutoSave Tab](#).



For more information about each of these options, see [AutoSave Tab](#).

## File Filter Options

The File Filters tab, shown below, provides the ability to change the file specifications that appear in the **List files of type** text box at the bottom left of the Open dialog box. The wildcard patterns specified in the File Filters dialog box is used to initialize the Open dialog box. The default is to list all files. You might want a smaller list initially that displays the source files you typically edit. To edit file filters, from the main menu, select **Tools > Options > File Options**, then select the [File Filters Tab](#).



## Virtual Memory Options

To manage virtual memory, from the main menu choose **Tools > Options > General**, then select the **Virtual Memory** tab. See [Virtual Memory Tab](#) for information about these settings.



## Using the File Manager

---

The SlickEdit® File Manager offers a rich set of file listing, selecting, and operating capabilities. You can select, deselect, list, and unlist files by extension, attribute, and search pattern. Once you have listed and selected the files to be operated on, you can copy, move, delete, change attribute, or back up the selected files.

File Manager operations and options can be accessed from the main menu by choosing **File > File Manager**.

## Creating a New File List

Before managing files in the File Manager, you must first create a list of files to be managed. To create a new file list, use the **fileman** command, or from the main menu, select **File > File Manager > New File List**. The List Files dialog is displayed, with the following options:

- **File name** - Specify in this text box one or more files separated by spaces. Each file specification may contain wildcard characters. For example, "**\*.c \*.h**" will list all C and H files.
- **Include system files** - When checked, files with the system attribute set are included in the list. This option is ignored on UNIX. This option is always turned on under Windows if the **Show all files** option is set on the Open dialog box (see [Open Dialog - Windows](#)).
- **Include hidden files** - When checked, files with the hidden attribute set are included in the list. This option is ignored on UNIX. This option is always turned on under Windows if the **Show all files** option is set on the Open dialog box (see [Open Dialog - Windows](#)).
- **Subdirectories** - When checked, all subdirectories below each file specified in the **File Name** text box are searched.

The listing of files will appear in a new buffer window.

## Appending Files to the List

You can also append files to your current file list. From the main menu, choose **File > File Manager > Append File List** or use the **fileman** command. The Append File List dialog contains the same fields and options as the List Files dialog described above.

**TIP** To modify and close a file list without being prompted to save each time, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Throw away file lists**.

## Selecting Files in the File Manager

To select files in the file list, use the File Manager selection functions. To access these functions, from the main menu, choose **File > File Manager > Select**, then choose one of the following sub-menu items:

- **All** - Selects all files in the list.
- **Deselect All** - Deselects all files that were previously selected.
- **InvertSelect** - Invert the selection.
- **Attribute** - Selects files that contain a specific attribute search string that you specify.
- **Extension** - Selects all files with a given extension. Enter the extension without the Dot character.

- **Highlight** - Selects all files within a marked area.
- **Deselect Highlight** - Deselects highlighted files.

## Operating on Selected Files

The operations described below are available on the File Manager menu for working with selected files. To access these operations, from the main menu, choose **File > File Manager**.

- **Sort** - (**fsort** command) Sorts the files listed in File Manager on a primary and optionally a secondary key. Check the **Secondary Sort** check box if you want to sort on a secondary key.
- **Backup** - (**fileman\_backup** command) Copies the selected files (lines with ">" character as first character of line) to a directory you choose. The directory structure on the source file is preserved. Typically only a drive letter is specified for the destination directory. However, you may specify a path if you wish to further nest the directory structure.
- **Copy** - (**fileman\_copy** command) Copies the selected files (lines with ">" character as first character of line) to a directory you choose.
- **Move** - (**fileman\_move** command) Moves the selected files (lines with ">" character as first character of line) to a directory you choose. The destination drive does not have to be the same as the source drive.
- **Delete** - (**fileman\_delete** command) Prompts whether to delete the selected files.
- **Edit** - (**fileman\_edit** command) Edits the selected files.
- **Select** - Displays a menu of selection commands. See [Selecting Files in the File Manager](#) above.
- **Files** - Displays a menu of the following file commands:
  - **Unlist All** - (**unlist\_all** command) Removes all files from the list.
  - **Unlist Selected** - (**unlist\_select** command) Removes selected files from the list.
  - **Unlist Extension** - (**gui\_unlist\_ext** command) Removes files with a specific extension from the list.
  - **Unlist Attribute** - (**unlist\_attr** command) Removes files with a specific attribute from the list.
  - **Unlist Search** - (**unlist\_search** command) Deletes lines which contain a search pattern you specify.
  - **Read List** - (**read\_list** command) This dialog box is similar to the Append List dialog box which adds files to the current list. The difference is that the Read List dialog box prompts you for a file name which contains the names of files to append to the current list. The file may be a list of file names or may be a file in the same format as a file manager list. You may use the Write List dialog box to write a list containing the selected file names.
  - **Write List** - (**write\_list** command) Writes a list of the currently selected file names to a file you choose. The Open dialog box is displayed to prompt you for a file.
- **Attribute** - (**fileman\_attr** command) Sets the Read Only, Hidden, System, and Archive attributes of the selected files.
- **Repeat Command** - (**for\_select** command) Runs internal or external commands on selected files.
- **Global Replace** - (**fileman\_replace** command, or Alt+Shift+G) The Global Replace dialog box performs a search and replace on the selected files in the File Manager. The following options are available:



- **Search for** - Enter the string you want to search for here.
- **Replace with** - Search string is replaced with this string.
- **Match case** - When checked, a case sensitive search is performed.
- **Match whole word** - When checked, a word search is performed. Before a search is considered successful, the characters to the left and right of the occurrence of the search string found are checked to be non-word characters. To change the word characters for a specific extension, select **Tools > Options > File Extension Setup**, then select the [Advanced Tab](#).
- **Regular expression** - When checked, a regular expression search is performed. See [Find and Replace with Regular Expressions](#) for more information.
- **Place cursor at end** - When checked, the cursor is placed at the end of the occurrence found.
- **Global Find** - (`fileman_find` command) Performs a search on selected files.



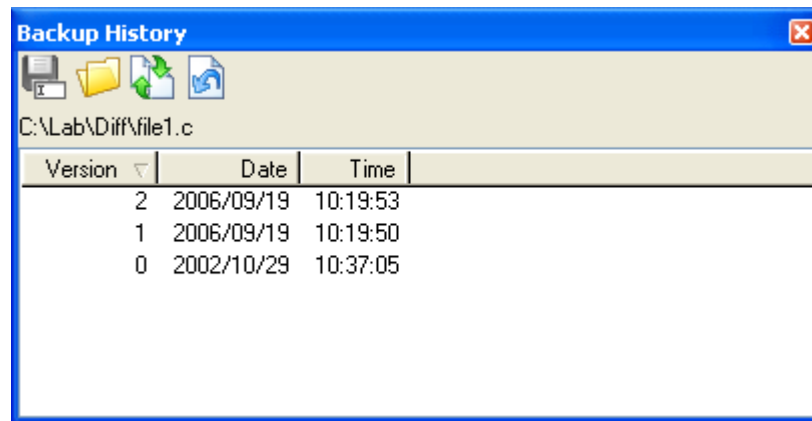
## File History and Backups

---

### Using Backup History

The Backup History feature allows you to view the differences between a previous version of a file and the version with which you are currently working. To enable Backup History, from the main menu choose **Tools > Options > File Options**, then choose the **Backup** tab. Select the option **Make backup files**.

To access Backup History for the current file, from the main menu choose **File > Backup history for**. The Backup History tool window is displayed (**View > Toolbars > Backup History**), as shown below.



SlickEdit® creates and stores deltas to conserve disk space. The **Run Diff on Selected Backups** button enables the comparison by using DIFFzilla® dialog, and the **Open Selected Backup** button allows you to open the version that is highlighted in the list of files. The **Save Selected Backup As** button can be used to save a file from the history with a different file name.

For more information on diffing files and using the DIFFzilla dialog, see [DIFFzilla®](#). See [Setting Backup Options](#) below for more information on Backup History options.

## Backing Up Network Files

### Windows Backups

Sometimes the default backup style does not work correctly when a network file is saved. By default, backups are placed in a subdirectory named "backup" under the SlickEdit® installation directory. For this backup style to work, make sure the application has access rights to this directory and that this directory exists. To specify the backup directory, select **Tools > Options > File Options**, select [Backup Tab](#), and enter a value in **Backup directory**.

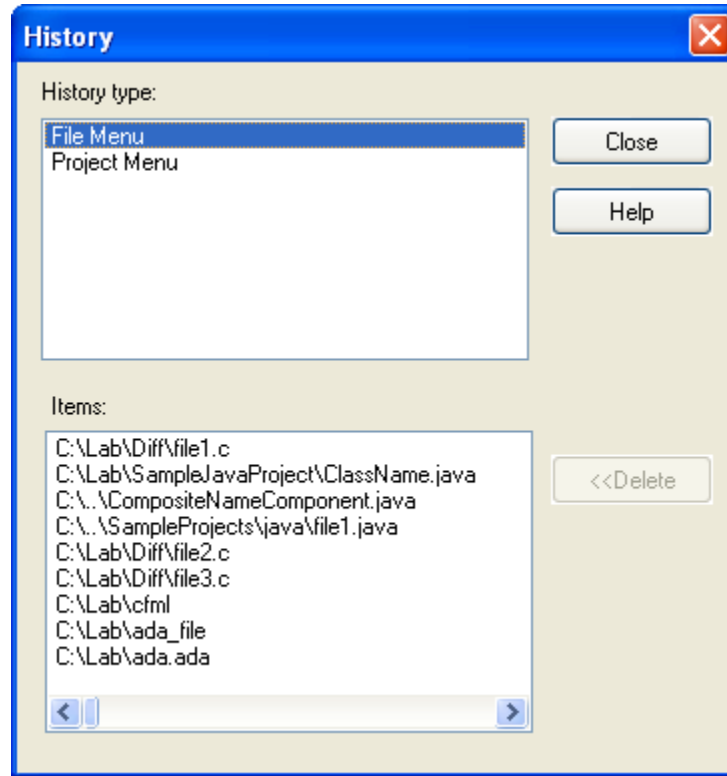
### UNIX and Mac OS X Backups

The default backup style will work properly under UNIX because backup files are placed in the \$HOME/.vslick directory, which should always have read-write permissions.

A backup file is created the first time a file is saved in a given edit session. Therefore, the backup file may not represent the last save.

## Viewing the History of Opened Items

To see the names of files and projects that have been recently opened, from the main menu, choose **Tools > Options > History**. The History dialog appears, as shown below.

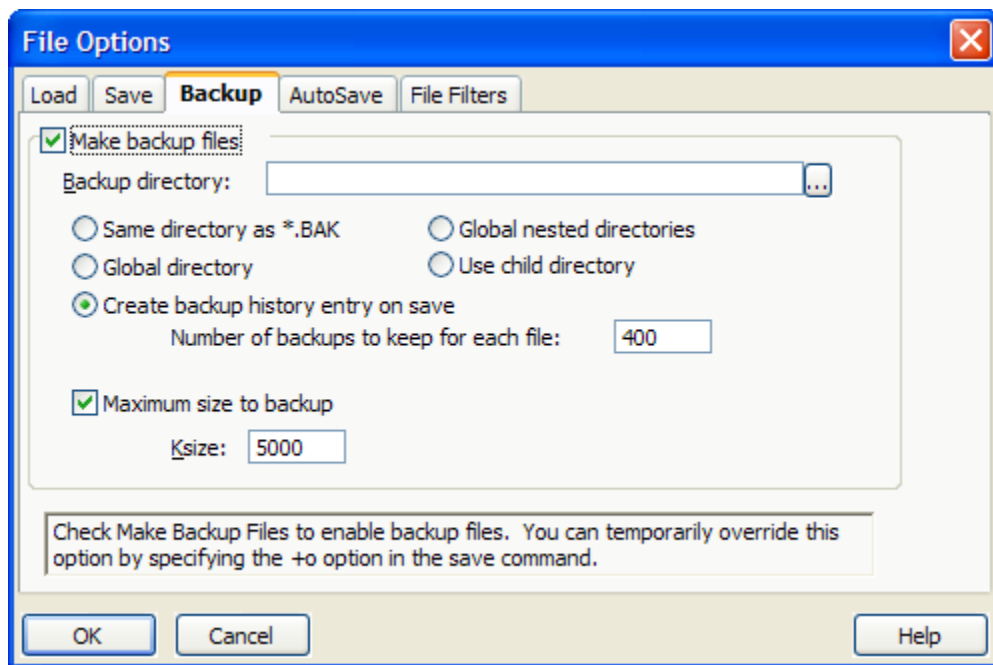


The **History type** allows you to select the history items displayed in the **Items** list box. Click **Delete** to delete selected items in the Items list box.

## Setting Backup Options

To enable backups of files and backup history, and to set backup preferences, from the main menu, select **Tools > Options > File Options**, then select the **Backup** tab. See [Backup Tab](#) for more information about the options.

A backup of a file is made the first time you save your file in a given edit session. Your backup file may not represent your last save. For more up-to-date backups, use the AutoSave feature (see [AutoSave Options](#)).





# Code Templates

---

Code templates are pre-defined units of code that you can use to automate the creation of common code elements, like a standard class implementation or design patterns. You can create templates for a whole file or multiple files. Templates can contain substitution parameters that are replaced when the template is instantiated—when a new element is created from that template. Some parameters are replaced with calculated or pre-defined values, like date or author. If a value is not known, you will be prompted for a value when the template is instantiated.

Code templates are composed of one or more template source files and a metadata file providing additional information, like the name of the template, a description of the template, prompts for substitution parameters, and default values for substitution parameters. The following is an example of a single file source template. The items surrounded by dollar signs, “\$”, are the substitution parameters.

```
/*
 * $copyright$
 */

package $package$;

/**
 * @author $author$
 * @version $version$
 */
public class $safeitemname$ {

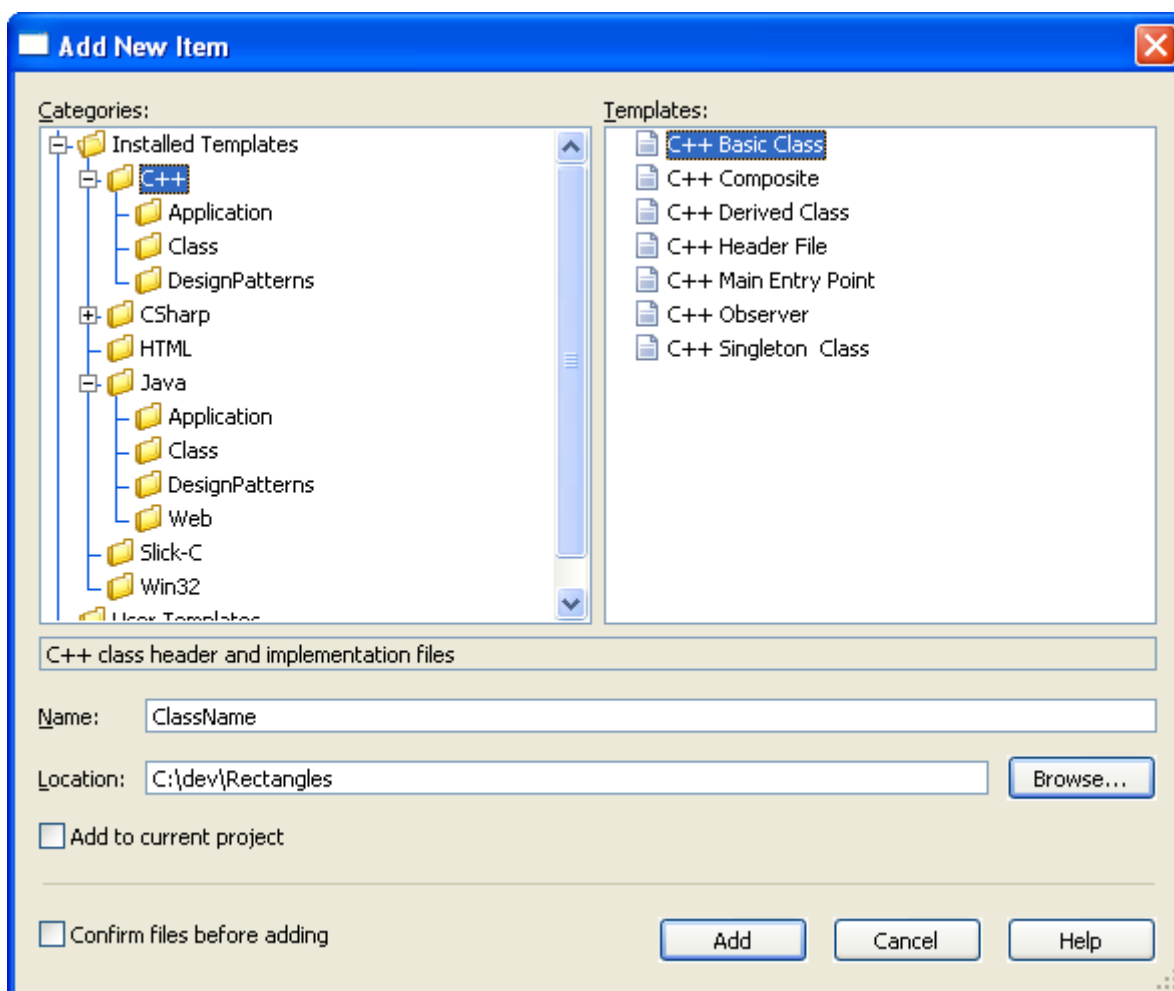
    /**
     * Default constructor.
     */
    public $safeitemname$(){
    }

}
```

Templates can be organized into Categories to make them easier to manage. The templates shipped with SlickEdit® are organized into categories by language and then by purpose. Use the Template Manager dialog to add, edit, and delete user templates. The Template Manager dialog is accessed by choosing **File > Template Manager**.

## Instantiating a Template

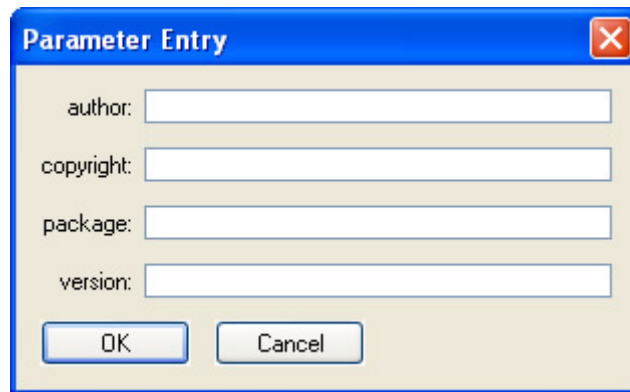
You can add an item to your current project by choosing **Project > Add New Item from Template**. If you want to create a new item from a template without adding it to your current project, then choose **File > New Item from Template**. The Add New Item dialog box is shown below.



We call the process of creating new files from a template “instantiating a template”. When a template is instantiated, you are prompted for the name of the new item. This name is often used heavily in the template. For a class template, the name will likely be the class name or a part of the class name. In the sample template, `$safetitemname$` is a form of this name that strips out any spaces, making it safe to use as part of an identifier. This value can even be used as part of the file name when the template is instantiated.

If any of the values in the template are not known at instantiation time, the Parameter Entry dialog box, shown below, will prompt you for values.





## Creating Templates

Creating templates is very much like writing code. To create a new code template, complete the following steps:

1. Create the template source files.
2. Insert substitution parameters into the template files.
3. Use the Template Manager to create a new template.
4. Add the template files to the newly-defined template.

### Create the Template Source Files

This is the same process as writing any source file. Use SlickEdit® to write a file from scratch or to modify an existing file. Make sure your file is syntactically correct to minimize compile errors after it is instantiated.

In many languages, the `$name$` syntax used by SlickEdit Code Templates is legal for identifiers, so you will be able to compile and run your template source files prior to instantiating them. In other languages, you will have to use temporary identifier names while writing the templates, and then put in the substitution parameters once you are sure the source is correct.

You can store these source files in any directory and copy them to the templates directory during Step 4.

### Insert Substitution Parameters into the Template Files

Use substitution parameters for any part of the source code that can differ from instantiation to instantiation. This includes class names, author names (if several people are sharing the same template files), or creation dates.

In our sample, we put in a substitution for copyright statement. See [Substitution Parameters](#) for more details.

### Use the Template Manager to Create a New Template

Select **File > Template Manager** to bring up the Template Manager. Select the “User Template” folder in the tree, and right-click in either the Categories pane or the Templates pane to create a new template.

There are different operations based on whether you want to create a new category or not. You will be prompted for the name of the new template. Fill in a name and click OK. Now you can use the Template Manager to enter a description, add files, or set values for Custom Parameters.

## Add the Template Files to the Newly-Defined Template

Select the Files tab on the Code Template Manager dialog and click the plus sign icon (“+”) to add the files you created in Step 1 to this template. You will have the option to link to the source in its current location or copy it to the template directory. You will also be prompted for a target file name. If you want the name of the instantiated template to appear in the file name, you should use a substitution variable in the name, like “My\$safeitemname\$Class.java”.

## Substitution Parameters

Substitution parameters provide the real power in Code Templates. Without them, you would simply be making copies of static files. You can use substitution parameters to replace any text in the template's source code. You can also use substitution parameters in file names, which is useful in Java where a class must be defined in a file by the same name.

Substitution parameters are written as identifiers surrounded by a delimiter. The default delimiter is “\$”. Use a double delimiter to represent the delimiter character in a template source file, “\$\$”. You can specify a different character to use as the delimiter. Select **File > Template Manager** and click on the Custom Parameters tab to change the value for the “Delimiter” field.

We provide a set of predefined substitution parameters for items related to item name, project name, directories, date, and time. We can determine the value for these items rather than having to prompt for them. See the list at the end of this section for all the predefined substitution parameters.

You can define substitution parameters that are common to all templates. For example, you might want to define an “author” parameter where the parameter value is your name. You could then create code templates that fill in a header comment with the author's (your) name. You would only have to define the substitution parameter once. To define these parameters, open the Template Manager and select the Custom Parameters tab.

If no value is provided for a substitution parameter, you will be prompted for one when the template is instantiated. This is useful for things like class name or other values that are different each time the template is instantiated.

## Predefined Substitution Parameters

The following substitution parameter names and values are pre-defined for use in an item template. The default delimiter “\$” is used:

Parameter Name	Description
\$itemname\$	Name of item entered, as on the Add New Item dialog.
\$fileinputname\$	Name of item entered, as on the Add New Item dialog, without file extension.
\$safeitemname\$	Name of item entered, as on the Add New Item dialog, with all unsafe characters replaced with safe characters. For example, if the item name was: My Custom Class, then the <b>\$safeitemname\$</b> would evaluate to My_Custom_Class for a C++ source code file.
\$upcasesafeitemname\$	Same as <b>\$safeitemname\$</b> with all characters upper-cased.

Parameter Name	Description
<code>\$lowcasesafeitemname\$</code>	Same as <b>\$safeitemname\$</b> with all characters lower-cased.
<b>\$tempdir\$</b>	Location of operating system temp directory. No trailing file separator.
<code>\$rootnamespace\$</code>	Root namespace or package for the current project. This is typically used for C# and Java projects to find the namespace containing <b>Main()</b> (or <b>main()</b> in the case of Java).
<code>\$ampmtime\$</code>	Time of day in the form hh:mm[am pm]. Example: 11:34pm
<code>\$localtime\$</code>	Time of day in locale-specific format.
<code>\$time\$</code>	Time of day in the form hh:mm:ss.
<code>\$localdate\$</code>	Current date in locale-specific format.
<code>\$date\$</code>	Current date in the form mm/dd/yyyy.
<code>\$projectname\$</code>	Current project name (no path, no extension).
<code>\$safeprojectname\$</code>	Current project name (no path, no extension), with all unsafe characters replaced with safe characters. For example, if the project name was: <code>My Project.vpj</code> , then <b>\$safeprojectname\$</b> would evaluate to <code>My_Project</code> for a C++ source code file.
<code>\$workspacename\$</code>	Current workspace name (no path, no extension).
<code>\$safeworkspacename\$</code>	Current workspace name (no path, no extension), with all unsafe characters replaced with safe characters. For example, if the workspace name was: <code>My Workspace.vpw</code> , then <b>\$safeworkspacename\$</b> would evaluate to <code>My_Workspace</code> for a C++ source code file.
<code>\$projectworkingdir\$</code>	Current project working directory. No trailing file separator.
<code>\$projectbulddir\$</code>	Current project build (output) directory. No trailing file separator.
<code>\$projectconfigname\$</code>	Current project configuration name.
<code>\$workspaceconfigname\$</code>	Current workspace configuration name. This will be the same as <b>\$projectconfigname\$</b> except for MS Visual Studio workspace which will have a separate workspace/solution configuration name.
<code>\$projectdir\$</code>	Location of current project file. No trailing file separator.
<code>\$workspacedir\$</code>	Location of current workspace file. No trailing file separator.
<code>\$username\$</code>	Operating system login name.

## Organizing Templates

Templates are organized into category hierarchies as shown on the Add New Item dialog. These category hierarchies map exactly to the directory structure under the locations for installed and user templates.

To create a new template item category:

1. Create a new folder under the user templates directory. For example, if you wanted to create a Dialogs category for Java project items, you would create the following directory:  
`<ConfigDir>/templates/ItemTemplates/Java/Dialogs/`
2. Place all templates for the category under this directory.
3. Create a new project or open an existing one.
4. From the main menu choose **Project > Add New Item**.
5. Verify that your new category appears in the Categories list on the Add New Item dialog box.

**CAUTION** We do not recommend creating new categories or re-organizing categories under installed templates since the next patch or upgrade would overwrite any customizations you have made. If you want to customize an installed template, then we suggest you copy it to the user templates directory and perform your customization on the copy.

## Template Manager Dialog

Use this dialog to add, edit, and delete templates. Use the Categories list to select a category. Selecting a category populates the Templates list with templates for that category. You can show this dialog by choosing **File > Template Manager**.

### Creating a New Category

To create a new category under the selected category, right-click in the Categories tree and select New Category. You will be prompted for a category name. After clicking OK, you can add templates in the new category.

### Creating a New Template

To create a new template, select the category in which to create the template, then right-click in the Templates list and select New Template. You will be prompted for a template name which is used to create the new template file. After clicking OK, you can edit the new template the lower half of the dialog.

### Editing an Existing Template

To edit an existing template, select a template from the Templates list, and edit its properties in the lower half of the dialog.

### Deleting a Template

To delete a template, select the template you want to delete from the Templates list, right-click and select "Delete Template" from the context menu.

## Categories

Lists a hierarchy of item categories for installed and user template items.

**NOTE** Installed templates can be viewed but not modified.

## Templates

Lists the templates for the currently selected category. When you select a template, you are able to edit its properties in the lower half of the dialog.

## Template file

File name of the currently selected template.

## Details tab

### Name

Specifies the name for the template item. The name is used in the Templates list of the Add New Item dialog.

### Description

Specifies the description for the template item. The description is displayed on the Add New Item dialog when the template is selected.

### Default name

Specifies the default item name when using the Add New Item dialog box.

### Sort order

Specifies an order number that is used to sort the template item in relation to other template items in a list. Used to sort template items in a category on the Add New Item dialog box. Lower sort orders are placed ahead of higher sort order values in a sorted list.

## Files tab

Use the Files tab to add, edit, order, and delete files in a template. Files are created from a template when using the Add New Item dialog, as when adding an item template to a project.

Add, Edit, Order, and Delete operations are accessible from the buttons on the right side or from the context menu inside the list of files.

## Custom Parameters tab

Use the Custom Parameters tab to add, edit, and delete substitution parameters in a template. Substitution parameters are used to replace parameter names in the content of files created from a template with a pre-defined value. Substitution parameters can also be used to form target file names (Files tab).

Add, Edit, and Delete operations are accessible from the buttons on the right side or from the context menu inside the list of parameters.

## Template Options Dialog

Use this dialog to edit options that are common to all templates. You can launch this dialog from the Template Manager dialog box.

### Global Substitution Parameters

Lists the substitution parameters that are common to all templates. A common substitution parameter, for example, could be “author” where the parameter value is your name. You could then create code templates that automatically fill in a header comment with the author's (your) name.

Add, Edit, and Delete operations are accessible from the buttons on the right side or from the context menu inside the list of parameters.

## Add File Dialog

Used to add a file to a template. This dialog is launched when performing an Add operation from the Files tab of the “Template Manager Dialog”.

### Source file name

When a file is created from a template, as when adding an item template from the Add New Item dialog, it is created from the source file with this file name.

### Copy source file to template directory

Check this option to place a copy of the file in the current template's directory and change the source file name to point to the new file in the template. The file is not copied until you click OK.

### Target file name

When a file is created from a template, as when adding an item template from the Add New Item dialog, the file name of the file that is created on disk is formed from the target file name in the location you specify. Use the menu button to the right of this field to insert common pre-defined substitution parameters. For example, `$fileinputname$` is the item name provided on the Add New Item dialog when adding an item template to your project.

### Replace parameters in target file content

Check this option if you want substitution parameters embedded in the content of the target file to be replaced when the file is created from the template, as when adding an item template to your project from the Add New Item dialog.

### Preview

Previews how the file would be copied when creating the file from a template as if the source file name and target file name were fully resolved.

## Add Parameter Dialog

Used to add a custom substitution parameter to a template. This dialog is launched when performing an Add operation from the Custom Parameters tab of the “Template Manager Dialog”. When files are created from a template, as when adding an item template to your project from the Add New Item dialog box, you

can configure your template to replace all substitution parameters with values. For a list of pre-defined substitution parameters, see [Predefined Substitution Parameters](#).

## Name

This is the name of the substitution parameter WITHOUT delimiters. For example, if the delimiter is “\$” (the default), then a substitution parameter that inserts a copyright string would have a name of “copyright” and NOT “\$copyright\$”. Do not use quotes in the name. Valid characters for a parameter name are: A-Za-z0-9\_

## Value

This is the value that the substitution parameter evaluates to when a string or file is created from the template and has its substitution parameters replaced with values.

## Prompt for value

Check this option if you always want to be prompted for the value of a substitution parameter. When set, the Value field becomes a default value field and is used to pre-populate the value when you are prompted.

## Prompt string

Specifies the prompt string to display when being prompted for a substitution parameter value.

# Add New Item Dialog

Use this dialog to add an item to your current project.

Use the Categories list to select a category. Selecting a category populates the Templates list with template items for that category. You can then select an item from the Templates list, enter a unique Name for the item, and enter a Location. Click Add to instantiate the template with the name and location you provided.

You can show this dialog from the main menu by choosing **Project > Add New Item** or **File > New Item from Template**. You can manage your templates from the Template Manager dialog box by choosing **File > Template Manager**.

## Categories

Lists a hierarchy of item categories for installed and user template items.

## Templates

Lists the template items for the currently selected category. When you select a template item, a brief description for that item is displayed just above the Name field.

## Name

Enter the name of the file you want to create.

**NOTE** For single file templates (templates that create a single file) this is the name of the file. Multi-file templates use the name of the item entered to form names of files in the template. For more information about creating multi-file templates, see [Creating a Multi-file Template](#).

## Location

Enter the location to which to save the item.

## Add

After you have selected a template item, provided a name and a location, click Add to instantiate the template item.

# Locating Templates

## Installed Templates

Templates that are installed with the product are located at:

```
<SlickEditInstallDir>/sysconfig/templates/ItemTemplates/
```

For example, the following directory under Windows contains item templates for the C++ language:

```
c:\SlickEdit\sysconfig\templates\ItemTemplates\C++\
```

## User Templates

User templates are templates that the user creates and are located at:

```
<ConfigDir>/templates/ItemTemplates/
```

**TIP** You can locate your Configuration Directory from the main menu by choosing **Help > About SlickEdit**.

# Manually Creating a Template

SlickEdit® Code Templates are represented as files stored in specific directories. A template is composed of the source file or files for the template and a metadata template file that provides additional information. Since these are just files, you can write them using SlickEdit.

To manually create an item template:

1. Choose a category folder under the user templates directory. Your user templates directory is at:  

```
<ConfigDir>/templates/ItemTemplates/
```

**TIP** You can locate your Configuration Directory from the main menu by choosing **Help > About SlickEdit**.

All files will be created relative to the folder you choose. For more information about how templates are organized, see [Organizing Templates](#).

2. Create or edit a code file (e.g. \*.cpp, \*.java, etc.). Replace occurrences of substitutable text with substitution parameter names. For example, you might want to make the name of a C++ or Java class into a substitution parameter, in which case you could use the **\$safeitemname\$** substitution parameter. For more information on substitution parameters, see [Substitution Parameters](#).
3. Create an XML file and give it an extension of `.setemplate`.
4. Insert template metadata into the `.setemplate` file. See the example below. For more information on template metadata elements, see [Code Template Metadata File Reference](#).



5. Create a new project or open an existing one.
6. Choose from the main menu **Project > Add New Item**.
7. Verify that your new template item appears in the Templates list on the Add New Item dialog box.

## Example

The following example illustrates the metadata for an item template for a custom Java class, along with the content of the Java source code file.

From the Add New Item dialog box, if the user entered `Foo.java` for the item name, then `$fileinputname$` would be replaced with “Foo” in the file name of the file created, and `$safeitemname$` would be replaced with “Foo” in the Java source code file.

`MyClass.setemplate:`

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File TargetFilename="$fileinputname$.java">MyClass.java</File>
    </Files>
  </TemplateContent>
</SETemplate>
```

`MyClass.java:`

```
class $safeitemname$ {
};
```

## Creating a Multi-file Template

A multi-file template is a template item that creates more than one file.

Multi-file templates require the use of substitution parameters to ensure that file name and extension parts are used when creating each file of the template item. For example, a C++ class typically consists of:

- A `.h` file that contains the class definition.
- A `.cpp` file that contains the class implementation.

Since you can only enter one name into the Name field on the Add New Item dialog box, you need a way to specify the target file name for each file created by the multi-file template. In the C++ class example below, the `.h` and `.cpp` files are created with the name you provide, while their extensions are preserved.

To create a multi-file item template from the Template Manager dialog, choose **File > Template Manager**.

To manually create a multi-file item template:

1. Create the item template the same way a single file template would be created. For more information on manually creating a template item, see [Manually Creating a Template](#).
2. Add TargetFilename attributes to each of the File elements in your template metadata file (.set-template). Set the value of each TargetFilename attribute to **\$fileinputname\$.<extension>**, where <extension> is the file extension of the target file name being created. When the files are created, their names will be based on the name you entered in the Name field of the Add New Item dialog box. See the example below.

## Example

The following example demonstrates a multi-file item template .set-template file. The item creates C++ class header (.h) and implementation (.cpp) files.

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My C++ Class</Name>
    <Description>My complete C++ class header and implementation</Description>
    <DefaultName>MyClass.cpp</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File TargetFilename="$fileinputname$.cpp">MyClass.cpp</File>
      <File TargetFilename="$fileinputname$.h">MyClass.h</File>
    </Files>
  </TemplateContent>
</SETemplate>
```

## Code Template Metadata File Reference

Template metadata describes the template item, its files, and how to create the template. Template metadata files have a .set-template extension.

The SETemplate element is the root element of a template file.

Summary of metadata elements:

Element	Child Elements	Attributes	Reference
"DefaultName"	-	-	See <a href="#">DefaultName</a> .
"Description"	-	-	See <a href="#">Description</a> .
"File"	-	ReplaceParameters, TargetFilename	See <a href="#">File</a> .
"Files"	File	-	See <a href="#">Files</a> .

Element	Child Elements	Attributes	Reference
"Name"	-	-	See <a href="#">Name</a> .
"Parameter"	-	Name, Value	See <a href="#">Parameter</a> .
"Parameters"	Parameter	-	See <a href="#">Parameters</a> .
"SETemplate"	TemplateContent, TemplateDetails	Type, Version	See <a href="#">SETemplate</a> .
"SortOrder"	-	-	See <a href="#">SortOrder</a> .
"TemplateContent"	Files, Parameters	Delimiter	See <a href="#">TemplateContent</a> .
"TemplateDetails"	DefaultName, Description, Name, SortOrder	-	See <a href="#">TemplateDetails</a> .

## Elements

### DefaultName

DefaultName is an optional child element of TemplateDetails.

Specifies the default item name when using the Add New Item dialog box. This element becomes more important in multi-file templates where you need to specify a DefaultName element in order to create file names from parts of the input item name. See the example below.

### Attributes

None.

### Child Elements

None.

### Parent Elements

TemplateDetails.

### Value

Text value is required.

The text value specifies the default name of the template item. Used to populate the name field with an initial value on the Add New Item dialog box.

### Example

The following example illustrates the metadata for an item template for a C++ class that creates a header file (.h) and implementation file (.cpp).

```

<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My C++ Class</Name>
    <Description>My complete C++ class header and implementation</Description>
    <DefaultName>MyClass.cpp</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File TargetFilename="$fileinputname$.cpp">MyClass.cpp</File>
      <File TargetFilename="$fileinputname$.h">MyClass.h</File>
    </Files>
  </TemplateContent>

</SETemplate>

```

## Description

Description is a required child element of TemplateDetails.

Specifies the description for the template item. See the example below.

## Attributes

None.

## Child Elements

None.

## Parent Elements

TemplateDetails.

## Value

Text value is required.

The text value specifies the description of the template item. The description is shown on the Add New Item dialog box.

## Example

The following example illustrates the metadata for an item template for a custom Java class.

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>

</SETemplate>
```

## File

File is an optional child element of Files.

Specifies a file for the template item. See the example below.

## Attributes

Attribute	Description
ReplaceParameters	Optional. Specifies whether parameter substitution takes place on the file contents when the file is created from the template. Note that parameter substitution always takes place on the TargetFilename attribute value (example: TargetFilename="\$fileinputname\$.cpp"). Possible values are "1" (true) or "0" (false). Defaults to "1" (true).
TargetFilename	Optional. Specifies the actual name of the item that is created from the template. This attribute is especially useful when creating a multi-file template where file names of files created from the template are assembled by parameter substitution.

## Child Elements

None.

## Parent Elements

TemplateContent.

### Value

Text value is required.

Value is the path of a file in the template item.

### Example

The following example illustrates the metadata for an item template for a C++ class that creates a header file (.h) and implementation file (.cpp).

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My C++ Class</Name>
    <Description>My complete C++ class header and implementation</Description>
    <DefaultName>MyClass.cpp</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File TargetFilename="$fileinputname$.cpp">MyClass.cpp</File>
      <File TargetFilename="$fileinputname$.h">MyClass.h</File>
    </Files>
  </TemplateContent>

</SETemplate>
```

### Files

Files is a required child element of TemplateContent.

Specifies files for the template item. See the example below.

### Attributes

None.

### Child Elements

File.

### Parent Elements

TemplateContent.

### Value

N/A

### Example

The following example illustrates the metadata for an item template for a C++ class that creates a header file (.h) and implementation file (.cpp).

```

<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My C++ Class</Name>
    <Description>My complete C++ class header and implementation</Description>
    <DefaultName>MyClass.cpp</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File TargetFilename="$fileinputname$.cpp">MyClass.cpp</File>
      <File TargetFilename="$fileinputname$.h">MyClass.h</File>
    </Files>
  </TemplateContent>

</SETemplate>

```

**Name**

Name is a required child element of TemplateDetails.

Specifies the name for the template item. See the example below.

**Attributes**

None.

**Child Elements**

None.

**Parent Elements**

TemplateDetails.

**Value**

Text value is required.

The text value specifies the name of the template item. The name is shown in the Templates list on the Add New Item dialog box.

**Example**

The following example illustrates the metadata for an item template for a custom Java class.

```

<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>

</SETemplate>

```

## Parameter

Parameter is an optional child element of Parameters.

Specifies a custom substitution parameter for the template item. For a list of pre-defined substitution parameters, see [Predefined Substitution Parameters](#).

See the example below.

## Attributes

Attribute	Description
Name	Parameter name. This is the name of the substitution parameter WITHOUT delimiters. For example, if the delimiter is "\$" (the default), then a substitution parameter that inserts a copyright string would be defined as "copyright" and NOT as "\$copyright\$".
Value	Parameter value. This is the value that the substitution parameter evaluates to when a string or File is created from the template.

## Child Elements

None.

## Parent Elements

Parameters.

## Value

N/A



**Example**

The following example illustrates the metadata for an item template for a custom Java class.

When `MyClass.java` is used to create the file from the template, all occurrences of **\$copyright\$** in the created file will be replaced with "(c)2005-2006".

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Parameters>
      <Parameter Name="copyright" Value="(c)2005-2006" />
    </Parameters>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>
</SETemplate>
```

**Parameters**

Parameters is a required child element of TemplateContent.

Specifies custom substitution parameters for the template item. For a list of pre-defined substitution parameters, see [Predefined Substitution Parameters](#).

See the example below.

**Attributes**

None.

**Child Elements**

Parameter.

**Parent Elements**

TemplateContent.

**Value**

N/A

**Example**

The following example illustrates the metadata for an item template for a custom Java class.

## CODE TEMPLATES

When `MyClass.java` is used to create the file from the template, all occurrences of **\$copyright\$** in the created file will be replaced with "(c)2005-2006".

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Parameters>
      <Parameter Name="copyright" Value="(c)2005-2006" />
    </Parameters>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>
</SETemplate>
```

### SETemplate

Root element.

Contains all metadata about template item.

#### Attributes

Attribute	Description
Version	Template version number. The current version is "1.0".
Type	Template type. Valid types are: "Item".

#### Child Elements

TemplateDetails, TemplateContent.

#### Parent Elements

None.

#### Value

N/A

#### Example

The following example illustrates the metadata for an item template for a custom Java class.

```

<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>

</SETemplate>

```

## SortOrder

SortOrder is an optional child element of TemplateDetails.

Specifies an order number that is used to sort the template item in relation to other template items in a list. Used to sort template items in a category on the Add New Item dialog box.

If no SortOrder is specified for a template item, then the SortOrder value defaults to "0".

## Attributes

None.

## Child Elements

None.

## Parent Elements

TemplateDetails.

## Value

Text value is required.

An integer that is greater than or equal to "0". When sorting in relation to other template items, low SortOrder values are placed ahead of higher values in a sorted list.

## Example

The following example illustrates the metadata for an item template for a custom Java class.

```

<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
    <SortOrder>100</SortOrder>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>

</SETemplate>

```

## TemplateContent

TemplateContent is a required child element of SETemplate.

Specifies the contents of a template item.

### Attributes

Attribute	Description
Delimiter	Optional. Delimiter used when replacing substitution parameters in content. Defaults to "\$".

### Child Elements

Files, Parameters.

### Parent Elements

SETemplate.

### Value

N/A

### Example

The following example illustrates the metadata for an item template for a custom Java class.

```

<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>

</SETemplate>

```

## TemplateDetails

TemplateDetails is a required child element of SETemplate.

Describes the template item. Details are used to display the template item on the Add New Item dialog box.

### Attributes

None.

### Child Elements

DefaultName, Description, Name, SortOrder.

### Parent Elements

SETemplate.

### Value

N/A

### Example

The following example illustrates the metadata for an item template for a custom Java class.

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/
setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>
</SETemplate>
```

# Context Tagging®

This chapter contains the following topics:

- [Context Tagging® Overview](#)
- [Building and Managing Tag Files](#)





## Context Tagging® Overview

---

Context Tagging is a feature set that performs expression type, scope, and inheritance analysis as well as symbol look-up within the current context to help you navigate and write code. Context Tagging uses an engine that parses your code and builds a database of symbol definitions and declarations—commonly referred to as *tags*. Context Tagging features work with *your* source code, not just standard APIs (application program interfaces), and the features are dynamic, in the sense that symbols are updated immediately or in the background as you edit your source code.

The Context Tagging feature set includes:

- [Tag-Driven Navigation](#)
- [List Members](#)
- [Parameter Information](#)
- [Auto List Compatible Parameters](#)
- [Completions](#)
- [Symbol Browsing](#)
- [Statement Level Tagging](#)

Before you begin working with these features, some configuration is required. See [Building Tag Files](#).

## Tag-Driven Navigation

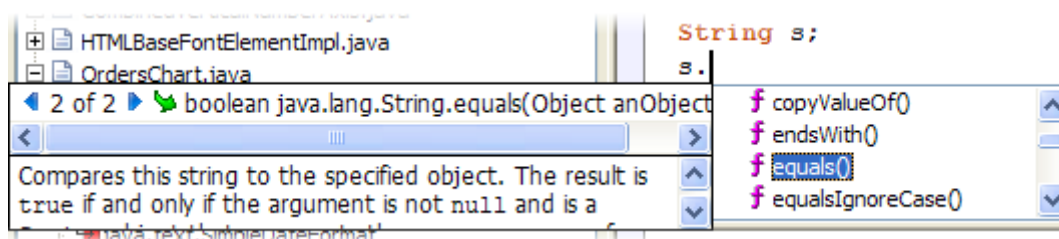
The Context Tagging® database enables you to navigate your code, jumping from a symbol to its definition or its references. For more information, see [Symbol Navigation](#).

## List Members

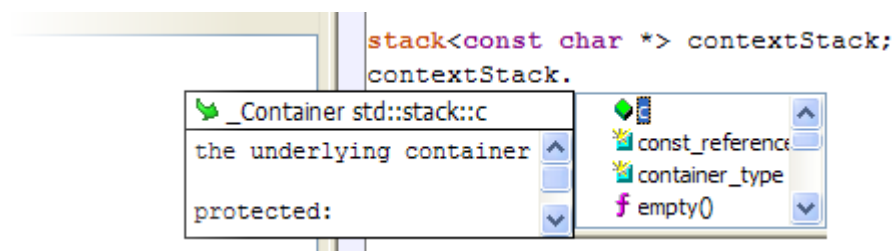
When typing a member access operator (Dot, Comma, “->”, and “.” for C++; Dot for Java; IN and OF for COBOL), members are automatically listed. You can access this feature on demand by pressing Alt+Dot, finding identifiers when there is no member operator (list locals, global variables, current class members, etc.). For example, for the C language, to find a string function, type the string on the command line and press **Alt+Dot**. If you want to disable automatic listing and only list members on demand, turn List Members off, as follows:

1. From the main menu, select **Tools > Options > File Extension Setup**.
2. Select the extension you want to affect from the **Extension** drop-down list.
3. Select the [Context Tagging® Tab](#).
4. Clear the **Auto-list members** check box.

The following example shows the results of what is displayed after typing a Dot when entering Java source. Notice that the Javadoc comments are displayed in a mini-HTML browser. To view documentation for Java APIs, you must install the source files as part of the JDK. If clicking on a URL, the default HTML browser starts. Clicking on other hypertext links navigates within the comment window. The equals method in the example below has two occurrences, one in the String class and one in the Object class. Press Ctrl+PgDn or Ctrl+PgUp to select the next or previous occurrence.



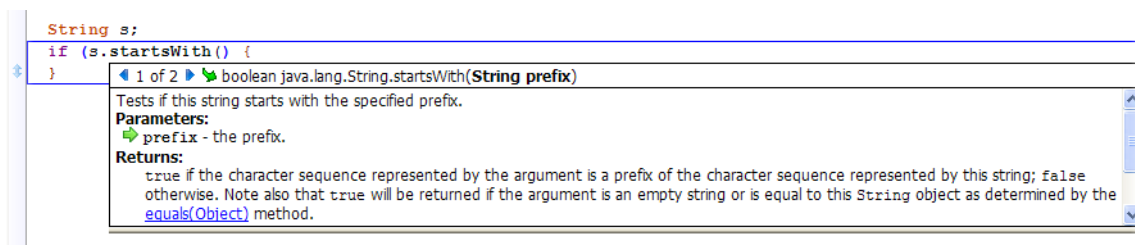
The example below shows the display after typing a Dot when entering C++ source code. The stack class is one of the C++ standard template library classes.



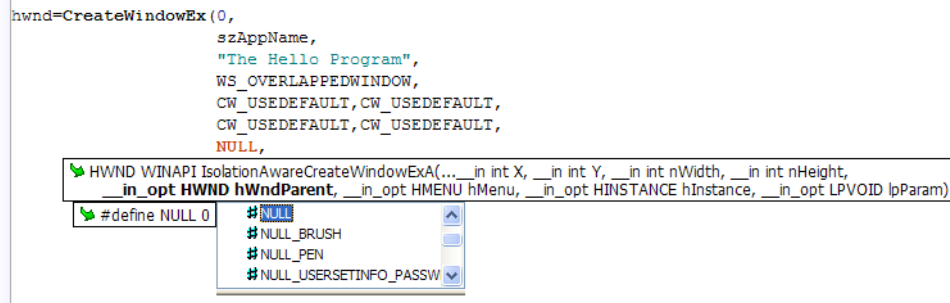
## Parameter Information

The prototype for a function is automatically displayed when typing a function operator such as the open parenthesis. This also highlights the current argument within the displayed prototype. When working with C++, parameter info is also automatically displayed when typing a template argument operator such as <.

The following example shows the result of pressing Alt+Comma inside the argument list of the Java API String method **startsWith**. The Javadoc comments are displayed in a mini-HTML browser. To view documentation for Java APIs, you must install the source files as part of the JDK. If clicking on a URL, the default HTML browser starts. Clicking on other hypertext links will navigate within the comment window. The **startsWith** method has two overloads that accept different arguments. Press Ctrl+PgDn or Ctrl+PgUp to select the next or previous occurrence.



The example below shows the result of pressing Alt+Comma inside the argument list of the WIN32 API function **CreateWindowEx**.

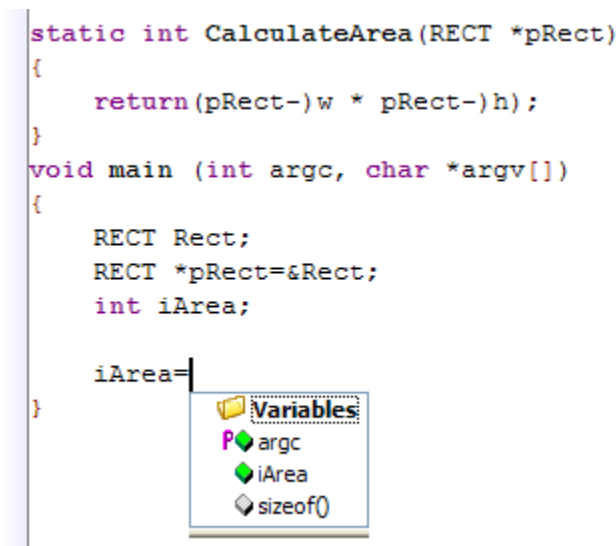


## Auto List Compatible Parameters

When typing a function operator such as the open parenthesis, a list of compatible variables and expressions for the current argument is displayed. Auto List Compatible Parameters can also be used instead of List Members, in assignment statements ( $x=<\text{Alt}+\text{Comma}>$ ) and when listing members of a class or struct. Keep in mind, not all possible variables and expressions are listed. Press  $\text{Alt}+\text{Dot}$  if the symbol that you want is not listed. To access Auto List Compatible Parameters on demand, press  $\text{Alt}+\text{Comma}$ . If you want to disable automatic listing and only list parameters on demand, turn Auto List Compatible Parameters off, as follows:

1. From the main menu, select **Tools > Options > File Extension Setup**.
2. Select the extension you want to affect from the **Extension** drop-down list.
3. Select the [Context Tagging® Tab](#).

The following example displays the results of pressing  $\text{Alt}+\text{Comma}$  after an assignment operator. The **Rect**, **pRect**, and **argv** are not listed because their types do not match.



## Completions

Completions save keystrokes as you are typing code by providing a way to automatically complete partially-typed text. Press Ctrl+Space to complete (type the rest of) the current symbol. If a unique match is not found, a list is displayed allowing the selection of the exact match. See [Completions](#) for more information about working with this feature.

## Symbol Browsing

SlickEdit® gives you the ability to browse and view symbols in your files or workspaces. There are several tool windows that display information as you work to help you find what you need exactly when you need it:

- **Class** - This tool window provides an outline view of both the members of the current class as well as any visible inherited members. It also shows the inheritance hierarchy of the current class. The Class tool window is docked as a tab on the left side of the editor by default.
- **Current Context** - Current Context displays the logical location of the cursor within your code. If it is within a class, it displays the class name. If it is within a function, it displays the function name. If the function is within a class, it displays the class and the function name. The tool window is docked in the top upper-right section of the editor by default.
- **Defs** - The Defs (Definitions) tool window contains the defs (definitions) browser, which provides an outline view of symbols in the current workspace. It is docked as a tab on the left side of the editor by default.
- **Find Symbol** - This tool window is used to locate symbols (tags) in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match. This window can be displayed by selecting **Search > Find Symbol** or by using the **gui\_search** command.
- **Preview** - The Preview tool window provides a portal for viewing information in other files without having to open them in the editor. It automatically shows this information when you are working with certain features. This window is docked as a tab on the bottom of the editor by default.
- **References** - This window displays the list of symbol references (uses) found the last time that you used the Go to Reference feature (**Ctrl+/** or **push\_ref** command—see [Symbol Navigation](#) for more information).
- **Symbols** - The Symbols tool window contains the symbol browser, which lists symbols from all of the tag files. It is docked as a tab on the left side of the editor by default.
- **Symbol Properties** - This window displays detailed information about the symbol at the cursor location. It can be displayed by selecting **View > Toolbars > Symbol Properties** or by using the **activate\_tag\_properties\_toolbar** command.

For more detailed information about these tool windows and how SlickEdit can help you browse symbols, see [Symbol Browsing](#). For information about how to navigate between symbols in files, see [Symbol Navigation](#).

## Statement Level Tagging

Statement Level Tagging is a feature of Context Tagging® that provides a more detailed view of items in the Defs tool window for C/C++, Java, Python, and Visual Basic .NET. Along with definitions, view constructs like **if**, **while**, and **for** statements. It also displays every non-comment line of code. To see this feature in action, from the [Defs Tool Window](#), right-click and select **Show Statements**.

## Building and Managing Tag Files

Context Tagging® creates tag files to store information about symbols and, optionally, cross-reference information from your source code. Many of the most powerful features of SlickEdit® use this information to speed your coding.

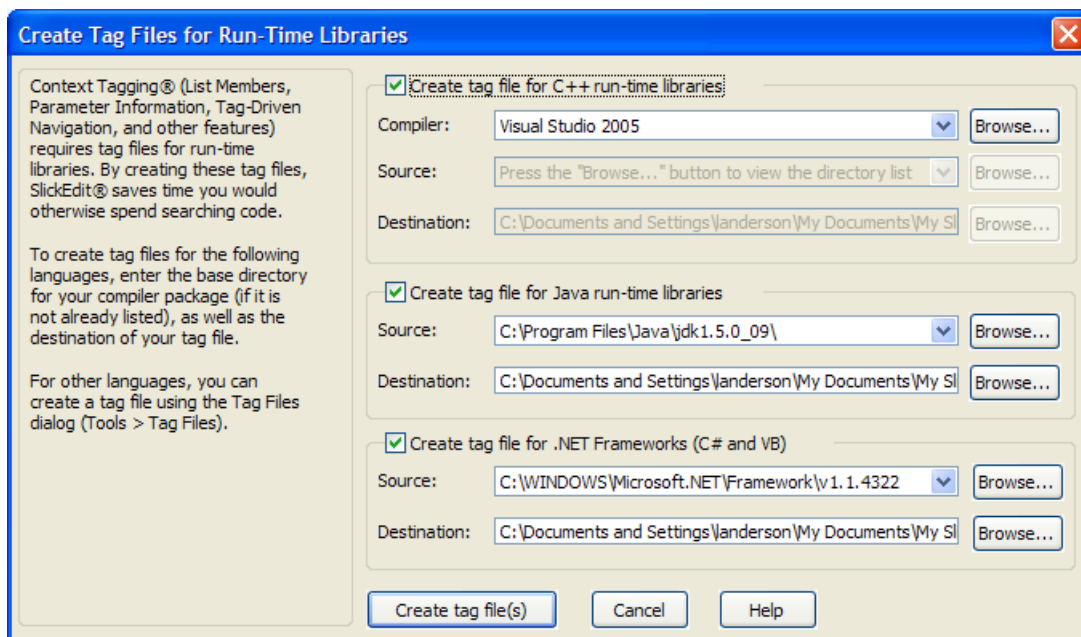
### Building Tag Files

Tag files are automatically created and maintained for files in the workspace (see [Creating Tag Files for Run-Time Libraries](#)). You may need to create extension-specific tag files for compiler includes or other libraries that you have (see [Tagging Run-Time Libraries](#)).

After a tag file is created, it is updated in the background when you make modifications. If you modify some source files using an application other than SlickEdit, you will need to rebuild the tag file. Tag file names have the extension `.vtg`. By separating tag files for different languages, the Context Tagging features can identify symbol information for the file that you are currently editing.

### Creating Tag Files for Run-Time Libraries

The Create Tag Files for Run-Time Libraries dialog appears when SlickEdit® is run for the first time. It allows you to build tag files for commonly used languages and their libraries, including C, C++, Java, and .NET. You can access this dialog at any time in order to create tag files, from the [Context Tagging® - Tag Files Dialog](#) (select **Tools > Tag Files**, then click **Auto Tag**).



To create tag files for the languages listed, enter the base directory for your package (if it is not already filled in), as well as the destination of your tag file. Click **Create tag file(s)** and the Building Tag Files dialog box opens showing the progress as the tag file is built.

For source files other than these languages, use the Add Tag File dialog, which allows you to choose from a list of languages the source type for which to insert the tag file. See [Creating Extension-Specific Tag Files](#) below.

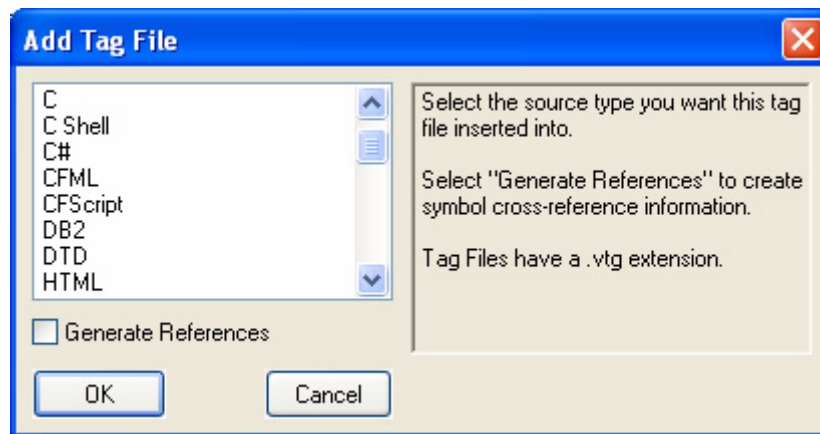
## Creating Extension-Specific Tag Files

Extension-specific tag files provide the same symbolic information for libraries that is provided for code in your projects. A library is a pre-built unit of code that is not edited as part of this development effort. These tag files are accessible from any project written in the same language.

You need to create an extension-specific tag file if your project uses a compiler whose standard libraries are not tagged by the Create Tag Files for Run-Time Libraries dialog (see [Creating Tag Files for Run-Time Libraries](#)) or if you are using a third-party library. Additionally, you may have local libraries that are reused from project to project.

To create an extension-specific tag file, complete the following steps:

1. From the main menu, select **Tools > Tag Files**. The [Context Tagging® - Tag Files Dialog](#) is displayed.
2. Click **Add Tag File** to open the Add Tag File dialog.

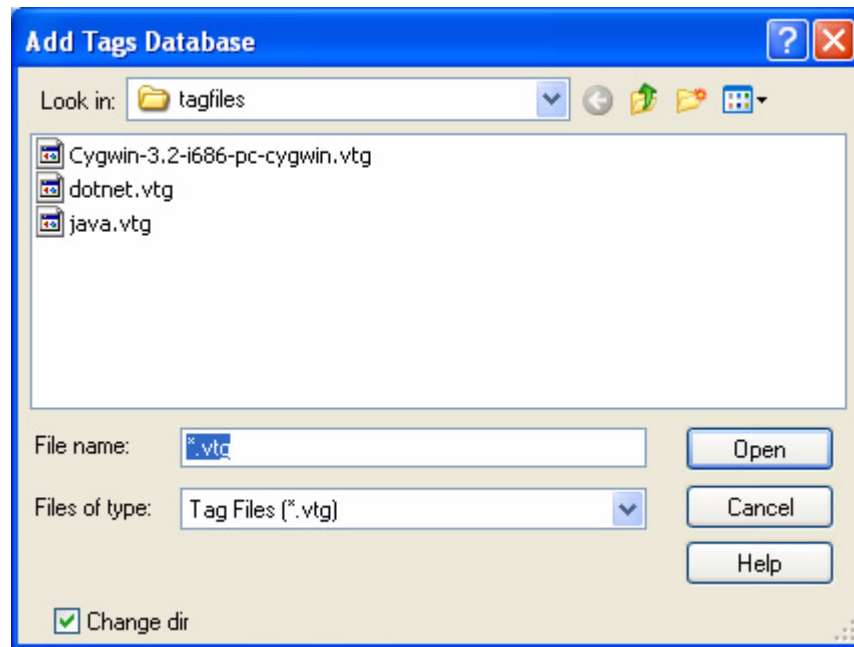


3. Select the source type into which you want the tag file inserted. Select **Generate References** only if you want library functions to be shown when you list references.

**NOTE Generate References** creates an inverted file index so that you can quickly find which files contain which symbols. Workspace tag files create this index by default. This information is used to build a list of references (using the **push\_ref** command, bound to Ctrl+/- in the CUA emulation). In general, it's better to have the reference list contain functions that are part of this workspace and not in libraries. If **Generate References** is not checked, you will still be able to jump from a symbol to its definition in a library using Ctrl+Dot (**push\_tag**).

This option is off by default since most programmers do not want to see library functions shown in the reference list.

4. Click **OK**. The Add Tags Database dialog opens.



5. Select an existing tag file or enter the name for the new tag file. Tag files have the extension `.vtg`.
6. Click **Open** to display the Add Tree dialog. Navigate to the root of the library source code and click the **OK** button.
7. The Building Tag File dialog opens showing the progress as the tag file is built. When finished, the contents are displayed in the Context Tagging® - Tag Files dialog.

See [Managing Tag Files](#) for more information.

## Tagging Run-Time Libraries

Create extension-specific tag files for include files of compiler packages or utility libraries or both. This enables the Context Tagging® feature set to work for all symbols, not just those symbols in the project. Context Tagging needs all symbol information to work properly.

A tag file is automatically built for the run-time libraries of C#, InstallShield, JavaScript, Perl, PV-WAVE, Slick-C®, Tornado, TCL, and Visual Basic .NET, and usually it is not necessary to build tag files for the run-times of these languages. If you already built a tag file for run-times during installation, you can skip this section. If you are using Perl, Python, or TCL, and the compiler cannot be found in PATH (or registry for Windows), you need to build tag files for these run-time libraries.

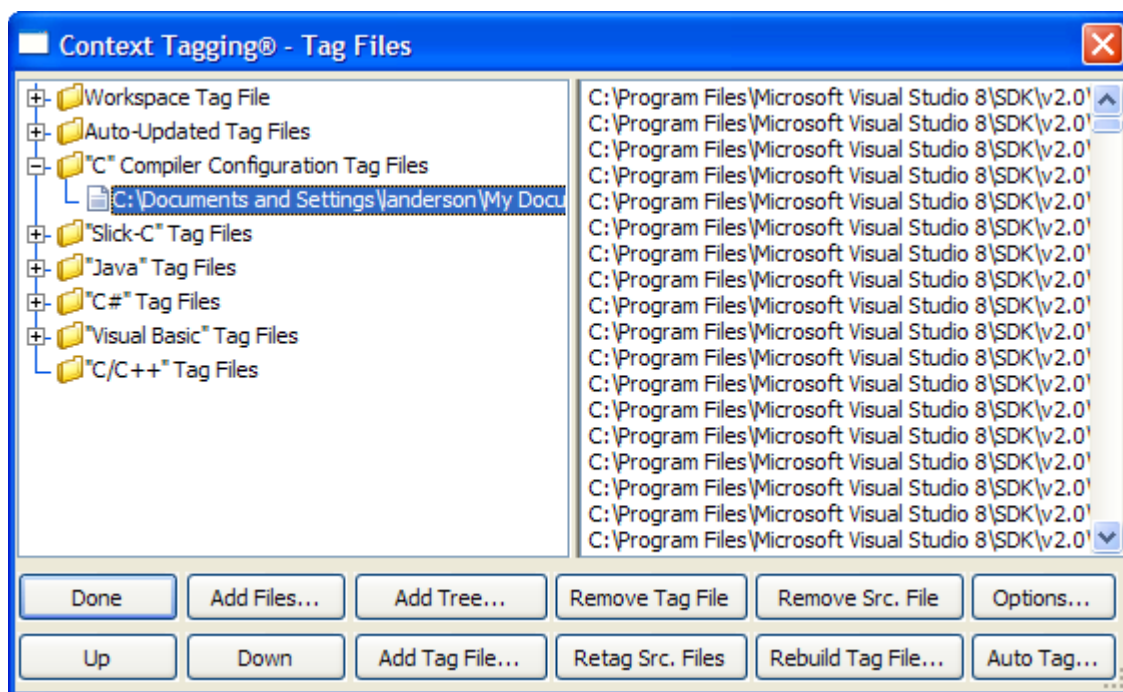
## Configuring Context Tagging® for COBOL

All of the Context Tagging features for COBOL, except Parameter Information, are provided by scanning COBOL source file and the copy books that are included. This information is used by List Members, completions, tag-driven navigation, symbol preview, and in the Outline view. Parameter Information for COBOL commands and intrinsic functions are provided by the COBOL built-ins file created during product installation. To provide Parameter Information for subroutines, you must build a tag file that will hold linkage information from the subroutine's point of view.



## Managing Tag Files

The [Context Tagging® - Tag Files Dialog](#) (**Tools > Tag Files**) is used to manage your tag files.



The left pane of the dialog lists all of your tag files, separated into categories (see [Tag File Categories](#) below). A tag file having a file icon with blue arrows indicates the tag file is built with support for cross-referencing. The right pane of the dialog lists all the source files indexed by the currently selected tag file.

For information about the buttons available, see [Context Tagging® - Tag Files Dialog](#).

### Tag File Categories

The Tag File categories, described below, are listed on the left side of the Context Tagging® - Tag Files dialog (**Tools > Tag Files**).

- **Workspace Tag File** - The tag file for the current active workspace. Visible only if a workspace is open.
- **Auto-Updated Tag Files** - These tag files are designed to be shared by multiple users of SlickEdit® on a network. You can use the **vsmktags** utility to rebuild these tag files as part of your nightly build process. When SlickEdit detects that a newer version of an auto-updated tag file is available, it will automatically copy in the newer version and begin using it.
- **"C" Compiler Configuration Tag Files** - These tag files correspond to one of the C/C++ compiler configurations. These may be configured by using the C/C++ Compiler Properties dialog box (**Tools > C++ Refactoring > C/C++ Compiler Options**). See [C/C++ Compiler Settings](#) for more information.
- **Extension-Specific Tag Files** - The tag files listed under each of the language-specific categories apply to that language only. Use these categories to add tag files for third-party libraries.



## Tag File Search Order

When doing tag lookups, the tag files are searched in a specific order, which affects the tags found. The following are examples of the order in which tag files are searched.

### Example: Java Tag File Search Order

If a Java source file is open, when a tagging-related operation is performed, the tag files are searched in the following order:

1. Workspace tag file, providing it contains other Java source files.
2. Auto-updated tag files containing other Java source files.
3. Extension-specific Java tag files, in the order that they are listed in the [Context Tagging® - Tag Files Dialog](#).

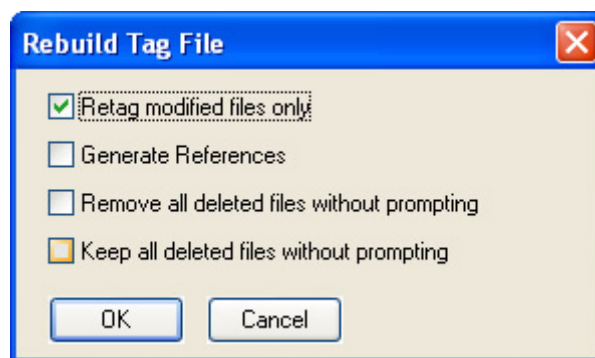
### Example: C/C++ Tag File Search Order

If a C/C++ source file is open, when a tagging-related operation is performed, the tag files are searched in the following order:

1. Workspace tag file, providing it contains other C/C++ source files.
2. Auto-updated tag files containing other C/C++ source files.
3. The “C” Compiler Configuration tag file corresponding to your default C compiler configuration as specified in your project (**Tools > C++ Refactoring > C/C++ Compiler Options**), or global default.
4. Extension-specific C tag files, in the order that they are listed in the [Context Tagging® - Tag Files Dialog](#). Note that if you have a “C” Compiler Configuration tag file, `cpp.vtg` will be excluded from this list.

## Rebuilding Tag Files

The Rebuild Tag File dialog box contains options for rebuilding the selected file. To display the Rebuild Tag File dialog, choose **Tools > Tag Files**. When the [Context Tagging® - Tag Files Dialog](#) is displayed, select a file to rebuild, then click **Rebuild Tag File**.



The following settings are available:

- **Retag modified files only** - If checked, SlickEdit® will incrementally rebuild the tag file, only retagging files that have been modified since the last time they were tagged. If not checked, SlickEdit will rebuild the entire tag file from scratch.

- **Generate References** - If checked, the tag file will be built with support for cross-referencing. Tag files with support for references are slightly larger and take slightly more time to build. They will also be included in all symbol references searches, which may not be necessary, especially for third-party libraries.
- **Remove all deleted files without prompting** - If checked and the tag file contains a source file which no longer exists on disk, the source file will be removed from the tag file without prompting for confirmation.
- **Keep all deleted files without prompting** - If checked and the tag file contains a source file which no longer exists on disk, the source file will not be removed from the tag file without prompting for confirmation.

**NOTE** The options **Remove all deleted files without prompting** and **Keep all deleted files without prompting** are mutually exclusive—selecting one will deselect the other.

## Context Tagging® Options

### General Context Tagging® Options

The [Context Tagging® Options Dialog](#) allows you to set general parameters for the Context Tagging feature set. Here, you designate how tagging is done, how the references function within the application, and you can also tune the application to maximize performance. To display this dialog, choose **Tools > Options > Context Tagging® Options**. See [Context Tagging® Options Dialog](#) for descriptions of the options.

**TIP** To improve tagging performance, you may need to adjust the **Tag file cache size** on the Virtual Memory tab of the General Options dialog (**Tools > Options > General**). See [Virtual Memory Tab](#) for more information.

### Extension-Specific Context Tagging® Options

Context Tagging options can be configured for each file extension type. This allows you to enable and disable particular features on a per-language basis. To set options, from the main menu choose **Tools > Options > File Extension Setup**. Select the **Context Tagging®** tab, then from the **Extension** drop-down list, select the extension you wish to work with. Options described in the section [Context Tagging® Tab](#).

# Building, Running, and Debugging

This chapter contains the following topics:

- [Building and Compiling](#)
- [Running and Debugging](#)



# Building and Compiling

---

SlickEdit® contains a comprehensive project management and build system. Build commands can be easily configured for any external compiler. You may choose to allow SlickEdit to manage project dependencies, or you can use an external build system such as **make** or **ANT**.

## Using Build and Compile Operations

SlickEdit® provides the capability to build a project or compile single files.

- To build the active project, select **Build > Build** from the main menu, press Ctrl+M, or use the **project\_build** command.
- To build a different project, open the Projects view, right-click on a project and select **Build**.
- To compile the file in the active editor window, select **Build > Compile** from the main menu, press Shift+F10, or use the **project\_compile** command.
- To compile a different file, display the Projects tool window, right-click on a file, and select **Compile**.

If your workspace contains multiple projects, sometimes one or more projects must be compiled before a particular project can be compiled. Select **Project > Project Properties** and then select the [Dependencies Tab](#) to view or set dependencies for the active project. Alternatively, you can right-click on a project in the Projects view and select **Dependencies**, allowing you to set dependencies for a project that is not active.

Before you can execute the Build or Compile commands you must set the current project or define an extension-based project. To define an extension-based project command, use the Extension Options dialog box (**Tools > Options > File Extension Setup**). Select the [Advanced Tab](#), then select the extension from the Extension drop-down list, then click **Extension Specific Project**. You will probably want the Build command to be based on the current project and not the current extension. Use the **workspace\_new** command (**Project > New**) to create a workspace or project. If the current project has a Compile command defined, the extension-specific project Compile command will be ignored.

By default, the Build or Compile command is executed in the Build window. This allows you to continue editing while the compiler runs. You can process the error messages as they appear in the Build window instead of waiting until the compile process finishes. Use the **stop\_process** command or select **Build > Stop Build** to stop the compiler running. To send the compile output to an editor window (named ".process"), right-click in the Build window and select **Send Compile Output to Editor Window**.

To customize the Build and Compile commands, select **Project > Project Properties**. Select the [Tools Tab](#), then select an operation from the list: Build, Compile, or Rebuild. Depending on the language and your other project settings, either a command line or an Options button will be displayed allowing you to configure the operation.

## Compiling a Project

The Build menu items **Compile** and **Build** start the compile and build commands respectively for the current project. If you selected a compiler package, you can try these commands now. To change these commands and a few other project options, use the [Tools Tab](#) of the Project Properties dialog box (**Project > Project Properties**). The **Build > Next Error** and **Build > Previous Error** menu items enable quick navigation of compiler errors.

## Using VSBUILD to Compile

Use the utility program **vsbuild** to compile files in a project and process dependencies between projects. This tool is intended to help implement project support. It has a built-in make facility for Java and C++, performs project dependencies, and processes pre- and post-build commands. For example, if `file1.java` references `file2.java` which references `file3.java` and `file3.java` is modified, then when you invoke the Build command, `file1.java`, `file2.java`, `file3.java` will be recompiled. Invoking **vsbuild** with no parameters displays invocation options.

## Compiling a Visual C++ Project

For Visual C++ v5.x and v6.x, the default compile command uses the `nmake` program which requires a makefile (`.mak` extension). Visual C++ v5.x and v6.x do not automatically create a makefile for you. Use **Project > Export Makefile** in Visual C++ to create or update the makefile. For Visual C++ v5.x or higher, the default build and rebuild commands do not need a makefile.

You can customize the compile, build, and rebuild commands from the [Tools Tab](#) of the Project Properties dialog box (**Project > Properties**).

If you get an error when you run `nmake`, you need to run the `VCVARS32.BAT` file (shipped with Visual C++) in a DOS box that you start the editor from. This will set the environment so that the editor can run these compiles.

## Specifying Build on Save

A build can be automatically launched upon saving the file or files within a project. To specify this option and to toggle it on/off, from the main menu choose **Build > Build Automatically on Save**. By default this option is not selected.

## Specifying Open Commands

The [Open Tab](#) of the Project Properties dialog (**Project > Project Properties**) lets you enter commands that are executed when the project is activated. This information is stored per project, not per configuration. This tab is disabled for extension-based projects.

To enter a new command to be opened for a project, simply type the command(s) in the editor control window. Each line should contain a command just as you would type it on the command line. You can set environment variables in the concurrent build window with the **set** command. For example, the command **set xxx=yyy** sets the environment variable `xxx` to the value `yyy`. This automatically supports different UNIX shells. Use **concur\_command** to send a command string to the concurrent build window, for example: **concur\_command export name=value**.

# Language-Specific Build Methods

## Build Methods for GNU C/C++

There are three build methods available for GNU C/C++. With these build options you will not need to convert the current build methods to use the GNU debugger. You can select one of these build methods when you create a new GNU C/C++ Wizard project:

- **Build without a makefile (dependencies automatically checked)** - When you create a GNU C/C++ Wizard and select this build option, no makefile is ever generated. Instead, our **vsbuild** utility program determines what needs to be compiled dynamically. We recommend using this option when you are not worried about how the build gets done. Make sure the project include

directories (**Project > Project Properties**, select the [Directories Tab](#)) are set up correctly so include files may be found.

- **Build with a user-maintained makefile or custom build command** - When you create a GNU C/C++ Wizard and select this build option, no makefile is ever generated and by default the build command is set to **make**. You can change the build command to anything you want using the Project Properties dialog (**Project > Project Properties**, select the [Tools Tab](#), select **Build** for the tool name). Choose this option when you already have your own method for building the source.
- **Build with an auto-generated, auto-maintained makefile** - When you create a GNU C/C++ Wizard and select this build option, a makefile is automatically generated and updated when files are added to the project. We recommend using this option when you need a makefile and do not want to use the built-in **vsbuild** utility. Make sure the project include directories (**Project > Project Properties**, select the [Directories Tab](#)) are set up correctly so include files may be found. To start a build from outside the application, execute the following command where **make** is the name of the make program, **Makefile** is the name of the makefile, and **ConfigName** is the name of the configuration: **make -f Makefile CFG=ConfigName**.

## Build Methods for Xcode

When SlickEdit® opens an Xcode project, it creates a view of the project that is consistent with other SlickEdit workspaces. This creates a few discrepancies between from the view of the project that Xcode provides. The most noticeable difference is that the files in the project cannot be viewed in a single tree, rather the files are always separated by the target that uses the file.

There are build methods available when using Xcode. To open an Xcode project, complete the following steps:

1. From the user interface, select **Project > Open Other Workspace > Xcode Project**.
2. In the Directory window, select the `.xcode` directory. This directory appears as a file inside the Finder.
3. From the File window, select the `project.pbxproj` file.
4. From the main menu, select **Project > Set Active Project**.
5. Select the project that you want to use.
6. Select **Build > Set Active Configuration**.
7. Select the style that you want.
8. Select **Build > Build**.
9. The project is then built, and you can work with your project.

## Working with Build Errors

### Viewing and Navigating Errors

Error markers (displayed as red "X" bitmaps) are placed on the lines of the errors in the editor windows after a build or compile is completed. To clear these markers, fix the errors and rebuild. You can also clear the markers by choosing from the main menu **Build > Clear All Error Markers** or by using the **clear\_all\_error\_markers** command.

Execute the **next\_error** command (Ctrl+Shift+down arrow, or **Build > Next Error**) to place the cursor on the line of the file containing the next error. Open the build window by clicking on the Build tab to view the

compiler's error messages. You can use the **cursor\_error** command (Alt+1, double-click, or **Build > Go to Error or Include**) to set the next error starting search position and go to a specific error.

You can also use the **cursor\_error** and **next\_error** commands to process messages from the **grep** (UNIX: **sgrep**) program provided. For example:

1. Activate the concurrent build window (**Project > Activate Build**).
2. Type **grep main \*.c** (UNIX: **sgrep main \*.c**) and press **Enter**.
3. Press Ctrl+Shift+Down arrow to cursor through the located occurrences.
4. Press PgUp to move the cursor off the concurrent build window command prompt so you can use the **cursor\_error** command (Alt+1 or double-click) on a specific occurrence.

### Listing Errors

To see a list of errors that have occurred during the current editing session, use the **list\_errors** command. The Error File dialog box will be displayed.

Move the cursor in the editor control to the error message you want to go to and click **Go To Cursor Error** to view the source code.

### Parsing Errors with Regular Expressions

SlickEdit® provides the capability to scan the Build window for errors and warnings (using **Build > Next Error** and **Build > Previous Error**). You can also navigate from an error or warning to the corresponding location in the source code (using **Build > Go to Error or Include** or by double-clicking on the error in the Build window).

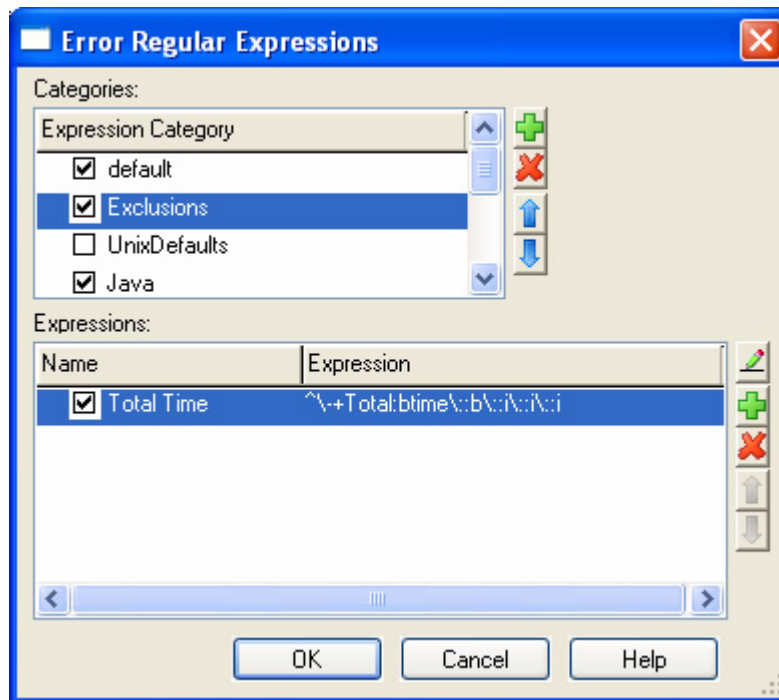
SlickEdit uses regular expressions to parse the contents of the Build window and retrieve the file name or path, line number, column number, and error message. A set of default regular expressions are included that can parse error messages from supported compilers like Visual Studio, GCC, and Java. For other tools, you may have to write additional regular expressions.

Error parsing regular expressions are written using the SlickEdit regular expression syntax (see [Regular Expression Syntax](#)). They are stored in the `ErrorRE.xml` file located in your configuration directory. If the file is deleted, SlickEdit will create a new one with the default values. Rather than modifying the XML by hand, you can use the Error Regular Expressions dialog to create new regular expressions or manage the list of existing ones.

### Configuring Error Parsing

To configure error parsing, use the Error Regular Expressions dialog. This dialog is accessed from the main menu **Build > Configure Error Parsing**.





The **Categories** list displays all the expression categories that are defined in the `ErrorRE.xml` configuration file. Highlighting a category will show the individual expressions for that category.

## Enabling Expressions

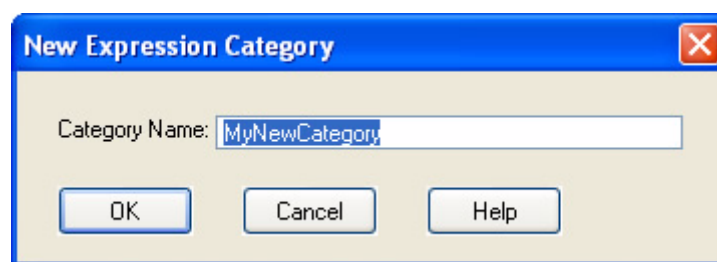
To enable or disable an expression, or a whole category of expressions, simply click the checkbox to the left of the expression or category. If a category is unchecked (disabled), then the expressions are not used to parse build output, regardless of their checked or unchecked status.

## Setting Priority

To optimize performance for your development needs, you may re-prioritize either expressions or whole categories by using the up and down blue arrow buttons. The default category cannot be re-prioritized.

## Adding New Categories

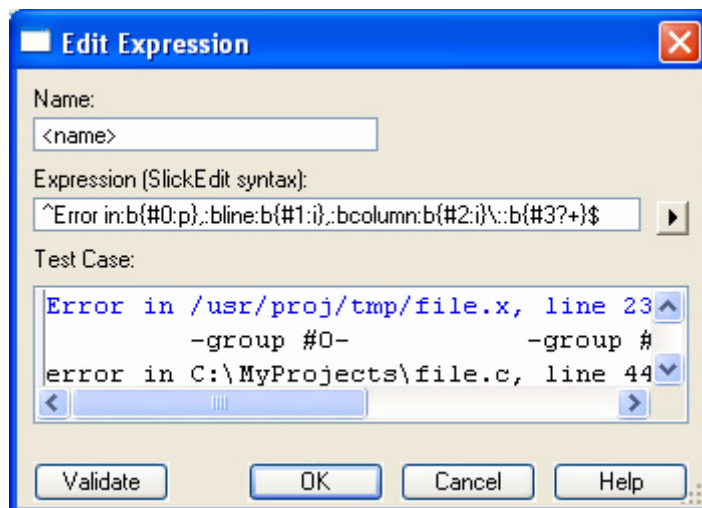
Click the green **Plus** icon next to the category listing. The following prompt is shown.



Enter the name for your new category and click **OK**. Category names must be unique and the dialog will prevent you from adding duplicate entries.

## Adding New Expressions

You can add new expressions to any category. Highlight the category you want the new expression to be under, then click the green **Plus** icon to the right of the Expressions listing. The following dialog is displayed.



Enter a name for your new expression. The regular expression must be authored using SlickEdit® Regular Expression syntax. The arrow to the right of the entry field will display a menu of common regular expression syntax constructs to assist you. A “starter” expression is provided for you, as well as some sample error output lines. See the following sections on how to author and test your expression.

Once you have created and tested your new expression, click **OK** to save the expression. You must also click **OK** when quitting the main configuration dialog to save your changes.

### Exclusions

Some of the error parsing expressions may match lines that you do not want recognized as errors. To eliminate these “false positive” matches, define a new expression in the Exclusions category. The default configuration file contains an expression to match the “Total Time” build output line that is generated by SlickEdit®’s internal build system, **vsbuild**. Any new exclusion expressions you write should be very strict to prevent real error lines from being skipped. You do not have to define match groups in the exclusion expressions since they will not be used to extract file name and line number information.

## Editing Expressions

To edit an existing expression, double-click the expression in the expression listing, or highlight the expression and click the small **Edit** icon to the right of the listing. This launches the same dialog that is used to create a new expression.

## Error Expression Groups

In order to navigate to the file that caused the build error or warning, the regular expression needs to be able to identify the file name, and optionally the line and column number, as well as the error message.

This is accomplished by using four *Tagged Expressions*, also known as *match groups*. The following table documents the match groups used to identify specific portions of an error message.

Group Number	Group Syntax	Purpose
0	{#0:p}	Retrieves the file name or file path.
1	{#1:i}	Retrieves the error line number.
2	{#2:i}	Retrieves the error column.
3	{#3}	Retrieves the error message text.

The expression for Group #3 can match any portion of the error message you like. The sample expression **{#3?+}\$** is just matching all remaining characters up to the end of the line. The groups can occur in any order in your expression. For example, if the build tool output places the file name, line, and column after the error message, like the following hypothetical example:

```
Error E509: Bad format: found in /usr/tmp/file.x, line 23, column 13
```

then your expression might look something like the following:

```
^Error {#3?+} found in {#0:p},:bline:b{#1:i},:bcolumn:b{#2:i}$
```

## Sample: Creating a New Error Parsing Expression

The steps below demonstrate creating a new regular expression to support error output from a Lint tool. Below are some samples of the tool's output.

Sample 1:

```
file.cpp (5) : Warning 200: Possible dereferencing of null pointer
```

Sample 2:

```
includes\file.h (17) : Warning 003: Macro not parenthesized
```

1. Create a new expression category, and name it "Lint".
2. Highlight the newly created Lint category. The Expressions listing is empty.
3. Create a new expression by clicking the New Expression (green **Plus** icon) button to the right of the expression listing. Copy and paste the sample output lines into the Test Case area.
4. The first thing to match is the file name at the beginning of the line. The group number reserved for the file is **{#0}**. SlickEdit® syntax for matching a file path is **:p**, and **^** represents the beginning of a line. Therefore, enter the following in the Expression entry field: **^{#0:p}**.
5. There is now one space, **:b**, followed by an integer, **:i**, enclosed in parentheses, **\(\)**. The group number reserved for the error line number is **{#1}**. Edit the expression to be: **^{#0:p}:b\({#1:i}\)**.
6. After the line number, there is a space, **:b**, a colon, **\:**, another space, and then the informative message on the remainder of the line. To match any number of characters you can use **?+**, and to match the end of the line is **\$**. The group number reserved for the output message is **{#3}**. Edit the expression to be: **^{#0:p}:b\({#1:i}\):b\:{#3?+}\$**
7. Now test the expression. Click the **Validate** button. You should see a popup message for each line of sample output.

8. Click **OK** to save the new expression, and click **OK** on the main dialog to save your changes to the configuration file.

## Testing Expressions

Copy some sample error or warning output lines from your compiler or build tool, and enter them into the Test Case area. Click the **Validate** button to validate the regular expression against the Test Case text lines. If the regular expression syntax is invalid, then the expression text is colored red, and an error message is displayed on the status line. If any of the lines in the Test Case area match the expression, a message box displays the details of the match, like the following sample.



This popup displays the line of the matched test case and value for each of the four tagged expression groups.

You may also want to use the Regex Evaluator tool window to test your expressions. From the main menu choose **Tools > Regex Evaluator**. Be sure to select the SlickEdit® syntax option when authoring expressions for error parsing. For more information on the Regex Evaluator tool window, see [The Regex Evaluator](#).

## Running and Debugging

SlickEdit® provides debugging capabilities when working in GNU C/C++, Java, and CLR programs, using GDB version 6.6. You can download this version from [www.slickedit.com/php/gdb](http://www.slickedit.com/php/gdb). Other programs will result in SlickEdit launching an external debugger.

**NOTE** The GDB shipped with SlickEdit on Windows is based on Cygwin. If you don't want to use Cygwin for GDB, you can download and install the MINGW-based version of GDB, `gdb-6.3-2.exe`, from [www.mingw.org/download.shtml](http://www.mingw.org/download.shtml). This version of GDB is known to work with SlickEdit and does not depend on `cygwin1.dll`. Use the **Configurations** tab on the Debugger Options dialog (**Debug > Debugger Options** or `debug_props` command) to make it the default native GDB debugger configuration.

## Running a Program

To run a program, complete the following steps:

1. From the main menu, select **Build > Execute**.
2. If there is more than one main program you are prompted to select the one to run.

## Debugging

Use one of the following methods for debugging your code. From the main menu, select **Debug** and then one of the following options:

- **Start** executes the program and will stop when a breakpoint is reached.
- **Step Into** places you on the first executable line of the program.
- **Restart** stops the current debugger session if necessary and then places you on the first executable line of the program (like Step Into).
- **Run to Cursor** will execute the program and will stop when the line under the cursor is reached.

Additional debug operations can be accessed through the Debug toolbar (**View > Toolbars > Debug**).

## Mixed Mode View in Debugger

When debugging, you can view your source code with the disassembled code displayed between each line of source. In this mode you can step execution at the assembly language level for greater control over debugging. The buffer is changed to read-only so that the SlickEdit® product can maintain synchronization between source and disassembled code. To view mixed mode, use the Debug toolbar (**View > Toolbars > Debug**) and click the button **Toggle Display of Disassembly**.

## Debug Key Bindings

The table below shows the key bindings that are available for Debug functions.

Key	Function
F5	Start/continue debugging
Shift+F5	Stop debugging

Ctrl+Shift+F5	Restart debugging
F9	Toggle breakpoint
Ctrl+F9	Toggle breakpoint enable
Ctrl+Shift+F9	Clear all breakpoints
F10	Step over
F11	Step into
Ctrl+F10	Run to cursor
Alt+* (on the numeric keypad)	Show next statement
Ctrl+Alt+B, Alt+F9	Activate breakpoints window
Alt+3, Ctrl+Alt+W	Activate watch window
Alt+4, Ctrl+Alt+V	Activate variables window
Alt+7, Ctrl+Alt+C	Activate call stack
Ctrl+Alt+H	Activate threads window

## Multiple Session Debugging

Multiple session debugging provides the ability to start more than one debugging session within a single instance of SlickEdit®. You can have one session debugging using GDB, and one using Java at the same time.

To create an additional debugger session, use the command **debug\_create\_new\_session**, or complete the following steps:

1. From the main menu, select **Build > Debug**.
2. From the Debug toolbar, select **Add a new Debug selection**.

### NOTES

- If you are using Java, you can only attach to remote JVM.
- CLR (.NET) allows only attaching to a running process.

## Named Sessions

The main debugging session always acquires the name of the current project. (Additional sessions can be created by typing **debug\_new\_create\_session**.) This name is to be numeric or derived from the executable name. The setup information and invocation information for each named session are stored in the workspace history file (.vpwhist). When you create a new session, you can reuse a named session to save time setting up a remote session. You must also confirm the process ID with each session.

A named session can be associated with a project in such a way that it will always be started when the project is debugged. The named session can be debugged using the Create New dialog.

If you detach from the main session, all sessions are stopped and you exit the debugging mode. If you detach from any other session, it simply detaches and control is assumed by another session.

## Attaching to a Running Process (GNU C++ only)

To attach to a running process, complete the following steps:

1. Select **Debug > Attach Debugger > Attach to Running Process**, then select a process to debug.
2. Enter the path to the executable (to pick up debug symbols).
3. Click **OK**.

To detach from a running process, select **Debug > Attach Debugger > Detach from Process**.

## Attaching to a Remote Process (GNU C++ only)

To attach to a remote GDB server or GDB stub process, complete the following steps:

1. Select **Debug > Attach Debugger > Attach to Remote Process**.
2. Enter the path to the executable (to pick up debug symbols).
3. Choose the attach method (socket or device).
4. Select the **Remote options** tab to adjust remote debugging options.
5. Click **OK**.

To detach from a remote debugging session, select **Debug > Attach Debugger > Detach from Process**.

## Attaching to a Core File (GNU C++, UNIX only)

To attach to a core file, complete the following steps:

1. Select **Debug > Attach Debugger > Analyze Core File**.
2. Type the path to the core file.
3. Type the path to the executable (to pick up debug symbols).
4. Click **OK**.

## Attaching to a Remote VM (Java only)

To attach to a remote VM, complete the following steps:

1. Start the remote VM with command arguments similar to the following example:

```
Java -Xdebug -Xnoagent -Xrunjdwp:transport=dt_socket,server=y,
suspend=y,address=8000 MainClass Arg1 Arg2
```

2. From the main menu, select **Debug > Choose Attach to Remote VM**.

To detach from a remote debugging session, choose **Debug > Detach from VM**.

## Setting Breakpoints

The quickest way to set or clear a breakpoint is to press **F9**. This toggles the breakpoint for the current line. You can also set a breakpoint using the appropriate button on the Debug toolbar (**View > Toolbars > Debug**).

A Breakpoints toolbar (**Debug > Windows > Breakpoints**) is also available that displays all of the breakpoints and lets you easily add, remove, and enable breakpoints.

## Setting Conditional Breakpoints

To set a conditional breakpoint, complete the following steps:

1. Set a breakpoint.
2. Select the **Breakpoints tab** on the Breakpoints toolbar.
3. Double-click on the breakpoint for which you want to set a conditional breakpoint.
4. Set the Expression to be evaluated or the Number of times to skip before stopping.
5. Click **OK**.
6. Click **Close**.

### Setting Java Exception Breakpoints

To set a breakpoint when an exception occurs, complete the following steps:

1. Select the **Exceptions tab** on the Breakpoint toolbar.
2. Click **Add** and select one or more exceptions from the list.
3. Click **OK**.

Once an exception breakpoint is added, double-click on it to display the exception properties dialog. This dialog allows you to specify an expression, number of times to skip before stopping, and a specific thread.

### Generate Debug

This feature supports C#, C++, Java, and Slick-C®. Place the cursor on a function name, then select **Tools > Generate Debug** to generate a statement that dumps the name of the current function and the value of the parameter(s) passed in. Place the cursor on a variable name, then select **Tools > Generate Debug** to generate a statement that dumps the contents of that variable. The results are as follows:

- In C#, this will generate a **System.Diagnostics.Trace.WriteLine()** statement.
- In C++, this will generate a **printf** statement.
- In Java, this will generate a **System.out.println** statement.
- In Slick-C, this will generate a **say** statement.

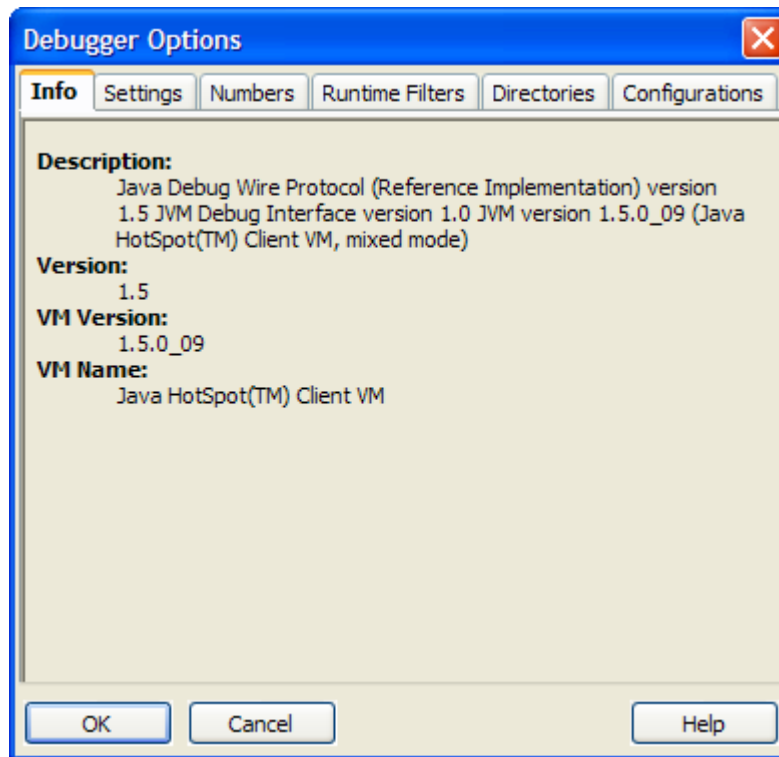
### Setting Debug Options

The Debugger Options dialog is used for tuning the runtime performance of the integrated debugger, examining the properties of the underlying debugger system, setting class filters, and controlling the directories searched for source files. This dialog box is accessed by selecting **Debug > Debugger Options**.

### Viewing Debugger Info

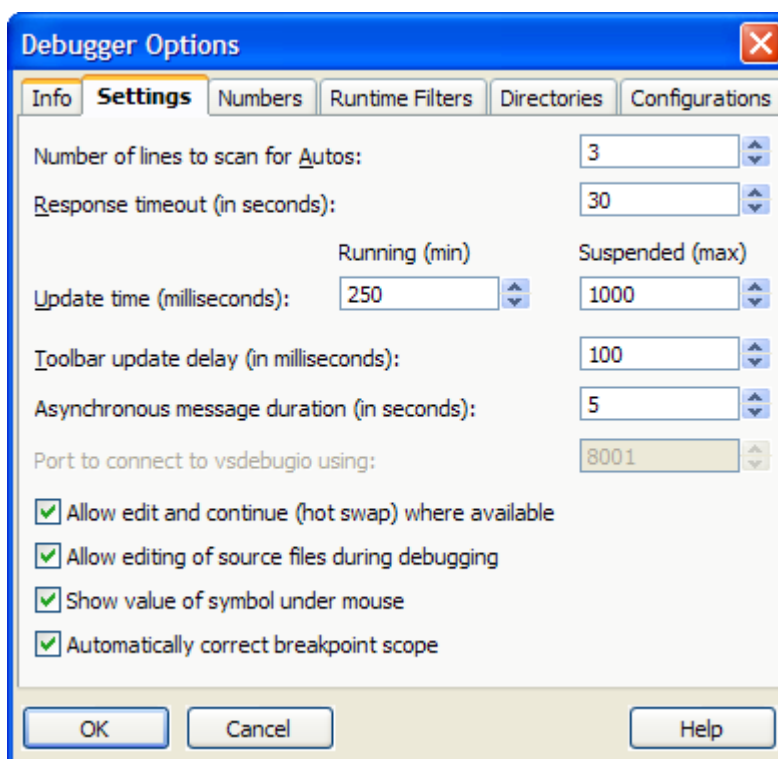
The **Info tab** on the Debugger Options dialog (**Debug > Debugger Options**) shows the properties of the underlying debugger system, including a general description retrieved from the debugger, version number, runtime version, and debugger name. All this information is read-only and may be copied to the clipboard.





## Fine-Tuning Debugger Performance

The **Settings tab** on the Debugger Options dialog (**Debug > Debugger Options**) is used to fine-tune the debugger performance. It is intended for advanced users.



For a list and descriptions of these options, see [Debugger Options - Settings Tab](#).

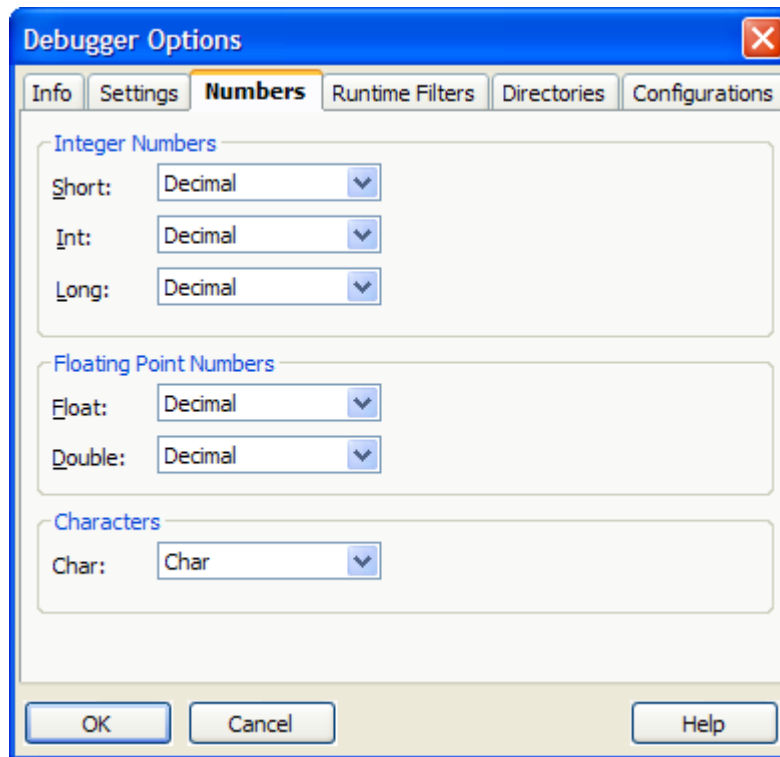
### Using the Hot-Swap Debugger

(Edit and continue - Java only) The Hot-Swap Debugger enables you to edit a file during a Java debugging session, compile or rebuild, and then, continue to debug. This feature is enabled by default when you install SlickEdit®. To disable the Hot-Swap Debugger, from the main menu, select **Debug > Debugger Options > Settings**. Clear the **Allow edit and continue (hot swap) where available** check box.

Keep in mind that when you are working with the Hot-Swap Debugger, that there are certain feature limitations that you might encounter that are defined by the Java Virtual Machine.

### Viewing Numbers in Multiple Bases

The **Numbers** tab on the Debugger Options dialog (**Debug > Debugger Options**) is available for configuring number settings for viewing numbers in multiple bases (hex, octal, and binary views).

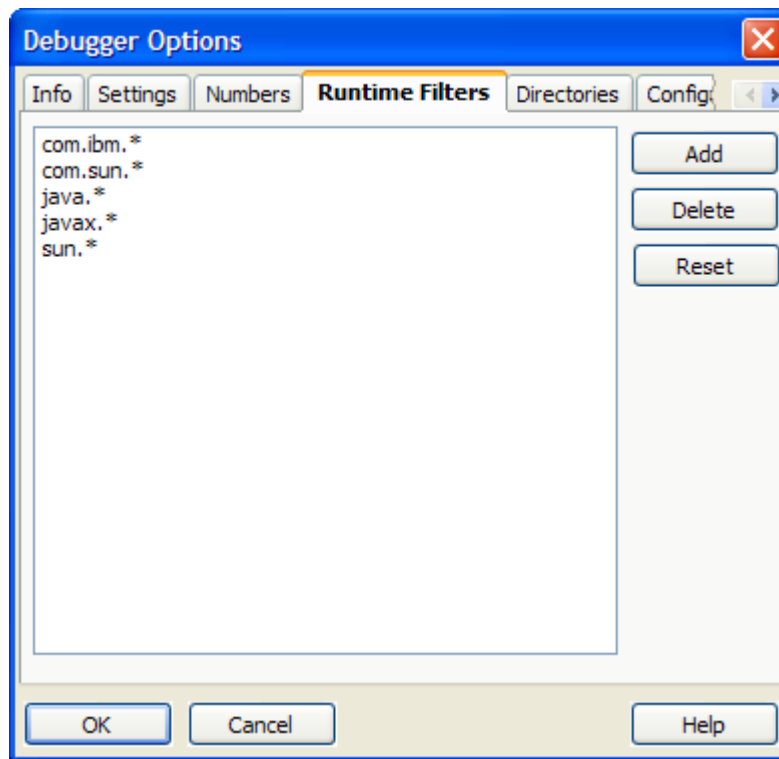


To set hex, octal, and binary view options for debugging, complete the following steps:

1. From the main menu, select **Debug > Debugger Options > Numbers tab**.
2. When the debugger options window is displayed, in the Integer Numbers group box, select **Short**, **Int**, or **Long**.
3. In the Output frame, you can right-click and change to view all as overriding everything or disabling the view variable.

## Setting Runtime Filters

To set runtime filters for debugging, use the **Runtime Filters tab** on the Debugger Options dialog (**Debug > Debugger Options**). This tab allows you to configure the Step Into command to skip certain runtime functions and methods.

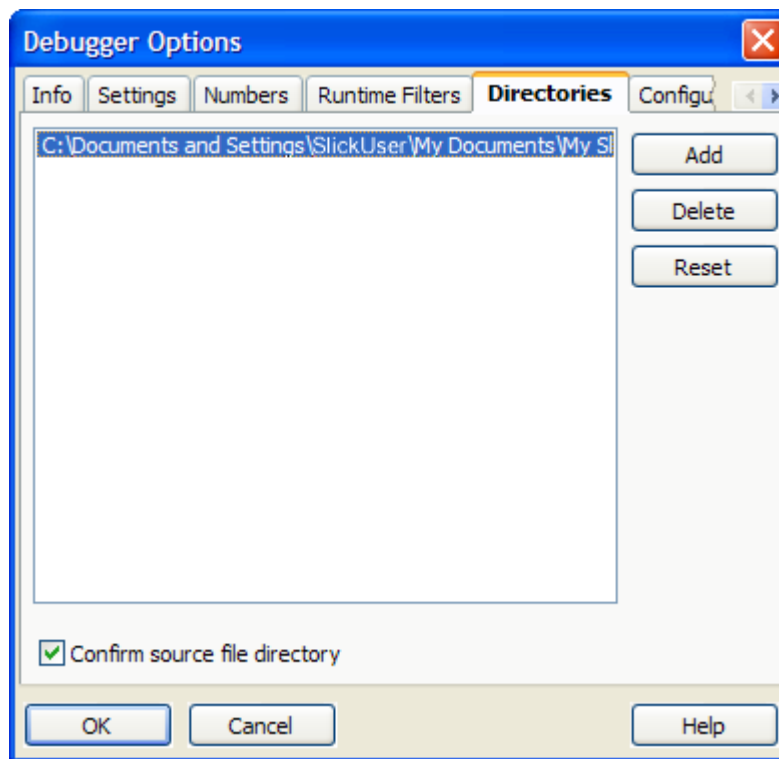


For Java, use this tab to specify packages or specific classes or class patterns for the debugger to consider as Runtime classes. This will affect the behavior of Step Into and the Debug Classes dialog. Step Into will step over statements that call into runtime classes. The **Reset** button will reset the filters back to the defaults, which are `java.*`, `javax.*`, `com.sun.*`, and `sun.*`, which correspond to the defaults used by Sun's JDB debugger.

For GNU C/C++, use this tab to specify a function, or class and method patterns (ex. `strcpy`, `str*`, `MyClass::*`, `MyClass::Insert*`) for the debugger to consider as Runtime functions. This will affect the behavior of Step Into. Step Into will step over statements that call into runtime functions. The **Reset** button will reset the filters back to the defaults.

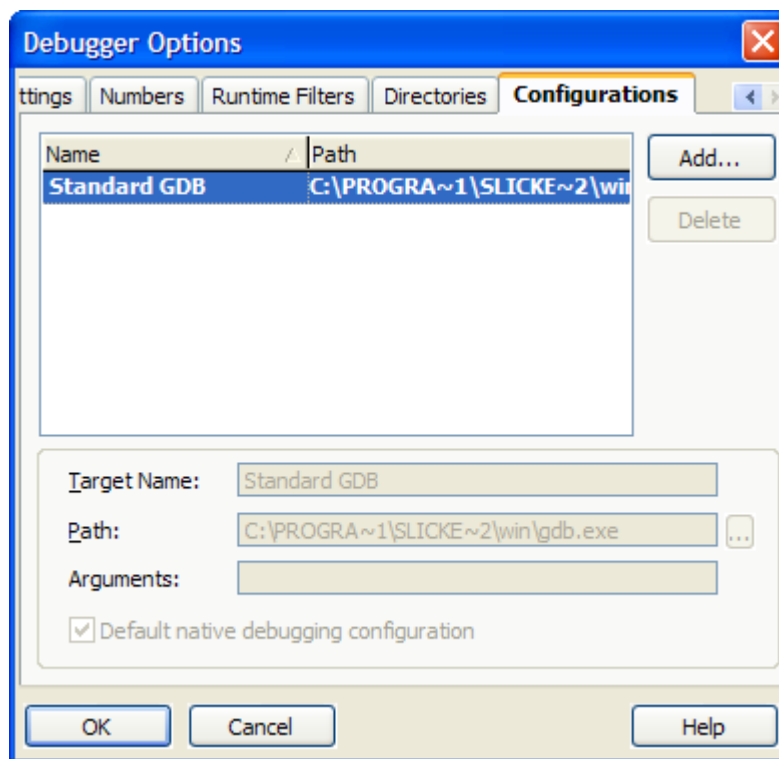
## Setting Debug Directories

The **Directories** tab of the Debugger Options dialog (**Debug > Debugger Options**), shown below, is used to tune the search path used to find source files while debugging. When a source file path can not be resolved using the current directory or the class path, the user is prompted for the source file. The debugger will then save the path in which the source file was found, so you will not be prompted again when that file or another in that directory are needed during a debugging session. The **Reset** button will clear all stored source paths.



## Setting GDB Debugger Configurations

The **Configurations** tab of the Debugger Options dialog (**Debug > Debugger Options**), pictured below, is valid only for projects utilizing the GDB debugger. These settings provide the ability to define multiple GDB debuggers, specify arguments to be passed to each debugger in the list, and define one of the debuggers in the list as the default.



This feature is especially useful for development teams who use cross-compiler platforms for applications such as embedded systems. This feature is also useful in case a newer version of the GDB debugger is available, as it allows the newer version to be added to the list and used in the debugging process. Click the **Add** button and a dialog box opens where you can select the debugger application.

The debuggers in this list also appear on the **Remote Options tab** of the **Debug > Attach Debugger > Attach to Remote Process** dialog box.

## Debugger Tool Windows

The toolbars and tool windows that can be used during debugging are listed in the section [Available Toolbars and Tool Windows](#). These can be accessed from the menu items **View > Toolbars** or **Debug > Windows** when the editor is in debug mode.

# Editing Features

This chapter contains the following topics:

- [Navigation](#)
- [Symbol Browsing](#)
- [Text Editing](#)
- [Color Coding](#)
- [Syntax Indent and SmartPaste®](#)
- [Completions](#)
- [Aliases](#)
- [Syntax Expansion](#)
- [Dynamic Surround and Surround With](#)
- [Bookmarks](#)
- [Code Annotations](#)
- [Commenting](#)
- [Find and Replace](#)
- [Beautifying Code](#)
- [Refactoring](#)
- [Viewing and Displaying](#)





# Navigation

---

There are two types of navigation in SlickEdit®: [Code Navigation](#), which provides in-depth symbol navigation and structure matching, and [Cursor Navigation](#), which pertains to more simple movements within text and files.

## Code Navigation

Some of the most powerful features in SlickEdit® are its code navigation methods, particularly [Symbol Navigation](#). These features allow you to navigate your code the way you think about it, rather than just as a set of files. If you aren't using SlickEdit's code navigation features, then you aren't getting the most out of the editor.

## Symbol Navigation

Symbol Navigation allows you to jump from a symbol to its definition or to a reference with a single keystroke. A pushed bookmark is set, allowing you to return to the symbol with another keystroke. You can chain a series of these navigation operations together, creating a stack of locations. Then pop your way back to the starting location.

To navigate between symbols use the following operations:

- **Go to Definition** - To quickly move the cursor from a symbol to its definition, pushing a bookmark in the process, press **Ctrl+Dot**. Alternatively, select **Search > Go to Definition** or use the **push\_tag** command.
- **Go to Reference** - To create a list of references and optionally jump to the first one, pushing a bookmark in the process, press **Ctrl+I**. Alternatively, select **Search > Go to Reference** or use the **push\_ref** command.
- **Pop Bookmark** - To pop the bookmark and return to the previous location, press **Ctrl+Comma**. Alternatively, select **Search > Pop Bookmark** or use the **pop\_bookmark** command. See [Pushed Bookmarks](#) for more information about working with bookmarks.

When you first call these operations, if a tag file does not exist for the current file, it will be built (see [Building Tag Files](#)).

### TIPS

- **Procs and prototypes** - In C and C++, navigating from a symbol to its definition will prompt you to select whether you want to go to the prototype or the function. You can tell SlickEdit® to always go to one or the other by setting one of the options **Go to Definition navigates to symbol definition (proc)** or **Go to Definition navigates to symbol declaration (proto)**. To set these options, choose **Tools > Options > File Extension Setup**, select the extension you want to affect from the **Extension** drop-down list, then select the [Context Tagging® Tab](#). When the cursor is in the prototype, pressing Ctrl+Dot will navigate to the function and vice versa. If you do not set one of these options, you will be prompted with the [Select a Tag Dialog](#) the first time you navigate from a symbol to its definition.
- **Auto-close visited files** - SlickEdit can automatically close a visited file, one that was opened through symbol navigation but not edited. Choose **Tools > Options > General**, select the [General Tab](#), then select the option **Automatically close visited files**. See [Automatically Close Visited Files](#) for more information.

## Navigating Between Multiple Instances

If more than one instance of the definition or reference is found, the Select a Tag dialog is displayed, from which you can select the instance to navigate to. To go to the next occurrence, press **Ctrl+G** (**Search > Next Occurrence** or **find\_next** command). To go to the previous occurrence, press **Ctrl+Shift+G** (**Search > Previous Occurrence** or **find\_prev** command).

Alternatively, press **Ctrl+Down** (**next\_tag** command) or **Ctrl+Up** (**prev\_tag** command) to place the cursor on the next or previous symbol definition.

## Using the Find Symbol Tool Window

The [Find Symbol Tool Window](#) (**Search > Find Symbol** or **gui\_push\_tag** command) is used to locate symbols (tags) which are declared or defined in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match. See [Find Symbol Tool Window](#) for descriptions of the options that are available.

## More Symbol Navigation Methods

There are several other methods for navigating to symbols:

- The [Symbols Tool Window](#) shows the symbols for all tag files. Right-click in the tool window and select **Find Tag** to search for a specific symbol. You can also use the **cb\_find** command to find the symbol under the cursor and display it in the Symbols tool window.
- At the SlickEdit® command line, use the **f** command and completion keys (spacebar and “?”) to enter a tag name. For example, if tagging the C run-time library, type **f str?** on the command line for a list of tag names starting with “str” (such as **strcpy**, **strcmp**, etc.).
- To navigate to a Slick-C® symbol, you can use the **fp** command (a shortcut for **find\_proc**). If editing a Slick-C macro, then enter the **push\_tag** command (Ctrl+Dot) to find the symbol at the cursor. The **push\_tag** command actually calls the **find\_proc** command with the symbol name at the cursor to perform the task.

## Begin/End Structure Matching

Begin/End Structure Matching moves the cursor from the beginning of a code structure to the end, or vice versa. This works for languages using curly braces “{ }”, “begin” and “end”, or any other defined begin/end pairs.

To place the cursor on the opposite end of the structure when the cursor is on a begin or end keyword pair, press Ctrl+] (**find\_matching\_paren** command) or from the menu choose **Search > Go to Matching Parenthesis**). The **find\_matching\_paren** command supports matching parenthesis pairs {},[], and ().

**TIP** For Python, SlickEdit® supports the matching of the colon (:) token and the end of context. See [Begin/End Structure Matching for Python](#) for more information.

## Viewing and Defining Begin/End Pairs

Use the [Extension Options Dialog](#) to view or define the begin/end pairs for any language. To access this dialog, choose **Tools > Options > File Extension Setup**. Select the extension you wish to work with from the **Extension** drop-down list, then select the [Advanced Tab](#).

In the **Begin/end pairs** text field, specify the pairs in a format similar to a regular expression.

**NOTE** This text box is disabled for languages that have special begin/end matching built-in.

The examples below illustrate the syntax for defining the begin/end pairs. The begin and end pair matching option is case-sensitive by default. Append “;I” to ignore case.

### Example 1

```
(begin),(case)|(end);I
```

The above begin/end pairs are for the Pascal language. The Pascal language requires a more sophisticated expression. This expression indicates the keywords “begin” or “case” start a block and the keyword “end” terminates the block. The “;” is used to specify multiple begins or multiple ends. The “I” operator is used to separate begins from ends.

### Example 2

```
(#ifdef),(#ifndef),(#if)|(#endif)
```

The above pairs are for the C language. The C language has the added complication that **#if** is a substring of **#ifdef**. Due to the implementation of begin/end matching, **#ifdef** must appear before **#if**.

More settings for begin/end pairs can be found on the Formatting Options dialog specific to the extension you are working with. From the main menu, choose **Tools > Options > File Extension Setup**. Select the extension you wish to work with from the **Extension** drop-down list, then click the **Options** button. See the individual language sections in the chapter [Language-Specific Editing](#) for more information about these options.

## Setting the Paren Match Style

As you type a closing parenthesis, highlight and matching options are available. To specify these options, choose **Tools > Options > General**, select the [More Tab](#), then select one of the **Paren match style** options.

The **Highlight** style option temporarily block-selects the text within the parenthesis pair. The **Cursor to Begin Pair** style option temporarily places the cursor on the matching begin parenthesis.

Select **Highlight matching blocks** to automatically highlight the corresponding parenthesis, brace, bracket, or begin/end word pairs under the cursor. To customize the highlighting color, go to **Tools > Options > Color**, and select the **Block Matching** screen element. To adjust the delay in milliseconds before the highlighting is updated, go to **Macro > Set Macro Variable** and modify the variable **def\_match\_paren\_idle**. See [Setting Colors for Screen Elements](#) and [Setting Macro Variables](#) for more information.

## Cursor Navigation

These cursor navigation methods pertain to simple cursor movement within files. We recommend creating key bindings for commands that you use frequently (if a key binding doesn't already exist by default). See also [Switching Between Buffers or Windows](#) for information about navigating between buffers and editor windows.

## Navigating in Pages and Files

The following commands control cursor navigation in pages and files:

- **top\_of\_window/bottom\_of\_window** (Ctrl+PgUp/Ctrl+PgDn) - Places the cursor at the top/bottom of the current editor window.
- **top\_of\_buffer/bottom\_of\_buffer** (Ctrl+Home/Ctrl+End) - The **top\_of\_buffer** command places the cursor at the first line and first column of the current buffer. The **bottom\_of\_buffer** command places the cursor at the end of the last line of the current buffer. If the option **Preserve column on**

**top/bottom** is on (**Tools > Options > General, [More Tab](#)**), the cursor is placed at the first line/last line of the buffer and the column position is unchanged.

**TIP** There is an option to make **top\_of\_buffer/bottom\_of\_buffer** push a bookmark, providing quick navigation between the top/bottom of the buffer and the previous location. See [Setting Bookmark Options](#) for more information.

- **top\_left\_of\_window/bottom\_left\_of\_window** - Places the cursor at the top left/bottom right of the current editor window.
- **page\_up/page\_down** (PgUp/PgDn) - Moves the cursor to the previous/next page of text.
- **page\_left/page\_right** - Changes the left edge scroll position by half the window width to the left/right. The cursor is moved half the window width to the left/right as well.

## Navigating in Statements and Tags

The following navigation commands are available for languages that support statement tagging:

- **next\_tag/prev\_tag** - Places the cursor on the next/previous tag definition, skipping any tags filtered out by the Defs tool window.
- **next\_proc/prev\_proc** - Places the cursor on the next/previous function heading.
- **find\_tag** - Displays a list of tags in the Select a Tag dialog, allowing you to pick the tag to which you want to navigate.
- **goto\_tag** - Prompts for a procedure tag name and places the cursor on the definition of the procedure name specified. This command is available in GNU Emacs emulation mode only.
- **end\_tag** - Places the cursor at the end of the current symbol definition. This is useful if you are in the middle of a large function or class definition and you want to jump to the end of it. In a class definition in C++, the end is where inline function definitions are usually stored.
- **end\_proc** - Moves the cursor to the end of the current procedure.
- **next\_statement/prev\_statement** - Moves the cursor to the beginning of the next/previous statement.
- **begin\_statement/end\_statement** - Places the cursor at the beginning/end of the current statement.
- **next\_sibling/prev\_sibling** - Moves the cursor to the beginning of the next/previous sibling. These are similar to the **next\_statement/prev\_statement** commands except they stay at one level of nesting.
- **goto\_parent** - Moves the cursor to the beginning of the enclosing statement or symbol scope relative to the current cursor position.
- **begin\_statement\_block/end\_statement\_block** - Moves the cursor to the beginning/end of the current statement block.

## Navigating between Words

To navigate between words, use the **next\_word** (Ctrl+right arrow) and **prev\_word** (Ctrl+left arrow) commands. The **next\_word** command moves the cursor to the beginning of the next word. The **prev\_word** command moves the cursor to the beginning of the previous word.

You can specify whether the cursor moves to the beginning or the end of the next/previous word. Choose **Tools > Options > General**, then select the [More Tab](#). Set the **Next word style** to **Begin** or **End**. This affects both **next\_word** and **prev\_word** commands.

## Navigating to a Specific Line

To view and place the cursor on a specific line number, from the main menu, choose **Search > Go to Line**. Enter the line number and click **OK**. Alternatively, you can use the **goto\_line** command in the syntax **goto\_line linenumber**.

## Navigating to an Offset

To seek to a byte offset in the current buffer, from the main menu choose **Search > Go to Offset**, or use the **gui\_seek** command. This function is the same as the C **lseek** function. However, if you have opened the file with tab expansion, the seek position on disk may be different.

When the Seek dialog appears, enter the position to seek for. You may specify a C syntax expression. In addition, you may prefix the expression with a plus or minus sign (+ or -) to specify a relative seek position.

Some examples are:

- **0x10+10** - Seek to offset 26
- **+8+4** - Seek to current offset + 12
- **-8+4** - Seek to current offset - 12

Select the **Decimal** option to enter the seek position in decimal number format. Select the **Hex** option to enter the seek position in hexadecimal number format. You can type an "x" as the first character in the **Position to seek for** text box and this option will automatically be selected.



## Symbol Browsing

---

SlickEdit® gives you the ability to browse and view symbols in your files or workspaces. Symbol browsing relies on Context Tagging®, so symbols are updated immediately or in the background as you edit. There are several tool windows that display information as you work to help you find what you need at exactly the time you need it:

- [Class Tool Window](#)
- [Current Context Tool Window](#)
- [Defs Tool Window](#)
- [Find Symbol Tool Window](#)
- [Preview Tool Window](#)
- [References Tool Window](#)
- [Symbols Tool Window](#)
- [Symbol Properties Tool Window](#)

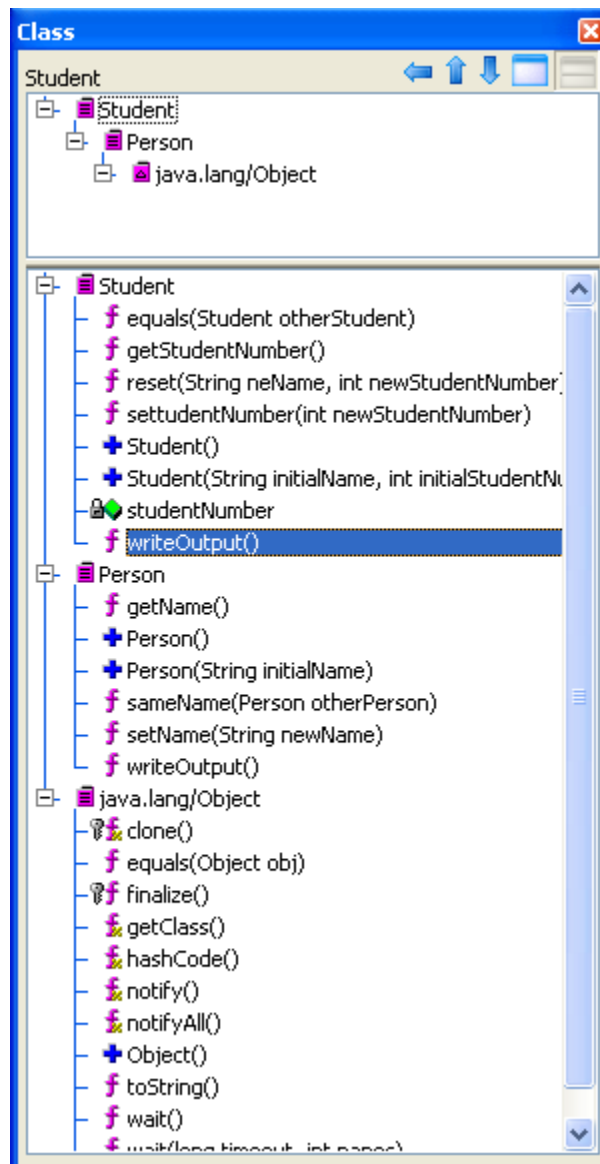
See also [Symbol Navigation](#) for information about how to navigate between symbols in files.

## Class Tool Window

**NOTE** The Class tool window is new in SlickEdit® 2007 and not to be confused with the tool window named “Classes” in previous versions. The formerly named “Classes tool window” has been renamed to “Symbols tool window”.

The Class tool window, docked as a tab on the left side of the editor by default, provides an outline view of both the members of the current class as well as any visible inherited members. This tool window also shows the inheritance hierarchy of the current class. This is useful for object-oriented programming languages such as Java.

Display of the Class tool window can be toggled on/off by selecting **View > Toolbars > Class** or by using the **toggle\_tbclass** command. To display the tool window on demand, use the **activate\_tbclass** command.



If you are coding within a class, the top pane (hierarchy pane) of the tool window shows the base class hierarchy for the current class. The bottom pane (members pane) shows all members of the current class, as well as all members visible from inherited superclass(es) and implemented interface(s). The name of the current class is displayed at the top of the tool window.

If you are not currently in a class (or enum or interface), the hierarchy pane is blank and the members pane shows the symbols in the current file. The name of the current file is displayed at the top of the tool window.

Hover the mouse over the bitmap of any item in the hierarchy or members panes to see a tooltip that shows the symbol's signature and scope.

To show or hide the hierarchy pane, use the two buttons located at the top-right of the tool window. If the hierarchy pane is hidden, the members pane is resized to take up the entire space of the window. Use the size bar to resize either pane.



Use the Up/Down buttons located to the left of the pane buttons to navigate up or down the class hierarchy. The Up arrow button will allow you to navigate to a child class (derived class or subclass) of the current class. The Down arrow allows you to navigate to a parent class (superclass or interface) of the current class. When using these buttons to navigate through code, the active buffer will switch to the destination class, and the hierarchy and members panes will update.

To jump to the definition of a class in the code, pushing a bookmark in the process, double-click on any member or class. Left-click or press Ctrl+Comma to go back.

## Filtering in the Hierarchy Pane

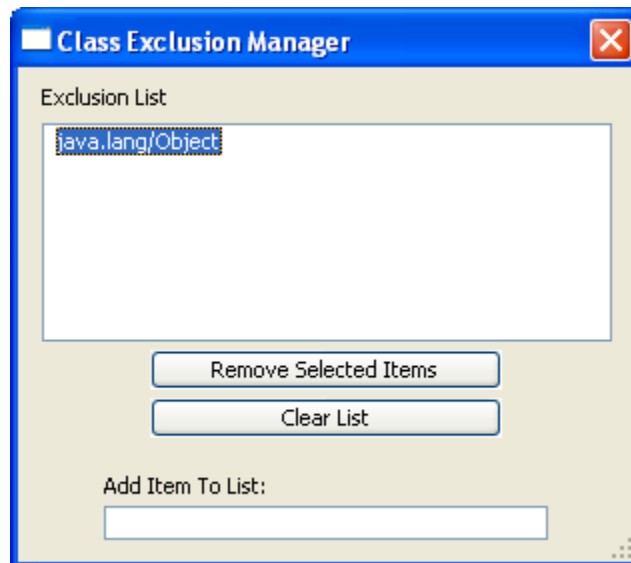
Right-click on a class in the hierarchy pane to display a list of filtering options. You can exclude entire namespaces or packages, anything above a certain level in the hierarchy, and anything outside of the current workspace. You can always include any class(es) you have excluded via the "Include" options.

By excluding a class or interface in the hierarchy view, the members of this class or interface are no longer displayed in the members pane, but they are still visible in the hierarchy as gray text.

Select **Show in Symbol Browser** to jump to the class in the symbol browser.

## Class Exclusion Manager

The Class Exclusion Manager, accessed by right-clicking on a class in the hierarchy pane, displays a list of any currently excluded classes, interfaces, namespaces, and packages. Exclusions are kept on a per-workspace basis.



To add an item to the list, type the name in the **Add Item To List** text box, then press **Enter**. Click the buttons to remove selected items or to clear the list.

## Filtering and Sorting in the Members Pane

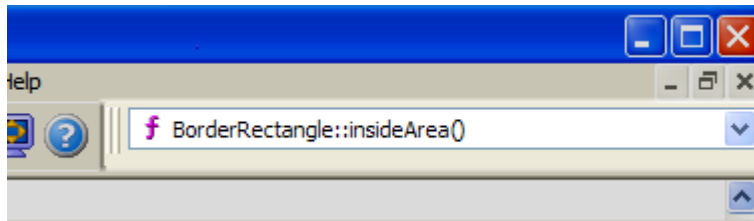
Right-click on a member in the members pane to access a list of filtering and sorting options as well as options for code navigation and modification. The following options are available:

- **C++ Refactoring** - Displays a menu of C/C++ refactoring options. See [C++ Refactoring](#) for more information.
- **Quick Refactoring** - Offers two Quick Refactorings: Rename and Modify Parameter List. See [Quick Refactoring](#) for more information.
- **Add Member Function, Add Member Variable, and Add Virtual Function** - (C/C++ only) When these options are selected for a class, you are prompted with a dialog to type a member function, member variable, or virtual function to be added into the source code at the top of the current class.
- **Organize imports** - (Java only) Organizes import statements in Java files. See [Organizing Java Imports](#) for more information.
- **Go to Tag** - Moves the cursor to the selected tag. See [Symbol Navigation](#) for more information.
- **References** - Brings the References tool window into focus, displaying the references for the symbol. See [References Tool Window](#) for more information.
- **Set Breakpoint** - Sets a debugging breakpoint. See [Setting Breakpoints](#) for more information.
- **Show in Symbol Browser** - Jumps to the member in the symbol browser. See [Symbols Tool Window](#) for more information.
- **Increase/Decrease Listed Members Limit** - Controls the number of members displayed in the members pane. When this option is selected, the command line will prompt you for a variable value. The default is 400.
- **Sort Classes By Hierarchy** and **Sort Classes By Name** - These options toggle the display of classes sorted either by hierarchy or alphabetically by name.
- **Sort Members By Line Number** and **Sort Members By Name** - These options toggle the display of members sorted either by line number or alphabetically by name.
- **Organize Members By Class** - Groups the members in the members pane by their class (or interface). When this option is selected, all “Sort” options are available. When this option is not selected, visible members in this pane will not be grouped at all. They will instead be displayed in one list, sorted by name.
- **Auto Expand All Top Level Classes** - Expands all top level class nodes in the members pane whenever the current class changes. The default behavior is to only auto-expand the node of the current class.
- **Auto Expand All Structs/Enums/Inner Classes** - Expands all struct, enum, and inner class nodes displayed in the members pane whenever the content is refreshed. By default this option is turned off, and these nodes are collapsed.
- **Quick Filters** and **Scope Filters** - Quick filters allow you to display only certain items in the members pane, such as functions, prototypes, etc. Scope filters allow you to display members only in certain scopes, such as public or global, private, protected, etc.

## Current Context Tool Window

Current Context displays the logical location of the cursor within your code. If it is within a class, it displays the class name. If it is within a function, it displays the function name. If the function is within a class, it displays the class and the function name.

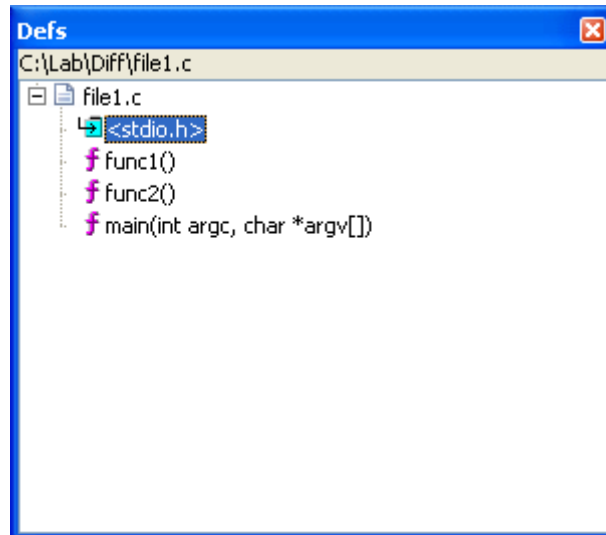
By default, the tool window is docked in the top upper-right section of the editor. Display can be toggled on/off by selecting **View > Toolbars > Current Context** or by using the `toggle_context` command. To display the tool window on demand, use the `activate_context` command.



## Defs Tool Window

The Defs (Definitions) tool window contains the defs (definitions) browser, which provides an outline view of symbols in the current file.

By default, the Defs tool window is docked as a tab on the left side of the editor. Display can be toggled on/off by selecting **View > Toolbars > Defs** or by using the **toggle\_defs** command. To display the tool window on demand, use the **activate\_defs** command.



The name of the file is displayed at the top of the tool window. Hover the mouse over the bitmap of any symbol in the window to see a tooltip that shows the symbol's signature and scope.

To jump to the definition of the symbol in the code, pushing a bookmark in the process, double-click on any symbol. Press Ctrl+Comma to go back.

## Defs Tool Window Options

Right-click on any symbol in the Defs tool window to access the following options:

- **C++ Refactoring** - Displays a menu of C/C++ refactoring options. See [C++ Refactoring](#) for more information.
- **Quick Refactoring** - Offers two Quick Refactorings: Rename and Modify Parameter List. See [Quick Refactoring](#) for more information.
- **Set Breakpoint** - Sets a debugging breakpoint. See [Setting Breakpoints](#) for more information.

- **Sort by Function Name** and **Sort by Line Number** - These options toggle the display of symbols sorted either alphabetically by function name or by line number.
- **Show Hierarchy** - Organizes symbols by their scope within the current file. Deselect this option to display all of the symbols in one flat list.
- **Show Statements** - (C/C++, Java, Visual Basic only) This option controls the [Statement Level Tagging](#) feature. When selected, the tool window shows an outline of all statements in each function within the current file. This allows you to see a primitive function flowchart or to navigate to a specific statement within a function.
- **Display Files** - Displays the names of the files that are open in the editor. Deselect this option to only show symbols in the current file, allowing you to use the window as a true outline view.
- **Auto Expand** - Automatically expands all levels within the current file. If this option is deselected, you will need to click to expand items manually.
- **Expand All** - Expands all symbols or levels in the current file.
- **Expand 1 Level** - Expands everything one level below the current symbol.
- **Expand 2 Levels** - Expands everything two levels below the current symbol.
- **Display Non-taggable Files** - Displays files that are open in the editor that are not taggable, such as text files.
- **Properties** - Displays the [Symbol Properties Tool Window](#), showing the properties of the selected item, such as visibility, whether it's static or final, etc. Note that you cannot use this window to change the properties.
- **Arguments** - Displays the return type and arguments for functions/methods in the [Symbol Properties Tool Window](#).
- **References** - Displays the list of references for the selected symbol, just as if you pressed Ctrl+/ in the editor window. See [Symbol Navigation](#) for more information.
- **Show Call Tree** - Displays a tree of symbols used by the selected symbol, for example, other functions called by the current function. See [Viewing Symbol Uses with the Calling Tree](#) for more information.
- **Contents** - Displays the following menu of save and print operations for the defs browser tree:
  - **Save** - Writes the items displayed in the defs browser to a text file, prompting you for a file name and directory location. The text file will then be displayed in the editor.
  - **Print** - Displays the Print dialog, where you can configure options for printing the tree.
  - **Save Subtree** and **Print Subtree** - These options function similarly to the above except they apply to the selected subtree.
- **Quick filters, Scope, Functions, Variables, Data Types, Statements, and Others** - All of these items are for filtering the data displayed in the Defs tool window.

## Find Symbol Tool Window

The Find Symbol tool window (**Search > Find Symbol** or **gui\_push\_tag** command) is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, Substring, or fast prefix match.

Searching for a symbol is faster than a normal text search because it is executed against the Context Tagging® database, rather than searching through your source files. Find Symbol also avoids false hits in

comments or string literals. Though [Syntax-Driven Searching](#) in the regular [Find and Replace Tool Window](#) provides this same capability, it cannot match the speed of Find Symbol.

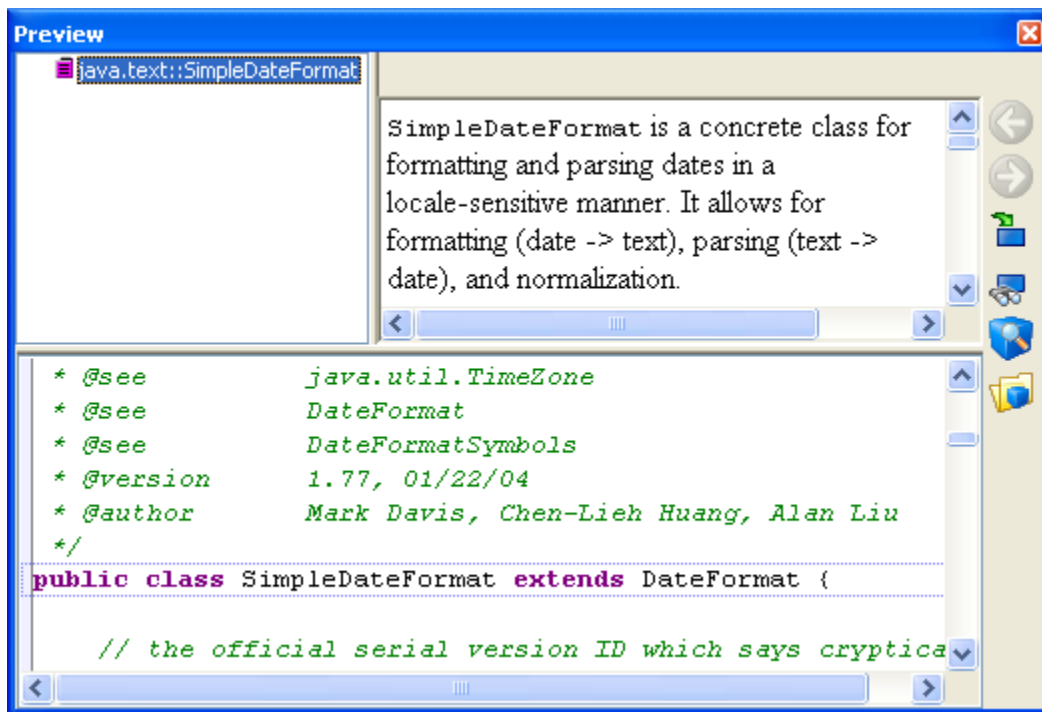
See [Find Symbol Tool Window](#) for information about the options that are available on the tool window.

## Preview Tool Window

**NOTE** In SlickEdit® 2007, the Preview tool window replaces the Symbol tool window found in previous versions. The Preview tool window does not have a search capability, so you should use the new [Find Symbol Tool Window](#) to search for symbols. This provides you with more power and more control over your symbol searching.

The Preview tool window provides a portal for viewing information in other files without having to open them in the editor. It automatically shows this information when you are working with certain features. See [Information Displayed by the Preview Window](#) for more information.

By default, the Preview window is docked as a tab at the bottom of the editor. Display can be toggled on/off by selecting **View > Toolbars > Preview** or by using the **toggle\_preview** command. To display the tool window on demand, use the **activate\_preview** command.



The Preview tool window contains the following components:

- **Symbol list** - This is the list of all symbols which are currently being previewed. In most cases, this is a single symbol. In some cases, such as for the symbol under the cursor, multiple matches are shown, such as the definition and declaration of a symbol. You can do a few things with the symbol list:
  - Hover the mouse over the bitmap of any item to see a tooltip that shows the symbol's signature and scope.

- Click on any symbol to preview that specific symbol or its comments.
- Right-click to adjust symbol search filtering options.
- Double-click to jump to a symbol. Press Ctrl+Comma to go back.
- You can create key bindings for the **preview-next** and/or **preview-prev** commands in order to scroll through the items in the symbol list without using your mouse. See [Creating Bindings](#) for more information.
- **File and line label** - Shows the file name and line number of the selected symbol.
- **Documentation comments pane** - This pane displays any existing comments for the symbol that is selected in the symbol list. If the comments are in Javadoc or XMLdoc format, they will be formatted in HTML. You can single-click on hypertext links within the comments to follow the links, such as “See also” sections.
- **Editor preview window** - Shows the contents of the actual source file at the line number of the selected symbol. Double-click to open the code in the editor. Right-click to adjust symbol search filtering options.
- **Size bars** - Use the size bars to adjust the width of the symbol list and/or the height of the documentation comments area.
- **Buttons** - The following buttons are found along the right edge of the Preview window:
  - **Back and Forward** - Allow you to navigate among the hypertext links that you have traversed in the documentation comments.
  - **Go to definition** - Opens the selected symbol in the editor.
  - **Go to reference** - Finds references to the selected symbol.
  - **Show in symbol browser** - Locates the selected symbol in the [Symbols Tool Window](#) (formerly known as the Classes tool window).
  - **Manage Tag Files** - Opens the [Context Tagging® - Tag Files Dialog](#) for building and maintaining tag files for indexing symbol information.

## Information Displayed by the Preview Window

The table below describes what the Preview window displays under different circumstances.

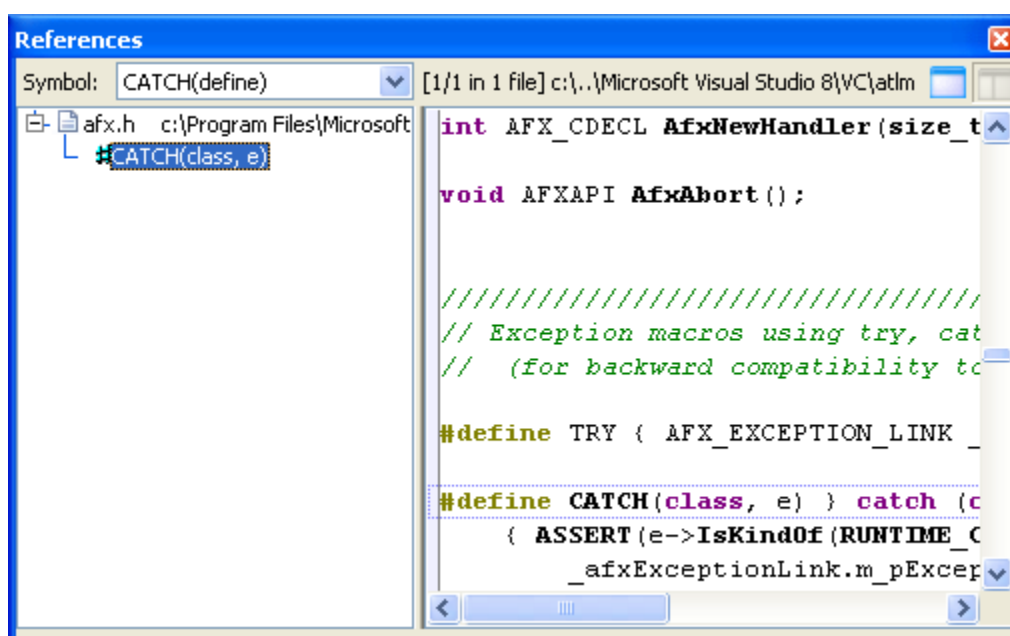
Editor Element in Use	Preview Window Display
Any source file open in the editor	The Preview window shows the definition or declaration of the symbol under the cursor, along with the symbol's documentation comments, if any exist.
The Defs, Symbols, Class, Current Context, and Find Symbol tool windows	Single-click on a symbol and the Preview window displays the selected symbol and its documentation comments, if any exist. See <a href="#">Defs Tool Window</a> , <a href="#">Symbols Tool Window</a> , <a href="#">Class Tool Window</a> , <a href="#">Current Context Tool Window</a> , and <a href="#">Find Symbol Tool Window</a> for more information.
Call Tree dialog and References tool window	The Preview window shows the location of the symbol references or use.

Editor Element in Use	Preview Window Display
The Base Classes and Derived Classes symbol browser dialogs	Single-click on a symbol and the Preview window displays the selected symbol and its documentation comments, if any exist. See <a href="#">Symbols Tool Window</a> for more information.
The Bookmarks tool window	Single-click on a bookmark and the Preview window displays the location of the bookmark. See <a href="#">Bookmarks Tool Window</a> for more information.
The Breakpoints tool window	Single-click on a breakpoint and the Preview window displays the location of the breakpoint. See <a href="#">Setting Breakpoints</a> for more information.
The Search Results tool window	Single-click on a line in the Search Results window and the Preview window displays the location of the selected search result. See <a href="#">Search Results Output</a> for more information.
List Members and Auto-Complete results	Cursor up or down through the list of items in auto-complete or list-members results and the Preview window displays the location of the selected symbol and its documentation comments, if any exist. See <a href="#">List Members</a> and <a href="#">Auto-Complete</a> for more information.

## References Tool Window

The References tool window displays the list of symbol references (uses) found the last time that you used the Go to Reference feature (Ctrl+/ or **push\_ref** command—see [Symbol Navigation](#) for more information).

By default, the References window is docked as a tab at the bottom of the editor. Display can be toggled on/off by selecting **View > Toolbars > References** or by using the **toggle\_refs** command. To display the tool window on demand, use the **activate\_refs** command.



The References tool window automatically comes into focus when you use the Go to Reference feature or when you select **References** from the right-click menu of the Class, Defs, or Symbols tool window.

**NOTE** Typically, you only want to view references that occur in project files, and not run-time libraries, which can be very large. For this reason, references are not generated automatically for run-time library tag files. If you want to view references that occur in a run-time library tag file, you need to generate references for the tag file. To do this, display the [Context Tagging® - Tag Files Dialog](#) (**Tools > Tag Files** or **gui\_make\_tags** command), choose the tag file, right-click to display the context menu, and select **Generate References**. See [Tagging Run-Time Libraries](#) for more information.

The left pane displays a tree view of the files and locations that contain the symbol references. Hover the mouse over the bitmap of a symbol to see a tooltip that shows the symbol's signature and scope. To jump to the location of a symbol reference in the code, pushing a bookmark in the process, double-click on it. Press Ctrl+Comma to go back.

The right pane displays a preview of that location in the source. The number of instances found and the file name and line number are displayed at the top. Use the size bar to resize either pane.

Use the buttons located at the top right corner of the tool window to toggle the preview pane on and off. Because source can also be previewed in the [Preview Tool Window](#), you may find it more efficient to use the References window with the preview pane off.

## References Tool Window Options

Right-click on a symbol or file in the left pane of the References window to display the following options:

- **Contents** - Displays the following menu of save and print operations for the references browser tree:
  - **Save** - Writes the items displayed in the references browser to a text file, prompting you for a file name and directory location. The text file will then be displayed in the editor.
  - **Print** - Displays the Print dialog, where you can configure options for printing the tree.



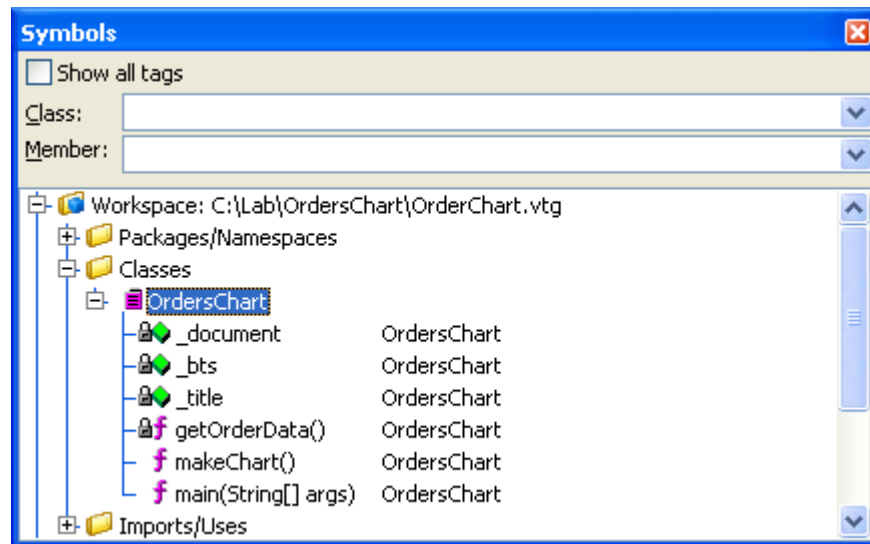
- **Save Subtree** and **Print Subtree** - These options function similarly to the above except they apply to the selected subtree.
- **Quick filters, Scope, Functions, Variables, Data Types, Statements, and Others** - All of these items for filtering the data displayed in the References tool window.

## Symbols Tool Window

**NOTE** In SlickEdit® 2007, the Symbols tool window replaces the Classes tool window found in previous versions. A new [Class Tool Window](#) is available.

The Symbols tool window contains the symbol browser, which lists symbols from all of the tag files.

By default, the Symbols tool window is docked as a tab on the left side of the editor. Display can be toggled on/off by selecting **View > Toolbars > Symbols** or by using the **toggle\_symbols** command. To display the tool window on demand, use the **activate\_symbols** command.



The top part of the window contains an option and combo boxes that are used for filtering. The bottom part of the window lists the symbols grouped by category. Symbols in your workspace are listed in the top group labeled "Workspace." The rest of the symbols are grouped by language or compiler.

Hover the mouse over the bitmap of a symbol to see a tooltip that shows the symbol's signature and scope. To jump to the definition of a symbol in the code, pushing a bookmark in the process, double-click on any symbol. Press Ctrl+Comma to go back.

## Filtering Symbols in the Symbols Tool Window

The symbols listed in the symbol browser can be filtered using the **Class** and **Member** combo boxes. The **Class** combo box filters the items listed under the Classes folder. The **Member** combo box filters the items listed under any displayed classes or under any of the other folders, like Global Variables, Static Variables, Defines, etc. Enter multiple words in either combo box to search for items containing either word.

For example:

- Enter **person** into the **Class** combo box to find all classes containing the word "person".

- Enter **person manager** into the **Member** combo box to find all members, variables, etc. containing the word “person” or “manager”.

### NOTES

- The filters are case-sensitive, so be sure to type the values in the same case.
- The items listed under the Classes folder are global classes that are not part of a namespace or package.

To clear the filters and see all items again, select the **Show all tags** option.

For non-object-oriented languages, use the **Member** combo box to search, since there are no classes. You can hide the combo boxes to save space by right-clicking and selecting **Filters**, then unchecking the corresponding check box.

## Symbols Tool Window Options

Right-click on a symbol in the Symbols tool window to access the following additional filtering options as well as code management options:

- **Go to Definition** - Moves the cursor to the symbol's definition (proc). See [Symbol Navigation](#) for more information.
- **Go to Declaration** - Moves the cursor to the symbol's declaration (proto). See [Symbol Navigation](#) for more information.
- **Quick Refactoring** - Offers two Quick Refactorings: Rename and Modify Parameter List. See [Quick Refactoring](#) for more information.
- **Organize imports** - (Java only) Organizes import statements in Java files. See [Organizing Java Imports](#) for more information.
- **Set Breakpoint** - Sets a debugging breakpoint. See [Setting Breakpoints](#) for more information.
- **Find Tag** - Searches for symbols and displays them in the symbol browser. Note that the Find Symbol tool window also provides this functionality.
- **Manage Tag Files** - Displays the [Context Tagging® - Tag Files Dialog](#) for use in managing your tag files.
- **Expand** and **Collapse** options - Expands/collapses symbols as specified.
- **Sort by** - Sorts symbols displayed by tag name, line number, or containers to top, which puts classes, structs, etc. at the top of the list.
- **Filters** - Filter by class or member, or select **Filtering Options** to display the Symbol Browser Filter Options dialog. See [Symbol Browser Filter Options](#) for information on the available options.
- **Contents** - Displays the following menu of save and print operations for the symbol browser tree:
  - **Save** - Writes the items displayed in the symbol browser to a text file, prompting you for a file name and directory location. The text file will then be displayed in the editor.
  - **Print** - Displays the Print dialog, where you can configure options for printing the tree.
  - **Save Subtree** and **Print Subtree** - These options function similarly to the above except they apply to the selected subtree.
- **Base Classes** - Displays the Base Classes dialog, which shows a list of base classes for the selected class on the left with the list of that class's members on the right. Base classes are displayed in a tree view, allowing you to explore up the inheritance hierarchy. See [Viewing Base and](#)

[Derived Classes](#) for more information. Note that the [Class Tool Window](#) provides this same functionality.

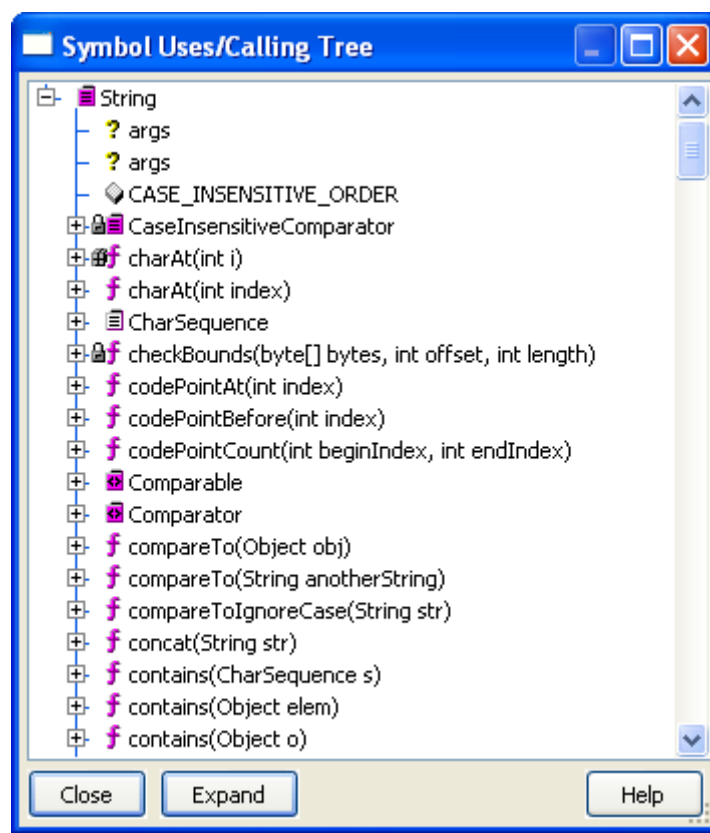
- **Derived Classes** - Displays the Derived Classes dialog, which works the same as above but for derived classes. See [Viewing Base and Derived Classes](#) for more information.
- **Properties** - Displays the [Symbol Properties Tool Window](#), showing the properties of the selected item, such as visibility, whether it's static or final, etc. Note that this window is read-only, so you can't use it to change the properties.
- **Arguments** - Displays the return type and arguments for functions/methods in the [Symbol Properties Tool Window](#).
- **References** - Displays the list of references for the selected symbol in the [References Tool Window](#), just as if you pressed Ctrl+/ in the editor window. See [Symbol Navigation](#) for more information.
- **Calls/Uses** - Displays a tree of symbols that are used by this symbol or called by this function. See [Viewing Symbol Uses with the Calling Tree](#) for more information.

## Viewing Symbol Uses with the Calling Tree

View symbol uses to see what symbols (variables, functions, methods, classes, etc.) are used by a specific function or method.

To view the symbols that a particular function or method uses, first create a project or open an existing project. Then from the Symbols tool window, right-click on the desired function or method and select **Calls or uses**. The Symbol Uses/Calling Tree dialog will be displayed.

**TIP** You can also access the Symbol Uses/Calling Tree from within the [Defs Tool Window](#) by right-clicking on a symbol and selecting **Show Call Tree**.

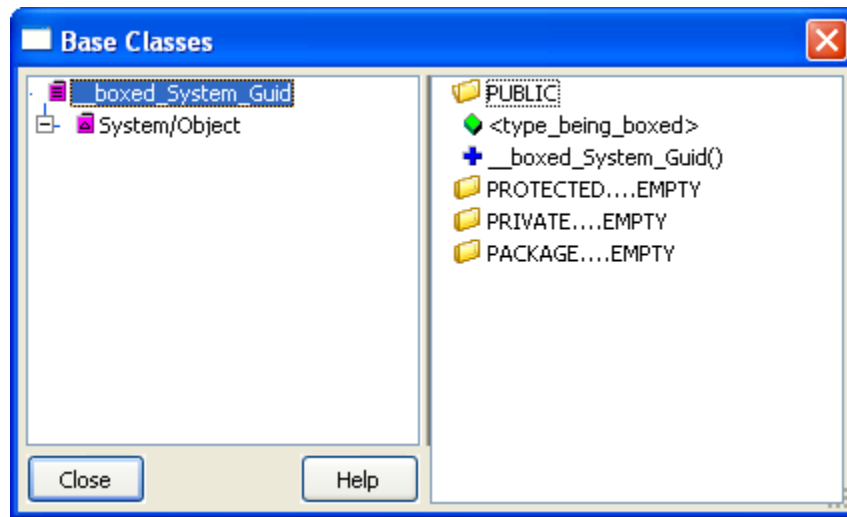


Right-click in this tree to display/modify the symbol filters. Items in the tree can be expanded to view uses recursively. Double-click or press the spacebar on an item in the tree list to go to an item. Double-click and Space are the same except when the item is a prototype that has a corresponding code section. Double-clicking will then go to the prototype's corresponding code section.

If the focus is in the Symbol Uses/Calling Tree dialog, the selected item will be shown in the [Preview Tool Window](#) tool window, just as it is in the [Symbols Tool Window](#).

## Viewing Base and Derived Classes

To see what classes are inherited by a particular class, right-click on the class in the Symbols tool window and select **Base Classes**. To see what classes are derived from a particular class, right-click on the class in the Symbols tool window and select **Derived Classes**. Both dialogs have the same interface.



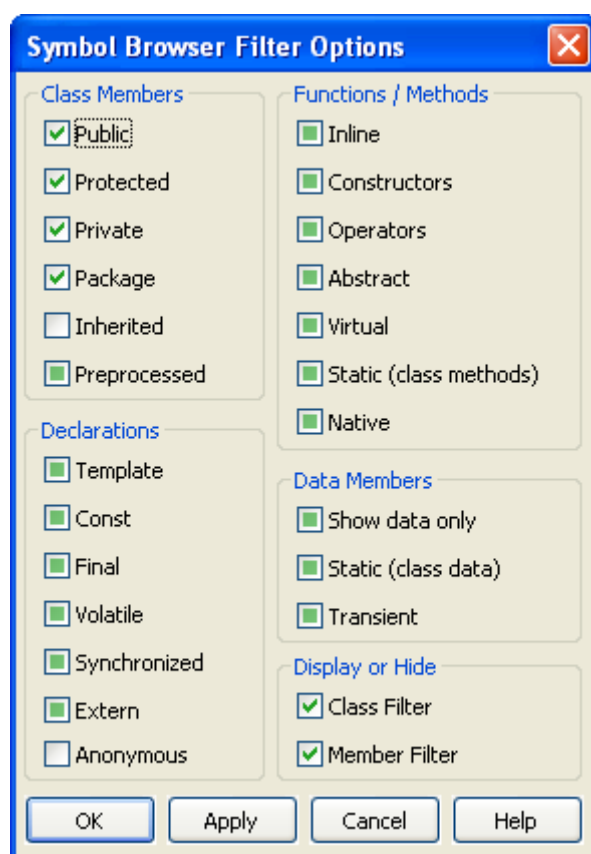
The left pane of each dialog contains a tree showing the class inheritance hierarchy (the class list). The right pane shows a list of the members of the selected class (the member list).

If the focus is in the class list, the selected class will be displayed in the member list, if it can be resolved. If the focus is in the member list, the selected item will be shown in the Preview window, and is the name as it appears within the class definition.

To jump to the symbol in the code, pushing a bookmark in the process, double-click on a symbol in either pane. Press Ctrl+Comma to go back. Right-click on a symbol for filtering options.

## Symbol Browser Filter Options

To access symbol browser filter options, right-click in the Symbols tool window and select **Filters > Filtering Options**.



Each option has three states: If the option is selected, only the specified items will be displayed. If the option is deselected, the specified item will not be displayed. If the option is in a neutral state, the item will not be considered in the filter.

The following options are available:

- **Class Members**
  - **Public** - When selected, public members are displayed.
  - **Protected** - When selected, protected members are displayed.
  - **Private** - When selected, private members are displayed.
  - **Package** - (Java only) When selected, package members are displayed. Java members have package scope if they do not specify public, protected, or private.
  - **Inherited** - When selected, only inherited members that this class can access are displayed. When unselected, only members of this class are displayed.
  - **Preprocessed** - When selected, only members expanded by pre-processing are displayed. This is specifically useful for MFC classes. When unselected, only non-preprocess members displayed.
- **Declarations**
  - **Template** - (C++ only) When selected, only template classes are displayed. When unselected, only non-template classes are displayed.

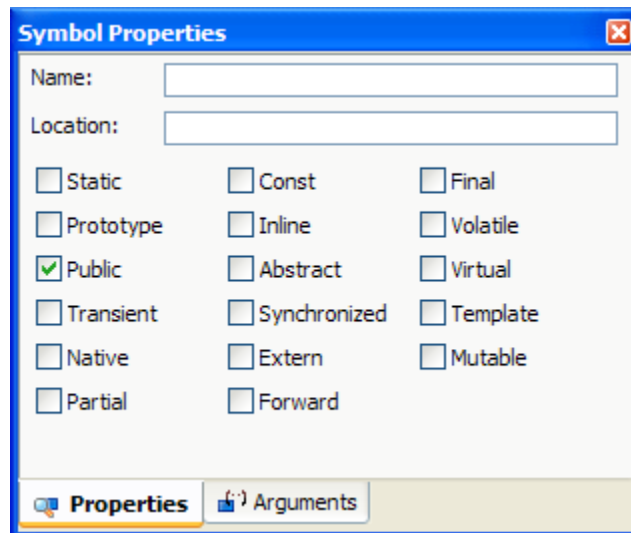
- **Const** - (C++ only) When selected, only methods which do not modify members (**method1() const**) are displayed. When unselected, only non-const methods are displayed.  
Use the [Symbol Properties Tool Window](#) (right-click in the Symbols tool window and choose **Arguments**, or from the main menu choose **View > Toolbars > Symbol Properties**) to view other **const** information for declarations (for example, **int const \* const \*pcpcvariable;**).
- **Final** - (Java only) When selected, only final members are displayed. When unselected, only non-final members are displayed.
- **Volatile** - (C++ only) When selected, only volatile method members (**method1() volatile**) are displayed. When unselected, only non-volatile members are displayed.
- **Synchronized** - (Java only) When selected, only synchronized members are displayed. When unselected, only non-synchronized members are displayed.
- **Extern** - When selected, only identifiers defined explicitly using the **extern** keyword are displayed. When unselected, only identifiers defined which do not explicitly use the **extern** keyword are displayed.
- **Anonymous** - When selected, only class names which are automatically generated by Context Tagging® are displayed. When unselected, only explicitly named classes are displayed.
- **Functions/Methods**
  - **Inline** - When selected, inline functions or methods are displayed.
  - **Constructors** - When selected, constructors are displayed.
  - **Operators** - When selected, overloaded operators are displayed.
  - **Abstract** - When selected, only abstract methods are displayed. When unselected, only non-abstract methods are displayed.
  - **Virtual** - When selected, only virtual methods are displayed. When unselected, only non-virtual methods are displayed. All non-static Java methods are implicitly virtual.
  - **Static (class methods)** - When selected, only static methods are displayed. When unselected, only non-static methods are displayed.
  - **Native** - When selected, only methods explicitly defined with the native keyword are displayed. When unselected, only non-native methods are displayed.
- **Data Members**
  - **Show data only** - When selected, only data members are displayed. When unselected, only methods are displayed.
  - **Static (class data)** - When selected, only static data members are displayed. When unselected, only non-static data members are displayed.
  - **Transient** - (Java only) When selected, only transient data members are displayed. When unselected, only non-transient data members are displayed.
- **Display or Hide**
  - **Class Filter** - When selected, the class filter is displayed in the Symbols tool window.
  - **Member Filter** - When selected, the member filter is displayed in the Symbols tool window.

## Symbol Properties Tool Window

The Symbol Properties tool window displays detailed information (properties and arguments) for the symbol at the cursor location. Note that this window is read-only, so you can't use it to change the properties.

## SYMBOL BROWSING

Display can be toggled on/off by selecting **View > Toolbars > Symbol Properties**. To display the tool window on demand, right-click on a symbol in the Symbols tool window and select **Properties** or **Arguments**, or use the `activate_tag_properties_toolbar` command.





# Text Editing

---

SlickEdit® provides familiar operations for selecting, copying, moving, and operating on text, with enhanced capabilities to meet the needs of developers. This section also describes how to work with lines, sort text, and insert literal characters into your text.

## Selections

SlickEdit® supports three kinds of selections: **character**, **line**, and **block**. Each provides different capabilities for different editing situations, and easy access for those who like to work using the keyboard only or the mouse. For a table of keyboard shortcuts, see [Selection Keys](#).

Selected text is rendered with a shaded background. To change the selection color, use the Color Settings dialog box (**Tools > Options > Color**). See [Setting Colors for Screen Elements](#) for more information on changing colors.

**TIP** When using the mouse for selections, you can switch the selection type from character to block or to line. While the left button is down, click the right button to toggle through the selection types.

## Character Selections

Character selections are used to select words, parts of a line, or a range of text between a starting location and an ending location. To create a character selection, use one of the following methods:

- **Keyboard method** - Position the cursor at the beginning of the text to be selected. Next, enter the **select\_char** command (F8 or **Edit > Select > Char**). Then, move the cursor to the end of the text to be selected by using the arrow keys (or by using the mouse).
- **Cursor method** - Press and hold Shift with any cursor key, including PgUp, PgDn, Home, and End, to create a quick character selection. For example, Shift+Home will create a character selection from the cursor position to the beginning of the line. Shift+End will create a selection from the cursor to the end of the line. You can also use Ctrl+Shift+Home to create a character selection from the cursor position to the top of the file, or Ctrl+Shift+End to create a selection from the cursor to the end of the file.
- **Mouse method** - Left-click and drag the text to be selected, or double-click to select a whole word. To extend a selection to the cursor, hold down the Shift key and left-click.

**NOTE** The key bindings are different if using the Vim emulation. See the Vim emulation chart (located in the `/docs` subdirectory of your installation directory) for a listing of selection keys.

## Block Selections

Block selections, also known as column selections, are used to process columns of text. To select a block, use one of the following methods:

- **Keyboard method** - Position the cursor at the beginning of the text to be selected. Next, enter the **select\_block** command (Ctrl+B or **Edit > Select > Block**). Then, move the cursor to the end of the block to be selected by using the arrow keys (or by using the mouse).
- **Mouse method** - Right-click and drag the text to be selected.

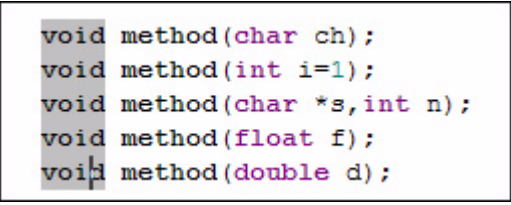
## Editing a Block of Text: Block Insert Mode

Block insert mode is useful when you need to edit a block of text instead of just copying or deleting it. Additionally, when in this mode, characters you type, as well as other edits (such as backspacing and deleting), apply to the entire block/column selection.

After a block selection is created, you can enter block insert mode by simply typing some characters to insert, or by entering the **block\_insert\_mode** command (**Edit > Other > Block Insert Mode**). If the block selection is more than one column wide, then the initial block selection will be deleted when you type the first character. This mode also supports use of the keys Tab, Shift+Tab, and Backspace.

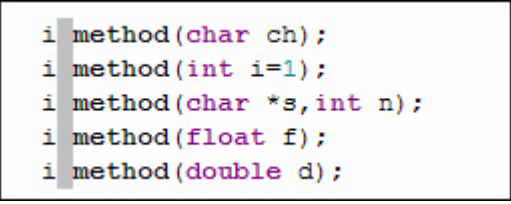
To cancel out of block insert mode, press the **Esc** key.

The figure below shows an example of a block selection created by right-clicking and dragging to select a block. Notice the cursor position.



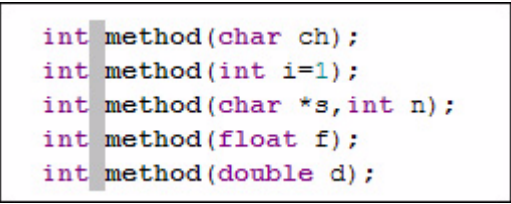
```
void method(char ch);
void method(int i=1);
void method(char *s, int n);
void method(float f);
void method(double d);
```

The figure below shows how the above example changes when you type “i” at the cursor while the block is selected.



```
i method(char ch);
i method(int i=1);
i method(char *s, int n);
i method(float f);
i method(double d);
```

The figure below shows how the original example changes when you type “int” at the cursor while the block is selected.



```
int method(char ch);
int method(int i=1);
int method(char *s, int n);
int method(float f);
int method(double d);
```

## Line Selections

A line selection is any selection that includes one or more complete lines of text. Line selections are usually the best way to edit lines of code, and they also work with SmartPaste®, which reindents pasted lines according to the surrounding code (see [SmartPaste®](#)). To select lines, use one of the following methods:

- **Keyboard method** - Position the cursor at the beginning of the line to be selected. Next, enter the **select\_line** command (Ctrl+L or **Edit > Select > Line**). Then, move the cursor to the last line to be selected by using the arrow keys (or by using the mouse).
- **Mouse method** - To select the current line, triple-left-click within a line (or choose **Edit > Select > Line**). To select multiple lines, left-click and drag in the space between the left edge of the buffer and the edit window border. The mouse cursor changes to point to the upper right instead of the upper left.

## Selection Keys

The following table contains the default keyboard shortcuts (CUA emulation) for selection functions.

Keys	Function
Ctrl+U	Deselect text.
F8	Start character selection.
Ctrl+L	Start line selection.
Ctrl+B	Start block or column selection.
Shift+Right Arrow	Start or extend selection to right.
Shift+Left Arrow	Start or extend selection to left.
Shift+Up Arrow	Start or extend selection up one line.
Shift+Down Arrow	Start or extend selection down one line.
Shift+PgUp	Start or extend selection up one page.
Shift+PgDn	Start or extend selection down one page.
Ctrl+Shift+Home	Start or extend selection to top of buffer.
Ctrl+Shift+End	Start or extend selection to bottom of buffer.
Ctrl+X	Cut selected text.
Ctrl+C	Copy selected text to clipboard.
Ctrl+V	Paste clipboard.
Ctrl+Shift+V	List clipboards.

Different selection styles found in non-CUA compliant editors, such as Brief, are also provided. To use a different selection style, use the [Selections Tab](#) of the General Options dialog box (**Tools > Options > General**).

## Modifying Selected Text

After you select text, you can invoke a key or type a command that modifies the selected text. Use the information in the following table to assist you when making modifications to selected text. When two com-

mands are displayed, the first command is the command line version of the command, and the second is the graphical version of the command.

Command	Key Combination or Menu Item	Description
<b>mou_click_copy</b>	Ctrl+L	Drags and copies the selected text. Click within the selected text and hold the left button down while moving the mouse to a new location. Line selections are inserted after the current line by default. If you want line selections inserted before the current line, change the line insert style. To access the line insert style, select <b>Tools &gt; Options &gt; General &gt; <a href="#">More Tab</a> &gt; Line insert style</b> .
<b>mou_move_to_cursor</b>	Ctrl+R	Moves the selected text to the cursor.
<b>mou_copy_to_cursor</b>	Ctrl+Shift+R	Copies the selected text to the cursor.
<b>mou_click L</b>	L	Drags and moves the selected text. You must click within the selected text, and keep the left button down while moving the mouse to a new location.
<b>list_clipboards</b>	Ctrl+Shift+V	Allows you to select a clipboard from a list of the most recently used (15 is the default maximum) clipboards to insert at the cursor.
<b>copy_to_clipboard</b>	Ctrl+C	Copies selected text to the clipboard.
<b>append_to_clipboard</b>	Ctrl+Shift+C	Appends selected text to the clipboard.
<b>append_cut</b>	Ctrl+Shift+X	Deletes the selected text and appends it to the clipboard.

Command	Key Combination or Menu Item	Description
<b>copy_to_cursor</b>	None	Copies selected text to the cursor. Line selections are inserted after the current line by default. If you want line selections inserted before the current line, change the line insert style. You can change the line insert style by clicking <b>Tools &gt; Options &gt; General</b> . Then, click the <a href="#">More Tab</a> , then set the <b>Line insert style</b> to <b>Before</b> .
<b>fill_selection,</b> <b>gui_fill_selection</b>	Edit > Fill	Fills the selected text with a character.
<b>indent_selection</b>	Tab	Indents the selected text by the indent for each level or by one tab stop. Indenting will be with tab or space characters depending upon whether the Indent with tabs check box is on.
<b>unindent_selection</b>	Shift+Tab	Unindents each line of selected text by the indent for each level or by one tab stop. Not all editing modes have an indent for each level.
<b>lowcase_selection</b>	Edit > Other > Lowcase	Converts the selected text to lowercase.
<b>upcase_selection</b>	Edit > Other > Upcase	Converts the selected text to uppercase.
<b>shift_selection_left</b>	Shift+F7	Shifts the text within the selection to the left. Supports the line and block text selection methods. If a character selection is used, it is converted to a line selection.
<b>shift_selection_right</b>	Shift+F8	Shifts the text within the selection to the right. Supports line and block selections. If a character selection is used, it is converted to a line selection.

Command	Key Combination or Menu Item	Description
<b>overlay_block</b>	Edit > Other > Overlay block	Overlays selected text at the cursor. Supports block selection only.
<b>adjust_block</b>	Edit > Other > Adjust block	Overlays selected text at cursor and fills the source selected text with blanks. Supports block selection only.
<b>reflow_selection</b>	Source > Format Selection	Word-wraps the text within the selected area. Line selections are word-wrapped with current margin settings. Block selections are word-wrapped within the columns of the block. Character selection is not supported.
<b>execute_selection</b>	Alt+=	Executes each line or sub-line of the selected text as if entered on the command line.
<b>put filename, gui_write_selection</b>	None	Writes the selected text to filename. Use <b>File &gt; Insert a File</b> to insert a file.
<b>append filename, gui_append_selection</b>	None	Appends the selected text to the file filename.
<b>sort_within_selection</b>	None	Sorts lines within a selected area.
<b>sort_on_selection</b>	None	Sorts lines of text based on text within columns specified.
<b>add</b>	None	Adds selected text and inserts result below the last line of the selection. Addition is performed for each adjacent line. If no operator exists between two adjacent numbers, addition is assumed.
<b>align_selection_left</b>	None	For block selections only (see <a href="#">Block Selections</a> ), aligns the text enclosed in the selection to the left edge of the selection.

Command	Key Combination or Menu Item	Description
<b>align_selection_center</b>	None	For block selections only (see <a href="#">Block Selections</a> ), centers the text enclosed in the selection.
<b>align_selection_right</b>	None	For block selections only (see <a href="#">Block Selections</a> ), aligns the text enclosed in the selection to the right edge of the selection.

## Adding Numbers to a Selection: Enumeration

You can automatically add incrementing numbers to a selection of code by using the **enumerate** command. To configure enumeration properties, from the main menu choose **Edit > Other > Enumerate**. For a list of the options that are available on the Enumerate dialog, see [Enumerate Dialog](#).

## Counting Selected Lines and Characters

SlickEdit automatically counts the number of lines and characters in a selection. This is useful to measure the length of a word or string, or the number of lines in a function. An indicator, located in the bottom edge of the editor, displays the following information based on your current selection:

- When nothing is selected, the indicator is dimmed and displays the text, “No Selection.”
- When the current selection is a **character selection**:
  - If the character selection is contained on one line, the indicator displays the number of columns selected.

**NOTE** Because columns are “virtual,” the number of columns displayed by the indicator is not necessarily the actual number of characters or bytes in the selection, if the selection includes tab characters, unicode characters, or extends beyond the end of the line.

- If the character selection spans more than one line, the indicator shows the number of lines, with a plus sign (+) to indicate if there are “extra” characters selected, or a minus sign (-) to indicate if there are fewer characters selected, depending on the start and end columns of the selection.
- When the current selection is a **line selection**, the indicator displays the number of lines.
- When the current selection is a **block selection**, the indicator displays the size of the block in the format **Lines x Columns**.

Click on the indicator, or use the **select\_toggle** command on the SlickEdit command line, to create successively larger common selections. For example, if you have a character selection, you can click on the indicator or use **select\_toggle** to extend the selection to include the entire word. Selections are cycled in the following order, starting with no selection:

1. Create empty character selection
2. Select current word
3. Select current line

4. Select current code block
5. Select larger code block
6. Select current function
7. Select entire file
8. Deselect

Except for empty character selections and line selections, the selections are locked so that the cursor remains stationary.

## Setting Selection Options

Many options are available for setting your selection preferences. To access these options, from the main menu, choose **Tools > Options > General**, then select the [Selections Tab](#).

# Cutting, Copying, and Moving Text

## Dragging and Dropping

Selected text can be copied or moved by dragging and dropping the selected text using the left mouse button. To set this functionality, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Allow drag drop text**.

## Using Clipboards

Use clipboards to copy or move text among files that are being edited. This includes files that are being edited, the command line, a dialog text box, or another application that supports text clipboards, such as a word processor. When using a cut or copy command, a clipboard is created. Insert this clipboard back into the buffer by pressing Ctrl+V.

Press Ctrl+K to copy the current word to the clipboard. Then use Ctrl+V to paste it anywhere. Pressing Ctrl+K multiple times in succession creates one clipboard. All clipboard-related commands are available on the Edit drop-down menu.

To move text between clipboards, complete the following steps.

1. Go to the beginning of a line with some text and press **Ctrl+E** to erase the text to the end of that line.
2. Press **Ctrl+V** to paste the text back in.
3. Move the cursor somewhere else in the buffer and press **Ctrl+V**. You have just made a copy of a line of text without selecting anything first.

Pressing the same cut key multiple times in succession creates one clipboard. If you press Ctrl+Shift+K three times to cut three words, one clipboard is created that you can insert with Ctrl+V. This is true for cut line (Ctrl+Backspace) and erase to end of line (Ctrl+E) as well.

**TIP** If using Brief emulation and a clipboard is wanted when cutting text, bind the commands **cut\_word**, **cut\_end\_line**, and **cut\_line** to the appropriate keys.

A stack of the last 15 (default maximum) of the most recently used clipboards is kept. You can change the maximum number of clipboards in the General Options dialog box (**Tools > Options > General**, select the



[General Tab](#)). To see a list of clipboards you have created, press Ctrl+Shift+V (**Edit > List Clipboards** or **list\_clipboards** command). The Select Text to Paste dialog box appears.

The numbers on the far left are used to help move the selection cursor. The number following the clipboard type indicates the number for complete or partial lines of text in the clipboard.

Only clipboards of one line can be inserted into the command line or text box. Both Ctrl+V and Ctrl+Shift+V key sequences insert clipboard text into the text area or the command line. The result of inserting a clipboard into the text area varies depending on the clipboard type.

A LINE type clipboard is inserted after the current line by default. If you want LINE type clipboards inserted before the current line, change the line insert style (**Tools > Options > General, [More Tab](#), Line Insert Style**). A BLOCK type clipboard is inserted before the current character and pushes over all text intersecting with the block. No lines are inserted. A CHAR type clipboard is inserted before the current character.

Use the **View** button to look at the complete text for the selected condensed clipboard. While viewing the clipboard, you can copy all or part of it to the operating system clipboard.

## Setting the Maximum Number of Clipboards

By default, a stack of your last 50 clipboards are kept, any one of which can be pasted with Ctrl+Shift+V. To change the maximum number of clipboards saved, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Enter a value in the **Max clipboards** spin box.

## Working with Lines

There are several options for working with lines as described below.

### Clicking Past the End of a Line

To have the ability to place the cursor past the end of a line, from the main menu, choose **Tools > Options > General**, then select the [General Tab](#). Select the option **Click past end of line**.

### Highlighting the Current Line

The current line can be highlighted by having a dotted box drawn around it, making the cursor easier to see. You can choose to make the highlight a box only, or you can choose a box with tab stops, syntax indent levels, or decimal points shown. To set these options, from the main menu, choose **Tools > Options > General**, then select the [General Tab](#). Select one of the **Current line highlight** options. Click on the color boxes to change the box color or the column marker color.

### Preserving the Column on Top/Bottom

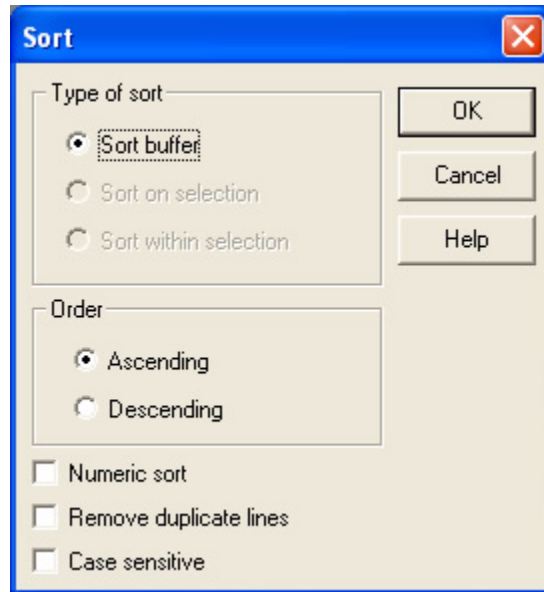
You can specify that the **top\_of\_buffer** (Ctrl+Home) and **bottom\_of\_buffer** (Ctrl+End) commands do not change the column position unless already at the top or bottom of the buffer. To set this option, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Select the option **Preserve column on top/bottom**.

### Setting the Line Insert Style

To set the line insert style, from the main menu, choose **Tools > Options > General**, then select the [More Tab](#). Set the **Line insert style** to **Before** or **After**. When the line insert style is set to **Before**, lines of text are inserted before the current line. When the line insert style is set to **After**, lines of text are inserted after the current line.

## Sorting Text

SlickEdit® uses a stable quicksort algorithm to sort text. It is recommended that at least half the text be in memory for best speed results. To sort text, from the main menu, choose **Tools > Sort**. The Sort dialog box is displayed, as pictured below.



The following options are available:

- **Type of sort** - Choose the type of sort that you prefer from the following options:
  - **Sort buffer** - When this option is selected, the entire contents of the buffer that you are working in are sorted.
  - **Sort on selection** - When this option is selected, each line intersecting with the selection is sorted based on the selected column. **Sort on selection** and **Sort within selection** have the same effect except when sorting a block or column selection.
  - **Sort within selection** - When this option is selected, the selected text is sorted. Text outside a block or column selection is not moved. The **Sort on selection** and **Sort within selection** options have the same effect except when sorting a block or column selection.
- **Order** - Choose **Ascending** or **Descending**. In an ascending sort, the lowest text item sorted is placed at the top.
- **Numeric sort** - When this option is selected, a numeric comparison is performed.
- **Remove duplicate lines** - When this option is selected, it removes adjacent lines that are identical. This option does not fully support column selection (it always compares complete lines).
- **Case sensitive** - When this option is selected, the sort is case-sensitive.

## Sort Commands

To use the command line for sorting, first activate the command line by pressing **Esc**. Sort command syntax is in the form **SortCommand OptionLetter(s)**. The following sort commands are available:

- **sort\_buffer** - Sorts the current buffer.

- **sort\_within\_selection** - Sorts text within a selected area. This command supports line and block selections only.
- **sort\_on\_selection** - To sort on a column field, press Ctrl+B to select an area of text, then invoke the command **sort\_on\_selection**. This command supports line and block selections only.

The table below describes the **OptionLetter(s)** that you can use with each command.

Option	Meaning
<b>A</b>	Sort in ascending order.
<b>D</b>	Sort in descending order.
<b>I</b>	Case insensitive sort (Ignore case).
<b>E</b>	Case sensitive sort (Exact case which is the default).
<b>-N</b>	Numeric sort. C-style floating point numbers with up to 32-digit mantissa are supported.
<b>-F</b>	File name sort.

## Inserting Literal Characters

Characters can be inserted at the cursor location in the current buffer. This is useful if you wish to insert non-ASCII characters (keys not on the keyboard). To insert a literal character, from the main menu, choose **Edit > Insert Literal**, or use the **insert\_literal** command. The Insert Literal dialog is displayed.

The text box to the right of the **Character Code** label displays the character. The spin box displays the decimal character code, hex character code, or ASCII character depending on which of those options is selected.



## Color Coding

---

This feature is designed to combine current line coloring, modified line coloring, and language-specific coloring. By default, the fundamental mode colors the current line. Languages with specific support already defined (i.e. keywords, comments, etc.) use language-specific coloring.

If modified line coloring is enabled, the left edge of the window displays a different color depending on whether the line was inserted or modified. To change the colors, use the Color Settings dialog box. For more information see, [Colors](#).

When current line color coding is enabled, language-specific color coding for the current line is not viewable.

If you want to know what lines have been modified, bind the command **color\_modified\_toggle** to a key. It will toggle display of modified lines in a different color on/off. You can bind the **color\_toggle** command to a key as well. This command toggles between current line, modified line, and language specific coloring individually.

## Resetting Modified Lines on Save

SlickEdit® can clear the modified and inserted line color when you save a file. To enable this feature, select the **Reset line modify** option on the [Save Tab](#) of the File Options dialog (**Tools > Options > File Options**).

## Adding Color-Coded Keywords to Supported Languages

You can add color-coded keywords to a supported language. To add color-coded keywords to a supported language, complete the following steps.

1. From the main menu, select **Tools > Options > Color Coding**.
2. Select the [Tokens Tab](#).
3. Choose the language you want to modify from the **Lexer Name** combo box list.
4. Click **New**.
5. Enter the new keywords separated with a space character.
6. Click **OK**.
7. Click **OK** on the Color Coding Setup dialog box.

For more information, see [Color Coding Configuration](#).

## Creating Color Coding for a New Language

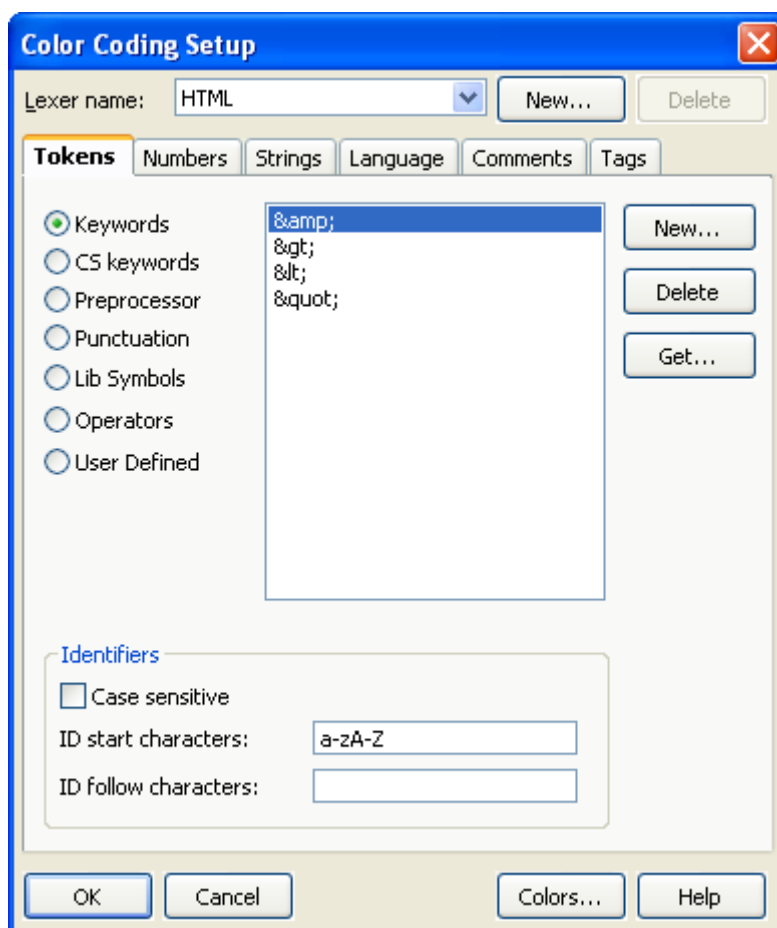
To create color coding support for your language, complete the following steps:

1. From the main menu, select **Tools > Options > Color Coding**. The Color Coding Setup dialog box is displayed.
2. Select the [Tokens Tab](#), then click **New**. The Enter New Keywords dialog box is displayed.
3. Enter the new lexer name. Usually this is a language name such as C or Java. Click **OK**.

4. On the Tokens tab, make sure the new keyword is selected, then correct the **ID start characters**. These are valid characters which can be the start of an identifier.
5. Correct the **ID follow characters**. These are additional characters which are valid after the start id character. For example, digits are usually allowed in identifiers, but not as the first character of an identifier.
6. Select the [Comments Tab](#). This lists the comments currently defined and allows you to define new multi-line and line comments. For each comment, click **New** to add a line or multi-line comment.
7. Select the [Numbers Tab](#) to display various numeric style options.
8. Select the [Strings Tab](#) to display various string literal options.
9. If you have not found all the options you need, click the [Language Tab](#). This displays some more advanced language-specific options.
10. Click **OK** on the Color Coding Setup dialog box.

## Color Coding Configuration

The Color Coding Setup dialog provides the capability to specify colors for identifying your code. To configure color coding, from the main menu, select **Tools > Options > Color Coding**. The Color Coding Setup box is displayed.



Select the language that you wish to work with from the **Lexer name** drop-down list.

Click **New** to prompt for a lexer name to start a new language-specific color coding definition (see [Creating Color Coding for a New Language](#)).

Click **Colors** at the bottom of the dialog to display the Color Settings dialog, which allows you to specify the color for color coding elements and other editor elements (see [Setting Colors for Screen Elements](#)).

The tabs on the Color Coding Setup dialog are described in the section [Color Coding Setup Dialog](#).

## Advanced Color Coding Configuration

The `vslick.vlx` file defines language-specific coloring support. For information about modifying this file, and how to create a new lexer name, see [VLX File and Color Coding](#).

## Color Coding Settings

There are several extension-specific settings that can be made to affect Color Coding. To access these options, from the main menu, choose **Tools > Options > File Extension Setup**. When the Extension Options dialog appears, select the extension you wish to work with from the **Extension** drop-down list, then select the [Advanced Tab](#).

Select from the following options:

- **Lexer name** - Specifies which lexer to use to recognize elements to use to be colored.
- **Color Coding** - Displays the Color Coding Setup dialog box allowing modification of language specific color coding for the current language. For more information, see [Color Coding Setup Dialog](#).
- **Modified lines** - Color codes the modified lines.
- **Current line** - Color codes the current line.





# Syntax Indent and SmartPaste®

---

[Syntax Indent](#) and [SmartPaste®](#) are two of the many SlickEdit® features designed to decrease typing, improving your coding efficiency.

Syntax Indent automatically indents code to the correct levels. There are two ways that code can be indented: by using the automatic Syntax Indent feature, and/or by using tabs.

SmartPaste reindents pasted text to the correct level based on surrounding code.

## Syntax Indent

By default, if you press Enter while you are editing a source file, Syntax Indent automatically indents the cursor to the next level if it is moved inside a structure block. For example, if you edit a C file and the cursor is on a line containing the text `for ( : ){` and you press Enter, a new line is inserted and the cursor is indented four spaces in from the letter “f” in the word “for”.

To change the Syntax Indent spacing, complete the following steps:

1. Select **Tools > Options > File Extension Setup**. The Extension Options dialog is displayed.
2. Select the [Indent Tab](#).
3. From the **Extension** drop-down list box, select the file extension that you wish to affect.
4. Change the value in the **Syntax indent** text box.

## Indenting with Tabs

By default, when you press the Tab key to indent, literal spaces are inserted. If you plan to indent your code using tab characters, or if you will be editing files that already contain tabs, you will need to specify these preferences.

To enable tab indenting, from the main menu, select **Tools > Options > File Extension Setup**, then select the [Indent Tab](#). From the **Extension** drop-down list box, select the file extension that you wish to affect. Then select the option **Indent with tabs**.

## Setting Tab Spacing

The default value of the Tab key is eight spaces. You can change this value in the **Tabs** text box. In general, the **Tabs** setting should match the **Syntax indent** value. For example, by default for the C language extension, the **Syntax indent** value is set to “4,” and the **Tabs** setting is set to “+4.” The plus sign (+) indicates that the editor will automatically expand the stops by four. By default, the Tabs setting is “+4”, which indicates that the default tab setting is eight spaces.

To work properly with the Sun Java API source code, the tab stops need to be in increments of eight, but the syntax indent must be set to four. The Syntax Indent affects not only the Tab key, but also the number of spaces to indent for each code block level.

#### NOTES

- When you change the tab stops and indent for all languages except COBOL, change the **Tabs** text box to **+value** where *value* is the same value used for the **Syntax indent** text box. The **Tabs** text box only affects how tab characters are expanded on the screen. This does not affect the indent when pressing Tab, or the amount of indent for statements inside a code block.
- For COBOL files, the **Tabs** text box also affects the Tab key. Syntax Indent still affects the indent for each code block level.

## Setting Tab to Indent Selections

For the Tab key to indent the selection when text is selected, select the option **Indent selection when text selected**.

## Setting Tabs for the Current File

To set tabs for the current buffer only, use the Tabs dialog box (**Document > Tabs** or **gui\_tabs** command). You can set tabs in increments or at specific column positions. For example, to specify an increment of three, enter **+3** in the text box. To specify columns, you could enter **1 8 27 44** to specify tab stops that have absolute locations.

By default, the Tab key inserts enough spaces to move the text to the next tab stop. The Shift+Tab key combination deletes enough spaces to move the text to the previous tab stop. See [Redefine Common Keys Dialog](#) for information on other Tab and Shift+Tab key bindings. Regardless of the Tab key binding, if the extension-specific setting **Indent with tabs** is on, a physical tab character is inserted (see [Indenting with Tabs](#)).

## Setting the Backspace Unindent Style

By default, pressing the Backspace key when the previous character is a tab, causes the rest of the line to be moved to the previous tab stop. If you want your Backspace key to delete through tab characters one column at a time, choose **Tools > Options > Redefine Common Keys**, then select the **Hack tabs backspace** option. See [Redefining Common Keys](#) for more information.

## SmartPaste®

When pasting lines of text into a source file, SmartPaste reindents the added lines according to the surrounding code. For example, if editing a C or C++ file, select some lines with a line selection (Ctrl+L), copy them to the clipboard (Ctrl+C), then paste them inside a for loop block (Ctrl+V). The added lines are correctly indented according to the for loop's indent level. SmartPaste will work for character/stream selections; however, the last line of the selection must include the end-of-line character. Use the mouse to copy and move lines and still take advantage of SmartPaste.

**NOTE** SmartPaste only works with line selections. For information about creating a line selection, see [Line Selections](#).

# Completions

Completions save keystrokes as you are typing code by providing a way to automatically complete partially-typed text. There are four types of completions in SlickEdit®:

- [Auto-Complete](#) - A feature set that includes syntax, keyword, and symbol completions.
- [Word Completion](#) - Completions that work for any text in an editor window.
- [Completion in Dialogs](#) - Completions that work in dialog text box fields.
- **Command-line completion** - Completions for command line entries. See [Command Line Completions](#) for more information about this type of completion.

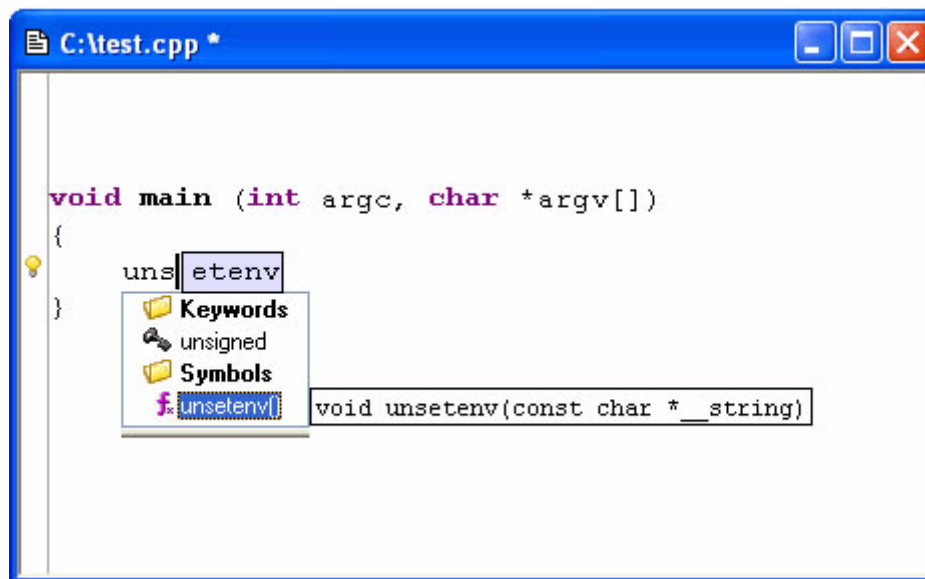
## Auto-Complete

Auto-Complete offers suggestions for how syntax, keywords, symbols, and lines of code may be completed by the editor. It works by looking at the word prefix under the cursor and using several different queries to find and suggest completion options. Each of these types of suggestions can be individually enabled or disabled, allowing you to customize auto-completion to your liking.

## Using Auto-Complete

Auto-Complete is activated when the editor is idle for a short period of time and there is a partially-typed word under the cursor. When Auto-Complete is active, the available completions are indicated in several ways:

- A light bulb appears on the left edge of the editor.
- A list of completions appears under the word being typed.
- The rest of the completed word or statement appears to the right of the cursor.



These visual hints can also be individually enabled or disabled through the Auto-Complete options. See [Auto-Complete Tab](#).

**TIP** Auto-Complete can be activated manually by using the **autocomplete** command. By default, this command is not bound to a key. Key bindings can be set from the main menu by choosing **Tools > Options > Key Bindings**. For more information, see [Creating Bindings](#).

To cancel out of Auto-Complete mode, use the Escape key.

To scroll through the items in the completion list, use the up arrow, down arrow, Page Up, and Page Down keys. Optionally, you can use Tab and Shift+Tab to cycle through the choices.

If a completion is selected, you can press Space, Enter, or any non-identifier key to cause the selected completion to be inserted along with the character typed (except for Enter).

Use Shift+Space to insert a real space rather than the completion. Use Ctrl+Shift+Space to insert the next character of the currently selected completion. This can be useful if you only want part of the word being completed and you do not want to type it yourself. Optionally, pressing Tab will cause auto-completion to attempt to insert the longest unique prefix match of all its completions.

If the completion has comments, you can use Shift+PageDown, Shift+PageUp, Shift+Home, or Shift+End to page through the comments. Use Ctrl+C to copy the comments for the current item to the clipboard.

Auto-Complete options can be configured for each file extension type. This allows you to enable and disable particular features on a per-language basis. To change Auto-Complete options, see [Auto-Complete Tab](#).

## Word Completion

Word Completions search the current editor window for text matching the prefix at the current cursor position. Most completions are driven by Context Tagging®, matching symbols such as function names and variables. Word Completions can match any text in the current editor window, including comments.

Auto-Complete also lists word completions, but it is often faster to use key bindings to search for and insert Word Completions. The following is a list of commands for these operations and the key bindings in the CUA emulation. See [Creating Bindings](#) to change them.

- **complete\_prev** (Ctrl+Shift+Comma) – Searches backwards through the current editor window to find a match.
- **complete\_next** (Ctrl+Shift+Dot) – Searches forwards through the current editor window to find a match.
- **complete\_more** (Ctrl+Shift+Space) – Adds subsequently more text from the matched line to the cursor position, allowing you to extend the amount of text inserted.

The following example of code shows how word completion is used:

```
if (pWindowView->pBuffer->LineNum>100) {  
  pW<Cursor is Here>  
}
```

Press Ctrl+Shift+Comma, Ctrl+Shift+Space, Ctrl+Shift+Space to obtain the following result:

```
if (pWindowView->pBuffer->LineNum >100) {  
  pWindowView->pBuffer->LineNum <Cursor is Here>  
}
```

Pressing Ctrl+Shift+Comma matched “pWindowView” in the previous line. If you wanted to match an earlier occurrence beginning with “pW”, press Ctrl+Shift+Comma to find the next previous match. This also changed “pW” on the second line to the matching text, “pWindowView”. Pressing Ctrl+Shift+Space extends that selection, matching “pWindow->pBuffer”. Pressing Ctrl+Shift+Space, again, extends the selection to include “pWindow->pBuffer->LineNum”.

You can easily see how this would save time typing in multiple lines that access structs, class members, arrays, etc.

## Completion in Dialogs

Many SlickEdit® dialogs contain fields that offer completions. Dialogs such as the Open dialog box (**File > Open**) and the New Project dialog box (**Project > New**) contain text fields for filenames or directory paths. SlickEdit uses completions to help you enter values for these fields. When you type a character, SlickEdit pops up a list of matching values.

When this list is displayed, the following keys have different behaviors:

Key	Behavior
Escape	Closes the list.
Enter	Uses the current value selected.
Tab, down arrow, or Ctrl+K	Moves to the next item in the list.
Shift+Tab, up arrow, or Ctrl+I	Moves up one item in the list.
Home or End	If an item is selected in the list, the cursor moves to the top or bottom of the list, otherwise the cursor moves to the beginning or end of the text box, respectively.
PageUp or PageDown	Moves up or down a page of items in the list.
Space	If an item is selected in the list, that item is used and you are advanced to the next argument.

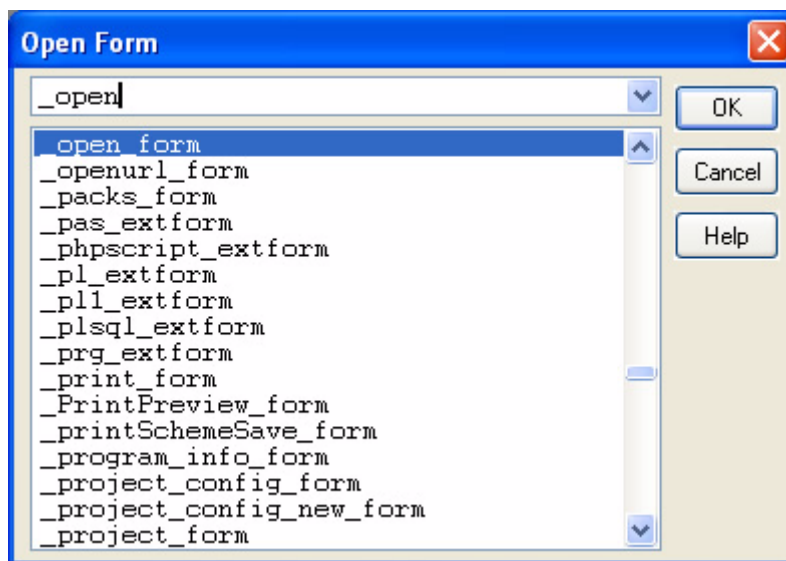
For a demonstration of how completions in dialogs work, complete the following steps:

1. From the main menu, select **Project > New**.
2. Press Tab to jump to the **Location** field.
3. Start typing a directory path. Notice the completion options.
4. Press **Esc** to cancel.

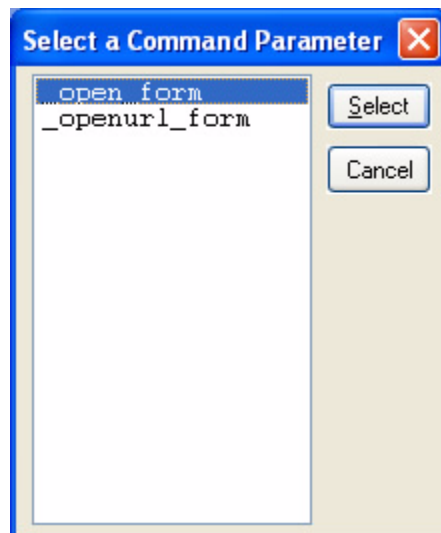
## Argument Completion

For a larger list of possible matches, type “?” to list the matches. For a demonstration of how this works, complete the following steps:

1. From the main menu, select **Macro > Open Form**.



2. Type **\_open** and the **\_open\_form** command is highlighted.
3. Press the question mark key (?) to display the Select a Command Parameter dialog. A selection list of possible matches to an argument that is partially-typed is displayed.



## Configuring Completion Settings

To configure Auto-Complete settings, from the main menu, choose **Tools > Options > File Extension Setup**. Choose the extension you wish to work with from the **Extension** drop-down list, then select the **Auto-Complete** tab. For a listing of these options and descriptions, see [Auto-Complete Tab](#).

# Aliases

Aliases are identifiers that you can quickly type which are then expanded into snippets of text. You can use aliases for commonly typed function names, statements, or to insert several lines of code. There are two types of aliases in SlickEdit®:

- [Directory Aliases](#) - Directory aliases are short identifiers for long directory names. They save you from having to type long path names when you are prompted for a file name or directory.
- [Extension-Specific Aliases](#) - These aliases are set up on a per-language basis, and are useful for inserting frequently used text, such as comment headers, into your code.

## Directory Aliases

Directory aliases take advantage of the fact that most users are constantly opening files from a small number of directories throughout the day. By using a directory alias when opening a file or changing directories, you do not have to type in long paths or click the mouse repeatedly in the **Directory** list box.

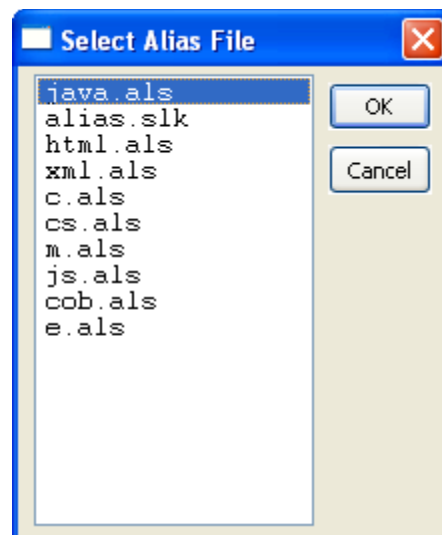
After typing the alias identifier, directory aliases can be expanded by pressing Ctrl+Space. These aliases are stored in the file `alias.slk`.

## Defining a New Directory Alias

Directory aliases typically consist of a short abbreviation of the last name in a long directory path. For example, if you had a directory called `c:\version20\src\project2\`, a good alias name might be **p2**. For compiler include files, define an alias called **inc** (**vinc** in Microsoft Visual C++, **binc** in C++ Builder®, or **ginc** for GCC) if you have multiple compilers.

To define a new directory alias, complete the following steps:

1. From the main menu, select **Tools > Options > Aliases**. The Select Alias File dialog appears.



2. Select `alias.slk` and click **OK**. The Alias Editor dialog appears.
3. Click **New**, then type the characters you wish to use for an identifier in the **Alias Name** text box.
4. Click **OK**. The identifier you entered is now displayed in the list box in the Alias Editor dialog.

5. Make sure your new identifier is selected, then in the large text box to the right, enter the alias value by typing in the directory path that you want the identifier substituted with.
6. Click **OK**.

## Using Directory Aliases

After the directory aliases are defined, you can use them in any text box or buffer, including the Build tool window and the Open and Change Directory dialogs. For example, to use a directory alias in the Open dialog, complete the following steps:

1. From the main menu, select **File > Open** (Ctrl+O).
2. In the **File name** field, type the alias name (identifier) for the directory you wish to go open.
3. Press **Ctrl+Space** to expand the alias.

## Embedding Environment Variables in Directory Aliases

If you keep source code in a version directory tree, you might want to set an environment variable and embed the environment variable in the alias value. For example, if you have a directory named `c:\version20\src\project2\`, define a **p2** alias and give it a value such as **%VERSION%\src\project2\**. Type the following command on the command line to set or create the VERSION environment variable:

```
set VERSION=c:\version20
```

For more information about setting environment variables, see [Environment Variables](#).

## Extension-Specific Aliases

You can set up aliases for any frequently used text, such as comment headers. Extension-specific aliases are set up on a per-language basis. Each extension can have one alias file, allowing aliases to be defined that do not affect other extensions.

After typing the alias identifier, extension-specific aliases can be expanded by pressing Ctrl+Space (**codehelp\_complete** command). Extension-specific aliases are stored in a file that you can specify and typically have the extension `.als`.

### TIPS

- If the alias is a Syntax Expansion modification, you can simply press Space to expand the alias. See [Syntax Expansion](#) for more information.
- An Auto-Complete option is available to show a tooltip of the matching alias for the word under the cursor. Choose **Tools > Options > File Extension Setup**, select the [Auto-Complete Tab](#), then choose the **Extension** from the drop-down list. Select the option **Alias expansion**. See [Completions](#) for more information.

## Creating an Extension-Specific Alias

When you define a new alias for a file extension, each time that you open or create a file with that extension, the aliases will be available.



## Choosing the Alias File

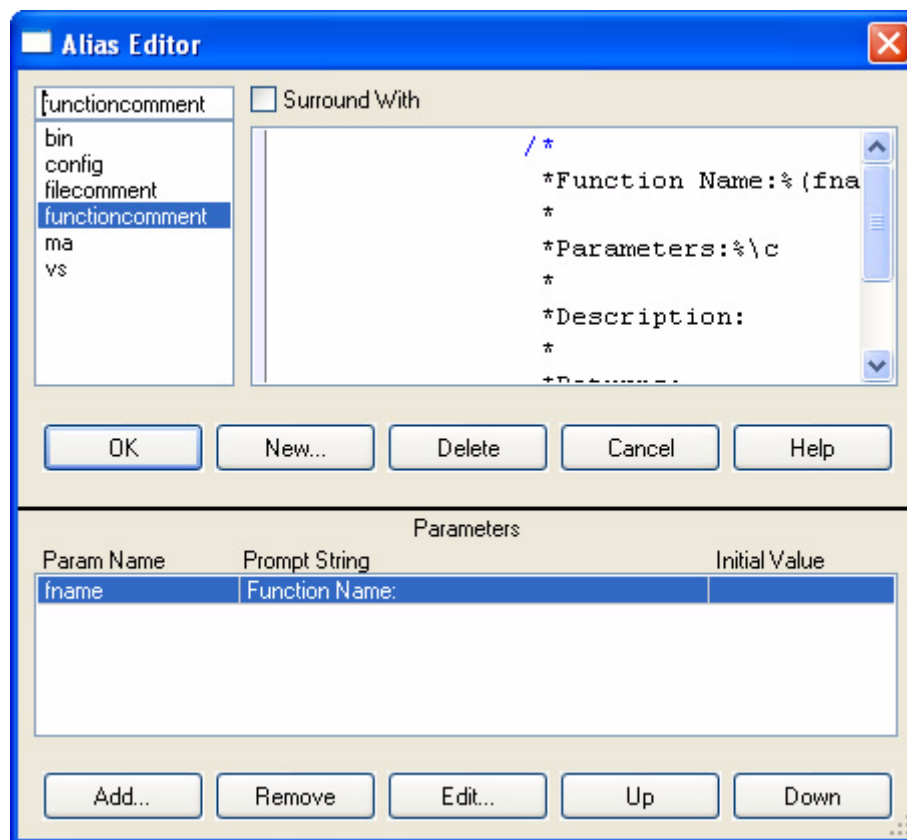
Before defining a new alias, you must specify the file in which the alias is to be stored. The quickest way to do this is to choose **Tools > Options > Aliases**, then pick an alias file from the predefined list. After you select the alias file, the Alias Editor dialog appears, allowing you to create the aliases.

If you wish to specify your own file to store aliases, or if you are using an extension that does not have a predefined file in the list, complete the following steps:

1. From the main menu, select **Tools > Options > File Extension Setup**. The Extension Options dialog appears.
2. Select the [General Tab](#).
3. In the **Extension** drop-down list, select the file extension that you wish to work with.
4. Note the **Alias file name**. If you wish to store the aliases in another file, type a new file name with the `.als` extension here.
5. Click **Aliases**. The Alias Editor dialog appears.

## Using the Alias Editor

After choosing the file used to store aliases, use the Alias Editor dialog to create or edit extension-specific aliases. The dialog is pictured below.



Alias names are displayed in the list box at the top left of this dialog box. The value for the selected alias name is displayed in the large text box at the top right of this dialog. Click **Delete** to remove a selected alias and its value.

To create a new alias, complete the following steps:

1. Click **New**, then type the characters you wish to use for an identifier in the **Alias Name** text box.
2. Click **OK**. The identifier you entered is now displayed in the list box in the Alias Editor dialog.
3. Make sure your new identifier is selected, then in the large text box to the right, enter the alias value by typing in the text that you want the identifier substituted with.

### TIPS

- You can use special escape sequences in your aliases, which will be substituted upon expansion with certain values. See [Alias Escape Sequences](#) for more information.
- You can also specify parameters in alias values. When the alias is expanded, you are prompted with a dialog to input the values. See [Parameter Prompting](#) for more information.

4. Click **OK**.

## Alias Escape Sequences

Alias escape sequences can be used in alias values. When the aliases are expanded, the sequences are replaced with their values. The following table contains a list of the escape sequences that can be used for aliases. For examples, see [Escape Sequence Examples](#) below.

Escape Sequence	Description
<b>%lc</b>	Places the cursor. This sequence can be used multiple times in the same alias value in order to create a series of “hot spots” within the alias. After the alias is expanded, press <b>Ctrl+[</b> ( <b>next_hotspot</b> command) to jump to the next cursor stop.
<b>%ld</b>	Inserts the date (locale-dependent).
<b>%le</b>	Inserts the date in MMDDYY format.
<b>%lt</b>	Inserts the time (locale-dependent).
<b>%%</b>	Inserts a percent character.
<b>%f</b>	Inserts the current file name.
<b>%ln</b>	Inserts the current function name.
<b>%lo</b>	Inserts the current function name with signature.
<b>%li</b>	Indents.
<b>%lb</b>	Unindents.
<b>%xColumnNumber</b>	Moves the cursor to the specified column number.
<b>%x+Increment</b>	Increment column by ddd.
<b>%x-Increment</b>	Decrement column by ddd.

Escape Sequence	Description
<b>%ls</b>	Preserves trailing spaces - place at the end of a line.
<b>%ll</b>	Preserves leading spaces - place at the beginning of the first alias line.
<b>%(ParameterName)</b>	Parameter replacement. See <a href="#">Parameter Prompting</a> .
<b>%\m MacroName ArgumentList%</b>	Calls the specified Slick-C® macro with a specified optional argument.
<b>%EnvironmentVariable%</b>	Inserts the value of the environment variable specified.
<b>%\m sur_text%</b>	Indicates where the text to be surrounded will be placed. See <a href="#">Dynamic Surround and Surround With</a> for more information.

## Escape Sequence Examples

The following table contains some examples of using escape sequences in alias values:

Alias Name and Description	Value
<b>comment</b> - A header comment to have the date and time inserted.	<pre> *****/ /* Date: %\d Time: %\t */ *****/ </pre>
<b>if</b> - A simple <b>if</b> statement, with indenting, support for surround, and a cursor position.	<pre> if(%\c){ %\i// Comment goes here %\i%\m sur_text% } </pre>
<b>ifelse</b> - An <b>if/else</b> statement with indenting and several cursor hot spots.	<pre> if(%\c){ %\i%\c } else { %\i%\c } </pre>
<b>wm</b> - A WinMain function template with indenting and a cursor position.	<pre> int APIENTRY WinMain(HANDLE hInstance,                      HANDLE hPrevInstance,                      LPSTR lpszCmdParam,                      int nCmdShow)  { %\i%\c } </pre>

## Parameter Prompting

Parameters can be set up for aliases, so that when the alias is expanded, you are prompted with a dialog to input the values. This is useful for reducing even more key strokes for repetitive tasks when using aliases that may require different values each time they are used.

To use parameter prompting, first define the parameters, then use them in your alias values by typing **%(ParamName)** where *ParamName* is the name of the parameter that you have defined (see [Creating an](#)

[Alias for Parameter Prompting](#) below). When the alias is used and expanded, the Parameter Entry dialog will appear, prompting you for the parameter values, which will then be inserted into your text.

## Creating an Alias for Parameter Prompting

To create an alias for parameter prompting, first select the alias file as described in the section [Choosing the Alias File](#), then use the Alias Editor to complete the following steps:

1. Click **New**, then enter the new alias name. In the aliases list box (on the left side of the Alias Editor), make sure the new alias is selected.
2. Click the **Add** button below the **Parameters** group box. The Enter Alias Parameter dialog is displayed.
3. Enter the following values:
  - **Parameter Name** - Enter the name that you wish to use in the alias value.
  - **Prompt** - Enter the text that you wish to be prompted with. This is the label that will appear on the Parameter Entry dialog that prompts for values after the alias is expanded.
  - **Initial Value** - (Optional) Enter the initial value of the parameter. This text will appear in the text field of the Parameter Entry dialog that prompts for values after the alias is expanded.
4. Click **OK**.
5. If you wish to add more parameters, repeat Steps 2 through 4.
6. On the Alias Editor dialog, the Parameters group box will now display a list of the parameters that you have added.
7. In the large text field on the right side of the Alias Editor, you can now type the alias value. In the places where you want parameter prompting to occur, type **%(ParamName)**, where *ParamName* is the parameter name that you entered in Step 3.
8. Click **OK** when you are finished.

### Example: Instantiating a Variable in Java with Parameter Prompting

In Java, instantiating variables can be a repetitive task. The following code shows a common Java code snippet:

```
public class {
    public static void main (String args[]) {
        String x = new String( arg[0] );
    }
}
```

You could define an alias for entering new class names with variables and arguments. That way, when you press Enter after the third line and type and expand the alias, you will be prompted for the values.

For this alias, in the Alias Editor dialog, first define three parameters: **class\_name**, **var\_name**, and **arg\_list**. Then, enter the following text for the alias value:

```
%(class_name) %(var_name) = new %(class_name)( %(arg_list) );
```

## Creating an Extension-Specific Alias from a Selection

You can create an extension-specific alias from a selection by following the steps below.

1. Select some code.

2. Right-click and select **Create Alias**.
3. Give the alias a name and click **OK**.
4. The Alias Editor dialog appears, from which you can edit the code to fine-tune, or add parameters.



## Syntax Expansion

Syntax Expansion is a feature designed to minimize keystrokes, increasing your code editing efficiency. When you type certain keywords and then press the spacebar, Syntax Expansion inserts a default template that is specifically designed for this statement. For example, if you are using the C language and type “for”, press Space and the following text expansion is inserted, with the cursor location between the parentheses:

```
for (
{
}

```

Additionally, for C, C#, C++, J#, Java, and Slick-C®, after the statement is expanded, you can use the **next\_hotspot** command (Ctrl+[) to jump the cursor to the next part of the statement. In the case of the **for** loop above, Ctrl+[ would move the cursor from the group in parentheses to the code block.

The structures **loop**, **if**, and **switch** or **case** are also expanded. You do not have to type the entire keyword for Syntax Expansion to occur. If there is more than one keyword that matches what you type, a list of possible keyword matches is displayed. To get the C template displayed above, type “f” followed by pressing Space.

To override the insertion of braces immediately for one line **if**, **for**, or **while** statements, type a semicolon immediately after the keyword. For example:

```
if;    =>  if ( <cursor here> ) <next hotspot>;
```

To override non-insertion of braces immediately for **if**, **for**, **while**, **foreach**, **with**, **lock**, **fixed**, and **switch** statements, type an open brace immediately after the keyword. For example:

```
if{    =>  if ( <cursor here> ) { <next hotspot> }
```

If the default behavior of Syntax Expansion does not match your coding style, for most languages, it can be customized. From the main menu, choose **Tools > Options > File Extension Setup**, select your language extension, then click the **Options** button. For more information on these options, see the topic for your language in the [Language-Specific Editing](#) chapter.

For further customization, for most languages, you can override the default keyword expansion by defining an alias for that keyword. See [Extension-Specific Aliases](#) for more information.

## Syntax Expansion Settings

To access Syntax Expansion settings, from the main menu, choose **Tools > Options > File Extension Setup**. Choose the extension you wish to affect from the **Extension** drop-down list, then select the [Indent Tab](#).

To turn Syntax Expansion on or off, select or deselect the option **Syntax expansion**.

To change the minimum expandable keyword length, enter the value by using the **Minimum expandable keyword length** spin box.

To set options such as brace style, click the **Options** button on the Extension Options dialog.

**TIP** SlickEdit® can display Syntax Expansion choices for the word prefix under the cursor. To turn this option on/off, select the [Auto-Complete Tab](#) on the Extension Options dialog, and select/deselect the **Syntax expansion** option. See [Completions](#) for more information.

## Modifying Syntax Expansion Templates

Syntax Expansion templates are essentially extension-specific aliases that have been pre-defined. You can modify these templates by replacing them with your own.

For example, to add a comment to the end of C **for**, **while**, **if**, and **switch** statements:

1. From the main menu, choose **Tools > Options > File Extension Setup**. The Extension Options dialog is displayed.
2. From the **Extension** drop-down list, select the **c** extension.
3. Select the [General Tab](#).
4. Click **Aliases** to display the Alias Editor dialog.
5. Click **New** and then type **for** as the alias name.
6. Type the following lines in the text box to the right of the alias name:

```
for (%\c;i) {
} /* for */
```

The **%\c** escape sequence above specifies the cursor placement after expansion is performed.

7. Repeat Steps 5 and 6 for the **while**, **if**, and **switch** keywords.
8. Click **OK** to save new aliases.

The above steps replace the default Syntax Expansion templates for these keywords. The C brace style options will not affect defined aliases.

For more information on working with aliases, using the Alias Editor, or using alias escape sequences, see [Extension-Specific Aliases](#).

## Adding Syntax Expansion for Other Languages

To add syntax expansion and indenting for other languages, complete the following steps:

1. Use the `prg.e` macro as a template. This file is located in the `macros` subdirectory of your installation directory. Make a copy of it and give it another name.
2. Change the `#define` constants **EXTENSION** and **MODE\_NAME** near the top of the file to reflect the new extension and mode name respectively. Do not use any spaces in these constants.
3. Change the name of the first five characters of the `_command` functions **dbase\_mode**, **dbase\_enter**, and **dbase\_space** to use the value given to the `MODE_NAME` constant in Step 2.
4. Modify the **prg\_expand\_enter** function to provide the Enter key the desired support .
5. Modify the **prg\_expand\_space** function to provide the spacebar key the desired support. If you can rely on extension-specific aliases, follow the comment in this function.
6. Use the load command **Macro > Load Module** to load new macro modules.

Steps 4 and 5 require a good understanding of the Slick-C® language and what this specific macro is doing. See the *Slick-C® Macro Programming Guide* for more information.



## Dynamic Surround and Surround With

Two SlickEdit® features that allow you to surround text with text are [Dynamic Surround](#), which lets you surround existing statements with block statements, and [Surround With](#), which lets you surround any selected text with predefined language structures, or any text that you specify. [Unsurround](#) is also available to remove outer code block structures from statements.

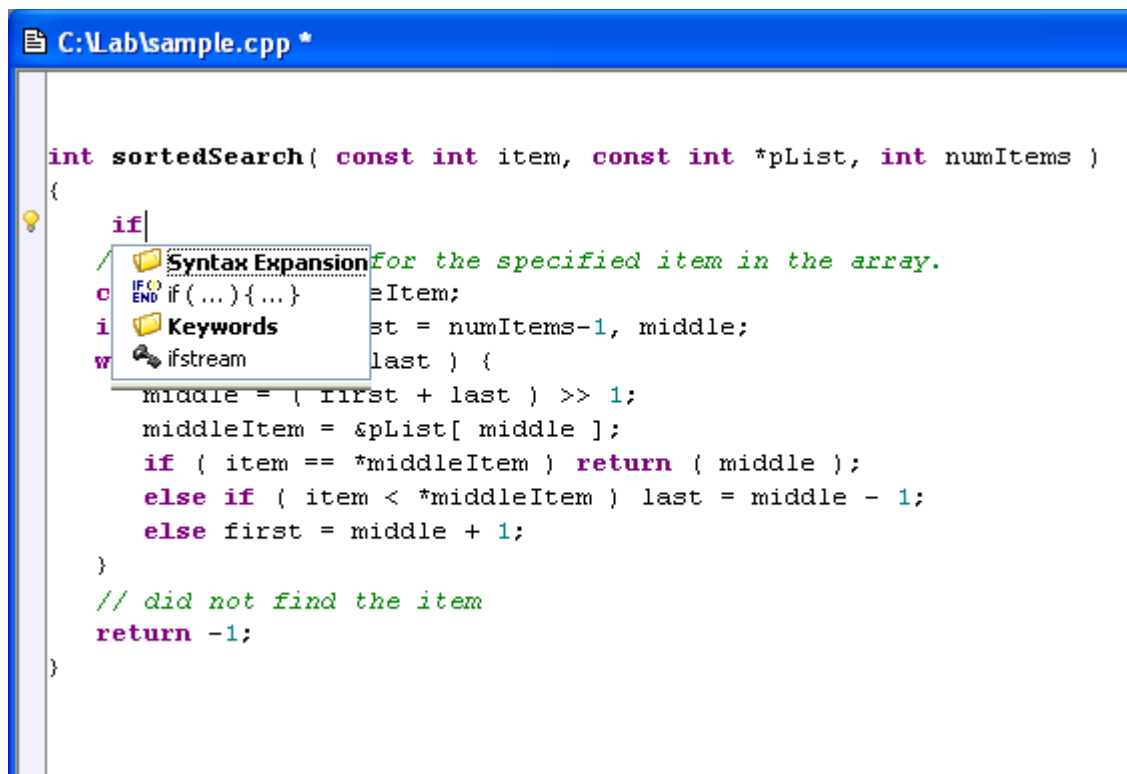
### Dynamic Surround

Dynamic Surround provides a convenient way to surround a group of statements with a block statement, indented to the correct levels according to your preferences. This feature works in conjunction with the syntax and alias expansion features (see [Syntax Expansion](#) and [Extension-Specific Aliases](#)), and is designed to help you keep your hands on the keyboard, thereby improving your speed and efficiency.

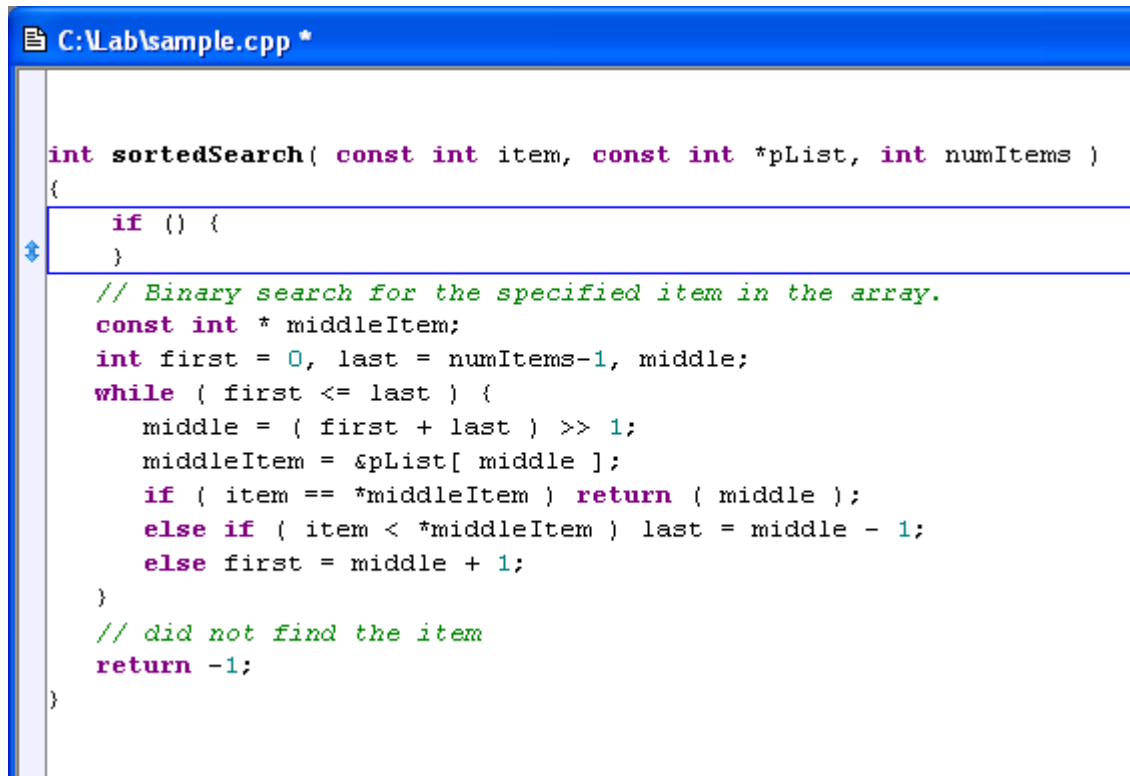
Dynamic Surround is supported for any language that uses block statements. Note that this feature is line-oriented and will not work for character or block selections.

SlickEdit® enters Dynamic Surround mode automatically, immediately after you expand a block statement (for instance, by typing "if" then pressing Space). After expanding the statement, a box is drawn around it as a visual guide, and you can pull the subsequent lines of code or whole statements into the block by using the Up, Down, PgUp, or PgDn keys. Pressing any other key or clicking with the mouse will exit Dynamic Surround mode.

The following screen shot shows the Syntax Expansion menu that appears after typing "if" in a C++ file:

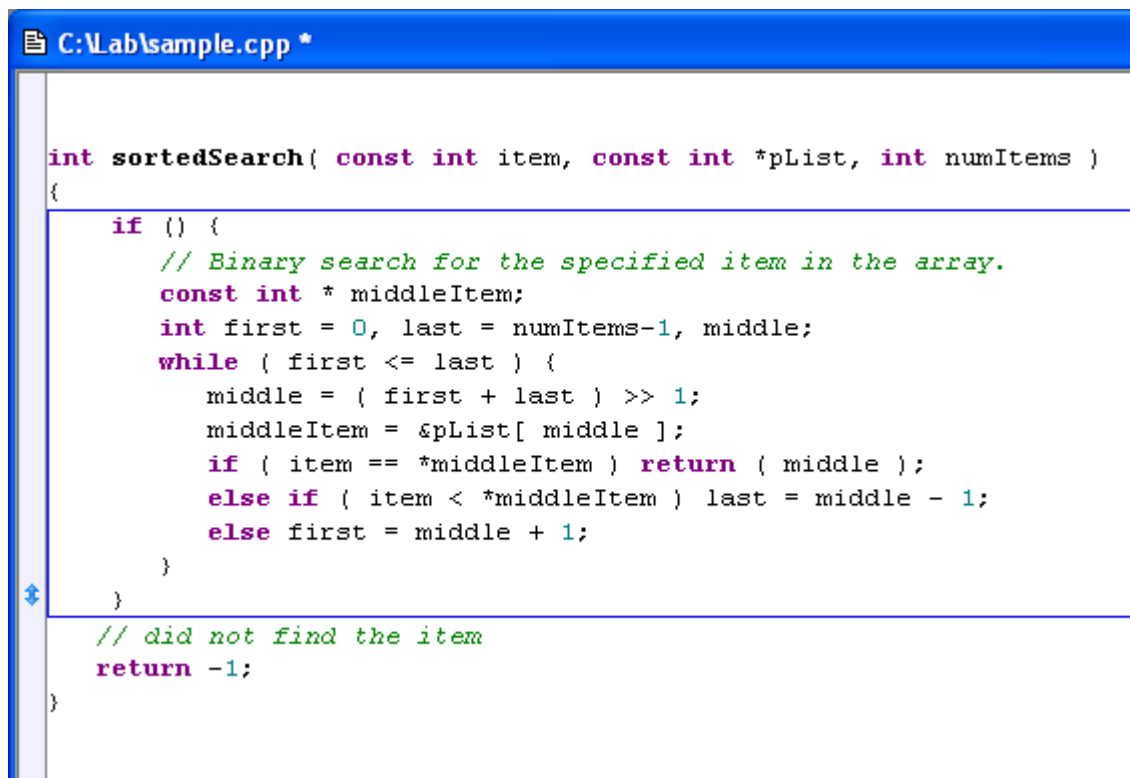


After pressing Space to expand the template, Dynamic Surround is activated, with a blue rectangle drawn around the expanded statement, as shown below:



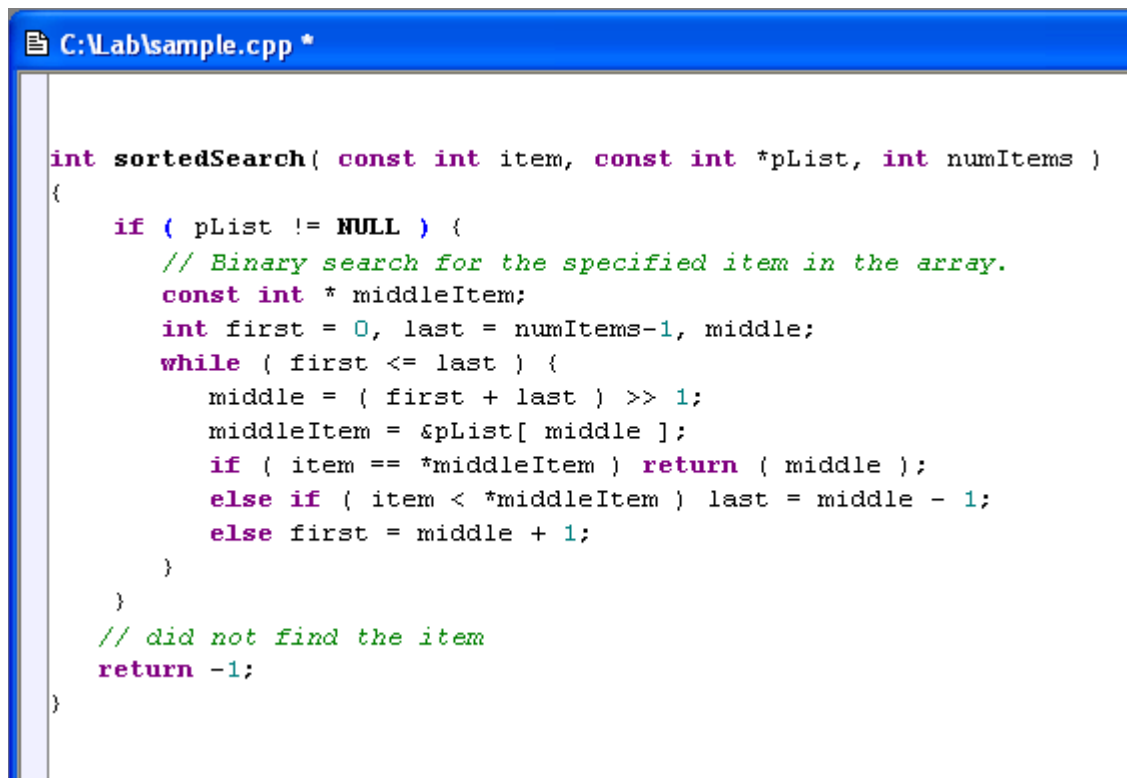
```
int sortedSearch( const int item, const int *pList, int numItems )
{
    if () {
        // Binary search for the specified item in the array.
        const int * middleItem;
        int first = 0, last = numItems-1, middle;
        while ( first <= last ) {
            middle = ( first + last ) >> 1;
            middleItem = &pList[ middle ];
            if ( item == *middleItem ) return ( middle );
            else if ( item < *middleItem ) last = middle - 1;
            else first = middle + 1;
        }
        // did not find the item
        return -1;
    }
}
```

Pressing the Down arrow key pulls the code block into the statement, indented to the correct levels, as shown below:



```
int sortedSearch( const int item, const int *pList, int numItems )
{
    if () {
        // Binary search for the specified item in the array.
        const int * middleItem;
        int first = 0, last = numItems-1, middle;
        while ( first <= last ) {
            middle = ( first + last ) >> 1;
            middleItem = &pList[ middle ];
            if ( item == *middleItem ) return ( middle );
            else if ( item < *middleItem ) last = middle - 1;
            else first = middle + 1;
        }
    }
    // did not find the item
    return -1;
}
```

The finished code is shown as follows:



```

int sortedSearch( const int item, const int *pList, int numItems )
{
    if ( pList != NULL ) {
        // Binary search for the specified item in the array.
        const int * middleItem;
        int first = 0, last = numItems-1, middle;
        while ( first <= last ) {
            middle = ( first + last ) >> 1;
            middleItem = &pList[ middle ];
            if ( item == *middleItem ) return ( middle );
            else if ( item < *middleItem ) last = middle - 1;
            else first = middle + 1;
        }
        // did not find the item
        return -1;
    }
}

```

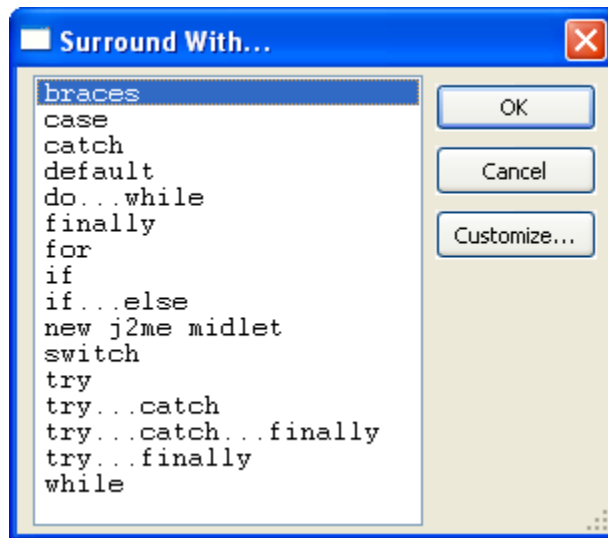
Statements that are pulled into the block are indented according to your settings on the [Indent Tab](#) of the Extension Options dialog (**Tools > Options > File Extension Setup**). The color of the rectangle box guide is controlled by the Block Matching screen element on the Color Settings dialog (**Tools > Options > Color**). See [Syntax Indent](#) for more information on setting indent styles, and [Colors](#) for more information on changing the colors of screen elements.

Syntax Expansion must be on for Dynamic Surround to work. Both options are on by default. To disable either of these options, select **Tools > Options > File Extension Setup**. Choose your extension from the **Extension** drop-down list, then select the [Indent Tab](#). Deselect the option(s) **Use Dynamic Surround** and/or **Syntax expansion**.

If you need to move or modify existing text, you can manually enter Dynamic Surround mode by using the **dynamic\_surround** command. This will cause the code block under the cursor to be selected, and you can use the navigation keys as described above to pull in statements. By default, this command is not associated with a key binding. See [Creating Bindings](#) for information on creating your own.

## Surround With

Surround With makes it fast and easy to wrap existing lines of code in a new block structure. Surround With is supported for the languages C, C++, C#, HTML, Java, JavaScript, and XML. Highlight the lines to surround, right-click, and select **Surround Selection With**, or use the **surround\_with** command. The Surround With dialog appears, with a pre-defined list of structures based on the current file extension.



Select the structure you wish to surround with, then click **OK**.

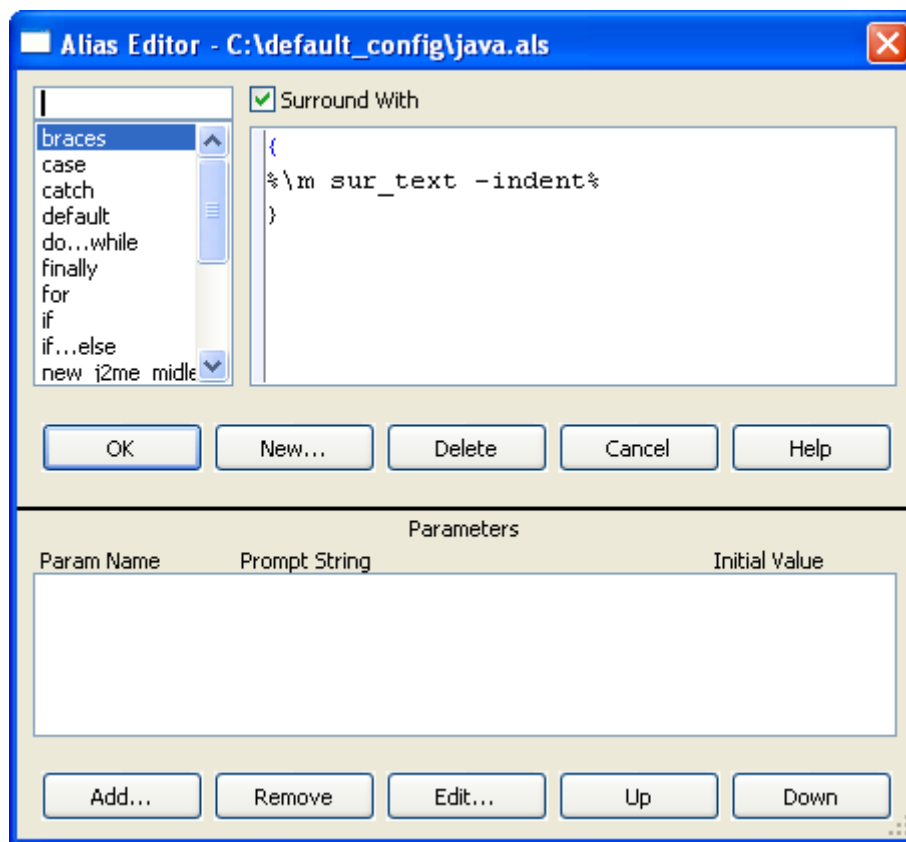
If there is no selection and you activate Surround With, the current line or code block will be automatically highlighted for surrounding (the same function performed by the **select\_code\_block** command).

**TIP** The **surround\_with** command has an icon available for toolbar customization. See [Customizing Toolbars](#) for more information about creating your own custom toolbars.

## Modifying Surround With Templates

Surround With templates are created and modified the same way as other aliases, with the addition of the **%\m sur\_text%** escape sequence. This sequence indicates where the selected text should be placed, and can be used multiple times within a single Surround With template. See [Surround With Commands](#) for more information on **sur\_text**.

To view or modify the Surround With templates, use the **surround\_with** command to display the Surround With dialog, then click the **Customize** button. This will display the Alias Editor dialog with the **Surround With** option selected. The list of Surround With structures for the chosen language is shown in the list box on the left.



To modify one of the Surround With structures, complete the following steps:

1. Select the structure that you wish to modify. Notice the template for the structure that appears in the text box on the right side of the Alias Editor.
2. Modify the template to suit your needs. For a list of escape sequences and template examples, see [Alias Escape Sequences](#). For more information about using the Alias Editor, see [Creating an Extension-Specific Alias](#).
3. When you are finished, click **OK** on the Alias Editor dialog.
4. Click **OK** on the Surround With dialog.

## Surround With Commands

There are three commands available for working with Surround With:

- **surround\_with** - This command is used to display the Surround With dialog, allowing you to pick a structure to surround selected text with. This command can be bound to a key—see [Creating Bindings](#) for more information.
- **sur\_text** - This is a Slick-C® function that can only be used inside of a Surround With template. It is used to indicate where the selected text should be placed and can be used multiple times within a single Surround With template. **sur\_text** can take several parameters, which can appear in any order. The available parameters are:
  - **-beautify** - This is the default for C, Java, and others. It beautifies the results of the template expansion.

- **-begin** *text* - Prefixes each line of the selection with *text*.
- **-deselect** - This is the default parameter. It specifies to leave the text deselected.
- **-end** *text* - Suffixes each line of the selection with *text*.
- **-ignore** *chars* - The **-begin**, **-indent**, and **-stripbegin** options will ignore any *chars* when finding the beginning of the selected line.
- **-indent** - Indents each line of the selection.
- **-nobeautify** - This is the default for HTML, XML, and others. It specifies that the editor should not attempt to beautify the results of the template expansion.
- **-notext** - Specifies that no text should be pasted.
- **-select** - Leaves the text selected.
- **-stripbegin** *text* - If any line begins with *text*, *text* is removed from the line. This option is applied before **-begin**.
- **-stripend** *text* - If any line ends with *text*, *text* is removed from the line. This option is applied before **-end**.
- **surround\_with\_if** - This is a wrapper command that expands the **if** template for the selected text. This command can be bound to a key—see [Creating Bindings](#) for more information.

The use of Surround With can be streamlined by using wrapper commands and key bindings. You can create your own wrapper commands. The following example is the definition of **surround\_with\_if**

```
_command void surround_with_if() name_info('');
```

VSARG2\_REQUIRES\_EDITORCTL | VSARG2\_MARK | VSARG2\_REQUIRES\_AB\_SELECTION)

```
{
    surround_with('if');
}
```

You must change the name of the command and the argument passed to **surround\_with**. The argument does not have to be an exact match with the template name. For instance, calling **surround\_with('i')** will prompt you to select the **if**, **if...else**, or **include once** template. If there is an exact match, that template will be used. In the case of **surround\_with\_if**, “if” matches the beginning of both the **if** and **if...else** templates, but the **if** template is used because it is an exact match.

After you create your wrapper command, you can bind it a key or invoke it from the command line.

For more information on working with commands, see the *Slick-C® Macro Programming Guide*.

## Unsurround

Unsurround is a feature that lets you remove the surrounding text from a code block. This is particularly effective when used with Dynamic Surround. Unsurround is supported for the following languages: ActionScript, AWK, C#, C++, CFML, HTML, Java, JavaScript, Perl, PHP, Slick-C®, Tcl, and XML.

To use Unsurround, right-click on a selected code block and select **Unsurround**, or use the **unsurround** command.

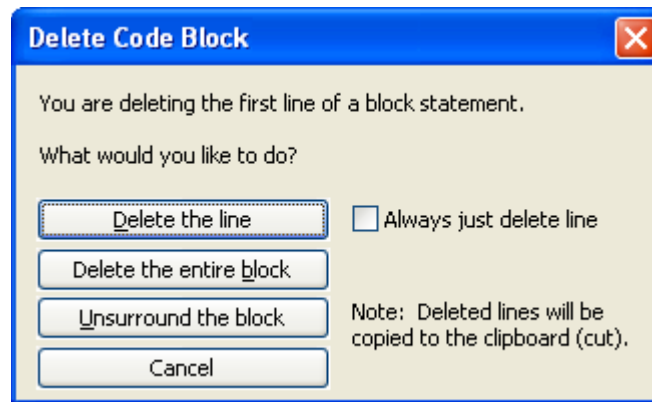
For example, to remove the **if** statement structure from a code block, select the code block or part of the code block, then right-click and select **Unsurround** (or use the **unsurround** command). The entire code block under the cursor is automatically highlighted and a dialog prompt appears to confirm the unsurround

operation. Click **OK**, and the **if** line of the code block as well as the line containing the closing brace are removed. The remaining code is unindented to the correct level.

**TIP** The **unsurround** command has an icon available for toolbar customization. See [Customizing Toolbars](#) for more information about creating your own custom toolbars.

## Deleting Code Blocks

Unsurround is also associated with the **cut\_line** (Ctrl+Backspace) and **delete\_line** (Ctrl+Del) commands. When one of these commands is invoked while the cursor is on the first line of a block statement, the Delete Code Block dialog appears, from which you can choose to delete the line, delete the entire block, or unsurround the block.



Each of these operations copies the removed text to the clipboard. This is useful if you want to paste the structure into a different location, because as soon as the text is pasted, SlickEdit® enters Dynamic Surround mode, allowing you to pull statements into the pasted block.

The Delete Code Block dialog also contains an option to **Always just delete line** when **cut\_line** or **delete\_line** operations are invoked. Selecting this option will prevent the dialog from appearing when these operations are used. To see the dialog again, use the **cut\_code\_block** command.



# Bookmarks

---

Bookmarks are used to save the current edit location, so you can quickly return to it later. There are two types of bookmarks:

- [Named Bookmarks](#) - Used to mark long-term, meaningful locations in the code, or to quickly set a temporary, named bookmark on the current line.
- [Pushed Bookmarks](#) - Used to set temporary “breadcrumbs” as you explore the code.

## Named Bookmarks

There are various ways to use named bookmarks:

- **Give them a specific name** - This is the best way to mark long-term, meaningful locations in the code. For example, you could set a bookmark named “main” to save the location of the **main** function.
- **Allow automatic naming** - This is the quickest way to set temporary, named bookmarks if you don’t care to spend the time naming them yourself.
- **Use a key binding shortcut for the name** - Bookmarks can be named according to a specific key binding. For example, you could bind Ctrl+1 so that it instantly sets a bookmark named “1”.

Bookmarks can be set through a variety of methods, depending on which way you want to use them. A green **Bookmark** icon, displayed in the left margin of the editor window, indicates a set bookmark.

## Naming Bookmarks

To set a bookmark on the current line and give it a name, choose **Search > Set Bookmark** (**set\_bookmark** command). The Bookmarks dialog is displayed. Type the name of the bookmark in the combo box, then click **Add**.

You can also use the Bookmarks tool window to create named bookmarks. See [Bookmarks Tool Window](#) below.

## Command Line Shortcut: **sb**

Power programmers may prefer to use the **sb** command, which is a shortcut for **set\_bookmark**. You can append **sb** or **set\_bookmark** with any character or text string, and the bookmark will be instantly set using the value for the name. For example, **sb 1** will allow you to create an instant bookmark named “1”, and **sb main** will let you create an instant bookmark named “main”. See also [Command Line Shortcut: \*\*gb\*\*](#).

## Allowing Automatic Naming

If you want to quickly set a named bookmark, and you would prefer the editor to automatically name them for you, press **Ctrl+Shift+J**, or choose **Search > Toggle Bookmark** (**toggle\_bookmark** command). This command also instantly toggles the new named bookmark on/off.

When a bookmark is set in this manner, the name is automatically generated and appears in the Bookmarks tool window in one of two formats: **SymbolName:LineNumber**, or **FileName:LineNumber**. The symbol name is used if the bookmark is inside of a symbol. The file name is used if there is no symbol on the line or if the file does not support Context Tagging®.

## Using a Key Binding for the Name

You can set a bookmark that takes its name from the key used to set it. There are two commands that can be used: **alt\_bookmark**, for setting a bookmark, and **alt\_gtbookmark**, for navigating to the bookmark.

The purpose of these commands is so that you can bind them to keys, providing a way for you to have one type of keyboard shortcut for setting the bookmarks, naming them in the process, and another for navigating to the bookmarks. These commands can be bound to any of the following keys/ranges:

- Ctrl+0-9, Ctrl+A-Z, Ctrl+F1-F12
- Alt+0-9, Alt+A-Z, Alt+F1-F12
- Ctrl+Alt+0-9, Ctrl+Alt+A-Z, Ctrl+Alt+F1-F12
- Shift+F1-F12

For example, you could bind **alt\_bookmark** to Ctrl+0-9 and **alt\_gtbookmark** to Alt+0-9, for a more efficient means of setting bookmarks named 0-9, and navigating back to them.

## Navigating Named Bookmarks

To navigate between the set bookmarks, choose **Search > Previous Bookmark** (**prev\_bookmark** command) or **Search > Next Bookmark** (**next\_bookmark** command).

You can also use the Bookmarks tool window to navigate between named bookmarks. See [Bookmarks Tool Window](#) below.

### Command Line Shortcut: gb

Power programmers may prefer to use the **gb** command, a shortcut for **goto\_bookmark**. This will display the Go to Bookmark dialog, from which you can select a specific bookmark to navigate to. Append **gb** with the name value to go directly to that named bookmark. For example, if you set a bookmark named "1" (for instance, by using the command **sb 1**), type **gb 1** to navigate back to that location. See also [Command Line Shortcut: sb](#).

## Deleting Named Bookmarks

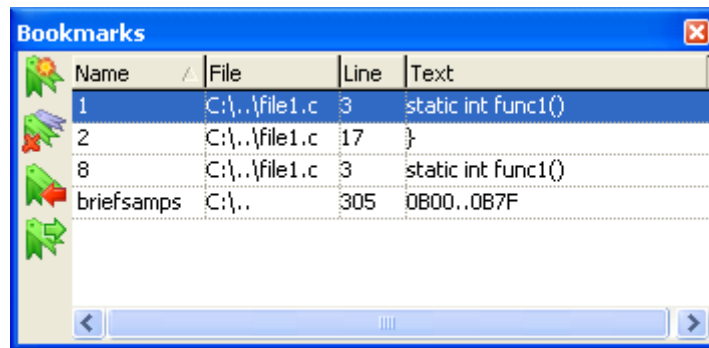
To remove a named bookmark, when the cursor is on the bookmark line, press **Ctrl+Shift+J** (or use the **toggle\_bookmark** command) to toggle the bookmark off. To remove all named bookmarks at once, use the **clear\_bookmarks** command.

Alternatively, the [Bookmarks Tool Window](#) contains options for deleting named bookmarks.

## Bookmarks Tool Window

The Bookmarks tool window (**Search > Bookmarks**, Ctrl+Shift+N, or **activate\_bookmarks** command), pictured below, is available for creating and managing named bookmarks. It contains a list of named bookmarks that have been set, showing each bookmark's name, file location, line number, and a preview of the text.

**NOTE** The tool window will show global bookmarks unless the option **Use workspace bookmarks** is selected on the [Search Tab](#) of the General Options dialog (**Tools > Options > General**).



Click on any header to sort the data by that type. Use the button controls to create a new bookmark, delete all bookmarks, or navigate between bookmarks. You can also jump to a bookmark by double-clicking the item. To delete an individual bookmark, select it and press the **Delete** key.

For more information about working with tool windows, see [Toolbars and Tool Windows](#).

## Pushed Bookmarks

Pushed bookmarks are stored on a stack. New bookmarks can be pushed onto the stack, preserving the current location. Popping the stack removes the top bookmark and navigates the cursor to the location of the previous bookmark. Pushed bookmarks do not have names and cannot be manipulated on the Bookmarks tool window.

### Pushing a Bookmark

To push a bookmark for the current line, choose **Search > Push Bookmark** or use the **push\_bookmark** command. Additionally, you can use the key binding **Ctrl+Dot** (**push\_tag** command) to move the cursor from a symbol to its definition, or **Ctrl+/** (**push\_ref** command) to navigate from a symbol to its reference, pushing a bookmark in the process. Pushing a bookmark will place the current line on the *bookmark stack*. A bookmark stack is simply an internal list of pushed bookmarks.

**NOTE** By default, **push\_bookmark** is not bound to a key. Key bindings can be set from the main menu by choosing **Tools > Options > Key Bindings**. For more information, see [Creating Bindings](#).

### Popping a Bookmark

Popping a bookmark will “pop back,” or return to, the location of the top bookmark pushed on the bookmark stack, removing the bookmark in the process. To pop a bookmark, press **Ctrl+Comma**. You can also use the menu item **Search > Pop Bookmark** or the **pop\_bookmark** command.

**NOTE** If the option **Automatically close visited files** is selected in the General Options dialog (**Tools > Options > General**, [General Tab](#)), the file will be closed if it is unmodified and the bookmark was created through symbol navigation. See [Symbol Navigation](#) for more information.

## Viewing Pushed Bookmarks

Pushed bookmarks do not appear in the Bookmarks tool window, and by default, no visual indicator is displayed in the editor window. To display a visual indicator, choose **Tools > Options > General**, select the

[Search Tab](#), then select the option **Show pushed bookmarks**. This will display a blue **Bookmark** icon in the left margin of the editor window for each new pushed bookmark.

## Setting Bookmark Options

Most bookmark options can be accessed from the main menu by choosing **Tools > Options > General**, then selecting the [Search Tab](#). See [Search Tab](#) for a listing of the available options.

Another option is available by setting a macro variable. SlickEdit® can push a bookmark whenever you jump to the top or bottom of the buffer (Ctrl+Home/Ctrl+End, or **top\_of\_buffer**/**bottom\_of\_buffer** commands, respectively). This is convenient, for example, in C++: if you jump to the top of the buffer to add a `#include` statement, a bookmark is pushed, so you can use Ctrl+Comma (**pop\_bookmark command**) to get back to your previous position. To use this option, select **Macro > Set Macro Variable**, and in the Variable combo box, select or type **def\_top\_bottom\_push\_bookmark**, then click **OK**. To turn off this option off, unset the variable by changing the **Value** to 0.

# Code Annotations

---

## Code Annotations Overview

Code Annotations provide a mechanism to store information about the code without actually modifying the code. Unlike code comments, code annotations are not stored in the source file but in an external file. The information you record is associated with a specific location in the code and can be viewed while you work on a source file.

You can use Code Annotations for recording various information, like comments about something that needs to be changed, tasks that need to be performed (see [Using Code Annotations to Record Tasks](#)), or comments in preparation for a code review (see [Using Code Annotations for Code Reviews](#)). Anything you can record in a code comment can be stored in a Code Annotation.

Because code annotations are not stored in the source code, you can use them to record private information you don't want to share with the rest of the team. Or you can use code annotations to record information that is shared with the team but should never be visible in the source code.

## Annotation Types

Each annotation contains a set of data fields specific to that type of annotation. All annotations contain a set of default fields, including the author who created or last edited this annotation, and the date this annotation was last changed. Code annotations used for different purposes also include specific fields related to that purpose. For example, a task annotation will also have a due date and a field to record the person who has been assigned this task. A code review annotation will include a text field for the proposed resolution and a status field indicating whether this change was accepted or rejected.

SlickEdit® provides some predefined annotation types, but you can create your own by specifying the set of fields contained. For each field, you specify the name of the field, the type of the field, and the default value. When an annotation of that type is created, you are prompted for these values. See [Managing Annotation Types](#) for more information.

## Purpose-Based Locations

When you create a code annotation, you specify where it is to be stored. You select the location based on the purpose of the annotation. The location can be one of these values:

- **Personal** - These annotations are for your own use and not intended to be shared with others. Personal annotations are stored in your configuration directory. They are not specific to any workspace.
- **Workspace** - Used to record information about files in the current workspace. The annotation file is stored in the same directory as your workspace file (`.vpsw`) and uses the same base file name but with a different extension. These are intended to be shared with anyone else working in this workspace, so the annotation file should be checked into source control.
- **Project** - Used to record information about files in a specific project. The annotation file is stored in the same directory as your project file (`.vpj`) and uses the same base file name but with a different extension. These are intended to be shared with anyone else working on this project.
- **User-Defined** - These annotations are stored in a file of your choice. These annotations cannot be readily shared since they store a full path to the referenced file. Unless two users have the same directory structure on their machines, the annotations will not be able to locate the referenced source files.

Workspace and Project annotations are very similar. If you don't create too many annotations, you could save all of your annotations as Workspace annotations. However, if you have projects that are shared between workspaces, you should use Project annotations so that the information is available regardless of which workspace you are using. If you create a lot of annotations, you may wish to use Project instead of Workspace so that you can view the list of annotations for a single project, providing a way to view a more manageable subset of annotations.

See [Managing Annotation Files](#) for more information.

## Private and Shared Annotations

Code annotations can be private or shared. Annotations are shared by sharing the annotation file with other users. Like source files, annotation files are most easily shared using a version control system.

Only Workspace and Project annotations can be shared effectively. Because they refer to files in the workspace or project, the path to those files is stored relative to the workspace or project. Even if users have the same workspace in different directories, SlickEdit® will be able to locate the source files referenced by annotations. Personal annotations and User-Defined annotations are not sharable. They store a fully qualified path to the referenced file. Unless two people have the same directory hierarchy on their machines, SlickEdit will not be able to locate the referenced source files.

## Relocatable Code Marker

Code Annotations use a relocatable code marker to store the location within the source code. This allows SlickEdit® to find the new location if someone makes edits to the file externally, like editing the file with a different editor. The next time you open the file, SlickEdit checks the location of each code marker and verifies that it is still correct. If necessary, SlickEdit uses stored information to locate the correct line of code for this annotation. If the code has changed too much, SlickEdit may not be able to find the new location, and you will be prompted to find the new location yourself. This will only happen when the code near the code marker has been heavily edited.

## Annotations Tool Window and File Manager

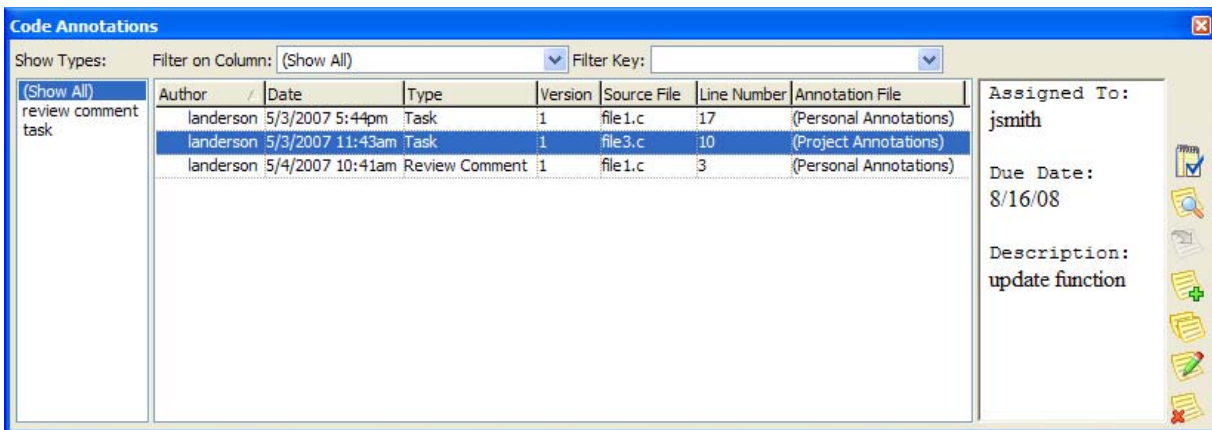
SlickEdit® provides a Code Annotations tool window that lists all of the currently visible annotations. This tool window allows you to filter the set of visible annotations. The primary filter is by type. If annotations of more than one type are displayed, only the default fields common to all annotations are displayed in the annotation list. You can use the tool window to create, edit, and delete code annotations as well. See [Managing Annotations](#) for more information.

Code Annotations are stored in files, and they become visible when the annotation file is opened. SlickEdit automatically opens the annotations from your personal annotations file along with those from the workspace and project annotation files. You can use the Annotation File Manager to open any other annotation files you like. See [Managing Annotation Files](#) for more information.

## Managing Annotations


The Code Annotations tool window provides a detailed view of annotations that you have created as well as operations for adding, modifying, copying, moving, and removing annotations. From this window, you can also manage annotation files and create your own annotation types.

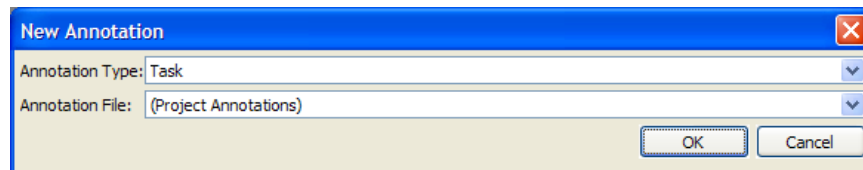
To display the tool window, from the main menu select **View > Toolbars > Code Annotations**, or use the **annotations\_browser** command on the SlickEdit® command line.



## Creating Annotations

Creating a code annotation is similar to setting a bookmark or breakpoint. To create a new annotation:

1. From within the editor window, position the cursor on the desired line of code.
2. There are three methods for initiating a new annotation, after which the New Annotation dialog is displayed:
  - o Right-click and select **Create Code Annotation**.
  - o Click the **Add**  button on the Code Annotations tool window.
  - o Use the **new\_annotation** command on the SlickEdit® command line.




3. On the New Annotation dialog, select an **Annotation Type** from the drop-down list.
4. Select where the annotation is to be stored from the **Annotation File** drop-down list (see [Managing Annotation Files](#)), then click **OK**.
5. SlickEdit displays a dialog with fields matching the selected annotation type. Some of the fields are common to all annotation types, while others are specific to the chosen annotation type. Some of the values, like author, creation date and time, and source file where the annotation marker is located, are presented read-only, since they cannot be changed. Each dialog also contains specific fields applicable to the annotation type:
  - o **Comment** - The Comment Annotation dialog provides a box for typing a text comment.
  - o **Task** - The Task Annotation dialog provides boxes for typing the name of the person this task is assigned to, the due date, and a description. Note that the **Due Date** field allows any text input.
  - o **Review Comment** - The Review Comment Annotation dialog provides boxes for typing an issue, a resolution, and a status.



6. After entering the annotation details, click **OK**.

## Viewing and Filtering Annotations

Annotations in your source code are indicated with a yellow Annotation bitmap  in the left margin of the editor window. Hover the mouse over a bitmap to see a preview of the annotations on that line.

The Code Annotations tool window is used to view and work with a list of your annotations and displays them in a table format, by default, in the order of last operation. Click on any column header to sort by that column. When you sort, an arrow on the column header shows the ascending or descending order. Drag the column size bars to resize columns to your desired width.

The **Show Types** area lets you choose what types of annotations to display in the tool window. Click and drag the separator bar to resize this area.

A preview pane, located on the right side of the tool window, shows the details of the selected annotation. This pane can also be resized by clicking and dragging the separator bar.

By default, the tool window view is set to show all annotations. To view only annotations of a specific type, select it in the **Show Types** list. Click **(Show All)** to display all annotations again. When **(Show All)** is selected, only the default fields common to all annotations are displayed in the annotation list.


To filter the list of visible annotations, use the filter boxes at the top of the tool window. Use the **Filter on Column** drop-down list to select a column to filter by. When you select a column, the **Filter Key** drop-down list is populated with all of the entries for that column, and you can select the item that you want displayed. For example, to see all annotations that share the same author, select **Author** in the **Filter on Column** drop-down, then select the author's name from the **Filter Key** drop-down list.

Double-click on an annotation in the tool window to go to the location of the annotation in the source code. You can also select **Go to Annotation** from the right-click context menu, or use the **show\_annotation\_source** command on the SlickEdit® command line. Note that this command only works from within the tool window.


## Copying and Moving Annotations


### Copying

You can create a new annotation by copying an existing one. This is useful if you have similar lines of code that need to have similar annotations. From within the Code Annotations tool window, select the code

annotation to copy, then click the **Copy** button . Alternatively, right-click on a selected annotation and select **Copy**. When you copy an annotation, your name is assigned as the author.

### Moving

Click the **Relocate** button  on the tool window to move the selected annotation to the current line in the current file.


Alternatively, you can click and drag the yellow Annotation bitmap  in the editor window margin to the desired location. If the line contains multiple annotations, the first annotation is moved.

**NOTE** You cannot move annotations between files. If the selected annotation is not located in the current file, the **Relocate** button on the tool window is unavailable.




## Editing Annotations


SlickEdit® allows you to edit the data that was entered for existing annotations. You cannot change the original source file, author, date, or annotation file, but the data entry fields can be edited.

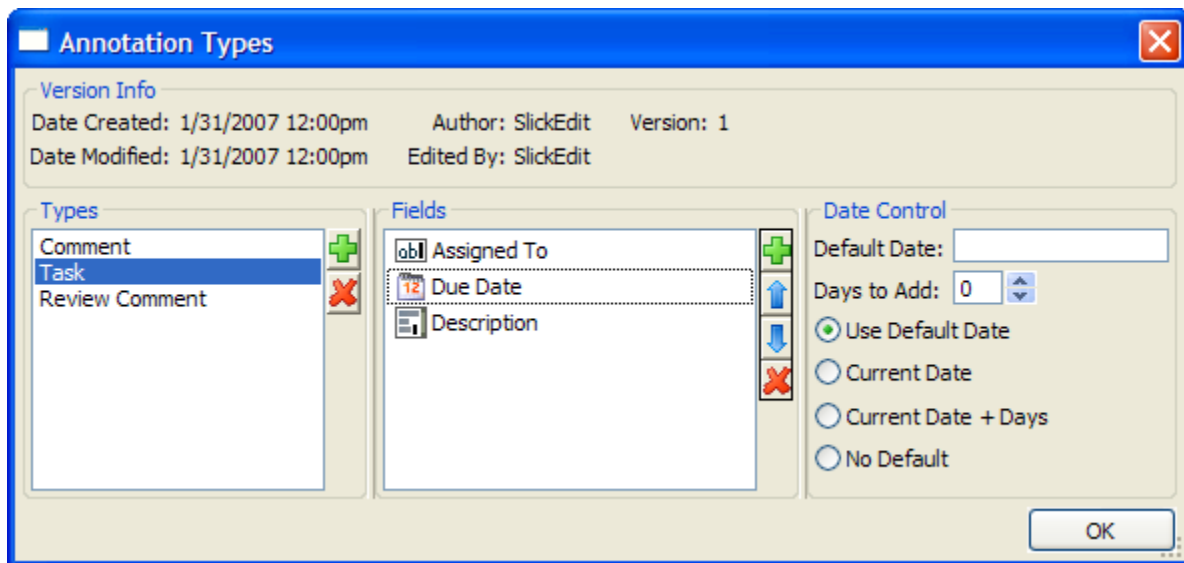
To edit an existing annotation, from within the Code Annotations tool window, select the annotation to edit then click the **Edit** button . You can also right-click in the tool window and select **Edit** or use the **edit\_annotation** command on the SlickEdit command line. Note that this command will only work for an item selected in the tool window.

## Deleting Annotations

To delete an annotation, from within the Code Annotations tool window, select the annotation to delete then click the **Delete** button . You can also right-click and select **Delete** or use the **delete\_annotation** command on the SlickEdit® command line. Note that this command will only work for an item selected in the tool window.

## Managing Annotation Types

SlickEdit® provides several predefined annotation types: **Comment**, **Task**, and **Review Comment**. To define your own annotation types, from the Code Annotations tool window click the Annotation Types button , or use the **annotation\_definitions** command on the SlickEdit command line. The Annotation Types dialog is displayed.



Creating a new annotation type is similar to creating a form. First you define a name, then you define the fields. The name that you give the new type will be the title of the dialog that appears when you create a new annotation of that type. The fields that you define will be the fields that are available on the dialog.

To create a new annotation type:

1. In the **Types** area, click the green **Plus** button.
2. Type a name for your new annotation type.

3. In the **Fields** area, click the green **Plus** button.
4. From the **Field Type** drop-down list, choose the type of control you want to use, then click **OK**.
5. The third area on the Annotation Types dialog is now populated with the Field Type that you just added. Click the green **Plus** button in this area to define values for the control.
6. Repeat Steps 3 through 5 to add more fields to your new type. Use the **Up/Down** arrows to control the order in which the fields should appear on the form.
7. Repeat these steps to define additional types.
8. Click **OK** on the Annotation Types dialog when finished.

Now, when you create a new code annotation, you can use the newly defined type(s) that will be displayed in the **Annotation Type** drop-down list on the New Annotation dialog.

To delete the selected type or a field, click the red **X** button next to the Types or Fields area.

**NOTE** In this version of Code Annotations, the **Date Control** field type just stores dates as text. Columns of this type are sorted lexicographically, not by date. We plan to introduce true date processing in a subsequent release. In the meantime, you can get the best result for dates by storing them in this format: yyyy/mm/dd. For example, 2007/07/04. This will allow dates to sort correctly.

## Handling Annotation Type Conflicts

Every annotation file contains the annotation type definitions for the annotations it contains. If a user modifies the type definition for an annotation, it will conflict with the type definition for annotations in other files. All annotation types by the same name must have the same definition. SlickEdit® detects any discrepancies between types and tries to rectify them.

When two conflicting types are found, SlickEdit prompts you for which definition to use. Once you have selected the master type, SlickEdit attempts to correct annotations that matched the other type. The resolution depends on the category of the change:

1. **Field added** - The new master type contains a field not present in the other version of this annotation type. SlickEdit will add the new field to all annotations of that type the next time they are saved.
2. **Field deleted** - The new master type is missing a field that is present in the other version of this annotation type. SlickEdit will delete that field from all annotations of that type the next time they are saved. This will result in the data in that field being lost.
3. **Field modified** - The new master type contains a field by the same name as a field in the other annotation type but the definition of the two fields differ. SlickEdit will attempt to coerce the data from the one type to the other. Since all data is stored as text, this will work in many cases. In some cases this will result in data loss. For example, if data is coerced from a text type to a discrete type (like Dropdown, List, or Checkbox), data will be lost if the text value doesn't match one of the predefined values for this control.

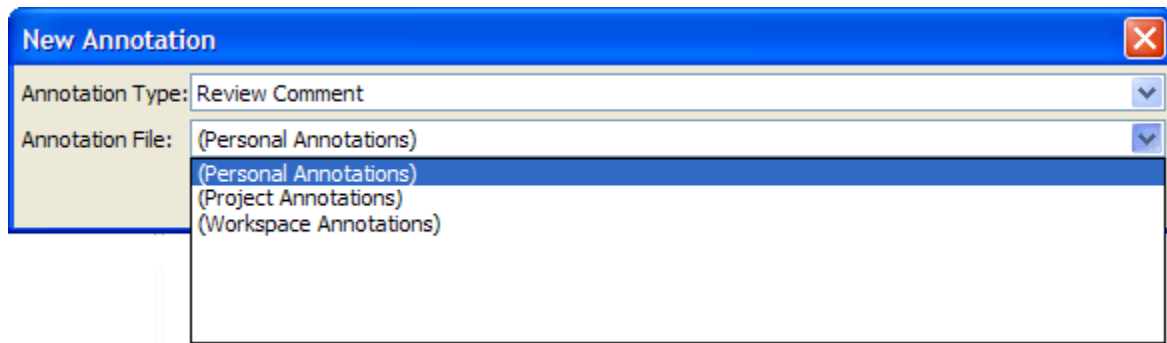
This system for resolving conflicts in Annotation Types does not allow you to merge two sets of changes to the same type. For example, if two users both add a field to the same type, one of them will be lost. Because of this, changes to annotation types should be made in a deliberate, planned manner and rolled out to the team in a way that avoids this issue. When a shared annotation type needs to be changed, have one person make that change and distribute a document with that type. The rest of the team can update their annotation files by opening that document while other documents are open. Remember, there is no centralized repository for annotation types, so conflict resolution is only performed on the set of open

documents. To update multiple workspaces, you will have to open each workspace and then open the document containing the changed type.

To avoid conflicts in your personal annotation types, you may want to create your own types. Conflict resolution is performed using the name of the annotation type to uniquely identify a type. If the types you use in your personal annotations have their own, unique names then you can avoid conflicts with annotation types in other documents.

## Managing Annotation Files

Code annotations are stored in files with a `.sca` extension. There are four types of locations for these files, based on the purpose of the annotation: Personal, Workspace, Project, and User-Defined. Each of these are “aliases” that correspond to a path on your computer where the annotation file is stored. You specify the location when you create new code annotations by selecting from the Annotation File drop-down list on the New Annotation dialog.



### Personal Annotations

Personal annotations are for your own private use and are not specific to any workspace or project. They are stored in the `personal.sca` file located in your configuration directory.

### Workspace Annotations

Workspace annotations are used to record information about files in the current workspace. They are intended to be shared with others using the same workspace, so you can check the annotation file into source control. These annotations are stored in a `.sca` file located in the same directory as your workspace file (`.vpw`). The base file name is the same as your workspace base file name, except it is appended with “\_workspace”. For example, if your workspace is named `Diff.vpw`, your Workspace annotations are stored in `Diff_workspace.sca` in the same directory.

### Project Annotations

Project annotations are used to record information about files in projects. They are also intended to be shared with others using the same project, and can be checked into source control. Project annotations are stored in a `.sca` file located in the same directory as your project file (`.vpj`). The base file name is the same as your project base file name, except it is appended with “\_project”. For example, if your project is named `Diff.vpj`, your Project annotations are stored in `Diff_project.sca`.

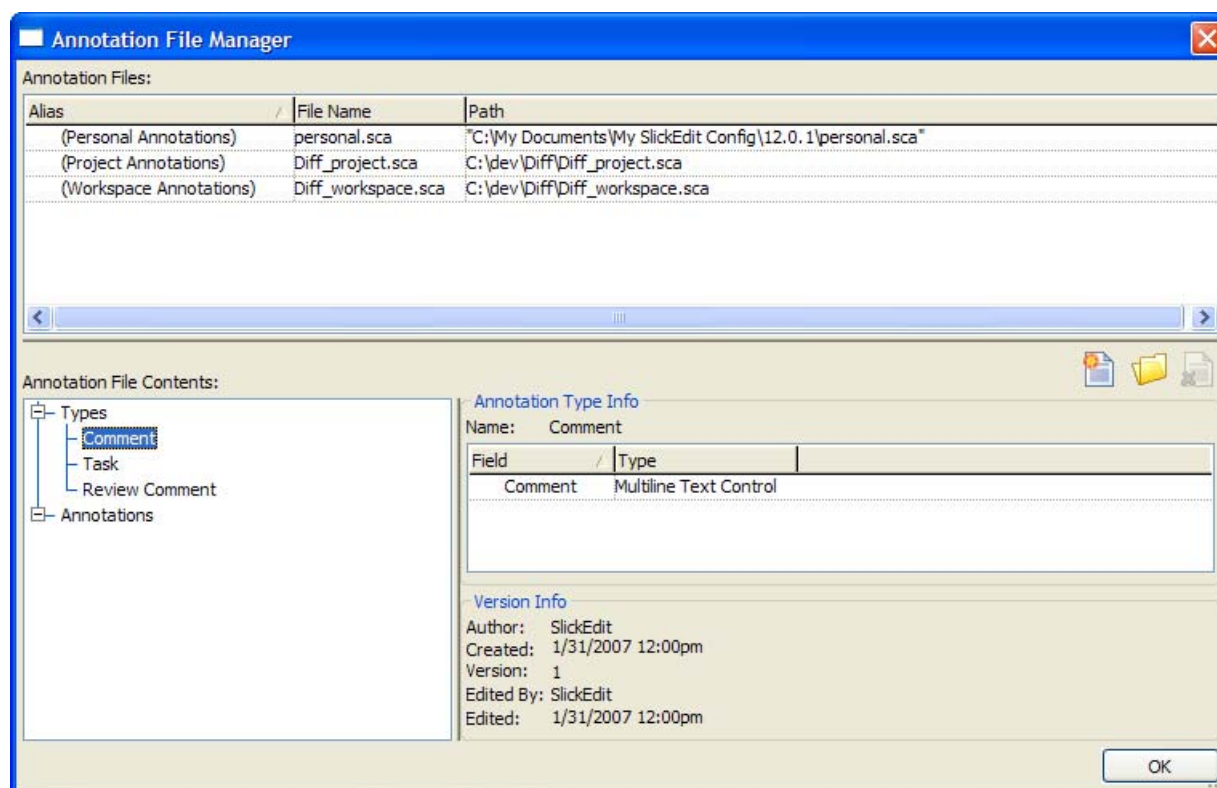
## User-Defined Annotations

User-defined annotations are stored in a file of your choice, and can be kept private or you can share them if you want. You will need to specify a location path for user-defined annotations prior to creating new annotations for this location type. This is done through the [Annotation File Manager](#).

## Annotation File Manager

You can view annotation file information and add, open, or close user-defined annotation files by using the Annotation File Manager. It can be accessed from the Code Annotations tool window by clicking the


Annotation Files  button.





The top of the dialog contains a list of all annotation files that are currently open, showing the alias, the name of the file, and the location path. Click on any column header to sort by that column. An arrow in the header indicates the ascending or descending sort order. Click and drag to resize any column. You can also click and drag the horizontal separator bar to resize the entire Annotation Files area.

Personal, Workspace, and Project annotation files are always open and displayed. User-defined annotation files are also displayed if you have created them and they are open.

Use the buttons on the Annotation File Manager to perform the following operations on User-defined annotations:

- Click the **New Annotation File** button  to create a new User-Defined annotation file. The New dialog is displayed where you can specify the name and path of the new file.

- Click the **Open Annotation File** button  to open the selected User-Defined annotation file. The Open dialog is displayed where you can specify an annotation file to open (note that annotation files have the extension `.sca`).
- Click the **Close Annotation File** button  to close the selected User-Defined annotation file. Closing an annotation file removes its associated types from the list when creating new annotations, and also removes the margin indicators for the associated annotations. This is useful to prevent cluttering, should the annotation file have many types or many annotations.

The bottom of the Annotation File Manager contains details about the selected file. The **Annotation File Comments** area shows a tree view of the annotation types and the specific annotations that are contained in the file. Click and drag the separator bar to resize this area. When you select a task or annotation in this list, details are shown on the right. If a task type is selected, the dialog displays the fields and details for that type. If an annotation is selected, the dialog displays the contents of that annotation.

The **Version Info** area displays the version information for the selected type or annotation, include the original author, date of creation, version number, and the author and date of the last edit, if applicable.

## Using Code Annotations to Record Tasks

Code annotations provide a convenient mechanism to record tasks associated with specific locations in the code. You can use Personal annotations to record your own tasks, or use shared annotations to assign tasks to another team member.

SlickEdit® includes an Annotation Type for tasks, called **Task**. Along with the standard fields, it adds three additional fields: **Assigned To**, **Due Date**, and **Description**. Follow the instructions in [Creating Annotations](#) to create a new task annotation, and select the **Task** annotation type.

If this is a personal task, select **(Personal Annotations)** for the location value. If this task is for another team member or you want others to see this task, select **(Workspace Annotations)** or **(Project Annotations)** for the location. See [Managing Annotation Files](#) for more information about annotation file locations.

## Using Code Annotations for Code Reviews

Code Reviews provide an excellent use for Code Annotations. In a typical code review process, code is reviewed by a number of team members, who record issues and forward their comments to a review coordinator. During the review, the comments are discussed and a resolution is recorded.

SlickEdit® includes an Annotation Type for code reviews, called **Review Comment**. Along with the standard fields, it also includes **Issue**, **Resolution**, and **Status**. If your review process requires a different set of fields, refer to [Managing Annotation Types](#) for information on how to create your own.

There are many ways to implement a code review using SlickEdit and Code Annotations. We will describe one approach that should be the easiest. You may find other methods that work best with your processes.

Prior to the review, the review coordinator creates a Workspace in SlickEdit for the files to be reviewed. Copy those files to the Workspace directory and then add the files to the Workspace. If you have modified the stock definition of the Review Comment, you can make sure that everyone is using the same definition by creating a Workspace Annotation using that definition. You might put one on the first line of code with instructions for how to perform the review.

Send a copy of this workspace to each of the reviewers. This can be done by compressing the directory into a ZIP, TAR, or other archive file. Each reviewer then reviews the code and records their comments

## CODE ANNOTATIONS

using the Review Comment annotation type as a Workspace annotation. When they are finished, they send the Workspace's `.sca` file back to the review coordinator.

The review coordinator will merge the `.sca` files together to produce a single workspace annotation file. Since these are XML files, they are easy to read and merge using many different tools. The consolidated annotation file is used for the review walkthrough. You can also accomplish the sharing and merging with most source control tools. Many provide automatic merging capabilities that will add the inserted lines into the master file. You'll have to test your system to make sure that the merges it performs are safe. If not, you can use SlickEdit's [DIFFzilla®](#) to compare and merge the files.

During the review meeting, the review coordinator opens the single, merged document and then walks through the issues by double-clicking on each in the Code Annotation tool window. The review team discusses the recorded comment, and appropriate resolution remarks and status are recorded for each.

# Commenting

---

SlickEdit® makes commenting your code easy. You can comment out selected text, or type the start characters for a new doc comment and have the doc comment skeleton automatically expanded. SlickEdit also makes your comments easier to read by automatically wrapping them as you type. Existing comments can be “reflowed” to match current comment wrap settings.

## Commenting Blocks and Lines

Existing text in your code can be commented out (or uncommented) as follows:

- To comment out a selected code block, from the main menu, click **Document > Comment Block** (or use the **box** command). This comments out the entire selection as a single block comment by surrounding the block with comment characters you have specified in your comment settings.
- To comment out selected lines, from the main menu, click **Document > Comment Lines** (or use the **comment** command). Each line in the selection is commented out as a single line comment. If there is no selection, the current line is commented out. If using a block selection where there are partially selected lines, comment characters are placed at the beginning and end of the selection. If using a character selection where there are partially selected lines, comment characters are placed based on your settings. The comment characters that are placed to the left and right of the text are also specified in your comment settings.
- To uncomment lines in a selection, from the main menu, click **Document > Uncomment Line** (or use the **comment\_erase** command). Surrounding line comment characters are removed from the line. If there is no active selection, the current line will be uncommented. Uncomment Line only works for well-formed comments, which means that every line in the selection is commented and that the comment characters occur in the same column.

Whether you are creating a comment block or a comment line, if the selected text already contains comments, another set of comment characters is added. SlickEdit® attempts to preserve the indentation level of the code and any existing comments when adding or removing comment characters.

## Comment Block and Line Settings

To specify the characters and other settings used for comment blocks and lines, from the main menu, click **Document > Comment Setup** (or use the **comment\_setup** command). The Extension Options dialog is displayed open to the [Comments Tab](#). Select the extension you wish to affect from the **Extension** drop-down list.

The **Comment block** group box provides eight fields to specify the characters used in your commenting style. If you want to apply a comment with no additional decoration, fill in the upper-left and lower-right fields with the characters to begin and end a block comment. To draw a box around the comment, fill in additional characters in the other fields. For example, you might put an asterisk in each of the other fields to draw a box of asterisks around the block comment.

The **Comment line** group box contains fields for you to specify the characters to be inserted at left and right sides of a line comment.

For code examples and descriptions of the other available options, see [Comments Tab](#).

## Creating Doc Comments

Doc comments are specially formatted comments that are processed by tools that extract and present the information in a formatted manner. Doc comments follow a predefined structure, based on the programming language and the tool processing the comments.

SlickEdit® supports the most common doc comment formats (Javadoc, XMLdoc, and Doxygen). When you type the start characters for one of these comment formats and press **Enter** on a line directly above a function, class, or variable, SlickEdit can automatically insert a skeleton doc comment for that style.

**NOTE** In C#, you do not need to press **Enter**, as the skeleton comment is inserted after you type the third slash.

To enable and configure automatic completion of doc comment skeletons, complete the following steps:

1. From the main menu, click **Document > Comment Setup** (or use the **comment\_setup** command). The Extension Options dialog is displayed, open to the [Comments Tab](#).
2. Select the extension you want to affect from the **Extension** drop-down list.
3. In the **Doc comments** box, check the option **Automatically expand doc comments**.
4. In the **For start characters** box, select the start characters for the doc comment style you plan to use for the selected extension. These are the characters that you type that will trigger automatic completion of the doc comment skeleton. For comments formatted in Javadoc, select **/\*\***. For XMLdoc, select **///**. For Doxygen, select **/\*!** or **///  
/**.
5. In the **Use style** box, select the tag style to use for the corresponding start characters that you selected in Step 4. This tag style is used when SlickEdit creates skeleton doc comments beginning with the matching start characters. Comments formatted in Javadoc usually use the **@param** style. XMLdoc uses the **<param>** style. Doxygen can read the **\param** style.

**TIP** You can repeat Steps 4 and 5 to assign a style for each start character set, and the setting will be remembered.

6. Click **OK** on the Extension Options dialog.

## Doc Comment Examples

### Javadoc Format

To use the Javadoc commenting format, select the start characters **/\*\*** and use style **@param**. Check **Insert leading \***. Using the following code sample:

```
/**[CURSOR_HERE]*/
int setDimensions(int length, int width, int height) {
    ...
}
```



Pressing **Enter** at the “cursor here” location results in the following automatic completion:

```
/**
 * [CURSOR_HERE]
 *
 * @param length
 * @param width
 * @param height
 *
 * @return int
 */
int setDimensions(int length, int width, int height) {
    ...
}
```

## XMLdoc Format

To use the XMLdoc comment format, select the start characters **///** and the **<param>** style. Using the following code sample:

```
///[CURSOR_HERE]
int setDimensions(int length, int width, int height) {
    ...
}
```

Pressing **Enter** at the “cursor here” location results in the following automatic completion:

```
/// <summary>
/// [CURSOR_HERE]
/// </summary>
/// <param name="length"></param>
/// <param name="width"></param>
/// <param name="height"></param>
/// <returns>int</returns>
int setDimensions(int length, int width, int height) {
    ...
}
```

## Doxygen Format

To use a Doxygen comment format, select the start characters **/\*!** or **/\*\*** (based on your preference) and the **\param** style. Using the following code sample:

```
/*![CURSOR_HERE]*/
int setDimensions(int length, int width, int height) {
    ...
}
```

Pressing **Enter** at the “cursor here” location results in the following automatic completion:

```

/ *!
 * [CURSOR_HERE]
 *
 * \param length
 * \param width
 * \param height
 *
 * \return int
 */
int setDimensions(int length, int width, int height) {
    ...
}

```

## String Editing

When the cursor is inside of a string, if you press Enter to split the line, SlickEdit® can automatically align the string with the original string as well as insert the closing and opening quotes and, if necessary, operators. To set this option, choose **Document > Comment Setup** (**comment\_setup** command). The Extension Options dialog is displayed, open to the [Comments Tab](#). Select the extension you wish to affect from the **Extension** drop-down list, then select **Split strings on Enter**.

## Comment Wrapping

Comments can be set to automatically wrap to the next line as you type. This feature is available for C, C++, C#, Java, and Slick-C® files.

To enable comment wrapping, from the main menu, click **Tools > Options > File Extension Setup**, select the extension you want to affect from the **Extension** drop-down list, then select the **Comment Wrap tab**. Check the option **Enable comment wrap**, then select the type of comments you want wrapped (block comments, line comments, and/or doc comments).

The Comment Wrap tab also provides options to control how comments are wrapped. There are three types of width settings:

- **Fixed** - Comments will be formatted to a specified width.
- **Automatic** - Comments will be formatted according to the width of existing comments.
- **Fixed right margin** - Lines will break before a specified number of columns has been reached.

For more details on comment wrapping configuration, see [Comment Wrap Tab](#).

## Reflowing Comments

After configuring comment wrap settings, you can use the Reflow Comment dialog to reflow block comments, paragraphs, or a selection of the current file. To display this dialog, choose **Document > Reflow Comment**. For more information on the available options, see [Reflow Comment Dialog](#).

# Find and Replace

---

SlickEdit® provides several different ways to search and replace:

- For the fastest method of searching and replacing, use Quick Search and Quick Replace (see [Quick Search and Replace](#) below).
- To find and replace text “on the fly,” or, as you type, use incremental searching (see [Incremental Searching](#) below).
- If you are more comfortable with keystrokes, you may prefer command line searching with the find and replace commands (see [Find and Replace Commands](#)).
- Use the Find and Replace tool window if you prefer working within an interface (see [Find and Replace Tool Window](#)).
- To search for symbols, use the Find Symbol tool window (see [Find Symbol Tool Window](#)).

Both the Find and Replace tool window and command line searching provide the same search and replace options for single or multiple files, and for searching and replacing text, wildcards and regular expressions, so you can choose which method works best for you.

This section also includes the topics [Find and Replace with Regular Expressions](#) and [Undoing/Redoing Replacements](#).

## Quick Search and Replace

### Quick Search

The fastest way to search is by using Quick Search. Quick Search looks through the current buffer for the word or selection at the cursor. You can find the next occurrence of a search item by selecting a string in an existing buffer or Search Results window, then selecting **Quick Search** from the right-click menu (or by using the **quick\_search** command). The commands **find\_next** (**Search > Next Occurrence** or Ctrl+G) and **find\_prev** (**Search > Previous Occurrence** or Ctrl+Shift+G) will find the next and previous instances of the item, respectively.

### Quick Replace

Quick Replace gets the current word or selection at the cursor, prompts for replacement text on the command line, then highlights each occurrence of the word and prompts if you want to replace the text.

To use Quick Replace, right-click on any word or selection and select **Quick Replace** (or use the **quick\_replace** command).

The **quick\_replace** command has a command line alias, **qr**. The **qr** command takes the replace string as an argument. For example, if the cursor is on the word “cat,” the command **qr dog** will prompt you to replace all the instances of “cat” with “dog” in the current buffer.

## Incremental Searching

During incremental searching, a string is searched for as it is typed. To start a forward incremental search using the command line, use the **i\_search** command (Ctrl+I). To start a reverse incremental search, use the **reverse\_i\_search** command (Ctrl+Shift+I).

## FIND AND REPLACE

The following key combinations (based on the default CUA emulation) take on a different definition during an incremental search:

Keys	Function
Ctrl+R	Search in reverse for the next occurrence of the search string.
Ctrl+S	Search forward for the next occurrence of the search string.
Ctrl+T	Toggle regular expression pattern matching on/off.
Ctrl+W	Toggle word searching on and off. To change the word characters for a specific extension, from the main menu, choose <b>Tools &gt; Options &gt; File Extension Setup</b> . Choose your extension from the <b>Extension</b> drop-down list, then select the <a href="#">Advanced Tab</a> .
Ctrl+Shift+W	Copy complete word at cursor to search string.
Ctrl+C	Toggle case sensitivity. The key bound to the Brief emulation command <b>case_toggle</b> will also toggle the case sensitivity.
Ctrl+M	Toggle searching within selection.
Ctrl+O	Toggle incremental search mode.
Ctrl+Q	Quote the next character typed.
Ctrl+S or F5	(Brief emulation) Search forward for the next occurrence of the search string.
Ctrl+R or Alt+F5	(Brief emulation) Search in reverse for the next occurrence of the search string.
Ctrl+W	(GNU Emacs emulation) Copy complete word at cursor to search string.
Ctrl+Shift+W	(GNU Emacs emulation) Toggle word searching on and off.

Incremental searching stops when you press a key that does not insert a character. You can press Esc to terminate an incremental search (only during prompting). Press and hold Ctrl+Alt+Shift to terminate a long search.

You can retrieve your previous search string by invoking the **i\_search** or **reverse\_i\_search** command and pressing Ctrl+S or Ctrl+R, respectively, before entering a search string.

## Find and Replace Commands

### Find and Slash (/) Commands

The command line is available for performing searches. You can use the forward slash (/) or **find** commands which provide the same functionality as the Find and Replace tool window. Press **Esc** to toggle the cursor to the command line.

The syntax of the slash (/) command is:

`/string[/OptionCharacters]`

The syntax of the **find** command is:

**find** `/string[/OptionCharacters]`

**OptionCharacters** is one or more of the following option characters:

Option Character(s)	Description
E	Exact case.
I	Ignore case.
-	Reverse search.
M	Limit search to selection.
<	If found, place cursor at beginning of word.
>	If found, place cursor at end of word.
R	Interpret search string as a SlickEdit® regular expression.
U	Interpret search string as UNIX regular expression.
B	Interpret string as a Brief regular expression.
N	Do not interpret search string as a regular expression.
P	Wrap to beginning/end when string not found.
W	Limit search to words. Used to search for variables.
W=SlickEdit-regular-expression	Specifies the valid characters in a word. The default value is [A-Za-z0-9_\$]. To change the word characters for a specific extension, from the main menu choose <b>Tools &gt; Options &gt; File Extension Setup</b> , select your extension from the <b>Extension</b> drop-down list, then select the <a href="#">Advanced Tab</a> .
W:P	Limit search to word prefix. For example, a search for “pre” matches “pre” and “prefix” but not “supreme” or “supre.”
W:PS	Limit search to strict word prefix. For example, a search for “pre” matches “prefix” but not “pre,” “supreme” or “supre.”
W:S	Limit search to word suffix. For example, a search for “fix” matches “fix” and “suffix” but not “fixit.”
W:SS	Limit search to strict word suffix. For example, a search for “fix” matches “suffix” but not “fix” or “fixit.”
H	Allow finding search string in hidden lines.

Option Character(s)	Description
Y	Binary search. This allows start positions in the middle of a DBCS or UTF-8 character. This option is useful when editing binary files (in SBCS/DBCS mode) which may contain characters which look like DBCS but are not. For example, if you search for the character “a”, it will not be found as the second character of a DBCS sequence unless this option is specified.
, (comma)	Delimiter to separate ambiguous options.
<i>XCCLetters</i>	Requires the first character of search string NOT be one of the color coding elements specified. For example, XCS requires that the first character not be in a comment or string. <i>CCLetters</i> is a string of one or more of the following color coding element letters: OOther KKeyword NNumber SString CComment PPreprocessing LLine number 1Symbol 1 2Symbol 2 3Symbol 3 4Symbol 4 FFunction color VNo save line AAttribute
<i>CCLetters</i>	Requires the first character of search string to be one of the color coding elements specified. See <i>CCLetters</i> above.
#	Highlight matched patterns with highlight color. The buffer is not automatically cleared when executing a new search, like it is with the Find and Replace tool window.
*	Used with the “Search hidden text” ( <b>H</b> ) or “Highlight matches” ( <b>#</b> ) options to find all matches and un-hide or highlight them.
&	Use Wildcard regular expression syntax (*, ?).
#	Highlight matched patterns with highlight color.
V	(Replace commands only) Preserve case. When specified, each occurrence found is checked for all lowercase, all uppercase, first word capitalized, or mixed case. The replace string is converted to the same case as the occurrence found except when the occurrence found is mixed case (possibly multiple capitalized words). In this case, the replace string is used without modification.

Option Character(s)	Description
\$	(Replace commands only) Replaced occurrences are highlighted with modified color.

To set default search options, see [Search Tab](#).

If the “\*” option is not specified, you will be prompted with the message “Yes/No/Last/Go/Quit/Suspend?” for each occurrence of the “Search for” string.

## Replace and c Commands

The replace commands, **replace** and **c**, can be used in the command line. The syntax of these commands is:

**c/string1/string2/options** or **replace/string1/string2/options**

The available options are the same as for the **find** and slash (/) commands (see [Find and Slash \(/\) Commands](#) above).

You can perform one of the following actions with the replace command (**c**) by pressing the corresponding key:

Key	Action
<b>Y or Space</b>	Make change and continue searching.
<b>N or Backspace</b>	No change and continue searching.
<b>L or Dot</b>	Make change and stop searching.
<b>G or !</b>	Make change and change the rest without prompting.
<b>Q or Esc</b>	Exit command. By default, the cursor is NOT restored to its original position. If you want the cursor restored to its original position, from the main menu choose <b>Tools &gt; Options &gt; General</b> , then select the <a href="#">Search Tab</a> . Select the <b>Restore cursor after replace</b> option.
<b>Ctrl+G</b>	Exit command and restore cursor to its original position.
<b>Ctrl+R</b>	Search in reverse for next occurrence of search string.
<b>Ctrl+S</b>	Search forward for next occurrence of search string.
<b>Ctrl+T</b>	Toggle regular expression pattern matching on/off. The key bound to the Brief emulation command <b>re_toggle</b> will also toggle regular expression pattern matching.
<b>Ctrl+W</b>	Toggle word searching on/off. To change the word characters for a specific extension, from the main menu choose <b>Tools &gt; Options &gt; File Extension Setup</b> , select your extension from the Extension drop-down list, then select the <a href="#">Advanced Tab</a> .

Key	Action
<b>Ctrl+C</b>	Toggle case-sensitivity. The key bound to the Brief emulation command <b>case_toggle</b> will also toggle the case-sensitivity.
<b>Ctrl+M</b>	Toggle searching within selection.
<b>F1 or ?</b>	Display help on Find and Replace tool window.

## Replace Command Search Examples

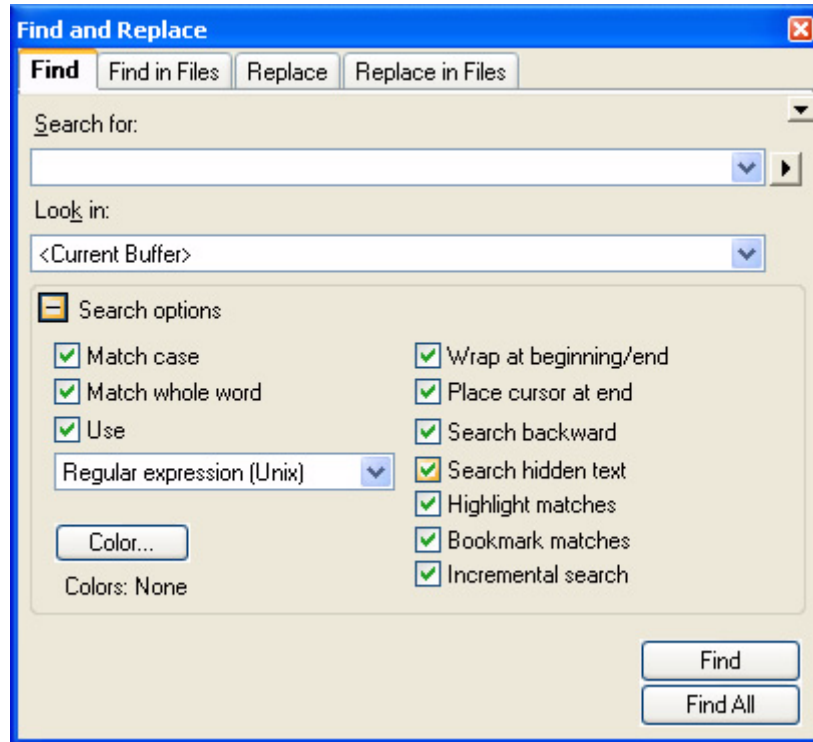
The table below provides examples of using command line replace.

Example	Description
c \$/\$\$	Replace occurrences of forward slashes with back slashes.
c/x/y/m	Replace occurrences of “x” in the selected area with “y” using default search case sensitivity.
c \$x\$y\$m	Replace occurrences of “x” in the selected area with “y” using default search case sensitivity. The string delimiter “\$” has been used requiring a space character after the “c.”
c/x/y/e*	Replace lowercase occurrences of “x” with “y” without prompting.
c/i/something_more_meaningful/w	Replace occurrences of the variable “i” with “something_more_meaningful.”
c/i/j/w=[A-Za-z]	Replace occurrences of the word “i” with “j” and specify valid characters in a word to be alphabetic characters.
replace/Test/TEMP/v	<p>Replace occurrences of the word “test” with the word “temp”, with the case preserved. For example:</p> <ul style="list-style-type: none"> <li>Occurrences of “Test” are replaced with “Temp”.</li> <li>Occurrences of “test” are replaced with “temp”.</li> <li>Occurrences of “tesT” are replaced with “TEMP” (because a mixed case will retain the actual replacement that you typed).</li> <li>Occurrences of “TEST” are replaced with “TEMP”.</li> </ul>



## Find and Replace Tool Window

You can use the Find and Replace tool window (Ctrl+F, **Search > Find**, or **View > Toolbars > Find and Replace**) to specify search and replace options and conduct search and replace operations on selections, single files, or multiple files.



## Docking the Tool Window

Like other SlickEdit® tool windows, this tool window is dockable. Docking options can be accessed by right-clicking on the tool window's title bar. When the tool window is docked, invoking any find or replace command will bring the window to the front focus. When it is not docked, invoking these commands will cause the window to float display. Whether docked or floating, search and replace operations will not close the tool window by default.

## Saving Search and Replace Values

When the Find and Replace tool window is invoked, the options that were used for your last search are displayed, providing a way to repeat the last search. Options also persist when switching between the tabs. Pressing F7 and F8 retrieves previous and next responses, respectively.

Search and replace values can be saved as named operations. Saving preserves the values of all of the fields in the Find and Replace tool window so that the search and/or replace operation can be repeated in the future with the same results. To save the search/replace, right-click in the Find and Replace tool window. Select **Saved Search Expressions**, then select **Save Search Expression** from the sub-menu. You will be prompted to name the operation. To access a saved operation, select **Saved List** from the sub-menu, then pick the saved operation to load.

## Syntax-Driven Searching

To reduce the number of false positives in your searches, you can restrict the search based on program syntax. Click the **Color** button on the **Find tab** of the Find and Replace tool window to specify the syntactic elements for filtering. Each check box has three states:

- **Neutral (the default)** - All check boxes start in the neutral state. These elements will be used in a search until deselected or until one or more other elements are selected. Putting a check in any checkbox essentially deselects all non-checked boxes.
- **Selected** - If the check box is selected, the search will be restricted to this element and any other selected elements. There is no need to deselect any other elements if any elements are selected. If any elements are selected, only selected elements will be searched. For example, to search for the word “result” only in comments, put a check only in the Comment check box. All other syntactic elements will be ignored as part of this search.
- **Deselected** - If the check box is clear, these elements will not be searched. For example, if you want to find the word “result” anywhere in your code except for in comments, clear the Comment check box.

## Setting Options

Options for individual search and replace operations are located on the Find and Replace tool window. Alternatively, you can set default options that are always used instead. To set the default options, right-click on the background of the tool window and select **Configure Options**. The default search options will always be used when the Find and Replace tool window is invoked, unless settings are changed on the Find and Replace tool window. If you change settings on the tool window and want to use the default options instead, right-click in the tool window and select **Use Default Options**. For information on each option on the Find and Replace tool window, see [Find and Replace Tool Window](#). For a listing of the default search options, see [Search Tab](#).

## Search Results Output

You can specify that multi-file search results are displayed in a new editor window or in a new Search Results tool window.

To send the results to a new editor window, select the **Find in Files tab**, click the **Result options** button to expand the options, then select **Output to editor window**.

To send the results of a multi-file search to a specific Search Results tool window, select the Find in Files tab, click the **Results options** button to expand the options, then use the **Search Results Window** drop-down list to select the window to be used. These are labeled starting at **Search<0>**. A new results tool window can be added with the **<New>** option up to a pre-set limit of open Search Results windows.

If **<Auto Increment>** is selected from the Search Results Window drop-down list, the search results will cycle through all of the open Search Results tabs in the Search Results tool window with each new search. For example, if you have Search<0>, Search<1>, and Search<2> open, then for each search operation, the results will be displayed in this order: Search<0>, Search<1>, Search<2>, Search<0>, Search<1>, and so on. If you only have one Search Results tool window open, then all results will go into the only open search windows. You can open and close search results windows by right-clicking on the Search Results tab in the Search Results tool window.

Right-click in the Search Results window to access more options. See [Find in Files Tab](#) for more information.

## Find Symbol Tool Window

The Find Symbol tool window (**Search > Find Symbol** or **gui\_push\_tag** command) is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match.

Searching for a symbol is faster than a normal text search because it is executed against the Context Tagging® database, rather than searching through your source files. Find Symbol also avoids false hits in comments or string literals. Though [Syntax-Driven Searching](#) in the regular [Find and Replace Tool Window](#) provides this same capability, it cannot match the speed of Find Symbol.

See [Find Symbol Tool Window](#) for information about the options that are available.

## Find and Replace with Regular Expressions

Sometimes searching for a string literal is too limiting. For instance, you cannot search for a quoted string, a blank line, a word starting at the beginning of a line, or two words separated by any number of spaces. A regular expression can describe these search strings and many more.

All search commands support regular expressions. The Find and Replace tool window contains options for turning on regular expression searching (see [Find and Replace Tool Window](#)). The key binding **Ctrl+T** toggles regular expression searching on/off while an incremental search is in progress (see [Incremental Searching](#)). The search commands slash (/) and **find** take R and U options to interpret the search string as a regular expression (see [Find and Slash \(/\) Commands](#)). The search and replace command **c** also takes R, U, and B options to specify a regular expression (see [Replace Command Search Examples](#)).

SlickEdit® supports four types of syntax:

- UNIX (see [UNIX Regular Expressions](#))
- SlickEdit (see [SlickEdit® Regular Expressions](#))
- Brief (see [Brief Regular Expressions](#))
- Wildcards (\*, ?)

All syntax types have the same features. In order to accomplish this, SlickEdit made several enhancements to the UNIX and Brief syntaxes. To select the regular expression syntax, from the main menu, choose **Tools > Options > General**, then select the [Search Tab](#). Select the option **Regular expression**, then pick the syntax from the drop-down list.

## Special Characters in Regular Expression Find/Replace

Characters have special meaning during search and replace operations.

- **UNIX regular expressions** - When regular expressions are turned on for a search and replace command, the backslash character (\) has special meaning in the replace string. A backslash in the replace string has the same meaning as in the search string except that **\c** and **\:char** are not supported. See [UNIX Regular Expressions](#) for a list of backslash (\) options. See [Using Tagged Search Expressions](#) for information on specifying tagged expressions in the replace string.
- **SlickEdit® regular expressions** - When regular expressions are turned on for a search and replace command, the number sign character (#) and backslash (\) have special meaning in the replace string. A backslash in the replace string has the same meaning as in the search string except that **\c**, **:char**, and **\gd** are not supported. See [SlickEdit® Regular Expressions](#) for a list of \ options. See [Using Tagged Search Expressions](#) for information on specifying tagged expressions in the replace string using the number sign (#) character.

- **Brief regular expressions** - When regular expressions are turned on for a search and replace command, the backslash character (\) has special meaning in that backslash in the replace string has the same meaning as in the search string except that **\c** and **\:char** are not supported. See [Brief Regular Expressions](#) for a list of backslash (\) options. See [Using Tagged Search Expressions](#) for information on specifying tagged expressions in the replace string.

The following table contains some examples of replace operations using regular expressions:

Search Example	Description
Search For: <b>hat\$</b> Replace With: <b>cat</b>	Search for occurrences of the string “hat” that occur at the end of a line and replace it with “cat”.
UNIX and SlickEdit® regular expression: Search For: <b>^\\n</b> Replace With:  Brief regular expression: Search For: <b>&lt;\\n</b> Replace With:	Delete blank lines.
UNIX and SlickEdit regular expression: Search For: <b>^\\n\\n</b> Replace With: <b>\\n</b>  Brief regular expression: Search For: <b>&lt;\\n\\n</b> Replace With: <b>\\n</b>	Replace occurrences of two consecutive blank lines with one.
UNIX regular expression: Search for: <b>^a+\$</b> Replace with: <b>\\d12</b>  SlickEdit regular expression: Search for: <b>^a+\$</b> Replace with: <b>\\12</b>  Brief regular expression: Search for: <b>&lt;a+\$</b> Replace with: <b>\\d12</b>	Search for lines containing “a” and replace the “a” with a form-feed character.

## Using Expressions to Search for Binary Characters

Search for a sequence of binary characters by using regular expressions to specify hex or decimal characters. Some examples are:

- **UNIX or Brief search expressions:**

`\x0d\x0a\x01\x02`

`\d13\d10\d1\d2`

- **SlickEdit® search expressions:**

`\x0d\x0a\x01\x02`

`\13\10\1\2`

## Using Tagged Search Expressions

When you use regular expressions to search for a string, you will often want the replace string to depend on what was found. Use tagged search expressions to insert parts of what is found into the replace string.

- **UNIX regular expressions** - Use parentheses `()` to denote a tagged expression in the search string. The replace string specifies tagged expressions with a `"\"` followed by a tag group number 1-9. Count the left parenthesis (`"(`) in the search string to determine a tagged expression number. The first tagged expression is `\1` and the last is `\0`.
- **SlickEdit® regular expressions** - Use curly braces `{ }` to denote a tagged expression in the search string. The replace string specifies tagged expressions with a `#` followed by a tagged expression number 0-9. Count the left braces (`"{"`) in the search string to determine a tagged expression number. The first tagged expression is `#0`.
- **Brief regular expressions** - Use curly braces `{ }` to denote a tagged expression in the search string. The replace string specifies tagged expressions with a `"\"` followed by a tagged expression number 0-9. Count the left braces (`"{"`) in the search string to determine a tagged expression number. The first tagged expression is `\0`.

The following table contains examples of using tagged search expressions:

Tagged Search Example	Description
<p>UNIX regular expression:            Search for: <b>(if while)</b>            Replace with: <b>x\1y\2</b></p> <p>SlickEdit® regular expression:            Search for: <b>{if while}</b>            Replace with: <b>x#0y#1</b></p> <p>Brief regular expression:            Search for: <b>{{if} while}}</b>            Replace with: <b>x\0y\1</b></p>	<p>Replace occurrences of “if” and “while” with “xify” and “xwhiley.” Unmatched groups are null. Note: The UNIX syntax <b>\2</b> (SlickEdit syntax <b>#1</b>, Brief syntax <b>\1</b>) is replaced with null.</p>
<p>UNIX regular expression:            Search for: <b>^(.*?),(.*)\$</b>            Replace with: <b>\2,\1</b></p> <p>SlickEdit regular expression:            Search for: <b>^{?},{?}\$</b>            Replace with: <b>#1,#0</b></p> <p>Brief regular expression:            Search for: <b>^{*},{*}\$</b>            Replace with: <b>\1,\0</b></p>	<p>Reverse text on lines containing a comma. Lines with “abc,def” will be changed to “def,abc.” Notice that the UNIX regular expression search string uses a minimal matching operator <b>*?</b> so that the comma actually matches the first comma in the line and not the last.</p>

## Minimal versus Maximal Matching

If you are using tagged expressions or regular expressions to perform a search and replace, you need to understand the difference between the minimal and maximal operators.

Take, for example, a line of text which contains a DOS file name: `\path1\path2\path3\name.ext`.

The regular expression **^\\.\*?\\** (UNIX), **^\\?\*\\** (SlickEdit®), or **<\\\*\\** (Brief) will match the string `\path1\`.

The regular expression **^\\.\\\*\\** (UNIX), **^\\?@\\** (SlickEdit), or **<\\:.\*\\** (Brief), which uses the maximal operator, matches the string `\path\path2\path3\`.

As a rule of thumb, you will usually want to use the minimal matching operators **\*?** (UNIX), **\*** (SlickEdit), or **@** (Brief) and **+?** (UNIX), **+** (SlickEdit/Brief) after a less-specific regular expression such as **.** (UNIX) or **?** (SlickEdit/Brief).

You will usually want to use the maximal matching operators after a regular expression which matches something more specific. For example, to search for a string of digits and prefix each string of digits with the character \$, you would specify the following in the Replace tab of the Find and Replace tool window:

- **UNIX regular expression:**  
 Search for: `([0-9]+)`  
 Replace with: `$\1`
- **SlickEdit regular expression:**  
 Search for: `{[0-9]#}`  
 Replace with: `$#0`
- **Brief regular expression:**  
 Search for: `{[0-9]\: +}`  
 Replace with: `$\0`

If the minimal matching operator (UNIX `+`?, SlickEdit and Brief syntax `+`) was used in the search for string instead of the maximal matching operator (UNIX `+`, SlickEdit syntax `#`, Brief syntax `\: @`), the above search and replace would prefix each digit in the file with a “\$” character, which is probably not what you want.

## Undoing/Redoing Replacements

To undo a replacement, select **Edit > Undo**, press Ctrl+Z, or use the **undo** command. To redo a replacement, select **Edit > Redo**, press Ctrl+Y, or use the **redo** command.

To undo replacements in multiple files, select **Edit > Multi-File Undo** or use the **mfundo** command. To redo replacements in multiple files, select **Edit > Multi-File Redo** or use the **mfredo** command.





# Beautifying Code

---

## Code Beautifiers

Code beautifiers, available for many languages, reformat the layout of existing text based on settings that you specify, such as begin/end styles and indenting.

To beautify selected lines of code, or to beautify the entire buffer, from the main menu, select **Tools > Beautify** (or use the `gui_beautify` command). A dialog box is displayed with functions specific to the type of project that is active. If an HTML project is active, then the HTML Beautifier dialog appears with options. If a GNU C/C++ project is active, then the C/C++ Beautifier dialog opens, and so on. Beautifying is supported for the languages listed below. Follow the cross-reference links to learn more about working with each beautifier.

- Ada - See [Ada Beautifier](#).
- C/C++, C#, Java, JavaScript, Slick-C® - These beautifiers contain the same options and settings. See [C/C++ Beautifier](#).
- CFML, HTML - These beautifiers contain the same options and settings. See [HTML Beautifier](#).
- Javadoc - See [Javadoc Beautifier](#).
- XML, XSD - These beautifiers contain the same options and settings. See [XML Beautifier](#).

## Reflowing Text

To reflow text in the current paragraph according to your margin settings, select **Document > Format Paragraph** or use the `reflow_paragraph` command. Margin settings are defined on the [Word Wrap Tab](#) (**Tools > Options > File Extension Setup**).

When you reflow a paragraph, the cursor will be kept at the same location within the current paragraph after reflow has occurred, unless the **Reflow next** option is selected (**Tools > Options > General**, [More Tab](#)). If **Reflow next** is selected, the `reflow_paragraph` command places the cursor on the next paragraph after it has reformatted the current paragraph.

Comments can also be reflowed according to the comment wrap settings. See [Reflowing Comments](#) for more information.



# Refactoring

---

Refactoring is a precise code editing feature that you can use to clean up and improve the understandability of your source code. Refactoring allows you to make disciplined, system-wide changes to code without affecting the external behavior.

There are two types of refactoring available within SlickEdit®: [C++ Refactoring](#) and [Quick Refactoring](#). C++ Refactoring supports the C++ language only, while Quick Refactoring supports C++, C#, Java, and Slick-C®. Quick Refactoring is generally faster and less stringent than C++ Refactoring.

For information about refactoring results, see [Reviewing Refactoring Changes](#).

## Quick Refactoring

Quick Refactoring supports C++, C#, Java, and Slick-C®, and performs refactorings using Context Tagging® rather than a formal language parser. Quick Refactoring is generally faster and less stringent than C++ Refactoring.

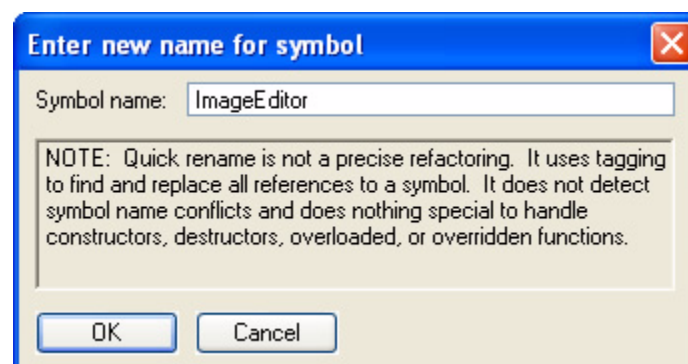
### Available Quick Refactorings

To access the Quick Refactorings, from the main menu, choose **Tools > Quick Refactoring**. The Quick Refactoring menu can also be accessed from the right-click menus within the Symbols and Defs tool windows.

**TIP** Refactoring operations can modify more than one file. You can undo all of the refactoring modifications in one step by selecting **Edit > Undo Refactoring**.

### Quick Rename

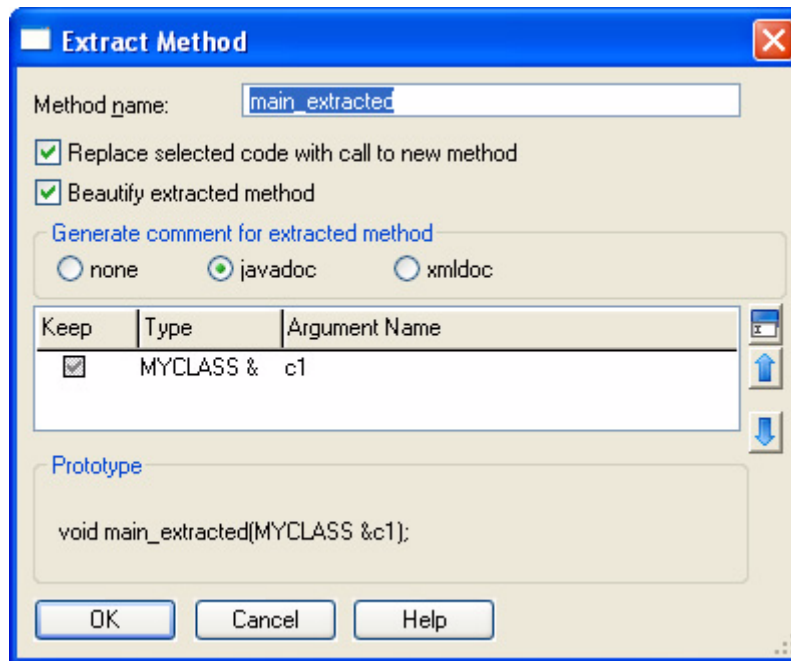
Quick Rename uses the Context Tagging® to rename a symbol under the cursor or any symbol selected in the Defs or Symbols tool windows. This operation works for all tagged languages. It is faster than the rename provided by C++ Refactoring, but less stringent. Quick Rename does not treat renaming classes, constructors, and destructors as a special case. Quick Rename will rename all of the overloads of a function. Quick Rename does not rename overridden methods (in parent and child classes).



### Quick Extract Method

After selecting a set of lines, Quick Extract Method creates a new method with the selected lines as the body. It discovers any undeclared variables and creates them as parameters to the new method. The

extracted method is created in the same scope as the original method. Quick Extract Method is only available for C++, C#, Java, and Slick-C®.



### Quick Modify Parameter List

This refactoring allows you to add, delete, and re-order parameters for a selected function. The refactoring will modify the parameter list for the selected function and all of its counterparts within the class hierarchy. Quick Modify Parameter List is only available for C++, C#, Java, and Slick-C®.

### Quick Replace Literal with Constant

Replaces the selected literal with a constant, replacing use of the literal with the new constant. Quick Replace Literal with Constant is only available for C++, C#, Java, and Slick-C®.

## C++ Refactoring

You can apply many commonly used refactorings in C++.

**NOTE** C++ Refactoring is supported on Mac OS X, but not for Objective-C or Objective C++. While Objective-C is closely related to C++, the refactoring engine cannot accommodate the syntactic differences.

## Available C++ Refactorings

To access the C++ refactorings, from the main menu, choose **Tools > C++ Refactoring**. The C++ Refactoring menu can also be accessed from the right-click menus within the Symbols and Defs tool windows.

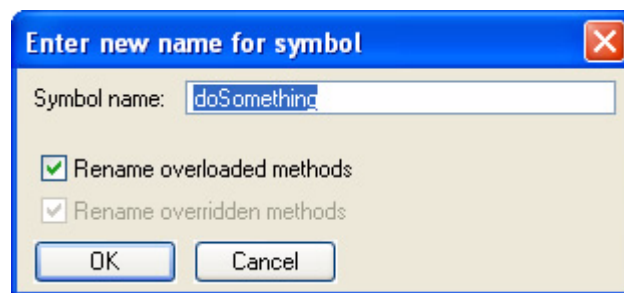
**TIP** Refactoring operations can modify more than one file. You can undo all of the refactoring modifications in one step by selecting **Edit > Undo Refactoring**.

### Rename

Rename is used to rename variables, methods, and classes. It uses Context Tagging® to identify:

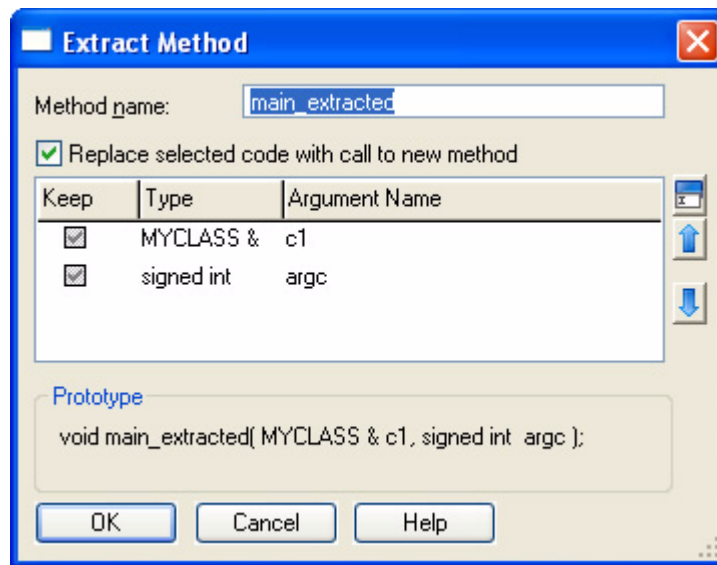
- the symbol under the cursor (or any symbol selected in the Symbols or Defs tool windows)
- all of the symbol overloads
- all other instances of the symbol within the class hierarchy

It then parses each file containing references to the selected symbol(s), and updates the rest of the code to use the changed name.



### Extract Method

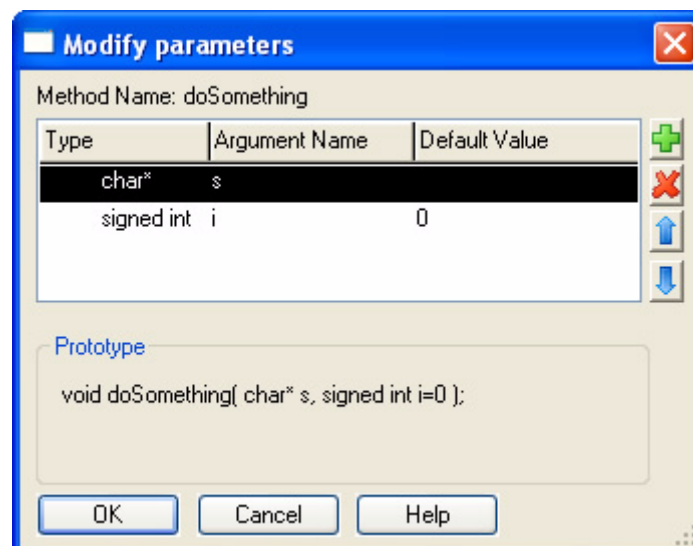
After selecting a set of lines, Extract Method creates a new method with the selected lines as the body. It discovers any undeclared variables and creates them as parameters to the new method. The extracted method is created in the same scope as the original method.



The **Method name** text box allows you to choose the name of the newly extracted method. Uncheck **Replace selected code with call to new method** if you do not want the original method to be modified. This option is disabled if the selected block contains a return, continue, or break statement.

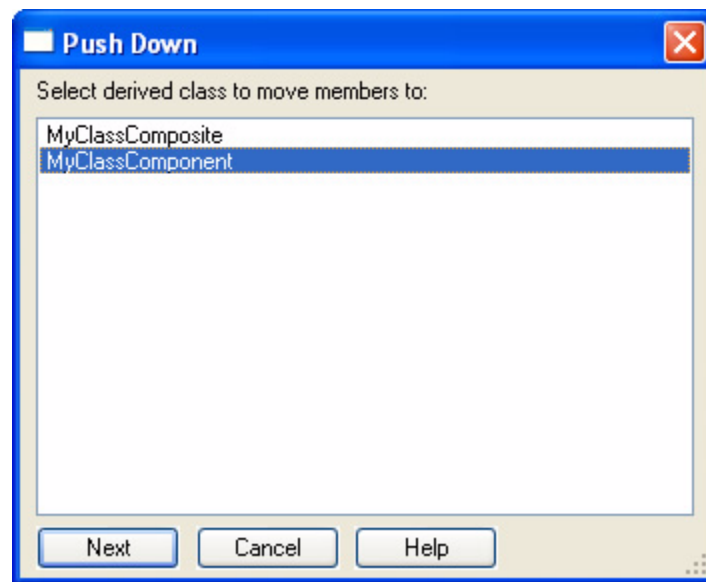
## Modify Parameter List

Modify Parameter List enables you to add, delete, and re-order parameters for a selected function. The refactoring will modify the parameter list for the selected function and all of its counterparts within the class hierarchy.



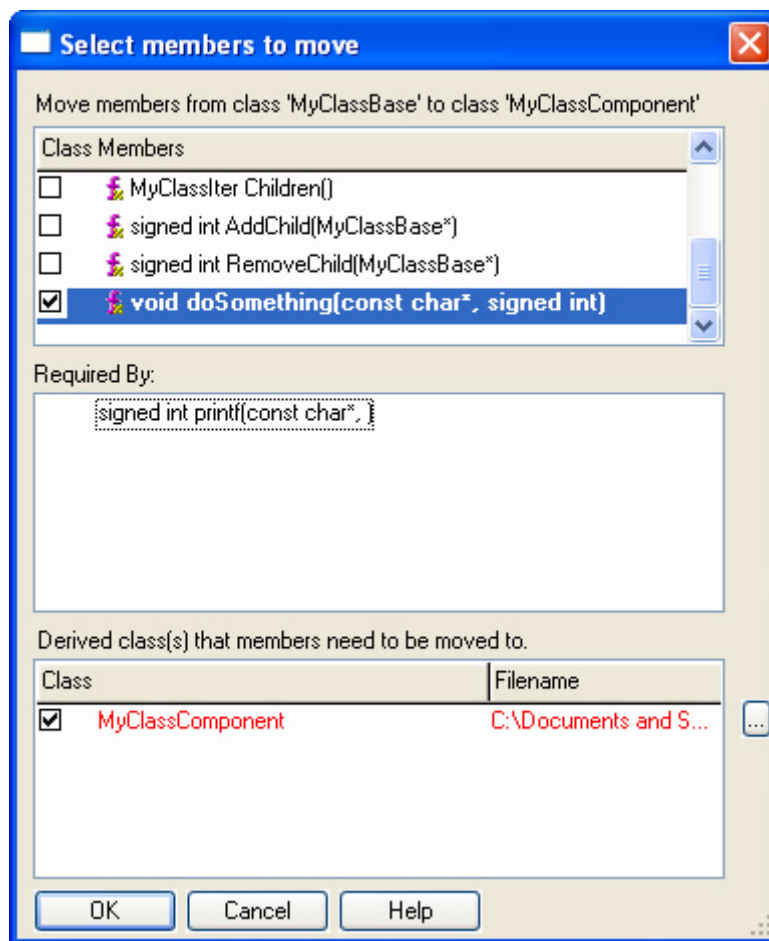
## Push Down to Derived Class

Moves class members to a class that inherits from the selected class.



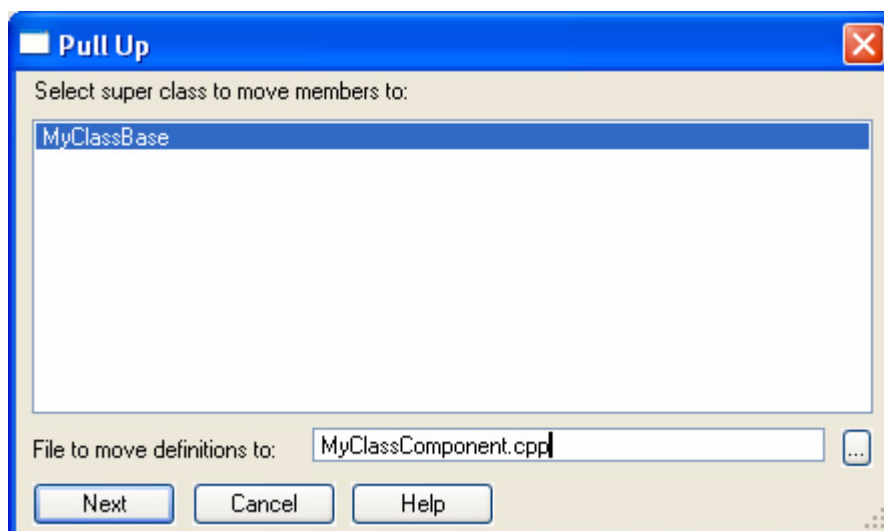
If a member of the super class is used by multiple subclasses, then the member is moved to all of the subclasses that access that member, so that when the refactoring is done, everything will compile. Any member that is explicitly accessed through an instance of the super class (or an instance that is cast to the super class) cannot be moved. Moving this will break the code.

Constructors and destructors cannot be moved. Members can only be moved one level at a time.



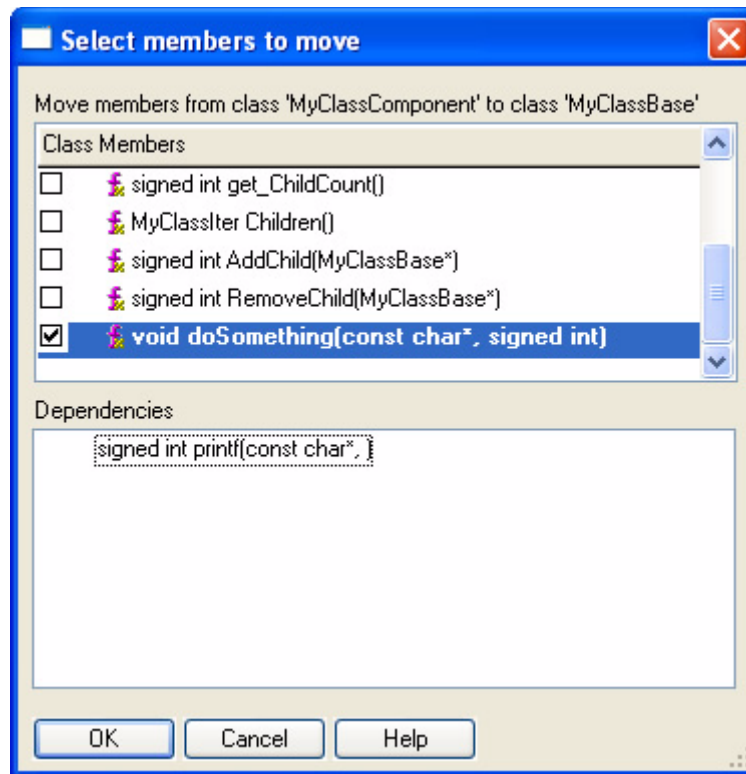
## Pull Up to Super Class

Automates moving members from a selected class to one of its directly-inherited base classes.



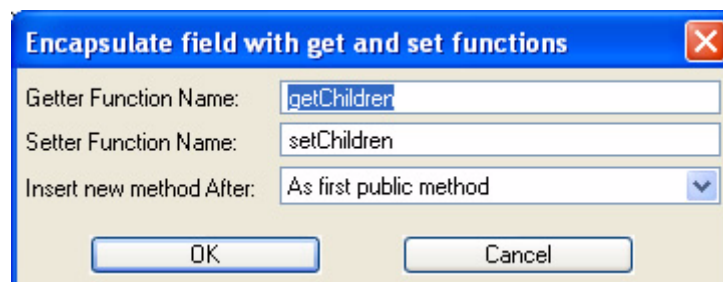


Class members are pulled up one level at a time. Constructors and destructors cannot be moved. Any dependencies that are not part of the original class will not be moved to the new base class, and might not be accessible, thereby causing compilation errors. This occurs, for example, when a function being moved uses a static global variable that is defined in the original class' .cpp file when the definitions are moved to a different .cpp file.



## Encapsulate Field

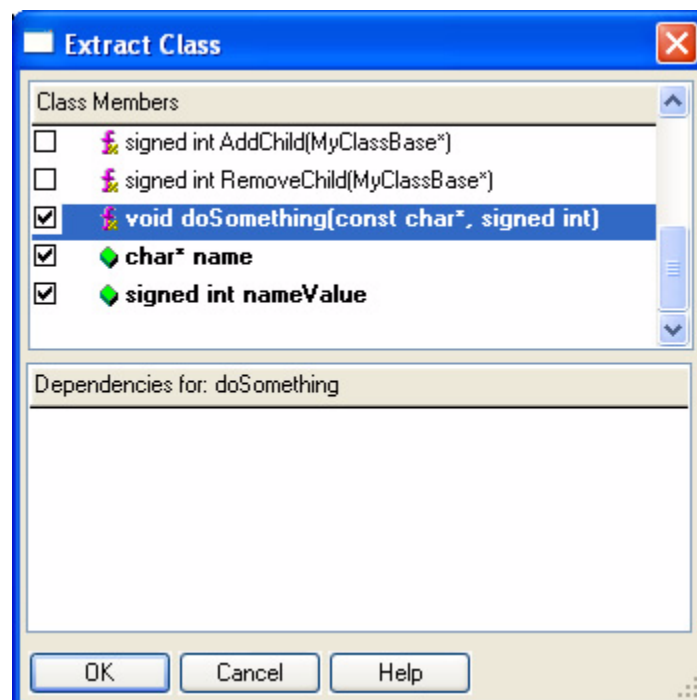
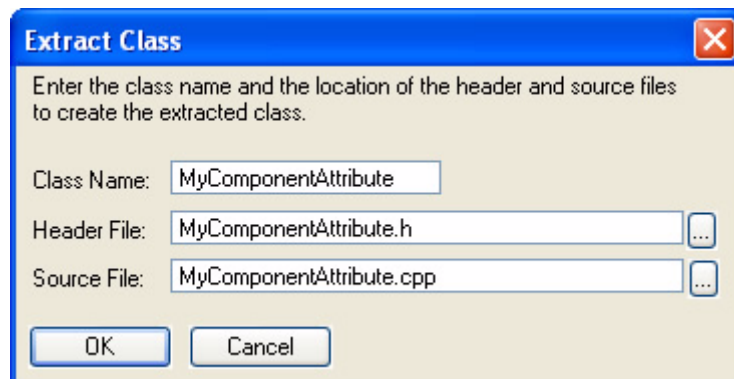
Generates get and set functions for the specified variable and makes that variable private. All references to the variable are replaced with references to the getter or setter, as appropriate.



## Extract Class

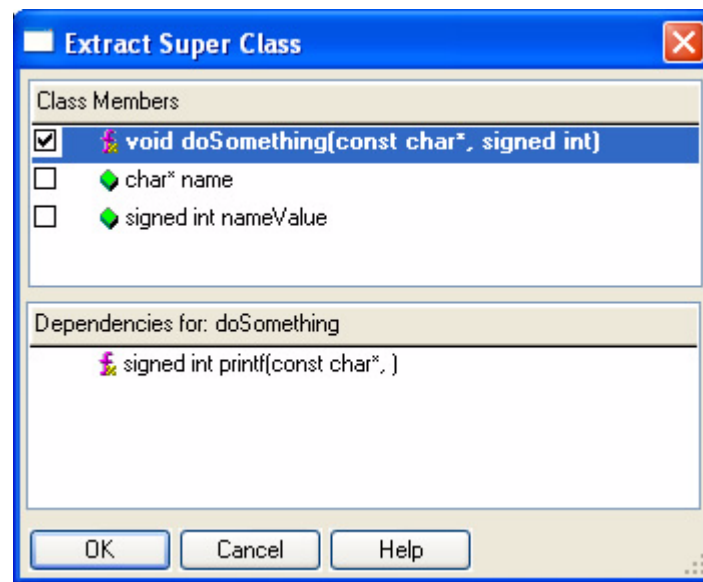
Breaks a large class into a better abstraction by moving some responsibilities into a new class or interface. This refactoring creates new files. When using the Extract Class refactoring, keep in mind the following information:

- The default path uses the same directory as the original class.
- The default filenames are the `<class_name>.(cpp/h).`
- Default paths and filenames can be changed.
- The files will be added to the current project.
- If version control is enabled, you are prompted to add the files to the version control system.
- The new class will generate a default constructor, and a constructor that takes a parameter per member variable moved. Initializers in the original class' constructors for moved member variables are moved to the new class' constructors.



### Extract Super Class

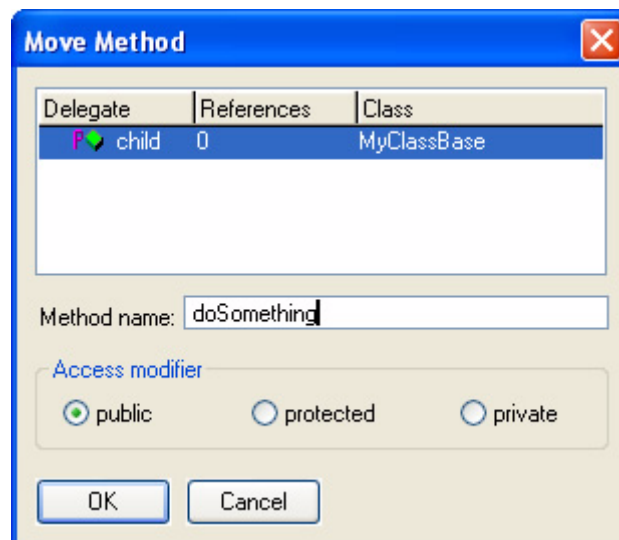
Breaks a large class into a better abstraction by moving some responsibilities into a new class/interface. The extracted class becomes the super class of the original class.



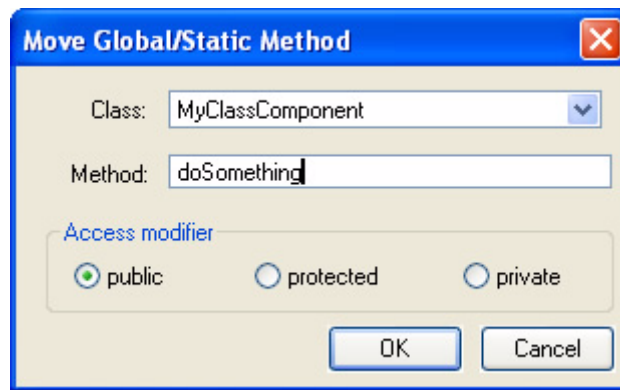
## Move Method

Moves a method from one class to another and updates references accordingly.

The original method may be converted to a delegate method if there are references to the original method that cannot be converted to a reference to the moved method.



If the method is static, you will be prompted for the target class.



## Move Static Field

Moves a static data member from one class to another and updates references accordingly.

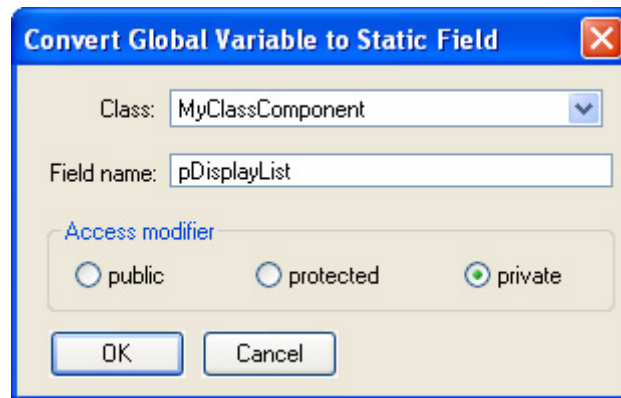


## Convert Static to Instance Method

Changes a static method to an instance method and updates any references to change how the method is accessed.

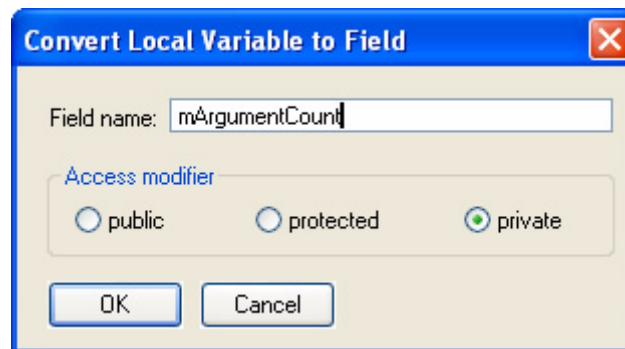
## Convert Global to Static Field

Moves globally-declared variables into a static field in a class, and updates references to refer to the new static variable.



## Convert Local to Field

Moves a local variable from the body of a method into an instance member variable in the current class. References to the local variable are replaced with references to the new data member. This refactoring cannot be used to convert a method parameter to a field.



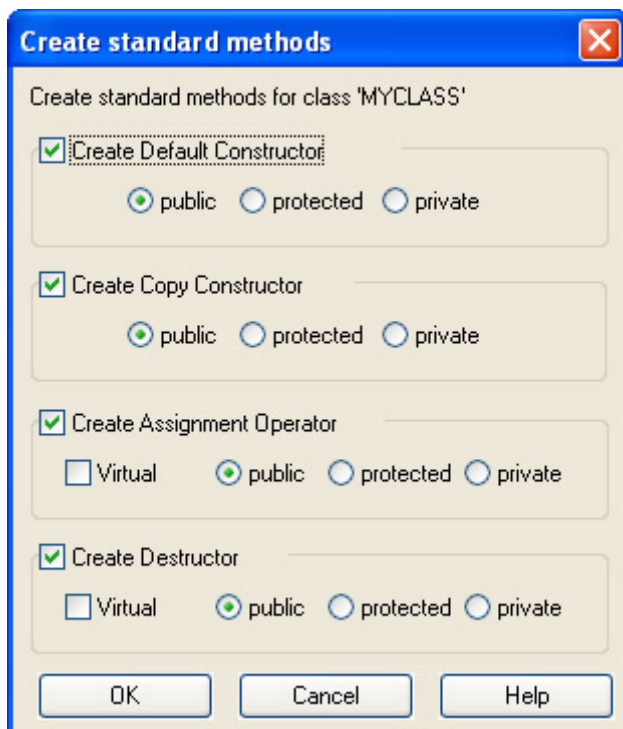
## Replace Literal with Constant

Replaces the selected literal with a constant, replacing use of the literal with the new constant.



## Create Standard Methods

Creates an assignment operator, copy constructor, default constructor, and destructor for the selected class.

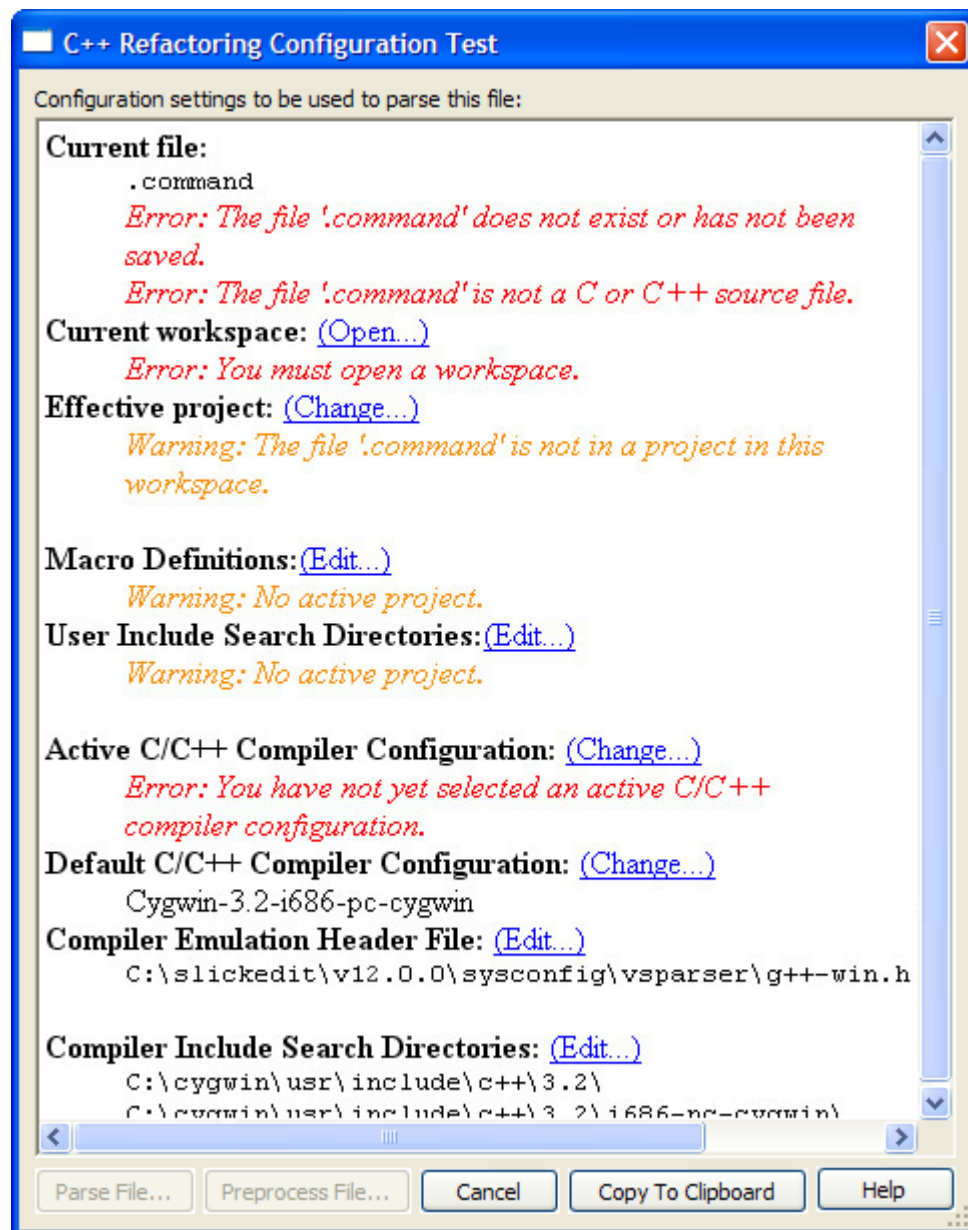


## Test Parsing Configuration

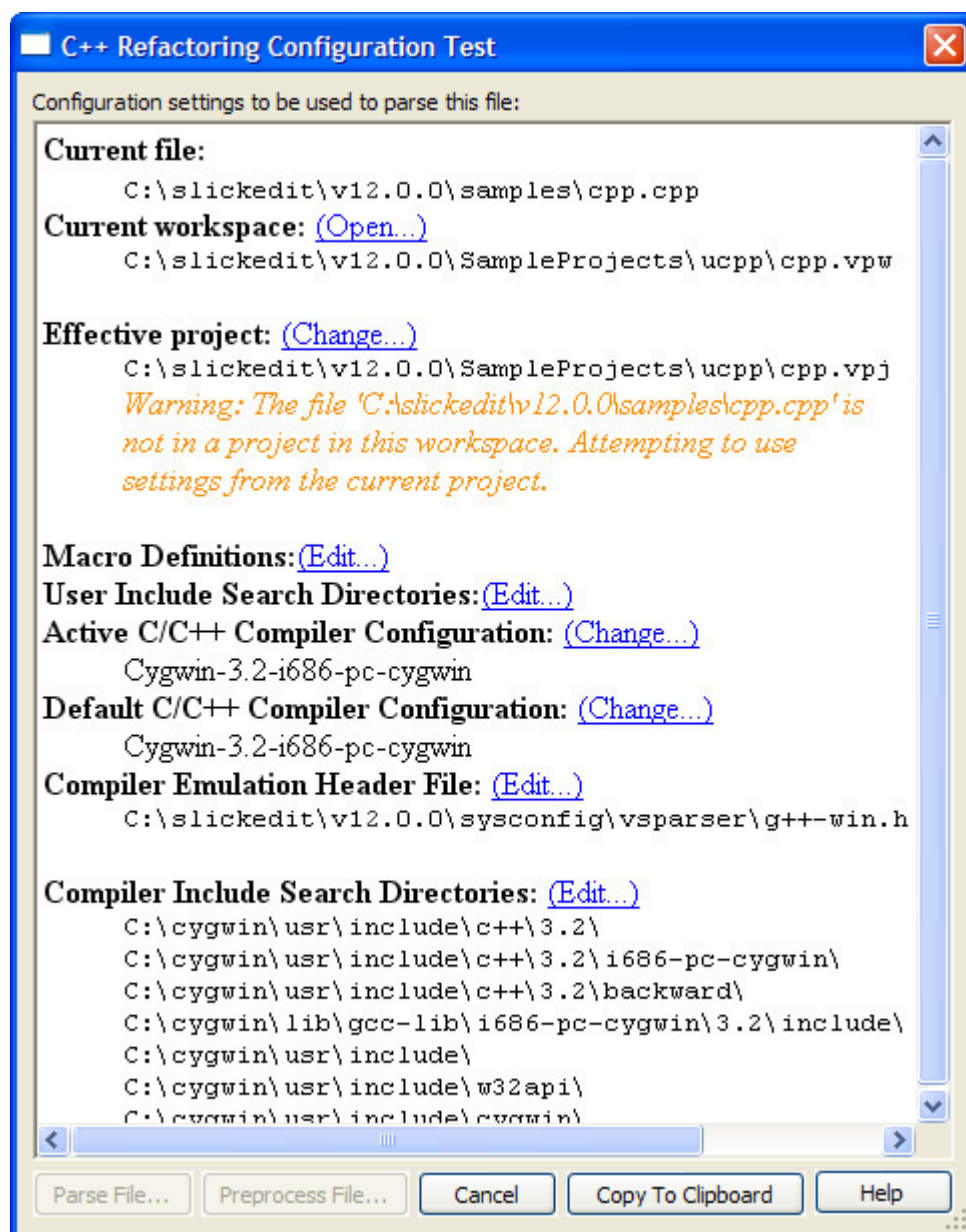
Test Parsing Configuration analyzes the C++ refactoring configuration that has been set up and reports key information and errors. Generally, it is designed as a debugging aid when a refactoring fails, but it can also be valuable as a means to view the parameters that are set for a refactoring. You can verify the refactoring setup before proceeding to avoid errors.

To access Test Parsing Configuration, from the main menu click **Tools > C++ Refactoring > Test Parsing Configuration** (**refactor\_parse** command), or right-click within the editor window and click **C++ Refactoring > Test Parsing Configuration**. On the Test Parsing Configuration dialog, click **Parse File** to parse the file. Any errors are displayed in the Output tool window. Click **Preprocess File** to preprocess the file, sending the results to a new editor window. Click **Copy to Clipboard** to copy the entire display shown in the window. This is useful for sending to SlickEdit Support if it is necessary to diagnose a problem.

The following example of the Test Parsing Configuration window shows an error stating that there is no open workspace, as well as various other warnings. These are severe enough to prevent parsing the file, which is why the **Parse File** button is disabled.

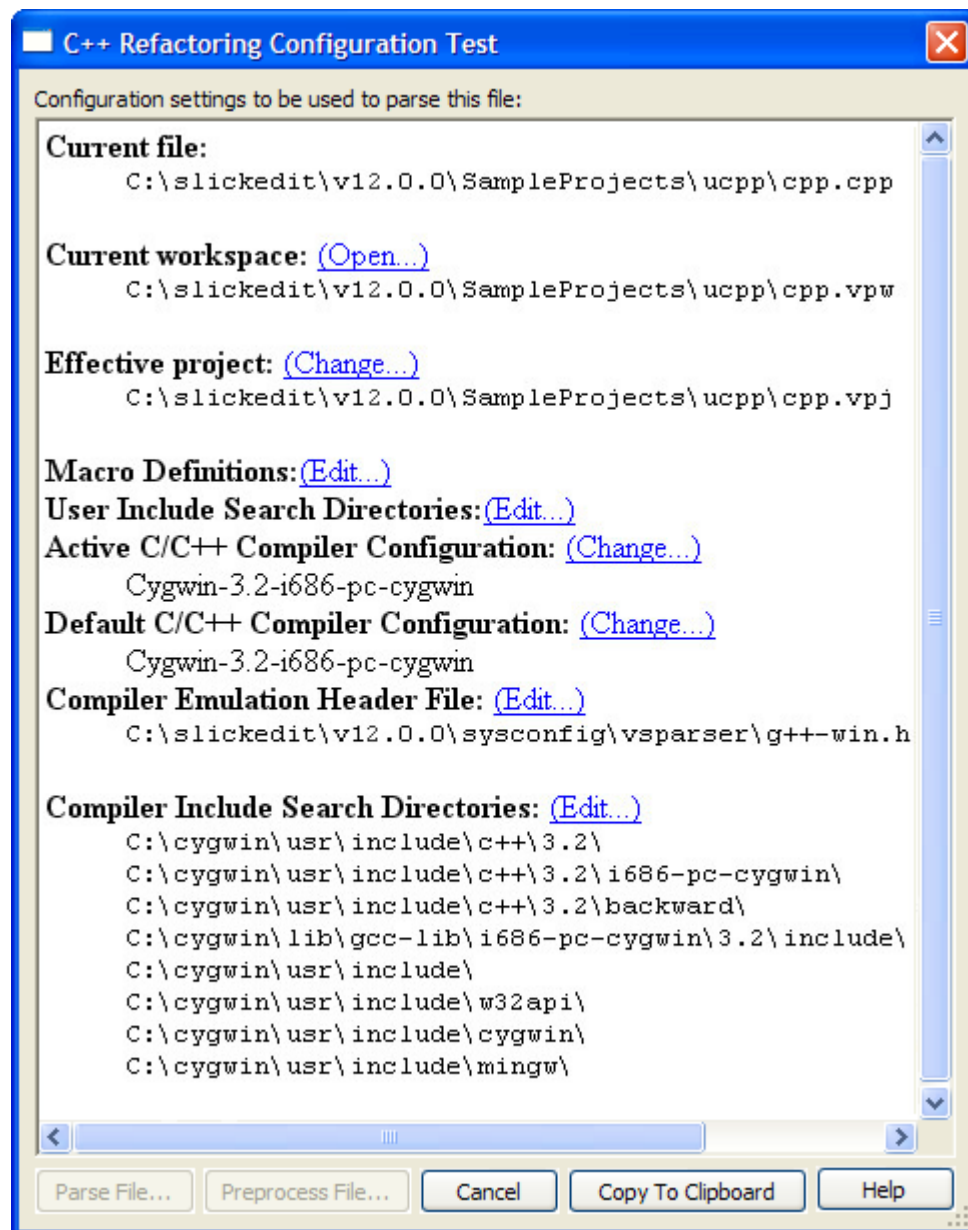


The following example shows a warning that is not severe enough to prevent parsing of the file.



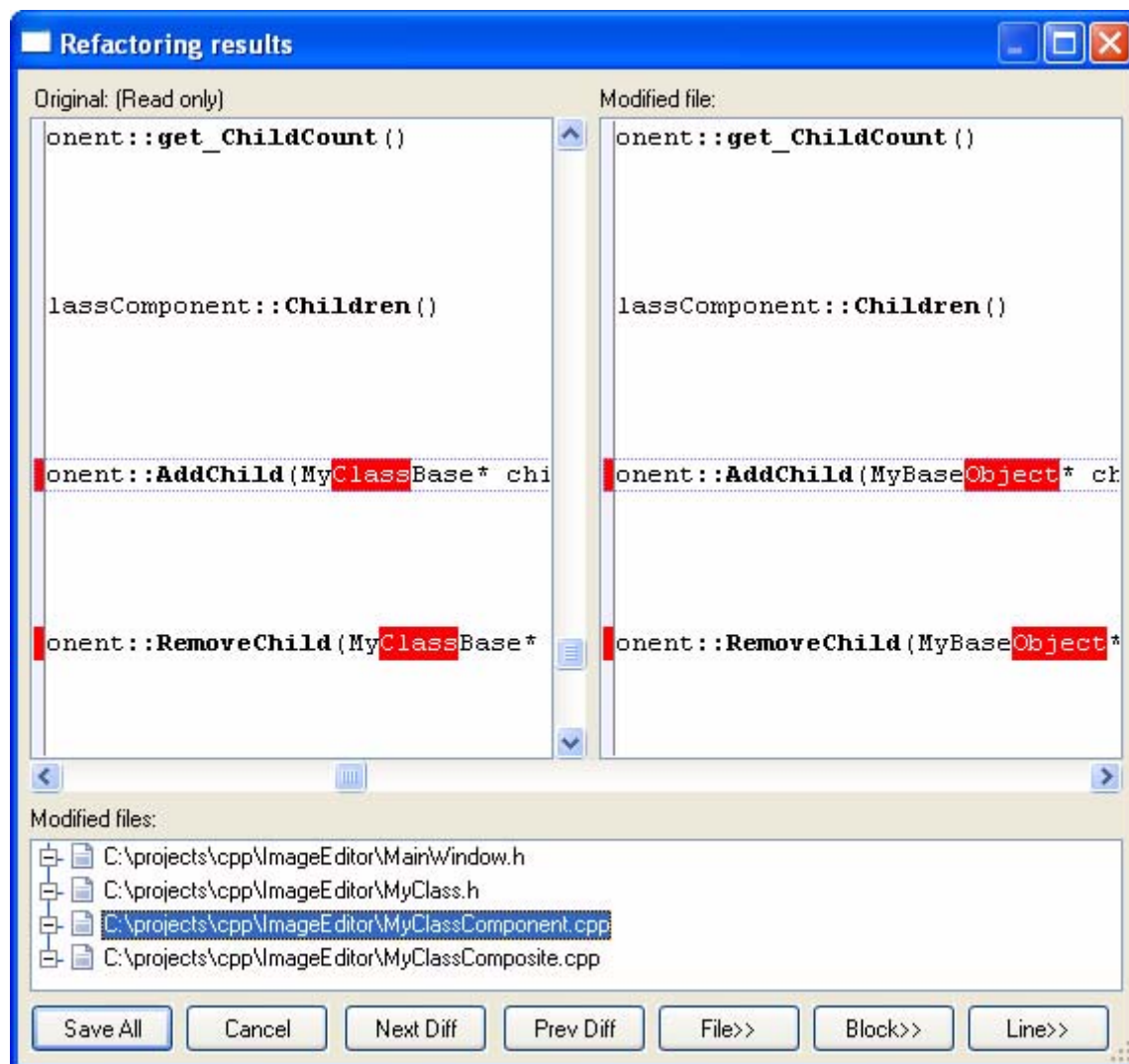
Finally, the following example shows a test resulting in no errors or warnings.





## Reviewing Refactoring Changes

When a refactoring finishes, the Refactoring results dialog box is displayed, allowing you to review the changes.



There are three panes in this window:

- The left pane is read-only and shows the original file(s).
- The right pane shows the refactored file(s). For convenience, this pane can be edited.
- The bottom pane lists all files that have been modified by the refactoring. Clicking on any file in this list brings that file into view, where it can be reviewed and edited.

Click **Save All** at the bottom of this window to save all the refactoring and editing changes that were made on all files. Click **Cancel** to discard changes and have all files remain the way they were before the refactoring process.

Click **Next Diff** or **Prev Diff** to advance to the next or previous change made by the refactoring. Click **File>>** to restore the contents of the current selected file to its original contents.

Click **Block>>** to restore an entire block of changes to the original contents. Click **Del Block** to remove a block of code inserted by the refactoring. Click **Line>>** to restore the current line to its original contents.

Some refactorings, in particular [Modify Parameter List](#), may require further user input. In this case each input will be displayed under the file it is in, and there will be two additional buttons: **Next Input** and **Prev Input**. You will not be able to save the refactoring results until you have resolved all of the input requests.



## Viewing and Displaying

---

SlickEdit® offers several features for viewing and displaying.

### Hexadecimal View and Edit Mode

To view a binary or text file in a hex/text view mode, select **View > Hex** or **View > Line Hex** (or use the commands **hex** or **linehex**). If closing a file while viewing it in hex mode, then the next time the file is edited, it will be displayed in hex/text view mode. When the cursor is in the hex data, it can be overwritten or hex nibbles (characters 0 through F) can be inserted. When the cursor is in the text data, overwrite it if you want, or insert text characters the same as if editing a text file. All of the search and replace commands work while hex editing. Only character selections are displayed when in hex editing mode.

### Hex Mode Key Bindings

Hex mode key bindings override normal key bindings for the emulation. Most of the other emulation keys will perform the same operation. However, keys that are bound to the commands **top\_of\_buffer**, **bottom\_of\_buffer**, **page\_up**, **page\_down**, **begin\_line**, **end\_line**, **begin\_line\_text\_toggle**, **cursor\_left**, or **cursor\_right** will perform hex/text cursor motion.

Keys	Function
<b>Tab and Shift+Tab</b>	Toggle cursor between hex data on left and text data on right.
<b>Home</b>	Move cursor to beginning of hex/text line.
<b>End</b>	Move cursor to last character of hex/text line.
<b>Backspace</b>	Delete a byte to the left of the cursor and move the cursor to the left.
<b>Delete</b>	Delete the byte under the cursor.

### Viewing Special Characters

By default, many important characters are not visible in the editor, like tabs, spaces, and line-ending characters. To view these, choose **View > Special Chars**. SlickEdit® will then display a visible character to represent these invisible characters. When using this option with **View > Line Hex**, the hex value for the actual character (like space) will be displayed, not the value for the character used to represent it (like a dot).

To define which characters to display, from the main menu choose **Tools > Options > General** and then select the **Special Characters tab**. See [Defining Special Characters](#) below.

**NOTE** Viewing special characters is only available for ASCII files.

### Special Character Toggles

The following toggles are available on the view menu:

- Hex Toggle
- Line Hex Toggle
- Special Characters Toggle
- New Line Characters Toggle
- Tab Characters Toggle
- Space Characters Toggle
- Line Numbers Toggle

## Defining Special Characters

To define what characters to display when viewing special characters, from the main menu, choose **Tools > Options > General**, then select the [Special Characters Tab](#). Enter the character codes that you wish to use.

To view the differences between a DOS format text file and another format when **View > New Line Chars** is enabled, choose something other than a space for the End-Of-Line (2) character. Under Windows, the recommended choices are **13** for End-Of-Line (1) and **10** for End-Of-Line (2).

To change the colors and styles of special characters, from the main menu choose **Tools > Options > Color**. Select **Special Characters** from the screen element drop-down list. For more information on color settings, see [Setting Fonts and Colors](#).

## Selective Display

Selective Display (also known as *code folding*) is a convenient way to display or hide regions of your code, so that you can view and manage only those regions that are relevant to your current editing session.

Use the Selective Display dialog to enable this feature and to specify the type of regions to display or hide. This dialog is displayed by selecting **View > Selective Display**, or by using the **selective\_display** command.

When Selective Display is enabled, a **Plus (+)** or **Minus (-)** bitmap is placed before hidden or expanded lines in the editor window margin. The following screenshot shows a sample file with two function definitions expanded and the rest collapsed.

```

#include <stdio.h>
Rectangle::Rectangle()
Rectangle::Rectangle(const Point& topLeft, const Point& bottomRight)
float Rectangle::getWidth()
{
    return m_BottomRight.m_x - m_TopLeft.m_x + 1;
}

float Rectangle::getHeight()
{
    // REFACTOR: Encapsulate Field: m_x, m_y
    return m_BottomRight.m_y - m_TopLeft.m_y + 1;
}

void Rectangle::printTitle()
void Rectangle::printBanner()
bool Rectangle::contains(Rectangle& rect)

```

When Selective Display is enabled, you can perform the following operations:

- To display or hide lines: Double-click on the Plus or Minus bitmaps. Alternatively, select **View > Expand/Collapse Block**, press **Ctrl+I**, or use the **plusminus** command.

**TIP** Selective Display icons can be expanded or collapsed with a single click, causing Selective Display to operate similar to Windows Explorer. Note, however, that you will not be able to select a line by clicking to the left of a text line which contains a Selective Display bitmap. To set this option, from the main menu, choose **Tools > Options > General**, then select the [General Tab](#). Select the option **Expand/collapse single click**.

- To copy visible text to the clipboard: Select **View > Copy Visible** or use the **copy\_selective\_display** command. Normally when you copy a selection that spans multiple lines, hidden lines are copied as well. This command ignores hidden lines and only copies visible text. This operation does not work with block selections.
- To redisplay all lines and remove the Plus and Minus bitmaps: From the main menu choose **View > Show All** (**show\_all** command).

## Expanding/Collapsing Code Blocks

SlickEdit provides a way to expand and collapse code blocks without having to clutter the gutter with Selective Display bitmaps. You can expand or collapse blocks of code by using the **plusminus** command, whether or not Selective Display Plus or Minus bitmaps are displayed.

The **plusminus** command expands or collapses code blocks under the following conditions:

- If the cursor is on the first line of a code block, the block is collapsed, creating a new Selective Display region.

- If the cursor is on a line that contains a Plus (+) bitmap, the block is expanded.
- If the cursor is on a line that contains a Minus (-) bitmap, the expanded block is collapsed.

### NOTES

- The definition of a “code block” is based on your language.
- The **plusminus** command is controlled by the **def\_plusminus\_blocks** configuration variable. The value is set to true (1) by default. For more information, see [Configuration Variables](#).
- The **plusminus** command uses the same logic to identify code blocks as the command **cut\_code\_block**. See [Deleting Code Blocks](#) for more information.

## Selective Display Regions

Using the Selective Display dialog, you can choose the regions you want to display or hide. Specific settings are provided for each region.

- [Search Text](#) - Displays lines that contain the specified search string or lines that do not contain the specified string.
- [Function Definitions](#) - Displays only function headings and optionally, function heading comments.
- [Preprocessor Directives](#) - Displays a source file as if it were preprocessed according to the define values specified here. If you do not remember your defines, use the **Scan for Defines** button.
- [Multi-Level](#) - Select this option to set multiple levels of Selective Display based on braces or indent.
- [Paragraphs](#) - Displays the first line of each paragraph. A paragraph is defined by a group of lines followed by one or more blank lines.
- [Hide Selection](#) - Select this option to hide the lines in the current selection.

The Selective Display dialog also contains static options for expanding/collapsing sub-levels. See [Selective Display Dialog](#) for more information and details about the available settings.

## Other Display Options

This section describes other general display options that you might find useful.

### Inserting a Top of File Line

You can specify that each buffer opened displays a line which contains the text “Top of File”. This indicator for the location of the top of the file is displayed at line “0”, which does not affect lines of code. This is useful when you need a line inserted before the first line of a buffer when in CUA or SlickEdit® text mode emulation. It is also useful when using Selective Display, because a plus sign bitmap can be displayed on line “0”.

To set this option, from the main menu, choose **Tools > Options > General**, then select the [General Tab](#). Select the option **Top of file line**.

Rather than using this option, you can use Ctrl+Shift+Enter (Ctrl+Enter in Visual C++ and Visual Studio emulation) to insert a new line above the line where the cursor is located.



## Displaying a Vertical Line

You can choose to display a vertical line in all files that are open for editing. To access this setting, from the main menu choose **Tools > Options > General**, then select the [General Tab](#). In the **Vertical line column** spin box, specify the column number at which you want the vertical line displayed. A value of “0” (default) displays no vertical line. Click on the colored box to the right of this option to change the color of the vertical line.

## Viewing Line Numbers

The line number of the current cursor position is always shown in the status line of SlickEdit® (along the bottom-right edge of the editor). You can also choose to display line numbers in the left gutter of editor windows.

To toggle display of line numbers for the current document, from the main menu click **View > Line Numbers**, or use the `view_line_numbers_toggle` command on the SlickEdit command line.

To always display line numbers for any file with a specific extension, complete the following steps:

1. From the main menu, click **Tools > Options > File Extension Setup**.
2. From the **Extension** drop-down list, select the extension to work with.
3. Click the [General Tab](#), then click the **Display line numbers** check box.



# Language-Specific Editing

This chapter contains the following topics:

- [Language-Specific Editing Overview](#)
- [C and C++](#)
- [Java](#)
- [XML and HTML](#)
- [XML/HTML Formatting](#)
- [Ada](#)
- [COBOL](#)
- [Pascal](#)
- [PL/I](#)
- [Python](#)



## Language-Specific Editing Overview

---

Many features in SlickEdit® are language-specific and based on the language editing mode. You can also configure different settings for different languages. See [Language Editing Modes](#) and [Extension Options](#) below for more information.

This chapter also includes specific information about extension options, beautifiers, and more for the following languages:

- [Ada](#)
- [C and C++](#)
- [COBOL](#)
- [Java](#)
- [Pascal](#)
- [PL/I](#)
- [Python](#)
- [XML and HTML](#)

## Language Editing Modes

SlickEdit uses the extension of the current file to determine what language you are using, thereby only making available the options and features that are possible in that language. If you have a file with a non-standard extension or no extension at all, you will need to manually specify the editing mode. To specify a mode, from the main menu choose **Document > Select Mode** (or use the `select_mode` command). The Select Mode dialog is displayed with a list of modes from which to select.

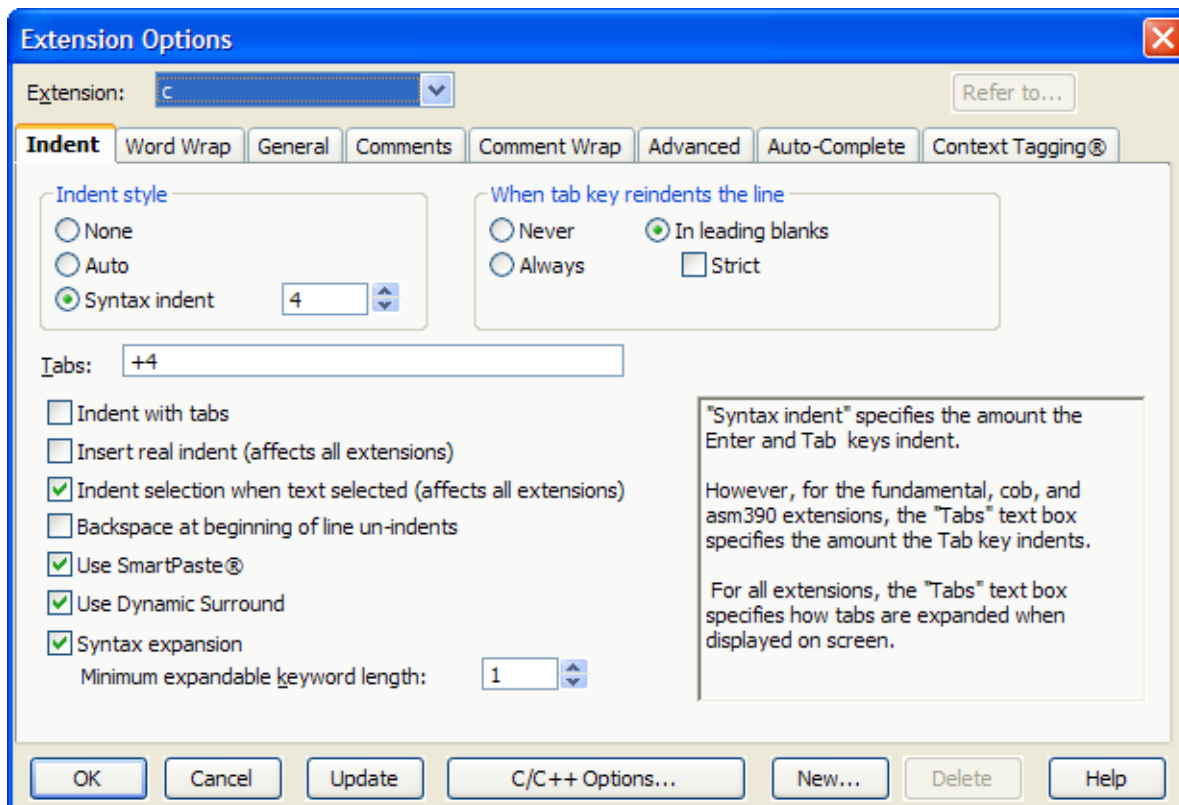
## Changing and Creating Modes

To change or create modes, from the main menu choose **Tools > Options > File Extension Setup**, then select the [General Tab](#). If you want to change the name of an existing mode, select the language extension from the **Extension** drop-down list. Then enter the new name in the **Mode name** text box.

If your language is not listed in the Select Mode dialog or in the **Extension** drop-down list, you can create a new mode. Click the **New** button on the Extension Options dialog, then type the language extension (without the Dot) in the **Extension** text box. If the language is similar to another language that is already available, you can select it from the **Refer to** combo box. This will cause the new extension's configuration to match the configuration of the referred existing language. See [Referring to Extensions](#) for more information.

## Extension Options

The behavior of the editor can be customized for files based on specific language extensions. The Extension Options dialog box (shown below) contains the settings that can be configured for file extensions. This dialog can be accessed from the main menu by choosing **Tools > Options > File Extension Setup**, or by using the `setupext` command.



Be sure to select the extension you want to affect from the **Extension** drop-down list before configuring any settings. For a complete of options and buttons on this dialog, see [Extension Options Dialog](#).

## Referring to Extensions

When an extension refers to another extension, both extensions operate exactly the same. That is, all Context Tagging®, template editing, word processing options, and all other extension options are the same. In addition, modifying the extension option information for either extension updates both extensions. For example, by default, the H and CPP extensions refer to the C file extension. Modify the H or CPP extension setup to modify the extension setup for all three extensions. In addition, the H and CPP extensions use the same Context Tagging settings as the C extension.

To have the setup data for one extension refer to another extension, click the **Refer to** button (located at the top right corner of the Extension Options dialog box). This button is disabled if other extensions already refer to this one. **Refer to** is also available on the New Extension dialog box to set when adding a new extension.

## Creating a New Extension

If SlickEdit® does not provide options for a language extension that you are working with, you can add the extension. On the Extension Options dialog, click the **New** button, and the following dialog is displayed.



Enter the new extension in the **Extension** text box (without the Dot). If the language is similar to another language that is already available, and you wish to have the new language configuration the same as an existing one, you can select the language to refer to from the **Refer to** combo box.

After a new extension is added, you can change its reference at any time by selecting it from the **Extension** drop-down list on the Extension Options dialog, and then by clicking the **Refer to** button. See [Refer-ring to Extensions](#) for more information.

## Deleting an Extension

To delete the selected file extension's setup information, click the **Delete** button (located at the bottom of the Extension Options dialog box). The Fundamental extension setup information cannot be deleted. An extension such as C, that has other extensions (such as H and CPP) that refer to it, also cannot be deleted until all of the extensions that refer to it are deleted.





## C and C++

---

This section describes some of the advanced features and options that are available in SlickEdit® for C and C++, including extension-specific formatting options, the C/C++ Beautifier, compiler settings, and pre-processing.

SlickEdit's default editing mode for C and C++ allows for programming in either language. If you are coding to strict ANSI C standards, you should configure the value of the macro variable **def\_ansi\_exts** to contain a space-delimited list of extensions for files you want interpreted as ANSI C. To set the macro variable, press Esc to bring up the SlickEdit command line, then type **set-var def\_ansi\_exts "<extensions>"**, where **<extensions>** is the space-delimited list of extensions.

For example:

```
set-var def_ansi_exts "c h"
```

Please note that if you also code in C++ and any of these extensions are used for C++, they will be interpreted as ANSI C.

## C/C++ Formatting Options

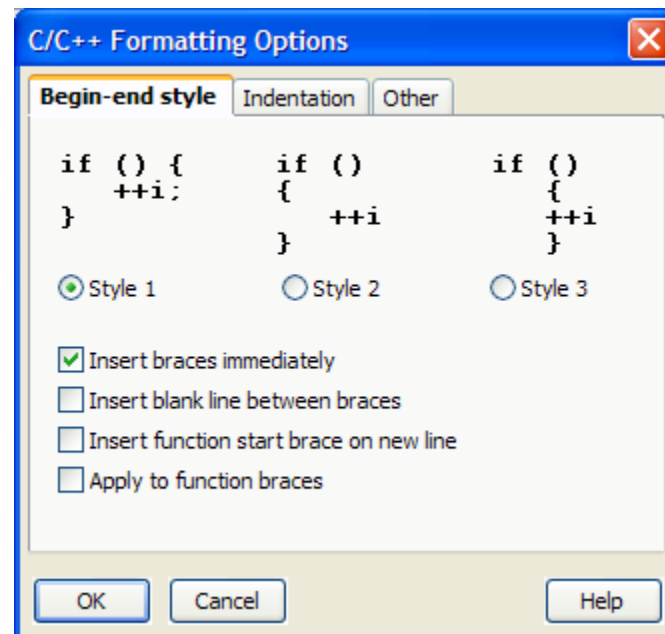
Options are available for C and C++ language file extensions for changing smart indenting and styles for template editing. To access these options, from the main menu, choose **Tools > Options > File Extension Setup**. Make sure the C/C++ language extension you want to work with is selected in the **Extension** drop-down list, then click the **Options** button. The Formatting Options dialog specific to the file extension you have selected is displayed.

**NOTE** Languages similar to C/C++ have similar Formatting Options dialogs which are not specifically documented.

The tabs on the C/C++ Formatting Options dialog are described below.

### Begin-End Style Tab

The Begin-End Style tab is pictured below.

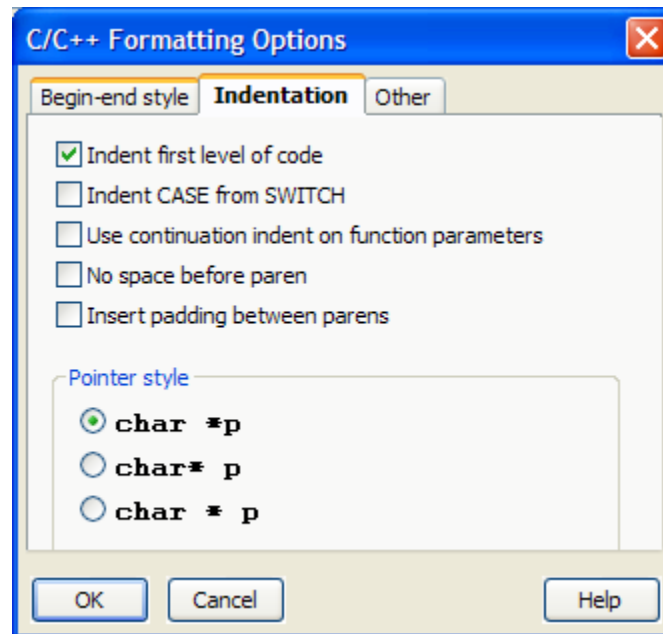


Use this tab to specify the brace style used by template editing and smart indenting. Choose the style you want to use then select from the following options:

- **Insert braces immediately** - Specifies whether template should be inserted with braces.
- **Insert blank line between braces** - Specifies whether a blank line should be inserted between braces when a template expands with braces.
- **Insert function start brace on new line** - Specifies whether a function start brace should be inserted after Enter is pressed to start a new line.
- **Apply to function braces** - When this option is selected, the your begin/end style will be applied to braces for function definition.

## Indentation Tab

The Indentation tab, pictured below, is used to specify indentation options.



The following options are available:

- **Indent first level of code** - Specifies whether smart indenting should indent the cursor after declarations such as functions.
- **Indent CASE from SWITCH** - When checked, template editing places the case statement indented from the switch statement column.
- **Use continuation indent on function parameters** - Determines whether function parameters should always use the continuation indent.

By default, we format multi-line function parameters as follows:

```
myLongMethodName ( firstarg,
                    secondarg,
                    thirdarg
                  );
myLongMethodName (
    firstarg,
    secondarg,
    thirdarg
  );
```

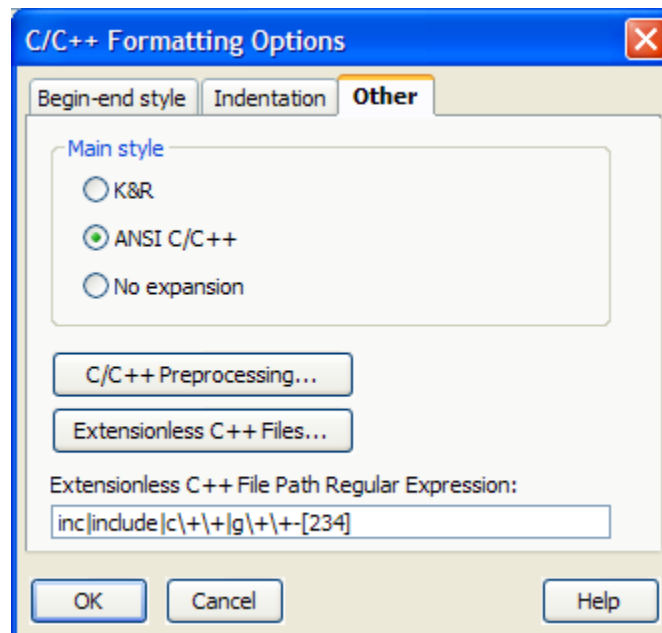
If **Always use continuation indent on function parameters** is selected, the format will change as follows:

```
myLongMethodName( firstarg,
    secondary,
    thirdarg
);
myLongMethodName(
    firstarg,
    secondary,
    thirdarg,
);
```

- **No space before paren** - Determines whether a space is placed between a keyword such as **if**, **for**, or **while** and the open paren when syntax expansion occurs. Example: **(if( or if (**).
- **Insert padding between parens** - When checked, a space is placed after the open paren, and before the close paren, providing padding for the enclosed text. For example, `if ( )` becomes `if ( )`.
- **Pointer style** - Specify the pointer style you wish to use.

## Other Tab

The Other tab of the Formatting Options dialog is pictured below.



The following options are available:

- **Main style** - specifies the **main** function declaration template inserted. Select **ANSI C/C++** if you want an old ANSI C main declaration inserted. You can define a template by using aliases, or you can write a replacement function for **c\_insert\_main**. The command **find-proc c\_insert\_main** will locate the macro source for this function.

- **C/C++ Preprocessing** - Click this button to customize the global preprocessing that is used when Context Tagging® creates tag files for C or C++. See [C/C++ Preprocessing](#).
- **Extensionless C++ Files** - Click this button to add names of extensionless C++ header files. SlickEdit® takes care of the standard STL headers automatically, but you can use this to add additional compiler specific headers, such as **unodered\_set** or **regex**. Note, this setting works in combination with the extensionless header file path regular expression (see below).
- **Extensionless C++ File Path Regular Expression** - In order for SlickEdit to safely recognize an extensionless C++ header file as C++ automatically, without accidentally attempting to open other extensionless files (such as executables) as if they were C++, in addition to requiring that you specify the names of the files (see above), the path that the files are located in must match this regular expression.

## C/C++ Beautifier

To beautify a C or C++ document, open the file you want to beautify, then from the main menu, choose **Tools > Beautify** (or use the **gui\_beautify** command). The C/C++ Beautifier will be displayed, which allows you to make settings for how the code will be beautified.

You can use the commands **c\_beautify** or **c\_beautify\_selection** to instantly beautify the file or the selection according to the settings on the Beautifier dialog.

Currently, this beautifier supports beautifying Slick-C® source if the statements are terminated with semicolons like C.

**NOTE** The C#, Java, JavaScript, and Slick-C Beautifiers contain the same options and settings as the C/C++ Beautifier.

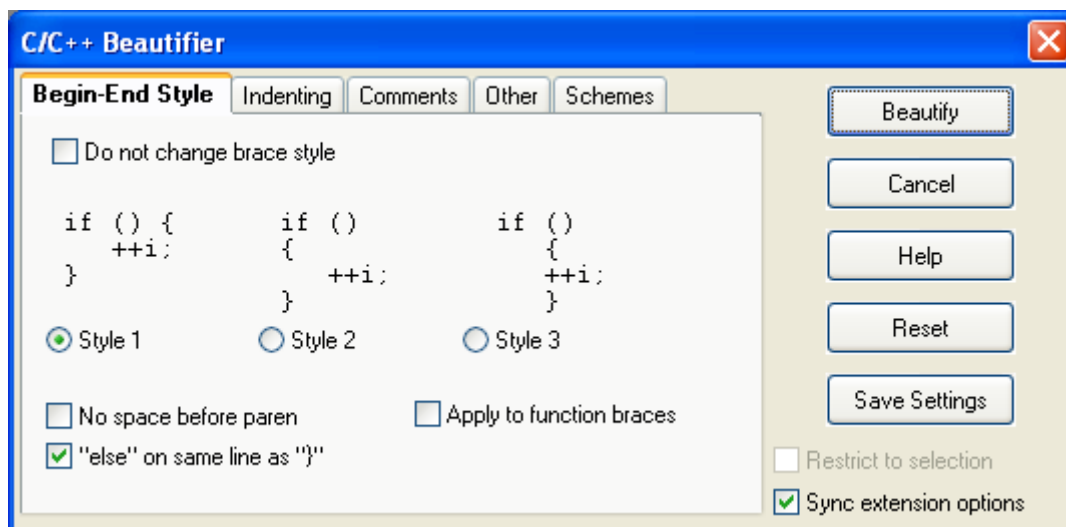
The following buttons and settings are available on the Beautifier:

- **Beautify** - Beautifies current selection or buffer and closes the dialog box.
- **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.
- **Save Settings** - Saves beautify options in `uformat.ini` file. These settings are used by the **c\_beautify** command.
- **Restrict to selection** - When this option is selected, only lines in the selection are beautified.
- **Sync extension options** - When this option is selected, the extension options are updated to reflect any changes that these dialogs have in common. For example, changing the begin-end style to **Style 2** will update your brace style for Syntax Expansion.

The tabs on the C/C++ Beautifier are described in the sections below.

### Begin-End Style Tab

The Begin-End Style tab of the C/C++ Beautifier is pictured below.



The following options and settings are available:

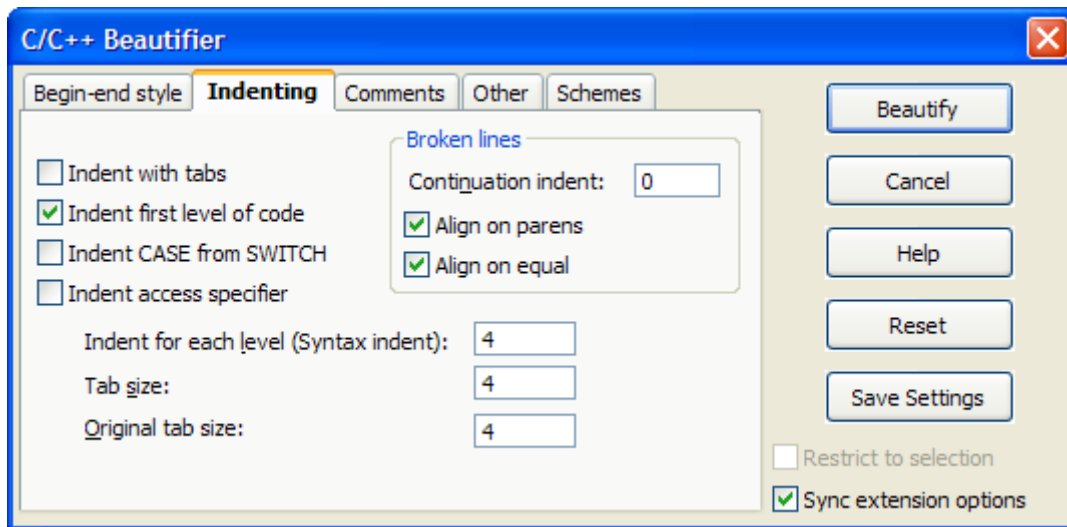
- **Do not change brace style** - Select this option if you do not want your brace style changed. This is useful if you are using a brace style that is not supported by SlickEdit®.
- **No space before paren** - Determines whether a space is placed between a keyword such as **if**, **for**, or **while** and the open paren.
- **else on same line as }** - When this option is selected, the beautifier will place **}** else on the same line. This is typical when using brace style 1. The following is an example of using Style 1 with an **else** clause:

```
if (i<j) {
} else {
}
```

- **Apply to function braces** - When this option is selected, the beautifier will apply your begin/end style to braces for function definition.

## Indenting Tab

The Indenting tab of the C/C++ Beautifier, pictured below, provides indenting parameters that you can use when working with C/C++ files in SlickEdit®.



The following options and settings are available:

- **Indent with tabs** - When this option is selected, tab characters are used for the leading indent of lines. This value defaults to the **Tabs** text box on the [Indent Tab](#) of the Extension Options dialog box (**Tools > Options > File Extension Setup**).
- **Indent first level of code** - Do not clear this check box. When this check box is selected, the first level of code inside a function or method definition is not indented.
- **Indent CASE from SWITCH** - When this option is selected, the case and default statements found inside switch statements are indented from the switch.
- **Indent access specifier** - When this option is selected, specifiers are indented under the class. When not selected, specifiers are aligned directly underneath the class.
- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level of code. We have put the words "Syntax indent" in parenthesis to help indicate that this field has the same meaning as the **Syntax indent** text box on the [Indent Tab](#) of the Extension Options dialog (**Tools > Options > File Extension Setup**). By default, we initialize this text box with your current extension setup setting.
- **Tab size** - The value in this field specifies the output tab size. The output tab size is only used if the option **Indent with tabs** is selected on the [Indent Tab](#) of the Extension Options dialog (**Tools > Options > File Extension Setup**). This value defaults to the **Syntax indent** text box on the [Indent Tab](#) of the Extension Options dialog.
- **Original tab size** - The value in this field specifies the size of the original expansion tab. SlickEdit uses the expansion size of your original file to handle reusing indent amounts from your original file. Currently the beautifier only reuses the original source files indenting for comments. This option has no effect if the original file has no tab characters.
- **Continuation indent** - The value in this field specifies how much to indent lines of statements that continue to the next line. This has no effect on assignment statements or parenthesized expressions. Lines that are a continuation of an assignment statement are indented after the first equal

sign. Lines that are a continuation of a parenthesized expression are indented after the open paren. For example:

```
unsigned
int i;

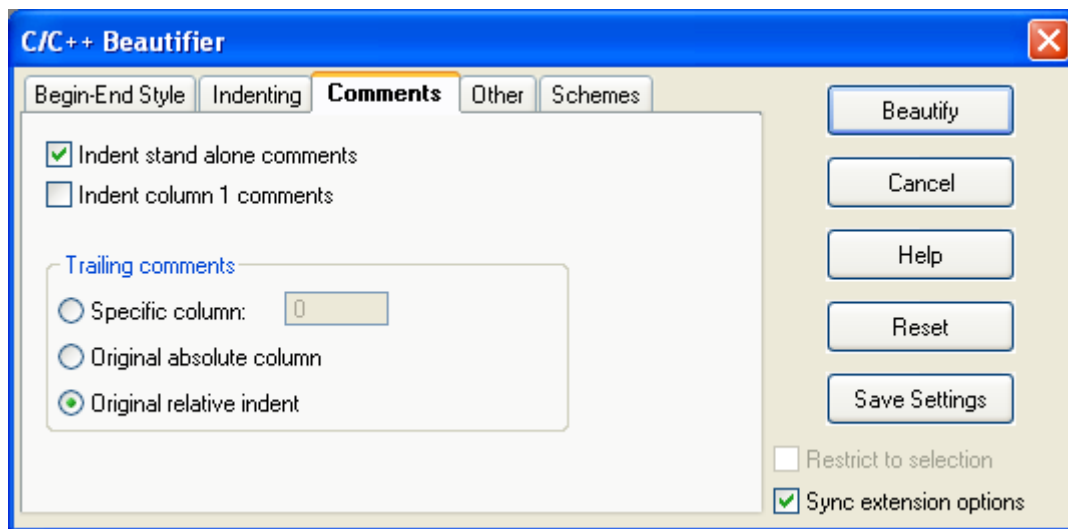
you would get

unsigned
<Continuation Indent>int i;
```

- **Align on parens** - When this option is selected, the text for parenthesized expressions that spans multiple lines is aligned on the first non-blank after the parenthesis or on the parenthesis itself.
- **Align on equal** - When this option is selected, the text for multi-line assignment is aligned on the first non-blank after the equals sign (=) or on the equal sign itself.

## Comments Tab

The Comments tab on the C/C++ Beautifier, pictured below, contains options for setting the parameters that you want for the trailing comments.



The following options are available:

- **Indent stand alone comments** - Indicates whether comments that appear on lines by themselves with no statement text to the left are indented to the current statement indent level. For example:

```
/* stand alone
   comment
*/
// another stand alone comment
i=1; // trailing comment
```

- **Indent column 1 comments** - Normally comments that start in column 1 are left alone. Select this option if you want the indent for these comments to be adjusted.



- **Specific column** - This text box specifies the column in which trailing comments should be placed. Trailing comments are comments that appear at the end of lines that contain statements or declarations. For example:

```
// another stand alone comment
/* stand alone
   comment
*/
i=1;  // trailing comment
if (x) {      /* trailing
               comment.
               */
}
```

- **Original absolute column** - When this option is selected, trailing comments are placed at the same column as the original source file. Trailing comments are comments that appear at the end of lines that contain statements or declarations.
- **Original relative column** - When this option is selected, trailing comments are indented by reusing the indent after the last character of the end of the statement or declaration of the original source file. Trailing comments are comments that are displayed at the end of lines that contain statements or declarations. For example, if the original code is as follows:

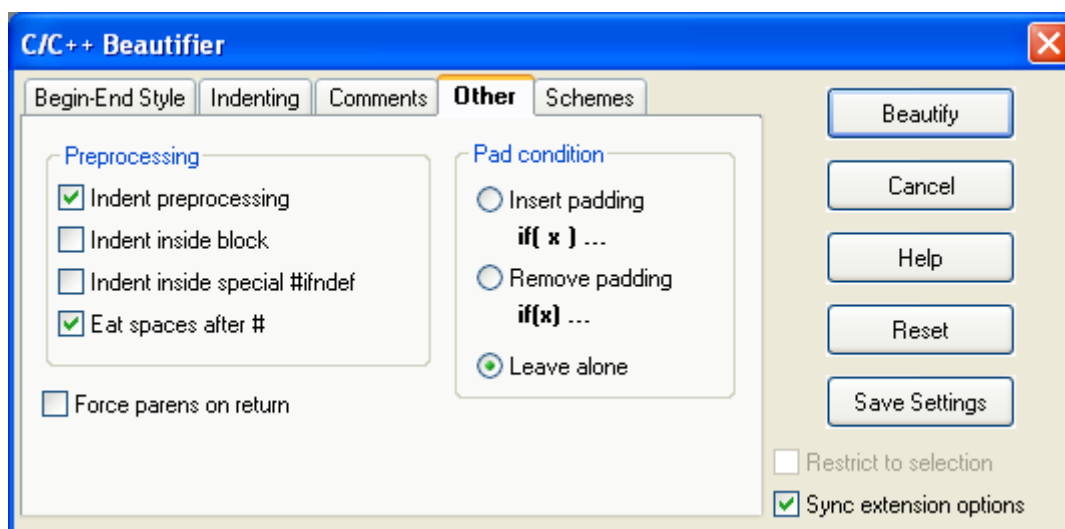
```
if () {
i=1;<four characters>//trailing comment
i=4;<four characters>/* trailing
                      comment.
                      */
}
```

The resulting code would be:

```
if () {
    i=1;<four characters>//trailing comment
    i=4;<four characters>/* trailing
        <four characters>  comment.
        <four characters>*/
}
```

## Other Tab

The Other tab on the C/C++ Beautifier, pictured below, contains the preprocessing and pad condition options.



The following options are available:

- **Indent preprocessing** - When this option is selected, the indent before the # character of preprocessing is set to indicate the preprocessing nesting level.
- **Indent inside block** - When this option is selected, preprocessing inside brace block is indented when inside preprocessing. Otherwise, preprocessing within a brace block start in column 1.
- **Indent inside special #ifndef** - Many C/C++ header files starts with the following lines of code:

```
#ifndef myheader_h
#define myheader_h

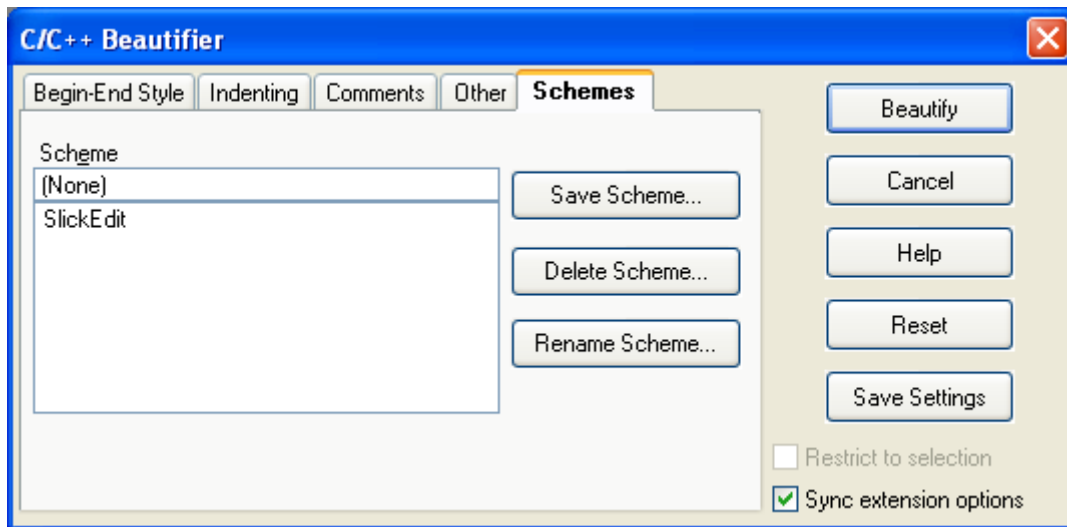
#endif
```

When this option is selected, preprocessing inside this special #ifndef case is indented.

- **Eat spaces after #** - When this option is selected, the spaces after a preprocessor #, but before the keyword (**if**, **ifdef**, **else**, **elif**, **endif**, etc.), are removed. This is useful for fixing old C code where the # character had to start in column 1 and spaces were used after the # to indicate the nesting level.
- **Force parens on return** - When this option is selected, parentheses are added to returns statements which do not have parentheses.
- **Pad condition** - These options indicate if parenthesized conditional expressions should have their spacing adjusted.

## Schemes Tab

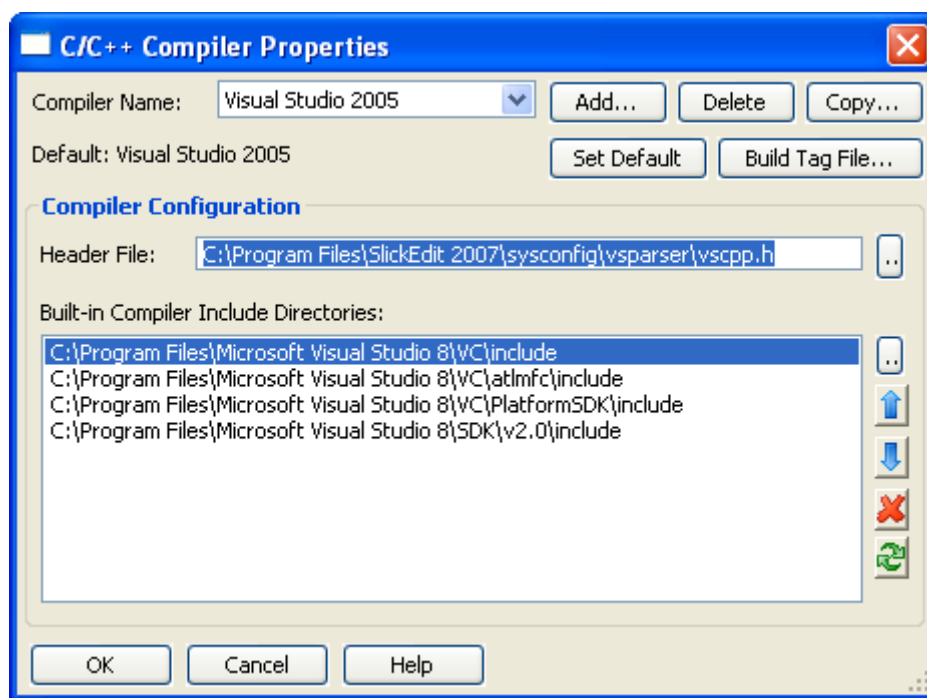
The Schemes tab of the C/C++ Beautifier is pictured below.



To define a new scheme, set the various beautify options, and press the **Save Scheme** button. User defined schemes are stored in `uformat.ini`.

## C/C++ Compiler Settings

In order to correctly perform full preprocessing, parsing, symbol analysis, and cross-referencing, SlickEdit® needs to emulate the implementation-specific parsing behavior of your compiler, including built-in functions, preset #defines, and include directories. This is accomplished by using the C/C++ Compiler Properties dialog box, shown below. To access the dialog, select **Tools > C++ Refactoring > C/C++ Compiler Options**. The dialog can also be displayed by selecting **Project > Project Properties**, selecting the [Compile/Link Tab](#), then clicking the **Triple Dot** icon button to the right of the **Compiler** combo box.



The C/C++ Compiler Properties dialog displays not only the chosen compiler, but also the associated header file and include directories. Collectively, this is a configuration. Configurations can be created and modified as needed.

In the **Compiler Name** drop-down list, select the compiler you wish to use. If this is to be the global default compiler for all projects, click the **Set Default** button.

**NOTE** It is possible to select other compilers for individual projects. In those cases, the project-specific compiler is used and overrides the global default.

SlickEdit ships with header files for each compiler, and the correct header file will appear in the **Header File** field. The header file configures the parser to emulate the compiler that is chosen in the **Compiler Name** field.

## Creating New Configurations

There are two ways to begin a new configuration. In both cases, a dialog box will be invoked, prompting for the name of the new configuration.

- Click **Copy** to copy the selected compiler configuration. This can be used as a template for creating a new configuration and makes the process of creating similar configurations more convenient.
- Or, click **Add** to create a configuration from scratch or to add a newly installed compiler.

If you wish to remove the selected compiler and associated configuration from the list, click **Delete**. This does not delete any files from disk.

## Building the Tag File

The **Build tag file** button on the C/C++ Compiler Properties dialog is used to build tag files from the header file found in the include directories for the selected compiler configuration. This is especially useful when new configurations are created. If you do not build the tag file here manually, it will be built on demand.

## C/C++ Preprocessing

Typically your source code base will include preprocessor macros that you use in your code for portability or convenience. For performance considerations, Context Tagging® does not do full preprocessing, so macros that interfere with normal C++ syntax can cause the parser to miss symbols. For example:

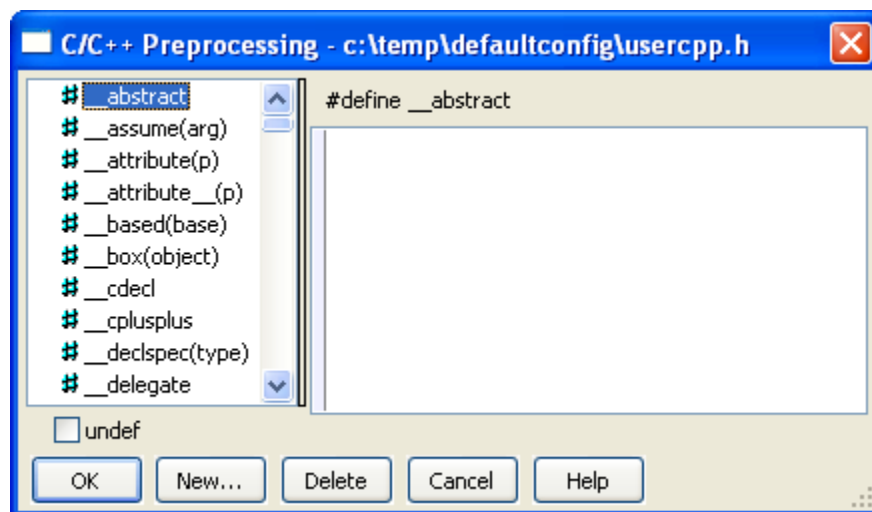
```
MYNAMESPACEDECL(my)
struct MYPACKEDMACRO BinaryTree {
    MYTYPELESS data;
    MYPOINTER(BinaryTree) next;
    MYPOINTER(BinaryTree) prev;
};
MYPOINTER(BinaryTree) proot = MYNULL;
MYENDNAMESPACE
```

This example uses the following preprocessor macros:

```
#define MYNAMESPACEDECL(name) namespace name {
#define MYPACKEDMACRO          __packed
#define MYTYPELESS             void*
#define MYPOINTER(t)           t*
#define MYNULL                  ((void*)0)
#define MYENDNAMESPACE         }
```

Among them, the only two that are harmless are MYTYPELESS and MYNULL, because they just create name aliases for types or constants. However, the other four are troublesome and cause the entire code snippet to be unparsable unless you configure SlickEdit to be aware of these preprocessor macros. To do so, complete the following steps:

1. From the main menu, select **Tools > Options > File Extension Setup**. The Extension Options dialog is displayed.
2. From the **Extension** drop-down list, select the **C** extension.
3. Click the **Options** button at the bottom of the dialog. The C/C++ Formatting Options dialog is displayed.
4. Select the **Other** tab.
5. Click the **C/C++ Preprocessing** button to display the C/C++ Preprocessing dialog.



6. Click **New** to add new preprocessing macros. Arguments are allowed (for example, **mymacro(a,b,c)** ).
7. When finished, click **OK**.
8. A prompt appears asking whether to rebuild your workspace tag file. Click **Yes**.

Preprocessor macros are stored in `usercpp.h`, located in your configuration directory. Rather than using the dialog, you can add large numbers of `#defines` directly to this file. You may want to make sure that your entire development team has an up-to-date copy of this configuration file once you have added all of your local preprocessor macros.

**NOTE** The `usercpp.h` file should only be used for `#defines` and `#undefs`—not `#includes`.

## Java

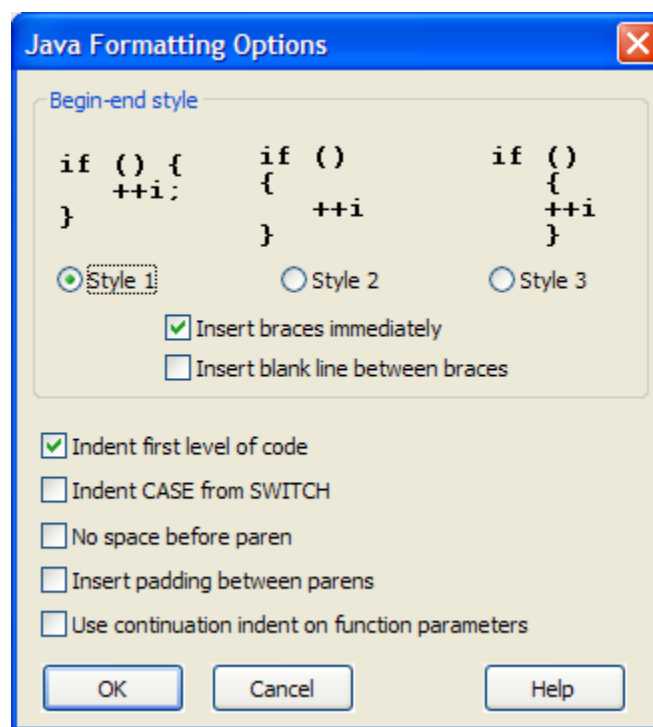
This section describes some of the features and options that are available for Java, including extension-specific options, the Javadoc Editor, and more.

### Java Formatting Options

Options are available for Java language file extensions for changing the smart indenting and template editing style settings. To access these options, from the main menu, choose **Tools > Options > File Extension Setup**. Choose the language extension you wish to work with from the **Extension** drop-down list, then click the **Options** button. The Formatting Options dialog specific to the file extension you have selected will be displayed.

**NOTE** Languages similar to Java have similar Formatting Options dialogs which are not specifically documented.

The Java Formatting Options dialog is pictured below.



The following settings are available:

- **Begin-End Styles** - Specify the brace style to be used for template editing and smart indenting, then choose from the following options:
  - **Insert braces immediately** - Specifies whether template should be inserted with braces.
  - **Insert blank line between braces** - Specifies whether a blank line should be inserted between braces when a template expands with braces.

- **Indent first level of code** - Specifies whether smart indenting should indent the cursor after declarations such as functions.
- **Indent CASE from SWITCH** - When checked, template editing places the case statement indented from the switch statement column.
- **No space before paren** - Determines whether a space is placed between a keyword such as **if**, **for**, or **while** and the open paren when syntax expansion occurs. Example: **(if( or if ( )**.
- **Insert padding between parens** - When checked, a space is placed after the open paren, and before the close paren, providing padding for the enclosed text. For example, `if ( )` becomes `if ( )`.
- **Use continuation indent on function parameters** - Determines whether function parameters should always use the continuation indent.

By default, we format multi-line function parameters as follows:

```
myLongMethodName(firstarg,
secondarg,
thirdarg
);
myLongMethodName(
    firstarg,
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            createdButtonFired(buttonIndex);
        }
    },
    thirdarg
);
myLongMethodName(new ActionListener() { // special case anonymous class first
argument
    public void actionPerformed(ActionEvent e) {
        createdButtonFired(buttonIndex);
    }
},
secondarg,
thirdarg
);
myLongMethodName(
    secondarg,
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            createdButtonFired(buttonIndex);
        }
    },
    thirdarg
);
```

If **Use continuation indent on function parameters** is selected, the format will change as follows:



```

myLongMethodName(firstarg,
    secondarg,
    thirdarg
);
myLongMethodName(
    firstarg,
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            createdButtonFired(buttonIndex);
        }
    },
    thirdarg
);
myLongMethodName(new ActionListener() {    // special case anonymous class first
argument
    public void actionPerformed(ActionEvent e) {
        createdButtonFired(buttonIndex);
    }
},
    secondarg,
    thirdarg
);

```

## Java Beautifier

To beautify Java source code, from the main menu choose **Tools > Beautify** (or use the **gui\_beautify** command). The Java Beautifier dialog appears, where you can specify preferences for how the code is beautified. The Java Beautifier contains the same options and settings as the C/C++ Beautifier. See [C/C++ Beautifier](#) for more information.

## Javadoc Beautifier

To beautify Javadoc comments or set up Javadoc Beautifier options, first invoke the Javadoc Editor by right-clicking within the edit window and selecting **Edit Javadoc Comments**. Then click the **Options** button. The Javadoc Beautifier Options dialog is displayed. The following settings are available:

- **Align parameter comments to longest parameter name** - If checked, the parameters are aligned to the length of the longest parameter name. If the parameter name length is less than the minimum length, the minimum length is used. If the parameter length is longer than the maximum parameter length, the description for the parameter will start on the next line.
- **Align exception comments to longest exception name** - If checked, the exceptions are aligned to the length of the longest exception name. If the exception name length is less than the minimum length, the minimum length is used. If the exception length is longer than the maximum exception length, the description of the parameter will start on the next line.
- **Align return comments** - Indicates whether @return comments should be aligned to the first line of comment text. No alignment is performed if tags which are indent-sensitive such as the **<pre>** tag are used.
- **Align deprecated comments** - Indicates whether @return comments should be aligned to the first line of comment text. No alignment is performed if tags which are indent-sensitive such as the **<pre>** tag are used.

- **Add blank line after parameter comment** - If checked, a blank line is added if a tag follows an @param tag.
- **Add blank line after parameter comment group** - If checked, a blank line is added if a tag follows an @param group.
- **Add blank line after return comment** - If checked, a blank line is added if a tag follows the @return tag.
- **Add blank line after description** - If checked, a blank line is added between the description and the first @ tag. This option is ignored if the description contains a custom or unsupported @ tag.
- **Add blank line after example** - If checked, a blank line is added if a tag follows the @example tag.

## Javadoc Editor

Use the Javadoc Editor to generate Javadoc syntax comments for Java, C, C++, JavaScript, and Slick-C®. To access the Javadoc Editor, right-click within the edit window and select **Edit Javadoc Comments**.

To add a custom or unsupported tag, append the tag (with an @ prefix) and its description into the **Description** text box. You can add @serial, @serialField, and @serialData fields this way.

For more information, see Sun's Javadoc documentation at <http://java.sun.com>.

## Organizing Java Imports

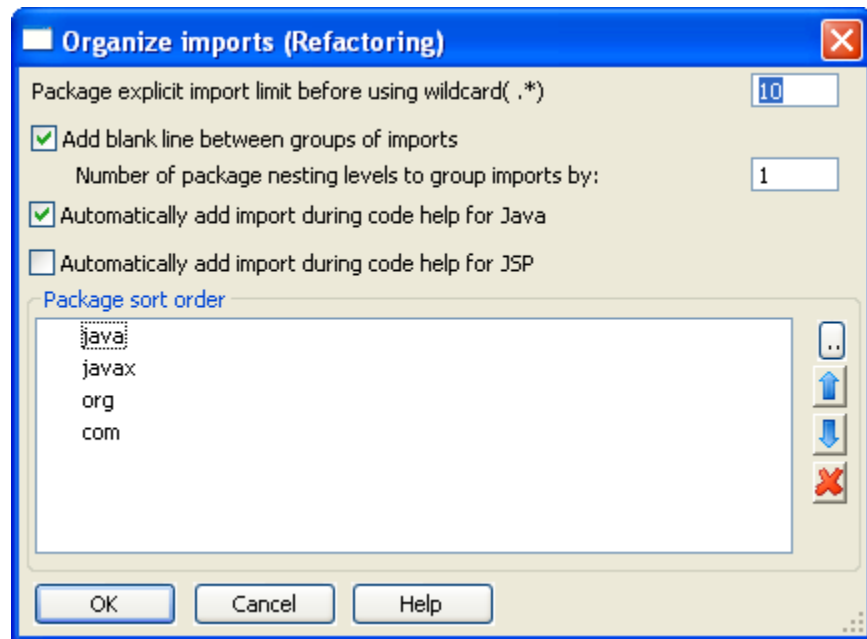
Organizing imports automates the management of import statements in Java files. This feature minimizes the amount of time that it takes to compile code by only importing the classes that are used. Existing import statements are also sorted in a readable format and are more consistent between different Java packages in the same project. Organizing of imports is applied to an entire file. To organize imports, from the main menu choose **Tools > Imports > Organize Imports**.

### Adding Imports

Add Import is used to add an import statement for the class name under the cursor in Java code. Invoke this feature by moving the cursor to the class name you want to import, then going to **Tools > Imports > Add Import**, or using the right-click context menu and selecting **Imports > Add Import**.

### Import Options

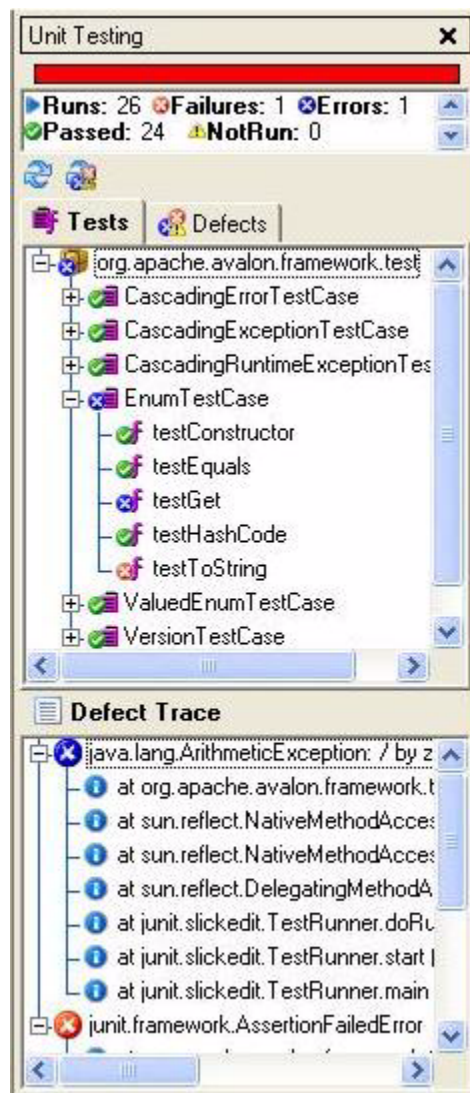
The behavior of the Organize Imports and Add Import features is controlled by the options on the Organize Imports Options dialog box, pictured below. This dialog can be accessed from the main menu by choosing **Tools > Imports > Options**. For a list with descriptions of the options on this dialog, see [Organize Imports Options Dialog](#).



## JUnit Test Support

JUnit tests can be run from within SlickEdit®. The results can be viewed and code that fails the testing can be easily reconciled. To work with the JUnit test support, complete the following steps:

1. From the Projects tool window, right-click the project, package or file that you want and select **Unit Test**.
2. Select **Run** or **Debug**. The results are displayed in the [JUnit Toolbar](#).



3. Double-click the items found on the Tests or Defects tab to be redirected to the code that needs to be debugged.

## JUnit Toolbar

The JUnit toolbar can be resized, docked, contains the Tests and Defects tabs. It also contains labels to display the number of tests: which ran, which passed, which failed, which had errors, and which did not run. The JUnit toolbar also contains a tree control to display the defect trace(s) for the currently selected test item, a button for rerunning the last set of tests, and a button that allows you to re-run only the tests with defects from the last set of tests run.

## Java Live Errors

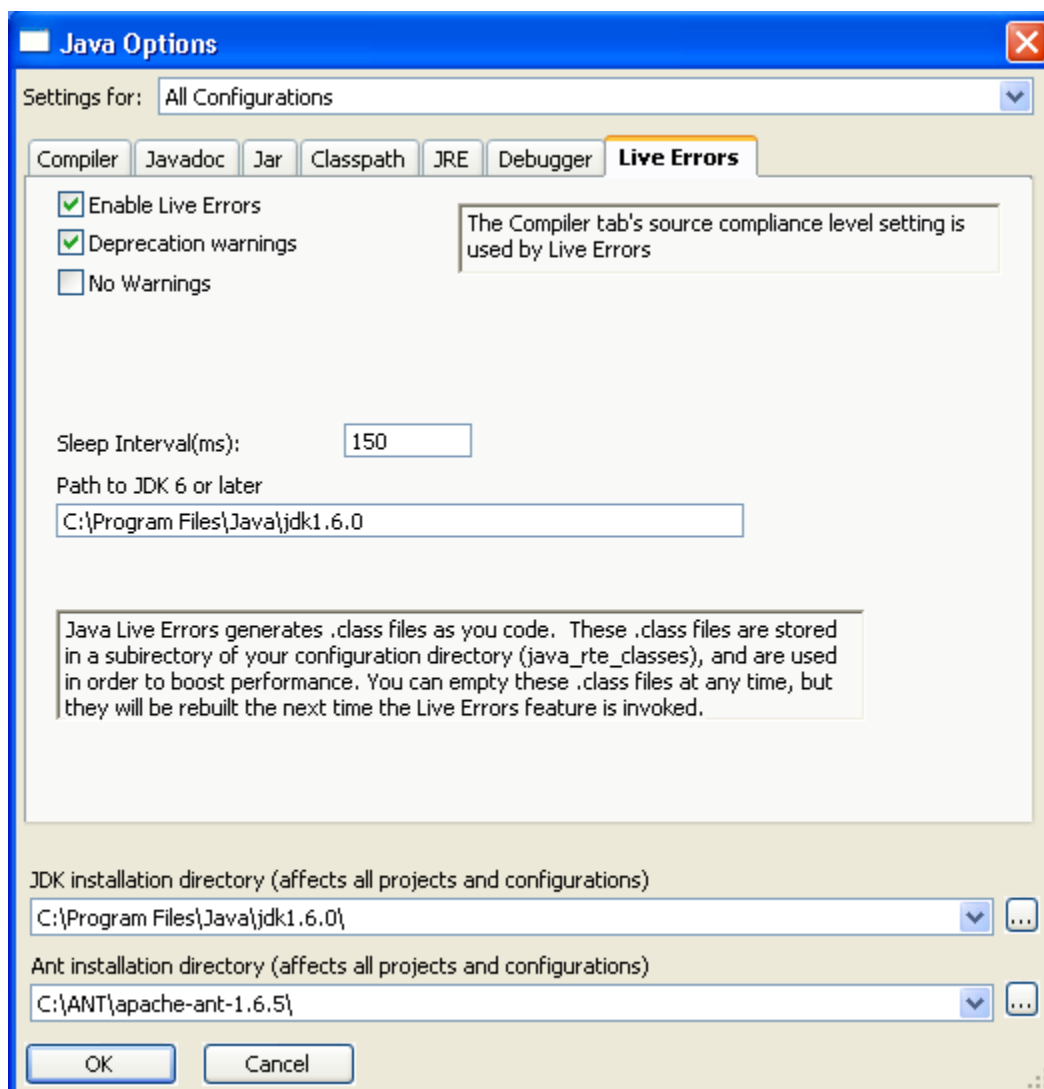
The Java Live Errors feature flags syntax and compilation errors as you edit your code. This feature also provides coding "best practice" warnings, and can be configured to accommodate any source compliance level.

Versions prior to SlickEdit 2007 used Jikes to compile your code, but Jikes does not support JDK 5 and later. Java Live Errors now uses javac from Sun's JDK 6.

Java Live Errors uses the source compliance level specified on the **Compiler tab** of the Java Options dialog (**Build > Java Options**). To enable Live Errors, use the **Live Errors tab**. After enabling Live Errors, you can use the `rte_next_error` command to jump through the live errors in the current file. Bind this command to a key sequence for more efficiency.

## NOTES

- For Live Errors to work, you must have the full JDK downloaded and installed from Sun, and you must specify the root of the JDK installation on the **Live Errors tab** of the Java Options dialog (**Build > Java Options**), unless it is automatically detected upon startup. There is no requirement that you build your code with JDK 6, only that it is available to Live Errors.
- If you have Live Errors running and wish to specify a different JDK 6 root, after changing the path on the Java Options dialog, you must restart SlickEdit.



The Live Errors tab of the Java Options dialog provides the following options and settings:

- **Enable Live Errors** - When checked, the Live Errors feature is activated. This setting is checked by default if SlickEdit has detected a valid JDK 6 installation on your system, the first time a Java project is opened.
- **Deprecation warnings** - When checked, a Warning icon appears when Live Errors encounters a keyword that has been deprecated. This is useful for people coding to strict Java standards.
- **No Warnings** - When checked, warning notices from Live Errors will not be displayed.
- **Sleep Interval(ms)** - This value is the amount of time in which a Live Errors thread will sleep before checking for errors (during times when an error check is not forced). The minimum and recommended value is 150 milliseconds.
- **Path to JDK 6 or later** - This value needs to be the root of a valid JDK 6 installation. Live Errors will not activate if this path is not correct. SlickEdit will attempt to fill in this path for you the first time a Java project is opened.

For more information about working with Java projects, see [Workspaces and Projects](#).

## Working with Apache Ant

Apache Ant is a popular make facility used to build Java components. If you are using Ant, you can either add Ant XML files to an existing project or open the Ant XML file directly. Either way, project files are required from within the application.

### Adding Ant XML Files to a Project

This is the preferred mechanism for working with Ant XML files since a project may contain multiple Ant XML files. After creating the project, add the Ant XML file(s) to the project.

### Opening an Ant XML File

To open an Ant XML file directly, select **Project > Open Other Workspace > Ant XML Build File**. This will either create a project and add the Ant XML file to it or open the existing project.

### Invoking Ant Targets

After you create a project and added the Ant XML file(s), you are now ready to execute Ant targets. To execute a single Ant target, pick the target menu item (**Build > Execute Ant Target**, then navigate to the target). If you need to specify arguments or execute multiple targets, select **Build > Execute Ant Target > Select Multiple Targets**. Alternately, you may right-click on an Ant project in the Projects tool window and either execute one target or multiple targets.

### Setting Shortcuts for Build and Rebuild

To set up the Build menu items or Rebuild menu items or both to invoke a specific set of targets, select **Build > Execute Ant Target > Select Multiple Targets**, and choose **Ant XML File**.

1. Check one or more targets and provide any additional arguments.
2. Check the **Remember and use these settings for** check box.
3. Select **Build** or **Rebuild** in the adjacent combo box.
4. Click **OK**.







# XML and HTML

---

Features for XML and HTML are described below. See also [XML/HTML Formatting](#).

## XML

XML features in SlickEdit® include Context Tagging®, validation, well-formedness checking, beautifier, color coding, URL Mapping, syntax expansion, and syntax indenting for XML, XSLT, and schemas (DTD or XSD).

For information about working with Ant XML files for Java, see [Working with Apache Ant](#).

## XML Toolbar

The XML toolbar is available for quickly accessing common XML operations. To display the XML toolbar, from the main menu, choose **View > Toolbars > XML**. By default, three button icons are available:

- **Beautify selection or entire buffer** - Use this button to instantly beautify the current file according to the Beautifier settings. See [XML Beautifier](#) for more information.
- **Validate XML document** - Use this button to validate an XML file against a DTD or schema. The results of the validation are displayed in the Output tool window. If there are errors during validation, you can double-click on the error line and the appropriate file will be opened and moved to the specified line.
- **Check for Well-Formedness** - Use this button to check if the document is well-formed according to XML syntax rules.

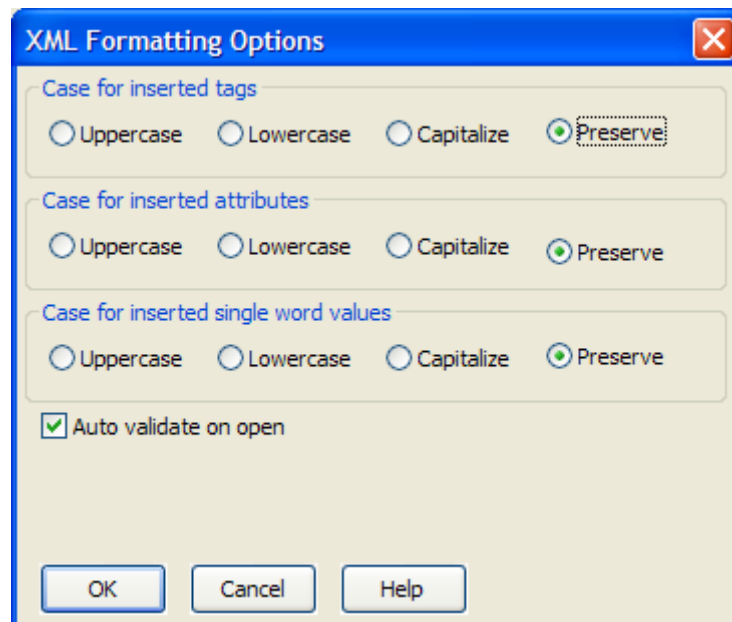
## XML Formatting Options

Content in XML and HTML files may be set to automatically wrap and format as you edit, through the XML/HTML Formatting feature. See [XML/HTML Formatting](#) for complete information.

Other miscellaneous tag and attribute options are still provided through the XML Formatting Options dialog. To access these options, select **Tools > Options > File Extension Setup**, choose **xml** from the **Extension** drop-down list, then click the **Options** button. The XML Formatting Options dialog is displayed.

**TIP** You can also display the XML Formatting Options dialog by clicking the **XML Options** button on the XML/HTML Formatting Scheme Configuration dialog (**Tools > Options > XML/HTML Formatting**).

The XML Formatting Options dialog is shown below.



The following settings are available:

- **Case for inserted tags** - This option is to specify if you want your tag names to be lowercase or uppercase. For example, if you select **Uppercase**, then `<tag>` would become `<TAG>`. Under normal circumstances, preserve the case of your XML tags, but for certain special cases (e.g. XHTML) you might want to change this setting.
- **Case for inserted attributes** - This option is to specify if you want attributes cased inside the body of a tag. For example, if you select **Uppercase**, then `<td align=right>` becomes `<td ALIGN=right>`. Under normal circumstances, preserve the case of your XML attributes, but for certain special cases (e.g. XHTML) you might want to change this setting.
- **Case for inserted single word values** - This option is to specify if the case used when inserting the single word values that appear after the equals sign of an attribute inside the body of a tag. This affects any attribute that has an enumerated type for its attribute values. Under normal circumstances you will want to preserve the case of your XML single word values, but for certain special cases (e.g. XHTML) you may want to change this setting.
- **Auto validate on open** - When this option is selected, XML files are automatically validated when they are opened. The result of the validation is displayed on the Output tool window.

## XMLdoc Editor

Use the XMLdoc Editor to generate Microsoft XML syntax comments for C#, C, C++, Java, JavaScript, and Slick-C®. Note that by default, when creating a new comment, the Javadoc Editor is displayed for all file types except C#. To work around this limitation, start an XML comment with `///` and then right-click in the edit window and select **Edit XML Comments**.

Unknown XML tags are left "as is" and not removed.

## XML Beautifier

To beautify XML source code, from the main menu choose **Tools > Beautify** (or use the **gui\_beautify** command). The XML Beautifier dialog appears, where you can specify preferences for how the code is beautified.

**CAUTION** The XML Beautifier is not affected by [XML/HTML Formatting](#). If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

You can use the commands **xml\_beautify** or **xml\_beautify\_selection** to instantly beautify the file or the selection according to the settings on the Beautifier dialog.

**NOTE** The XSD Beautifier contains the same options and settings as the XML Beautifier.

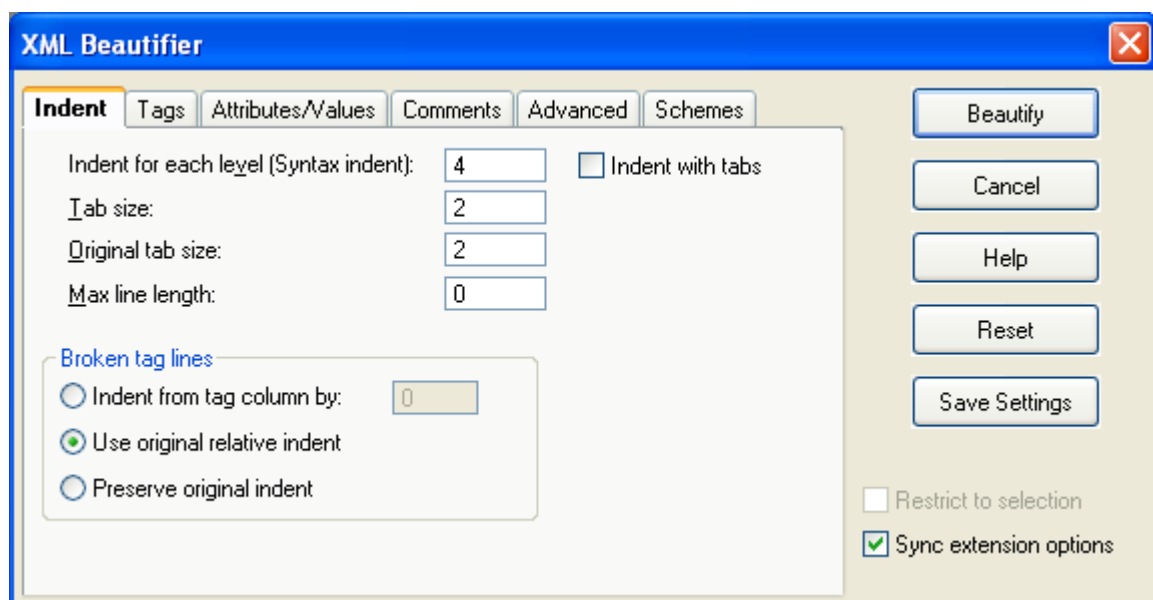
The following buttons and options are available on the Beautifier:

- **Beautify** - Beautifies current selection or buffer and closes the dialog box.
- **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.
- **Save Settings** - Saves beautify options in `uformat.ini` file. These settings are used by the **xml\_beautify** command.
- **Restrict to selection** - When on, only lines in the selection are beautified.
- **Sync extension options** - When on, the extension options are updated to reflect any changes that these dialogs have in common.

The tabs on the XML Beautifier are described in the sections below.

### Indent Tab

The Indent tab on the XML Beautifier is pictured below.

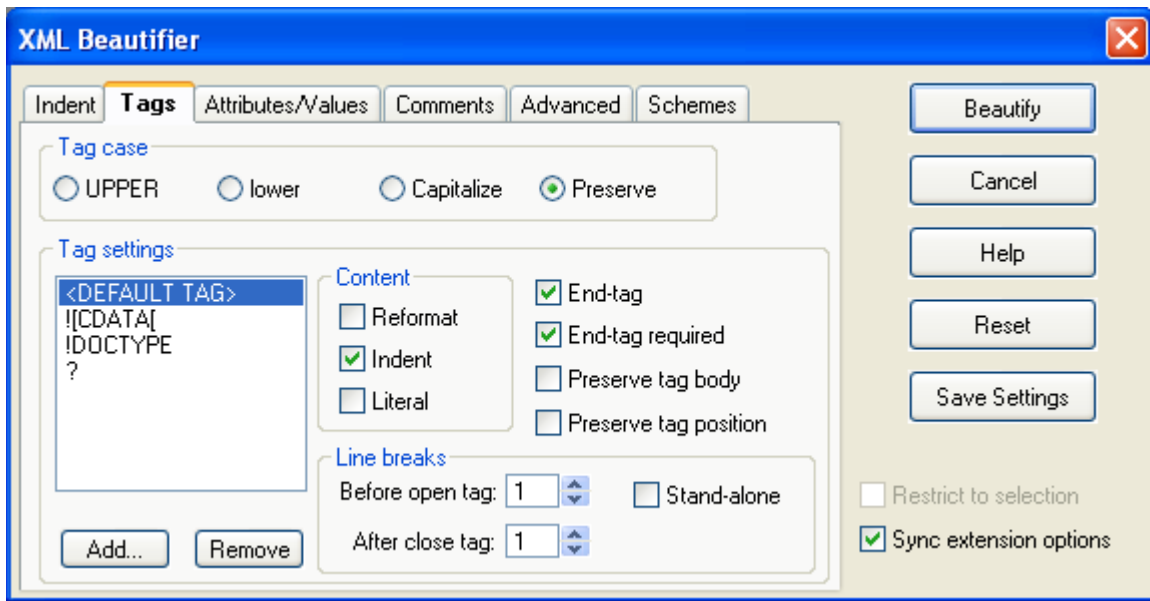


The following settings are available:

- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level of tags. We have put the words “Syntax indent” in parenthesis to help indicate that this field has the same meaning as the **Syntax indent** text box in the Extension Options dialog box (**Tools > Options > File Extension Setup**, [Indent Tab](#)). By default, we initialize this text box with your current extension setup setting.
- **Indent with tabs** - When on, tab characters are used for leading indent of lines. This value defaults to the **Tabs** text box in the Extension Options dialog box (**Tools > Options > File Extension Setup**, [Indent Tab](#)).
- **Tab size** - Specifies output tab size. The output tab size is only used if the **Indent with tabs** check box is on. This value defaults to the **Syntax indent** text box in the Extension Options dialog box (**Tools > Options > File Extension Setup**, [Indent Tab](#)).
- **Original tab size** - Specifies what the original file's tab expansion size was. We need to know the tab expansion size of your original file to handle reusing indent amounts from your original file. Currently the beautifier only reuses the original source file's indenting for comments. This option has no effect if the original file has no tab characters.
- **Max line length** - Specifies the maximum length a line can be before it is wrapped to a new line. This max line length is relative to the current indent level. For example, if you were inside an XHTML <td> block which was at an indent level of 30, and your max line length was set to 80, then that line would not be wrapped until it reached a total length of 30+80=110 characters. Set this value to 0 if you want your line breaks preserved.
- **Broken tag lines** - Specify how broken tag lines are treated from the following options:
  - **Indent from tag column by** - Specifies the amount to indent for broken tag lines from the starting column of the tag. Specify 0 to align broken tag lines with the starting column of the tag.
  - **Use original relative indent** - Reindent broken tag lines using the original relative indent amount from the starting column of the tag.
  - **Preserve original indent** - Preserve the original absolute indent amount on broken tag lines.

### Tags Tab

The Tags tab on the XML Beautifier is pictured below.



The Tags tab contains the following options and settings:

- **Tag case** - Specifies how you want your tag names cased. For example, if you choose UPPER, then <tag> would be beautified to <TAG>. Under normal circumstances you will want to preserve the case of your XML tags, but for certain special cases (e.g. XHTML) you may want to change this setting.
- **Tag settings** - The settings in this group box apply to the tag that is selected in the list box. The <DEFAULT TAG> tag item in the list of tags specifies settings to use when no settings exist for a tag found during beautification.
- **Add** - Display the Add Tag dialog. This dialog allows you to add a tag definition to the list and specify how it will be beautified.
- **Remove** - Used to remove the currently selected tag.
- **Content** - Specify how to beautify content from the following options:
  - **Reformat** - When off, all white space and line breaks are preserved. However, tags are formatted (tag case, attribute case, etc.).
  - **Indent** - When on, nested tags will be indented 1 syntax indent level. Furthermore, if **Reformat** is on, the selected tag's CDATA content (i.e. plain text), bounded by the opening and closing tag, will be indented 1 syntax indent level.
  - **Literal** - When on, all white space and line breaks are preserved. In addition, tags within the content are treated as literal text. If **Reformat** is on, then leading indent is adjusted. This option is useful for XHTML.

**TIP** Some examples of content settings for specific tags are:

- **style** - **Literal** (content is indented to the same level as the <style> open tag)
- **style** - **Reformat, Literal** (content is indented 1 syntax indent level from the <style> open tag)
- **pre** - <all **Content** check boxes off>
- **blockquote** - **Reformat, Indent**

- **End tag** - When on, the selected tag has an end tag. For XML you will normally want this to remain on.
- **End tag required** - When on, the selected tag's ending tag is required. For XML you will normally want this to remain on.
- **Preserve tag body** - When on, all properties of the body of the tag selected will be preserved. This is especially useful for processing instructions like `<?xml ... ?>` where you do not want the embedded text to be beautified.
- **Preserve tag position** - When on, the position of the tag within the document is preserved. This is especially useful with JSP/ASP tags where reindenting the tag would interrupt the flow of the script code.
- **Line breaks** - Select the way lines are broken:
  - **Before open tag** - Specify the number of line breaks before the opening tag. For example, if you were to set the number of line breaks before the opening tag to 3 for the XHTML `<td>` tag, and the original content was:

```
<tr>
<td>
</td>
</tr>
```

The resulting content would be:

```
<tr>

<td>
</td>
</tr>
```

Please note that the number of line breaks is not the same as the number of blank lines. If you wanted three blank lines, then you would set the number of line breaks to 4.

- **After close tag** - Specify the number of line breaks after the closing tag. For example, if you were to set the number of line breaks after the closing tag to 3 for the XHTML `<td>` tag, and the original content was:

```
<TR>
<TD>
</TD>
</TR>
```

The resulting content would be:

```
<TR>
<TD>
</TD>

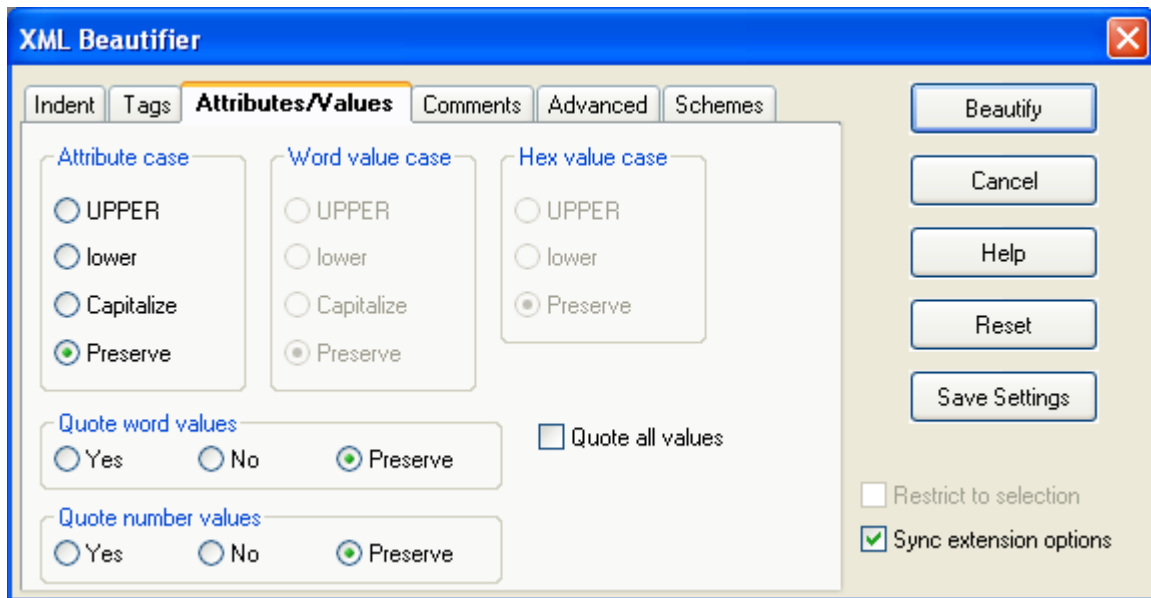
</TR>
```

Please note that the number of line breaks is not the same as the number of blank lines. If you wanted three blank lines, then you would set the number of line breaks to 4.

- **Stand-alone** - When on, the selected tag will always have at least one preceding and trailing line break on both its opening and ending tag when beautified. You can specify that there be more than one line break by setting **Line breaks** for the opening and closing tags.

## Attributes/Values Tab

The Attributes/Values tab of the XML Beautifier is pictured below.

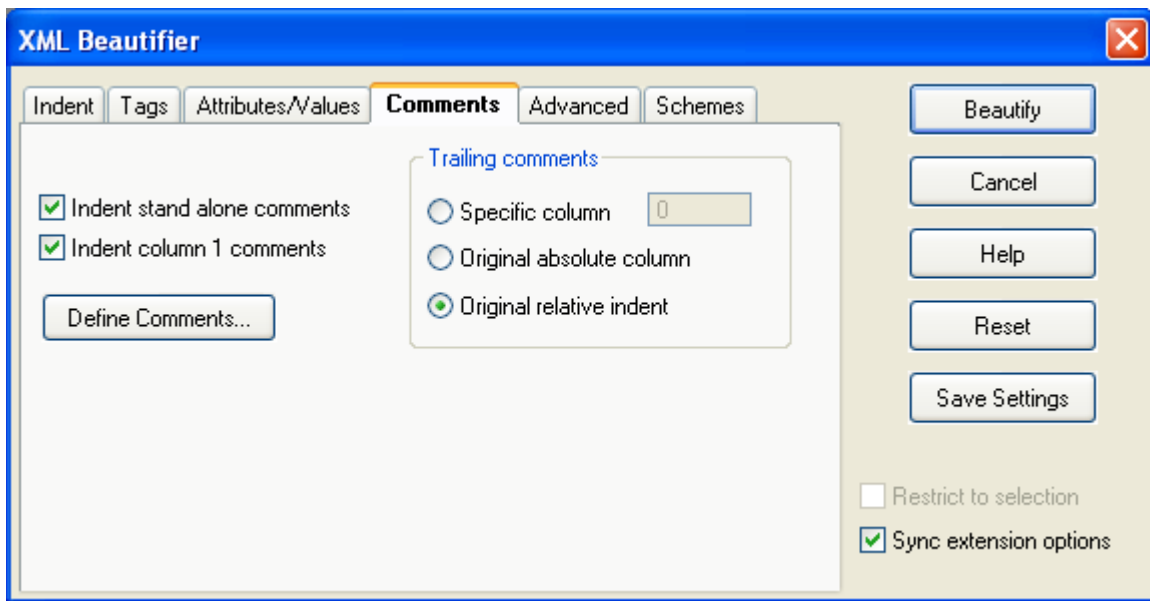


The Attributes/Values tab contains the following settings:

- **Attribute case** - Specifies how you want attributes cased inside the body of a tag. For example, if you choose UPPER, then `<td align="right">` would be beautified to `<td ALIGN="right">`. Under normal circumstances you will want to preserve the case of your XML attributes, but for certain special cases (e.g. XHTML) you may want to change this setting.
- **Word value case** - Not enabled for XML.
- **Hex value case** - Not enabled for XML.
- **Quote word values** - Specifies whether you want word values enclosed in double quotes after the = of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`. Select **Preserve** if you want word values left alone. Under normal circumstances you will want to preserve your XML values, but for certain special cases (e.g. XHTML) you may want to change this setting.
- **Quote number values** - Specifies whether you want number values enclosed in double quotes after the = of an attribute inside the body of a tag. For example, `<td width=590>` would be beautified to `<td width="590">`. Select **Preserve** if you want number values left alone. Under normal circumstances you will want to preserve your XML values, but for certain special cases (e.g. XHTML) you may want to change this setting.
- **Quote all values** - When on, all values will be quoted after the = of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`. Under normal circumstances you will want to preserve your XML values, but for certain special cases (e.g. XHTML) you may want to change this setting.

## Comments Tab

The Comments tab of the XML Beautifier is pictured below.



The Comments tab contains the following options and settings:

- **Indent stand alone comments** - When on, indicates whether comments which appear on lines by themselves with no content to the left are indented to the current content indent level. The following is an example of a stand-alone comment:

```
<!-- stand alone
      comment
-->
```

- **Indent column 1 comments** - Normally comments that start in Column 1 are left alone. Turn this on if you want the indent for these comments to be adjusted as well.
- **Define Comments** - Displays the XML Comments dialog. This dialog allows you to define what the beautifier recognizes as a comment. The sequence `<!-- -->` is defined as the XML comment by default. If you delete all comment definitions then all comments will be parsed as content.
- **Trailing comments** - Specify how trailing comments are treated from the following options:

- **Specific column** - This text box specifies the column that “trailing comments” should be placed at. By trailing comments, we mean comments which appear at the end of lines which contain tags. An example of a trailing comment is:

```
<TD> <!-- trailing comment -->
```

- **Original absolute column** - When on, “trailing comments” are placed at the same column as the original source file. By trailing comments, we mean comments which appear at the end of lines which contain tags.
- **Original relative column** - When on, “trailing comments” are indented by reusing the indent after the last character of the end of the statement or declaration of the original source file. By trailing comments, we mean comments which appear at the end of lines which contain tags.



The following is an example of code before beautifying trailing comments:

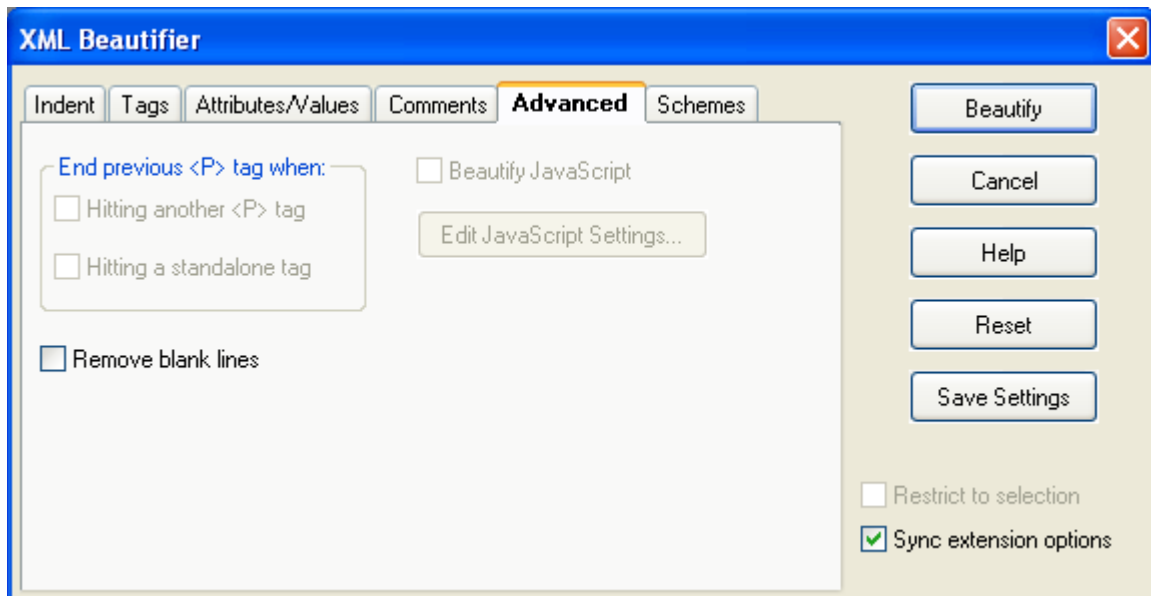
```
<Outer>
<Inner><four characters><!-- trailing comment -->
</Inner>
</Outer>
```

The resulting code would be:

```
<Outer>
  <Inner><four characters><!-- trailing comment -->
</Inner>
</Outer>
```

## Advanced Tab

The Advanced tab of the XML Beautifier is pictured below.

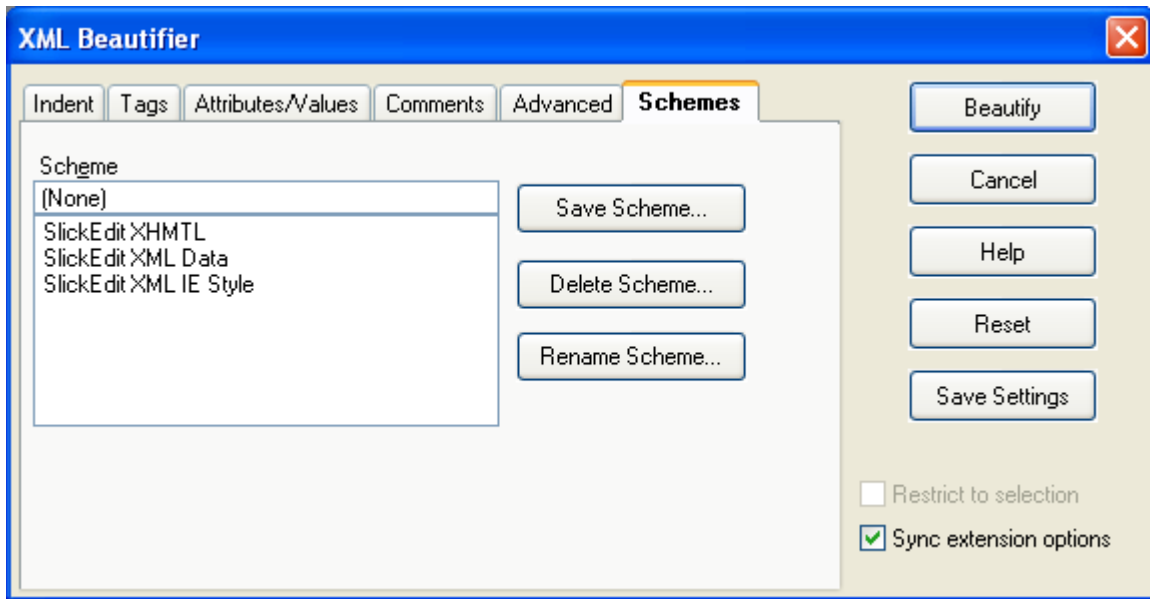


The following option is available on the Advanced tab:

- **Remove blank lines** - When on, blank lines are deleted.

## Schemes Tab

The Schemes tab of the XML Beautifier is pictured below.



To define a new scheme, use the Beautifier to set the various beautify options and then press the **Save Scheme** button on the Schemes tab. User defined schemes are stored in `uformat.ini`.

## DTD Caching

When you open an XML document that has a document type definition of (!DOCTYPE) that refers to a remote external DTD, the DTD file is downloaded and cached locally. The DTD is processed to provide Context Tagging® and better color coding. Currently, only HTTP (and not FTP) remote files are supported. This automatic caching allows you to work offline and edit XML documents that reference remote DTDs when you do not have an Internet connection. If you want to force re-caching of the DTD for the current XML document, right-click to open the context menu and select **Apply DTD changes**. Applying DTD changes is necessary after you create a new XML document and complete the document type definition (!DOCTYPE).

## Opening DTD Files from XML

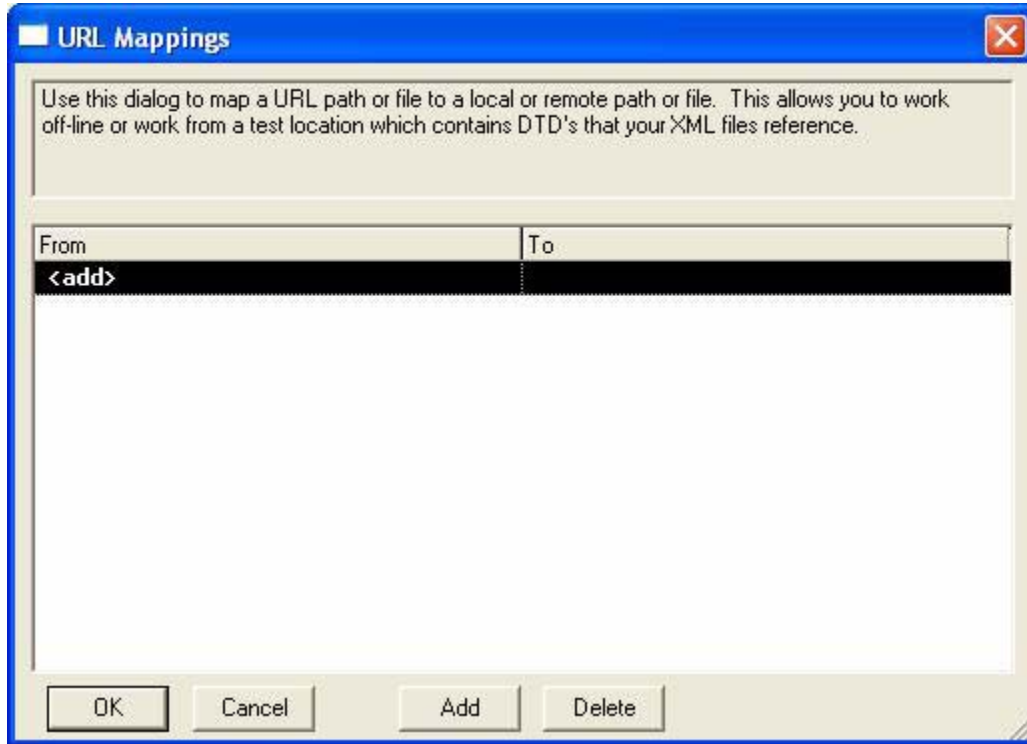
To open the external DTD referenced by document type definition (!DOCTYPE), place the cursor anywhere on the !DOCTYPE tag and press **Alt+1** (or right-click to display the context menu and select **Go to Error/Include File**).

## URL Mappings

Map URLs to a different location. Whenever opening a URL, the URL map is examined to see if this URL is mapped to a different location. If the URL is mapped to a different location, then that mapped location is used.

This feature allows you to work offline or from a test location. For example, if you need to work with XML documents that contain external DTDs while offline you can map the URL to the DTD to a local file. Simi-

larly, if you wanted to test changes to a DTD without modifying every XML documents DTD references, you can map the URL to the test DTD location.



To map a URL, complete the following steps.

1. From the main menu, click **Tools > Options > URL Mappings**.
2. Click in the **From** text box that reads **<add>**.
3. Type in the URL that will be mapped to a different location.
4. Click on the **To** text box and type in the location to use for this URL.
5. Click **OK**.

## Toggling Between Begin and End XML Tags

Place the cursor anywhere on the begin or end tag and press **Ctrl+]** to find the corresponding end or begin tag respectively.

## HTML

This section describes some of the features and options that are available for HTML, including extension-specific options, the HTML Beautifier, and more.

HTML support includes Context Tagging®, a beautifier, color coding, syntax expansion, and syntax indenting for HTML, JSP, and ASP. Many of the language features in SlickEdit® are supported for languages

embedded in HTML, including Context Tagging, color coding, SmartPaste®, syntax expansion, and syntax indenting.

**TIP** When working with HTML files, you can toggle between the begin and end HTML tags by pressing **Ctrl+]**.

## HTML Toolbar

The HTML toolbar is available for many common operations you may want to perform. To display the HTML toolbar, from the main menu, choose **View > Toolbars > HTML**.

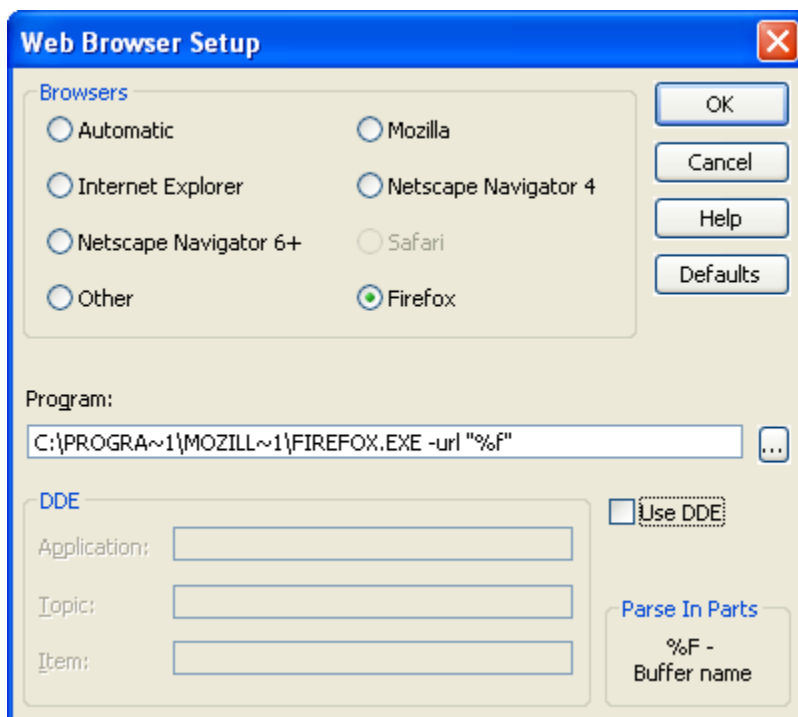
To invoke a Web browser or to display the current file in a browser, use the Web browser icon on the HTML toolbar. To configure the browser that is used, see [Configuring the Web Browser](#) below.

## Exporting to HTML

To save the current open buffer as HTML file with formatting and color coding, from the main menu, select File > Export to HTML (or use the **export\_html** command).

## Configuring the Web Browser

To specify the Web browser that is used for previewing, from the main menu choose **Tools > Options > Web Browser Setup**. The Web Browser Setup dialog is displayed, as shown below. See [Web Browser Setup Dialog](#) for a list of option descriptions.



## HTML Formatting Options

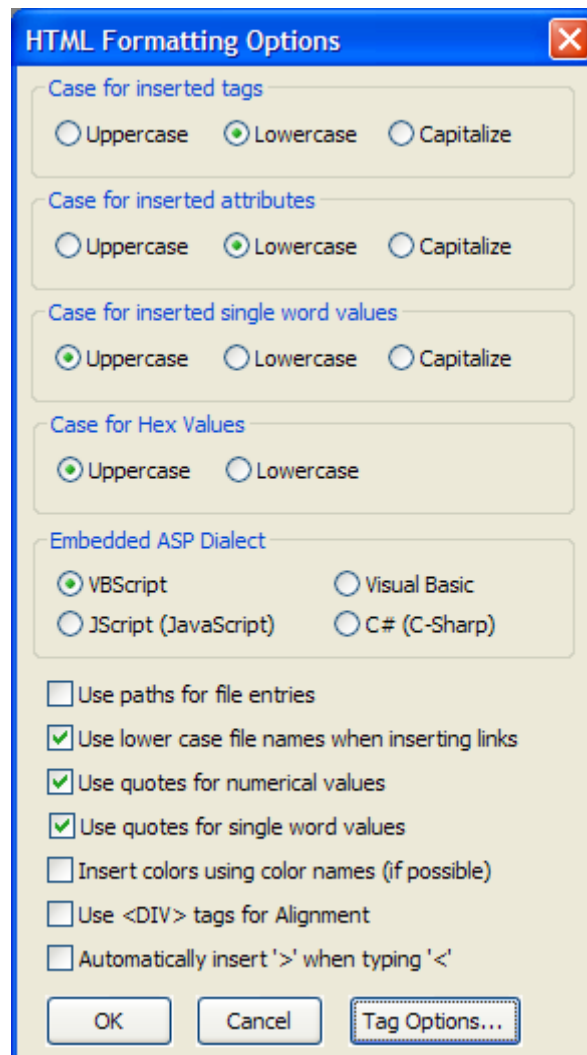
Content in XML and HTML files may be set to automatically wrap and format as you edit, through the XML/HTML Formatting feature. See [XML/HTML Formatting](#) for complete information.

Other miscellaneous tag and attribute options are still provided through the HTML Formatting Options dialog. To access these options, select **Tools > Options > File Extension Setup**, choose **html** from the **Extension** drop-down list, then click the **Options** button. The HTML Formatting Options dialog is displayed.

**TIP** You can also display the HTML Formatting Options dialog by clicking the **HTML Options** button on the XML/HTML Formatting Scheme Configuration dialog (**Tools > Options > XML/HTML Formatting**).

**NOTE** Languages similar to HTML have similar Formatting Options dialogs which are not specifically documented.

The HTML Formatting Options dialog is shown below.



The following settings are available:

- **Case for inserted tags** - This option is where you specify if you want the tag names to be lowercase or uppercase. For example, if you select Uppercase, then `<td>` becomes `<TD>`.
- **Case for inserted attributes** - This option is where you specify if you want attributes to be lowercase or uppercase inside the body of a tag. For example, if you select Uppercase, then `<td align=right>` would become `<td ALIGN=right>`.
- **Case for inserted single word values** - This option is where you specify if you want word values to be uppercase or lowercase when the = of an attribute is inside the body of a tag. For example, if you select Uppercase, then `<td align=right>` becomes `<td align=RIGHT>`.
- **Case for hex values** - This option is where you specify if you want hex values to be uppercase or lowercase after the = of an attribute inside the body of a tag. For example, if you select Uppercase, then `<body bgcolor=#ffffff>` would become `<body bgcolor=#FFFFFF>`.
- **Embedded ASP dialect** - The language that you select here determines the default embedded language mode for ASP files.

- **Use path for file entries** - This option is used by the HTML toolbar. When this attribute is selected, path information is included when inserting file names into the value of an attribute. For example, creating a link with this option turned on might result in the following example.

```
<A HREF=file:///c:/dev/html/index.htm#>sample link</A>
```

When this option is not selected, the result is the following example.

```
<A HREF= index.htm#>sample link</A>
```

- **Use lower case file names when inserting links** - This option is used by the HTML toolbar. When this option is selected, the lower case file names are used when inserting links into the value of an attribute. See the example for Use paths for file entries.
- **Use quotes for numerical values** - When this option is selected, word values are enclosed in double quotes after the = of an attribute inside the body of a tag.
- **Use quotes for single word values** - When this option is selected, number values are enclosed in double quotes after the = of an attribute inside the body of a tag.
- **Insert colors using color names (if possible)** - When this option is selected, colors are inserted by using the color names if possible. For example, instead of using #ff0000 to represent the color red, the color name red is used.

```
<BODY bgcolor=#ff0000>
<!-- and -->
<BODY bgcolor=red>
<!-- are identical -->
```

- **Use <DIV> tags for alignment** - This option is used by the HTML toolbar. When this option is selected, the <DIV> tag is used for aligning text. For example, rather than using a <CENTER> tag to designate alignment, use the following tagging:

```
<DIV ALIGN=CENTER>
</DIV>
```

- **Tag Options** - Opens the XML/HTML Formatting Scheme Configuration dialog. See [XML/HTML Formatting](#) for more information.

## HTML Beautifier

To beautify an HTML document, open the document you want to beautify, then from the main menu, choose **Tools > Beautify** (or use the **gui\_beautify** command). The HTML Beautifier dialog will be displayed, which allows you to make settings for how the code will be beautified.

**CAUTION** The HTML Beautifier is not affected by [XML/HTML Formatting](#). If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

You can use the commands **h\_beautify** or **h\_beautify\_selection** to instantly beautify the file or the selection according to the settings on the Beautifier dialog.

**NOTE** The CFML Beautifier contains the same options and settings as the HTML Beautifier.

The following buttons and options are available on the Beautifier:

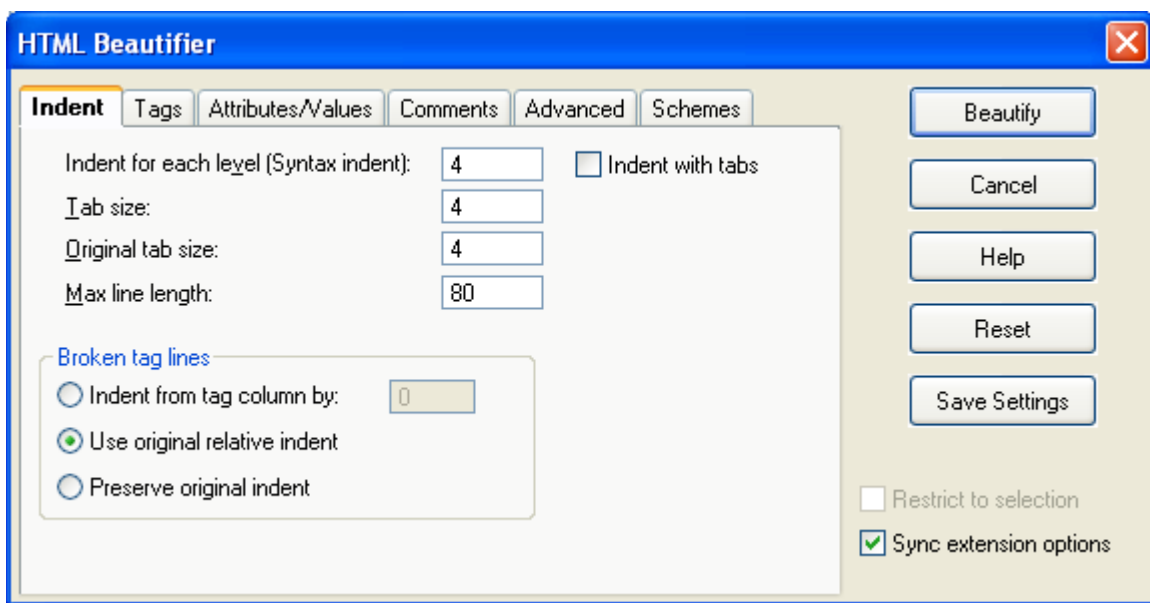
- **Beautify** - Beautifies current selection or buffer and closes the dialog box.

- **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.
- **Save Settings** - Saves beautify options in `uformat.ini` file. These settings are used by the `h_beautify` command.
- **Restrict to selection** - When on, only lines in the selection are beautified.
- **Sync extension options** - When on, the extension options are updated to reflect any changes that these dialogs have in common.

The tabs on the HTML Beautifier are described in the sections below.

## Indent Tab

The Indent tab on the HTML Beautifier is pictured below.



The following settings are available:

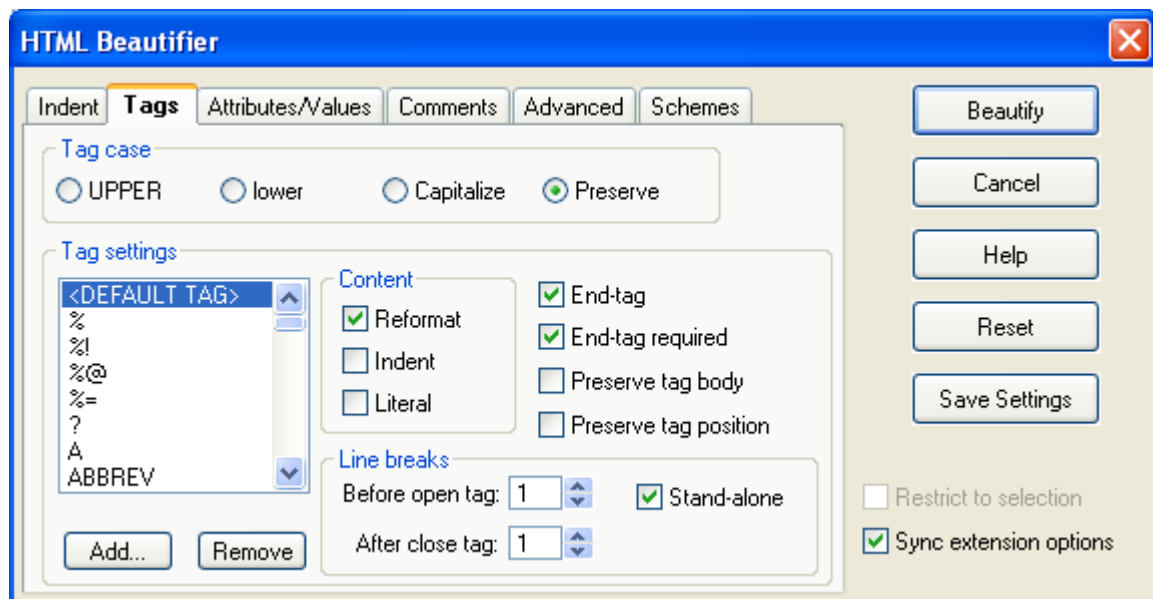
- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level. We have put the words "Syntax indent" in parenthesis to help indicate that this field has the same meaning as the **Syntax indent** text box in the Extension Options dialog box (**Tools > Options > File Extension Setup, Indent Tab**). By default, we initialize this text box with your current extension setup setting.
- **Indent with tabs** - When on, tab characters are used for leading indent of lines. This value defaults to the **Tabs** text box in the Extension Options dialog box (**Tools > Options > File Extension Setup, Indent Tab**).
- **Tab size** - Specifies output tab size. The output tab size is only used if **Indent with tabs** check box is on. This value defaults to the **Syntax indent** text box in the Extension Options dialog box (**Tools > Options > File Extension Setup, Indent Tab**).
- **Original tab size** - Specifies what the original file's tab expansion size was. We need to know the tab expansion size of your original file to handle reusing indent amounts from your original file. Currently the beautifier only reuses the original source file's indenting for comments. This option has no effect if the original file has no tab characters.



- **Max line length** - Specifies the maximum length a line can be before it is wrapped to a new line. This max line length is relative to the current indent level. For example, if you were inside a <TD> block which was at an indent level of 30, and your max line length was set to 80, then that line would not be wrapped until it reached a total length of 30+80=110 characters. Set this value to 0 if you want your line breaks preserved.
- **Broken tag lines** - Specify how broken tag lines are treated from the following options:
  - **Indent from tag column by** - Specifies the amount to indent for broken tag lines from the starting column of the tag. Specify 0 to align broken tag lines with the starting column of the tag.
  - **Use original relative indent** - Reindent broken tag lines using the original relative indent amount from the starting column of the tag.
  - **Preserve original indent** - Preserve the original absolute indent amount on broken tag lines.

## Tags Tab

The Tags tab on the HTML Beautifier is pictured below.



The Tags tab contains the following options and settings:

- **Tag case** - Specifies how you want your tag names cased. For example, if you choose UPPER, then <td> would be beautified to <TD>.
- **Tag settings** - The settings in this group box apply to the tag that is selected in the list box. The <DEFAULT TAG> tag item in the list of tags specifies settings to use when no settings exist for a tag found during beautification.
- **Add** - Display the Add Tag dialog. This dialog allows you to add a tag definition to the list and specify how it will be beautified.
- **Remove** - Used to remove the currently selected tag.
- **Content** - Specify how to beautify content from the following options:

- **Reformat** - When off, all white space and line breaks are preserved. However, tags are formatted (tag case, attribute case, etc.).
- **Indent** - When on, the selected tag's content, bounded by the opening and closing tag, will be indented 1 syntax indent level.
- **Literal** - When on, all white space and line breaks are preserved. In addition, tags within the content are treated as literal text. If Reformat is on, then leading indent is adjusted.

**TIP** Some examples of content settings for specific tags are:

- **XMP - Literal**
- **PRE** - <all **Content** check boxes off>
- **BLOCKQUOTE - Reformat, Indent**
- **STYLE - Reformat, Literal**

- **End tag** - When on, the selected tag has an end tag. For example, the tag <TD> has a ending tag that is </TD>, so **End tag** would be checked in this case.
- **End tag required** - When on, the selected tag's ending tag is required. This means that the ending tag is not optional. An example of a tag whose ending tag could be optional is <P>.
- **Preserve tag body** - When on, all properties of the body of the tag selected will be preserved. This is especially useful for JSP/ASP tags where you do not want the embedded Java or VBScript inside the <% ... %> to be beautified.
- **Preserve tag position** - When on, the position of the tag within the document is preserved. This is especially useful with JSP/ASP tags where reindenting the tag would interrupt the flow of the script code.
- **Line breaks** - Select the way lines are broken:
  - **Before open tag** - Specify the number of line breaks before the opening tag. For example, if you were to set the number of line breaks before the opening tag to 3 for the <TD> tag, and the original content was:

```
<TR>
<TD>
</TD>
</TR>
```

The resulting content would be:

```
<TR>

<TD>
</TD>
</TR>
```

Please note that the number of line breaks is not the same as the number of blank lines. If you wanted three blank lines, then you would set the number of line breaks to 4.

- **After close tag** - Specify the number of line breaks after the closing tag. For example, if you were to set the number of line breaks after the closing tag to 3 for the <TD> tag, and the original content was:

```
<TR>
<TD>
</TD>
</TR>
```

The resulting content would be:

```
<TR>
<TD>
</TD>

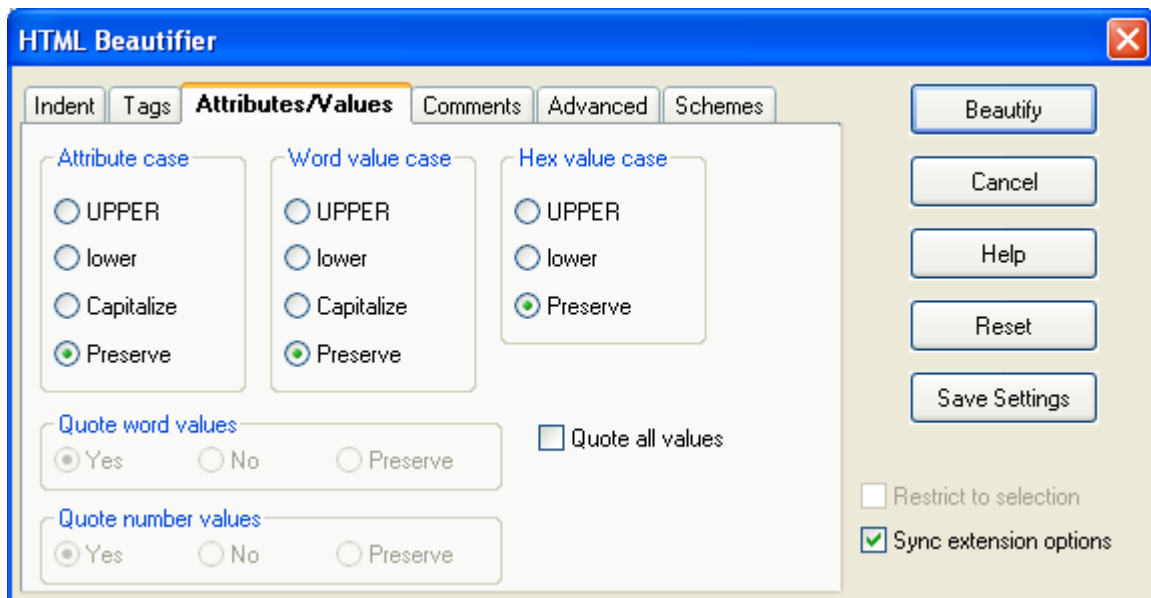
</TR>
```

Please note that the number of line breaks is not the same as the number of blank lines. If you wanted three blank lines, then you would set the number of line breaks to 4.

- **Stand-alone** - When on, the selected tag will always have at least one preceding and trailing line break on both its opening and ending tag when beautified. You can specify that there be more than one line break by setting **Line breaks** for the opening and closing tags.

## Attributes/Values Tab

The Attributes/Values tab of the HTML Beautifier is pictured below.



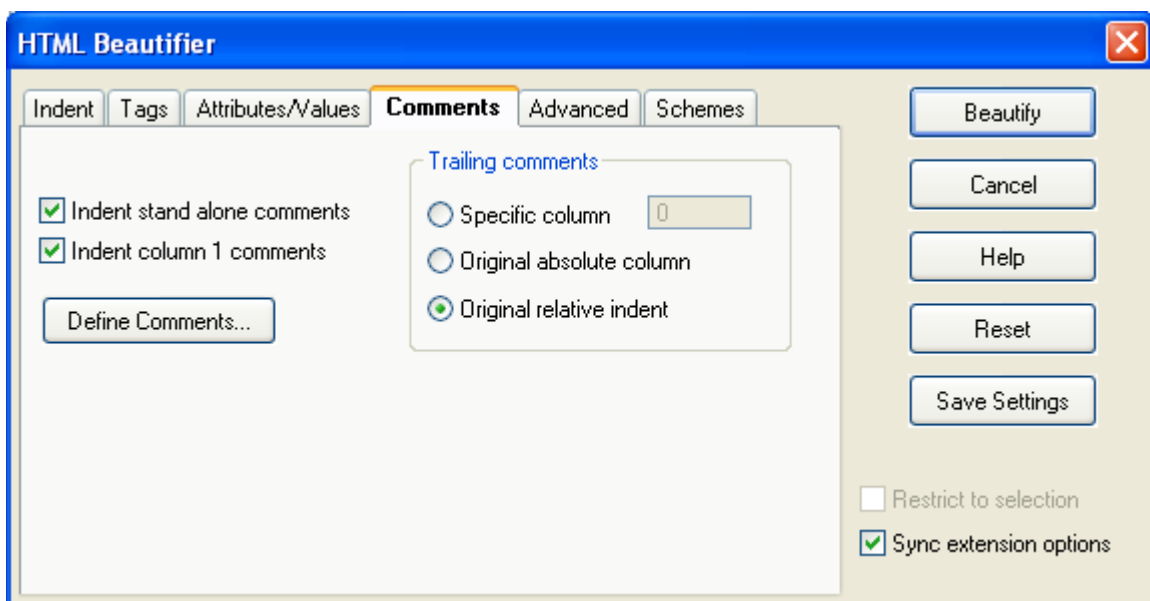
The Attributes/Values tab contains the following settings:

- **Attribute case** - Specifies how you want attributes cased inside the body of a tag. For example, if you choose UPPER, then <td align="right"> would be beautified to <td ALIGN="right">.

- **Word value case** - Specifies how you want word values cased after the = of an attribute inside the body of a tag. For example, if you choose UPPER, then `<td align="right">` would be beautified to `<td align=RIGHT>`.
- **Hex value case** - Specifies how you want hex values cased after the = of an attribute inside the body of a tag. For example, if you choose UPPER, then `<body bgcolor="#ffffff">` would be beautified to `<body bgcolor="#FFFFFF">`.
- **Quote word values** - Specifies whether you want word values enclosed in double quotes after the = of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`. Select **Preserve** if you want word values left alone.
- **Quote number values** - Specifies whether you want number values enclosed in double quotes after the = of an attribute inside the body of a tag. For example, `<td width=590>` would be beautified to `<td width="590">`. Select **Preserve** if you want number values left alone.
- **Quote all values** - When on, all values will be quoted after the = of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`.

## Comments Tab

The Comments tab of the HTML Beautifier is pictured below.



The Comments tab contains the following options and settings:

- **Indent stand alone comments** - When on, indicates whether comments which appear on lines by themselves with no content to the left are indented to the current content indent level. The following is an example of a stand-alone comment:

```
<!-- stand alone
      comment
-->
```

- **Indent column 1 comments** - Normally comments that start in column 1 are left alone. Turn this on if you want the indent for these comments to be adjusted as well.

- **Define Comments** - Displays the HTML Comments dialog. This dialog allows you to define what the beautifier recognizes as a comment. The sequence `<!-- -->` is defined as the HTML comment by default. If you delete all comment definitions then all comments will be parsed as content and possibly word-wrapped.
- **Trailing comments** - Specify how trailing comments are treated from the following options:
  - **Specific column** - This text box specifies the column that “trailing comments” should be placed at. By trailing comments, we mean comments which appear at the end of lines which contain tags. An example of a trailing comment is:

```
<TD> <!-- trailing comment -->
```

- **Original absolute column** - When on, “trailing comments” are placed at the same column as the original source file. By trailing comments, we mean comments which appear at the end of lines which contain tags.
- **Original relative column** - When on, “trailing comments” are indented by reusing the indent after the last character of the end of the statement or declaration of the original source file. By trailing comments, we mean comments which appear at the end of lines which contain tags.

The following is an example of code before beautifying trailing comments:

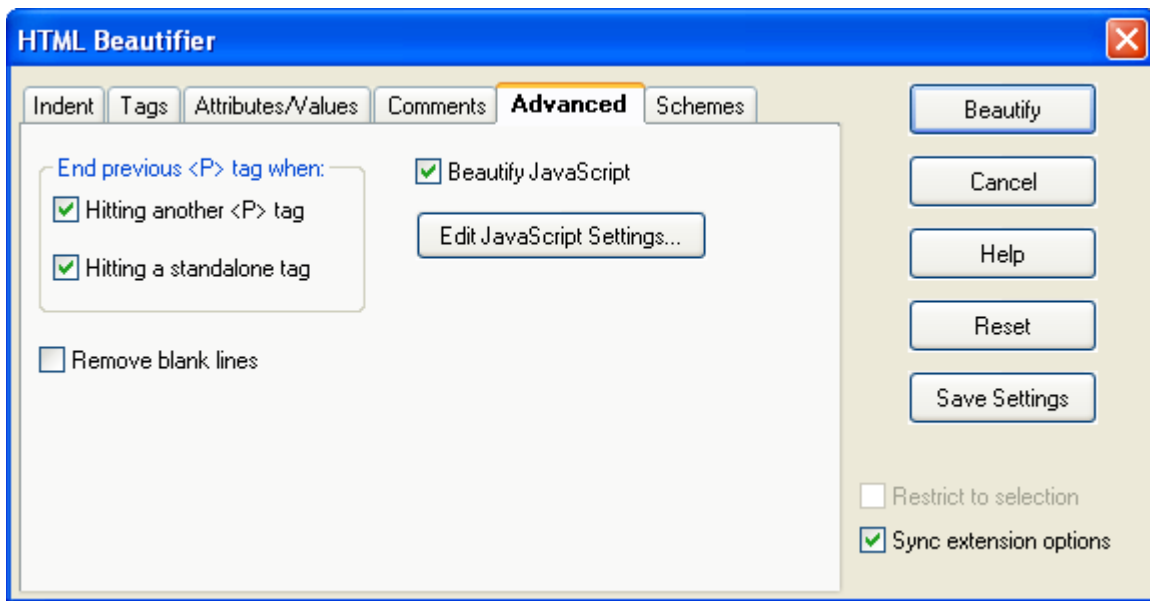
```
<TR>
<TD><four characters><!-- trailing comment -->
</TD>
</TR>
```

The resulting code would be:

```
<TR>
  <TD><four characters><!-- trailing comment -->
  </TD>
</TR>
```

## Advanced Tab

The Advanced tab of the HTML Beautifier is pictured below.



The following options are available on the Advanced tab:

- **End previous <P> tag when** - Select from the following options:
  - **Hitting another <P> tag** - When on, the beautifier will interpret a <P> tag as a signal of the end of any previous paragraphs.
  - **Hitting a standalone tag** - When on, the beautifier will interpret a standalone tag as a signal of the end of any previous paragraphs. For example, in the following content, the <TABLE> tag (assuming that the <TABLE> tag is a standalone tag) signals the end of the previous <P> paragraph. This has the benefit of cleaning up unwanted (and unexpected) indenting. For example:

```
<P>
This is a paragraph of content.  The paragraph will be ended by the
start of the table below it.

<TABLE>
  <TR>
    <TD>a table cell</TD>
  </TR>
</TABLE>
```

- **Remove blank lines** - When on, blank lines are deleted.
- **Beautify JavaScript** - When on, embedded JavaScript is beautified according to your JavaScript beautifier settings.
- **Edit JavaScript Settings** - Click on this button to configure the JavaScript Beautifier settings. The JavaScript Beautifier is the same as the C/C++ Beautifier - see [C/C++ Beautifier](#) for more information.

## Schemes Tab

To define a new scheme, set the various beautify options and press the **Save Scheme** button. User defined schemes are stored in `uformat.ini`.







## XML/HTML Formatting

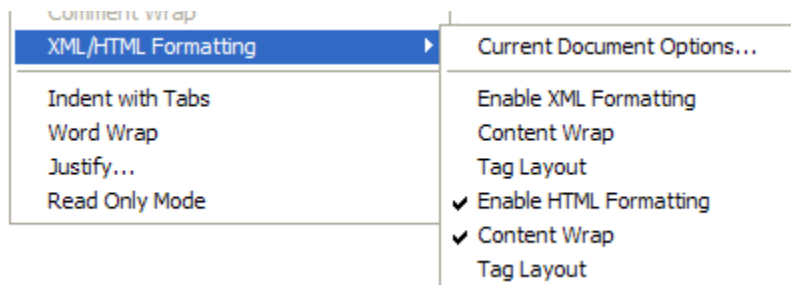
Content in XML and HTML files may be set to automatically wrap and format as you edit. XML/HTML Formatting is essentially comprised of two features: **Content Wrap**, which wraps the content between tags, and **Tag Layout**, which formats tags according to a specified layout. Both can be enabled individually for all XML and HTML files that are opened in SlickEdit®, or just for the current document.

*Formatting schemes* form the basis for how tags and content are formatted. A formatting scheme contains any number of XML or HTML tags, each of which can be configured individually for indent levels, wrapping, and tag structure. Multiple schemes can be defined—for example, you may want one scheme for HTML files and another for XML files, or perhaps you are required to code certain files to various standards. Schemes can be saved and imported, so they can be shared with your team. Tags for each scheme can be entered manually or you can import tags from the current file.

**CAUTION** XML/HTML Formatting does not currently affect XML or HTML Beautifier settings. If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

## Enabling/Disabling XML/HTML Formatting

XML/HTML Formatting is enabled by default for XML and HTML files that you open in SlickEdit®. You can enable and/or disable Tag Layout and/or Content Wrap for either file type on a global basis or on a per document basis. Options to turn these features on/off are located on the **Document > XML/HTML Formatting** menu.



## Enabling/Disabling Globally

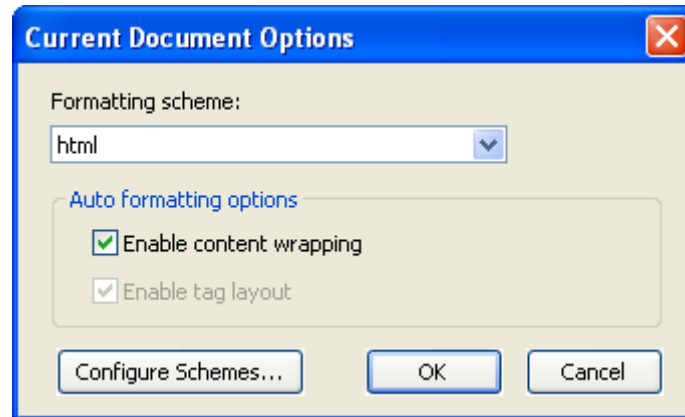
Automatic formatting can be enabled or disabled for every XML and/or HTML file that is created or opened in SlickEdit®. These XML- and HTML-specific global options are toggled on/off by placing a check next to **Enable XML Formatting** and/or **Enable HTML Formatting**. To toggle **Tag Layout** and/or **Content Wrap** on or off for either file type, check or uncheck those items. For example, if you want both Tag Layout and Content Wrap enabled for XML files, but you only want Content Wrap enabled for HTML files, place a check next to **Enable XML Formatting** and its sub-menu items **Tag Layout** and **Content Wrap**, then place a check next to **Enable HTML Formatting** and its sub-menu item **Content Wrap**.

The `xml_formatting_toggle` and `html_formatting_toggle` commands can also be used to toggle all respective formatting features on/off. For use in macros, two more commands are available: `xml_formatting` and `html_formatting`. When you use any of these commands, the command line prompts for yes (Y) or no (N). When these commands are used to enable/disable XML/HTML Formatting, both Tag Layout and Content Wrap are enabled/disabled for both XML and HTML files on a global basis.

## Enabling/Disabling for the Current Document

Current document options are available so that you can turn off aspects of global formatting for just the current document. For example, if you have both aspects of HTML formatting enabled globally, but you need to edit an old HTML file and do not want automatic tag layout to occur, you can disable HTML Tag Layout for that specific file. The current document settings are remembered each time you open that file.

To change XML or HTML formatting for just the current document, select **Document > XML/HTML Formatting > Current Document Options**, or use the `xml_html_document_options` command. The Current Document Options dialog is displayed.



Select the **Formatting scheme** that you want applied, then check or uncheck the **Auto formatting options** that you want enabled or disabled. Click **Configure Schemes** if you want to modify or create a new scheme to apply to the current document.

Global formatting must be enabled for the current file type in order for these options to be available. For example, see the two screen shots shown previously. The menu screen shot shows that global HTML formatting is enabled for Content Wrap only. This means that tag content for every HTML file that you create or open in SlickEdit® will be wrapped, but no Tag Layout settings will be applied. The second screen shot (the Current Document Options dialog) reflects that the global setting for Tag Layout is disabled. Therefore it cannot be enabled for the current document. To enable it for the current document, you would first need to enable the global setting by placing a check next to **Document > XML/HTML Formatting > Enable HTML Formatting > Tag Layout**. Then you can use the Current Document Options dialog to enable it for the current document.

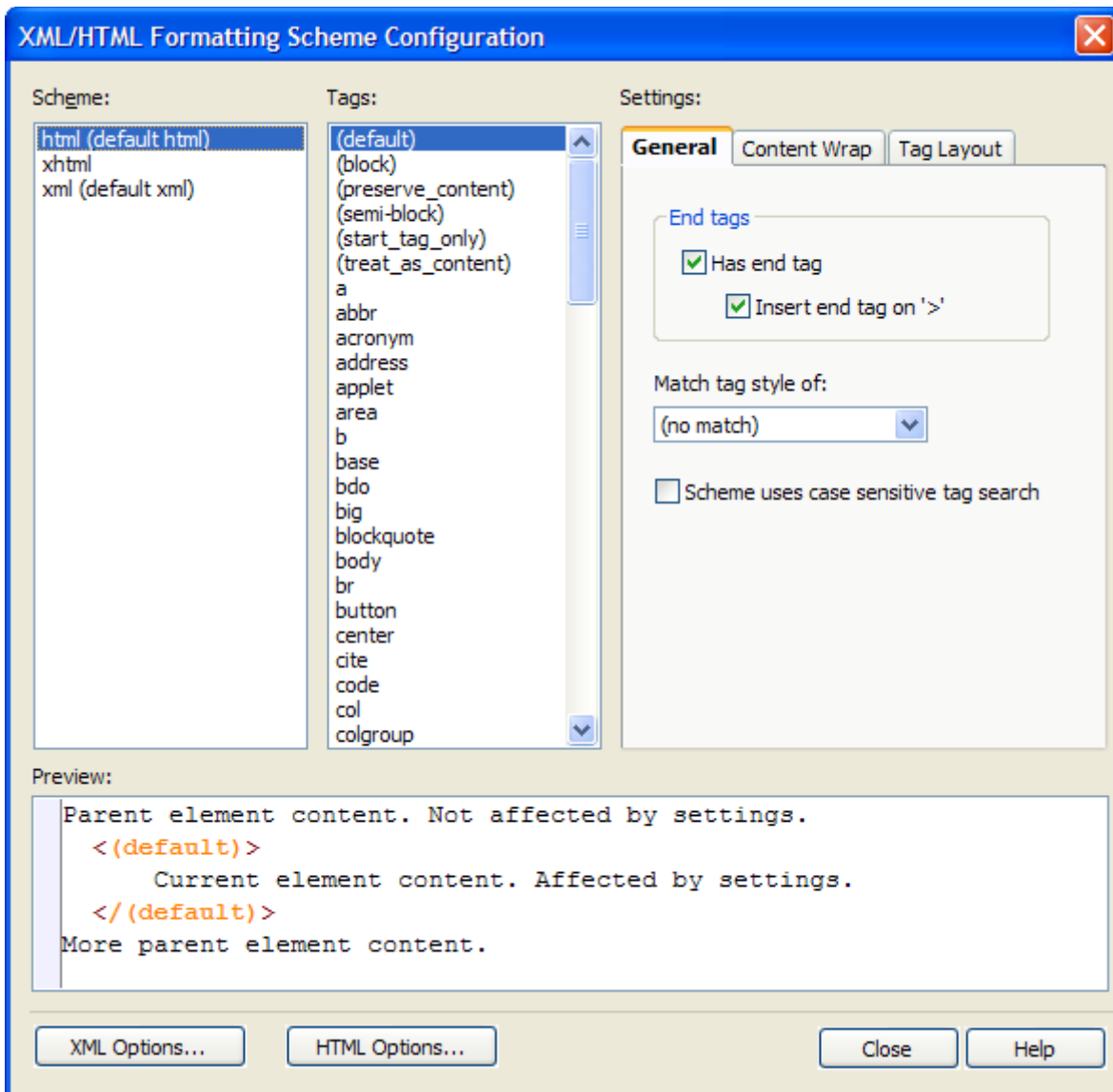
## Working with Schemes

A formatting scheme is comprised of any number of individually-configurable XML or HTML tags and controls the formatting of your text when XML/HTML Formatting is enabled. You can create different schemes for use with either XML or HTML, and/or different schemes for use with different individual files. For example, you may want one scheme for HTML and a different scheme for XML. Or, you may want one scheme for creating new files and another for editing existing files.

Schemes are stored as XML files in the format `SchemeName.xml`, and are located in the `formatschemes/xwschemes` subdirectory of your user configuration directory. Scheme files can be shared or checked into version control to ensure consistency in the formatting of your team's XML/HTML files.

Use the XML/HTML Formatting Scheme Configuration dialog to work with schemes. You can access the dialog from **Tools > Options > XML/HTML Formatting**, or by using the `xml_html_options` command.

Available schemes are listed in the **Schemes** column. The tags that make up each scheme are listed in the **Tags** column.



## Default Schemes

XML/HTML Formatting comes with two default schemes. Each time that you open an HTML file for editing, by default, the **html (default html)** scheme is used. Each time an XML file is opened, the **xml (default xml)** scheme is used. You can configure the settings for each default scheme or specify your own default schemes (see [Specifying a Different Default Scheme](#)).

The **html (default html)** scheme is comprised of a **(default)** tag as well as a list of commonly used HTML tags, that are preconfigured with standard settings. The **xml (default xml)** scheme is comprised of one **(default)** tag preconfigured with standard settings.

For any tag that does not appear in the Tags list, the **(default)** tag settings are used.

## Specifying the Scheme to Use

You can specify a scheme to use for just the current document, or a default scheme to use for all HTML and/or XML files that are created or opened in SlickEdit®.

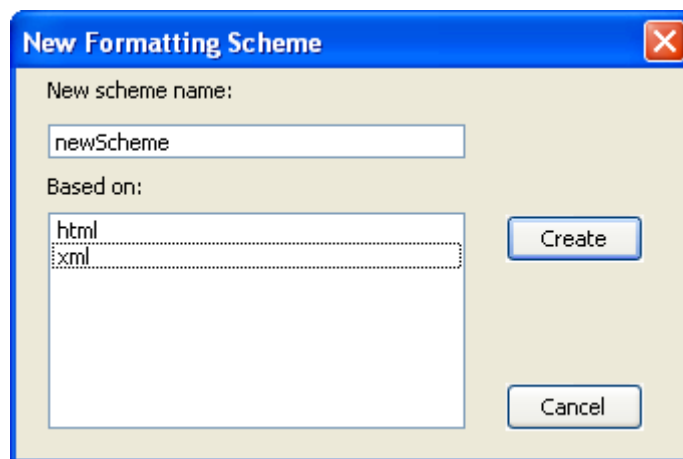
To specify the scheme for the current document, select **Document > XML/HTML Formatting > Current Document Options**, and pick the scheme to use from the drop-down list. The scheme you choose is remembered the next time that the document is opened.

## Specifying a Different Default Scheme

SlickEdit® has two predefined default schemes (see [Default Schemes](#)). You can specify your own default scheme for new XML or HTML files by selecting a scheme, then from the right-click context menu, choose **Set as Default XML Scheme** or **Set as Default HTML Scheme**. The name in the **Scheme** list will be appended with the text “(default xml)” or “(default html)”. For example, if you have an HTML scheme named **readmes** and set it as the default, the name in the Schemes list will change to **readmes (default html)**.

## Creating Schemes

To create your own scheme, right-click in the **Scheme** column of the XML/HTML Formatting dialog, and select **New Scheme**.



Type a name for your scheme, then select an existing scheme on which it should be based. This will “copy” all of the tags and settings from the selected existing scheme to your new scheme. Click **Create** when finished.

## Saving and Deleting Schemes

In the XML/HTML Formatting Scheme Configuration dialog, modified schemes are denoted by asterisks around the name (for example, **\*html\***). To save a modified scheme, right-click on it and select **Save Scheme Changes**. Alternatively, when you close the XML/HTML Formatting dialog, you are prompted whether to save modified schemes.

To delete a selected scheme, right-click and choose **Delete Scheme**.

## Working with Tags

As described previously, formatting schemes are comprised of individual tags with associated formatting settings. For example, in HTML, you may want start and end `<div>` tags to be on separate lines above/below the text, while start/end style tags (such as `<b>` and `<i>`) are formatted inline with the text.

In order to configure formatting for individual tags, you must first define a scheme (see [Creating Schemes](#)), or you can use one of the default schemes (see [Default Schemes](#)). The Tags column of the XML/HTML Formatting dialog shows a list of tags associated with the selected scheme.

### Default Tags

For all schemes, a **(default)** tag is included. It defines the settings that are applied to tags that are not specifically listed in the Tags list. It can also be used as a basis for other tags when you use the **Match tag style of** option on the General tab.

The default settings for the **(default)** tag are based on a block-style tag, in that the start and end tags are on separate lines, content and nested tags are indented according to your extension indent style, and content is wrapped to a fixed right margin at column 80. Use the Preview area at the bottom of the dialog to see how tags based on the **(default)** tag will be formatted in your code.

### Base Tags

Base tags are used to define groups of tags with similar behaviors. By creating a base tag and associating a set of actual tags with that tag, you can configure that set of tags by changing the settings for the base tag. The sample HTML schemes included with SlickEdit® include two base tags: (block) and (semi-block). When creating your own base tags, be sure to use unique tag names that are not used in formatting to make it easy to spot them.

The (block) tag is useful as a base tag for tags like `<div>`, where you want the start and end tags on separate lines from the content and aligned vertically. By default, this style indents nested tags and content according to your extension indent style, and content is wrapped to a fixed right margin at column 80. Because this is a common style in HTML, the (block) tag has the same settings as the (default) tag.

The (semi-block) tag is useful as a base tag for tags such as `<h1>`, where you want the start and end tags on the same line as the content. The default settings for (semi-block) are the same as for (block), except for the start and end tags.

Use the Preview area at the bottom of the dialog to see how tags based on these base tags will be formatted in your code. The following are examples:

```
<div>
    Sample text of a tag set to (block) style.
</div>

<h1>Sample text of a tag set to (semi-block) style. Notice how wrapping
occurs within this tag.</h1>
```

## Adding and Deleting Tags

To add individual tags to a selected scheme, right-click in the Tags column and select **New Tag**. Type the name of the tag without angle brackets or attributes. Click **OK** and the new tag is now listed in the Tags column.

**NOTE** Tag names cannot have spaces. If you create a new tag with spaces, SlickEdit converts the spaces to underscores in the dialog.

To add all of the tags from the current file to a selected scheme, right-click in the Tags column and select **Add Tags from Current File**.

To remove a tag from a selected scheme, right-click on the tag and select **Delete Tag**. Deleting SlickEdit® (default) tags, or any tag that is based on another, is not recommended. If you attempt to do this, you will be prompted whether to continue.

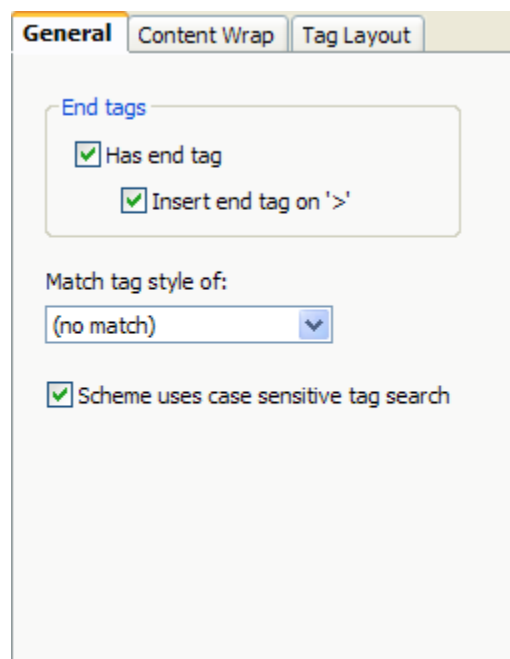
## Formatting Settings

The tabs on the XML/HTML Formatting dialog contain settings that control how your text is formatted when XML/HTML Formatting is enabled. The following sections describe the tabs and how each setting works. Before configuring settings, be sure the scheme and tag(s) that you want to affect are selected.

**CAUTION** XML/HTML Formatting does not currently affect XML or HTML Beautifier settings. If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

## General Settings

The General tab of the XML/HTML Formatting dialog is shown below.

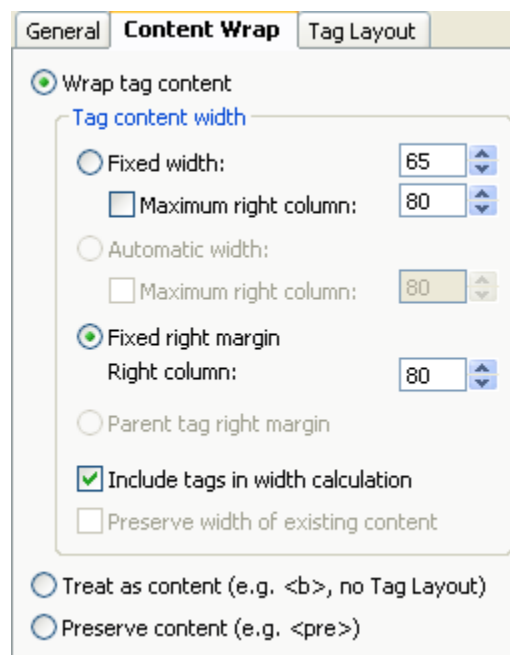


It contains the following general settings:

- **End tags settings** - These options control how the end tags for the selected tag are formatted:
  - **Has end tag** - This option tells SlickEdit® whether or not the specified tag is intended to be closed with an end tag. This information is used for SlickEdit to know when to start or stop calculating information based on the tag. For example, the <div> tag in HTML has a start and end tag, while the <br> tag does not have an end tag.
  - **Insert end tags on '>'** - When selected, SlickEdit automatically inserts the end tag after you type the closing brace (>) of the start tag. For example, when you type <div>, </div> is automatically inserted. The placement of the inserted tag depends on other settings you have specified, such as whether or not the end tag should be on a separate line (specified on the Tag Layout tab).
- **Match tag style of** - Select this option if you want the selected tag's settings to match the style of another tag, then pick the tag to use from the drop-down list. This can be a time-saver when adding a batch of new tags that should all have the same settings.
- **Scheme uses case-sensitive tag search** - When this option is selected, tags in the Tags column are displayed in the list exactly as you have typed them, with the case preserved. When you type the tags in the editor, the case must match exactly or the tag will not be recognized (and the (default) tag settings will be applied). This option is appropriate for XML. HTML is not case-sensitive.

## Content Wrap Settings

The Content Wrap tab of the XML/HTML Formatting dialog contains options for specifying how wrapping should occur for the selected tag's content.



There are three main (mutually-exclusive) options:

- **Wrap tag content** - When selected, content between tags is wrapped according to the settings specified in the **Tag content width** group box. See [Tag Content Width Settings](#) below for details.

- **Treat as content** - When selected, the tag as well as its content is wrapped according to the parent tag content. The tag is treated as “inline” with the surrounding text. This could be useful for style tags such as `<b>` or `<i>` that you want to appear “inline” with the content. For example:

```
<div>
  This is a sample paragraph with the <b>bold tags</b> being treated as
  content, “inline” with the rest of the text.
</div>
```

- **Preserve content** - When selected, content between start and end tags is not wrapped, but the tags are laid out properly with the parent tag. This could be useful for tags such as `<pre>`, where the content needs to be rendered exactly as it appears in the code.

#### NOTES

- When **Treat as content** is selected, the wrapping options in the **Tag content width** group box, as well as options on the Tag Layout tab, are unavailable.
- When **Preserve content** is selected, the wrapping options in the **Tag content width** group box are unavailable.

### Tag Content Width Settings

When **Wrap tag content** is selected, content between tags is wrapped according to the settings specified in this group box. The options **Fixed width**, **Automatic width**, and **Fixed right margin** are mutually exclusive.

- **Fixed width** - When selected, tag content is formatted to the specified width. The original left margin of the content is maintained, and the right margin is adjusted to meet the target width.  
If **Maximum right column** is used, lines will be wrapped when they reach the specified column, even if they have not reached the specified fixed width. This is useful if coding standards mandate that text should not exceed a specified column.
- **Automatic width** - When selected, the width of the longest multi-line paragraph in the tag's content is used as the width. This is useful for preserving the formatting of existing content.  
If **Maximum right column** is used, lines will be wrapped when they reach the specified column, even if they have not reached the specified automatic width. This is useful if coding standards mandate that text should not exceed a specified column.
- **Fixed right margin** - When selected, lines will break before the specified number of columns in the **Right column** field has been reached.

**TIP** If coding standards mandate that text should not exceed a specified column, you can still use Fixed or Automatic width settings. Select and set the **Maximum right column**, and lines will be wrapped when they reach the specified column, even if they have not reached the specified fixed or automatic width.

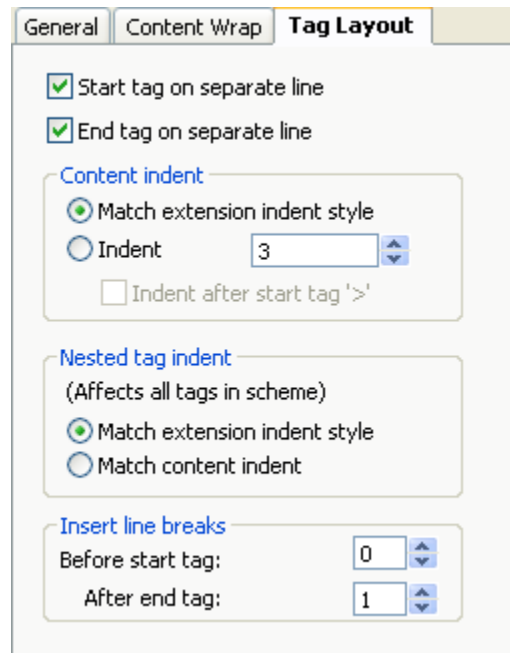
- **Parent tag right margin** - When selected, tag content is wrapped at the right margin to the width of the parent tag.
- **Include tags in width calculation** - When selected, the start and end tag characters are counted in addition to the content. The number of characters and spaces (including attributes) within tags are calculated, and the specified width is adjusted accordingly. This is useful for producing uniform blocks of text.



- **Preserve width of existing content** - When selected, SlickEdit® preserves the width of the existing content while editing. The width is determined by the length of the longest multi-line paragraph. If the width of the existing content cannot be determined, the formatting option specified (**Fixed**, **Automatic**, or **Fixed right**) will be used instead.

## Tag Layout Settings

The Tag Layout tab of the XML/HTML Formatting dialog contains options to control the location of the start tag, end tag, and the content between them.



- **Start tag on separate line** - When selected, the start tag occurs on a separate line, and the cursor will be placed on the line below for you to type the content. Note the cursor location (|) in the examples below.
- **End tag on separate line** - When selected, the end tag placed on a separate line below the content.

Example of both settings checked:

```
<div>
|
</div>
```

Example of both settings unchecked:

```
<div>|</div>
```

Example of start checked and end unchecked:

```
<div>
|</div>
```

**NOTE** [Dynamic Surround](#) is triggered when you type a tag that has both options **Start** and **End tag on separate line** checked. Note that Dynamic Surround cannot wrap content and only indents to match the indent style you have specified on the [Indent Tab](#) of the Extension Options dialog (**Tools > Options > File Extension Setup**). See [Syntax Indent](#) for more information.

- **Content indent group settings** - These mutually-exclusive options control how content between tags is indented:
  - **Match extension indent style** - When selected, tag content is indented according to the settings on the [Indent Tab](#) of the Extension Options dialog (**Tools > Options > File Extension Setup**). See [Syntax Indent](#) for more information on setting extension-specific indent styles.
  - **Indent** - When selected, indenting for the tag occurs at the column number specified in the spin box. When **Indent after start tag '>'** is selected, indenting is relative to the close bracket. Otherwise indenting is relative to the open bracket.
- **Nested tag indent settings** - Indenting is activated after the end tag is typed (or automatically inserted if **Insert end tags on '>'** is checked on the General tab). These mutually-exclusive settings apply to all tags in the selected scheme:
  - **Match extension indent style** - When selected, SlickEdit® indents the selected tag according to the indent style you have specified on the [Indent Tab](#) of the Extension Options dialog (**Tools > Options > File Extension Setup**). See [Syntax Indent](#) for more information on setting extension-specific indent styles.
  - **Match content indent** - When selected, SlickEdit indents the selected tag to the same level as its parent content.
- **Insert line breaks settings** - The values for **Before open tag** and **After close tag** specify the number of line breaks that are inserted before and after begin and end tags. Note that in order to insert one or more blank lines, the values should be set to 2 or higher.

## More Settings

The XML/HTML Formatting dialog contains two buttons along the bottom that allow you to configure even more settings for these languages. These buttons are shortcuts to the extension options that are usually accessed through the Options button on the Extension Options dialog. See [XML Formatting Options](#) and [HTML Formatting Options](#) for more information on these dialogs.

## Ada

---

This section describes some of the features and options that are available for Ada, including extension-specific options and the Ada Beautifier.

### Ada Formatting Options

Keyword casing options are available for Ada language file extensions. To access these options, from the main menu, choose **Tools > Options > File Extension Setup**. Choose the language extension you wish to work with from the **Extension** drop-down list, then click the **Options** button. The Formatting Options dialog specific to the file extension you have selected will be displayed.

**NOTE** Languages similar to Ada have similar Formatting Options dialogs which are not specifically documented.

**Keyword case** specifies the case of keywords used by template editing. If **Auto case keywords** is selected, the case of keywords are changed to the keyword case specified when you type them. For example, when you type the word “procedure” and the **Keyword case** is set to **Upper case**, the editor changes “procedure” to “PROCEDURE”.

### Ada Beautifier

You can beautify Ada files and change the beautify settings by using the Ada Beautifier dialog box. This dialog box can be accessed from the main menu by choosing **Tools > Beautify**, or by using the **gui\_beautify** command.

To instantly beautify Ada code according to the settings that are selected on the Ada Beautifier dialog box, use the **ada\_beautify** or **ada\_beautify\_selection** commands.

The following settings and operations are available on the Ada Beautifier:

- **Restrict to selection** - When checked, only lines in the selection are beautified.
- **Sync extension options** - When checked, the extension options are updated to reflect any changes that these dialogs have in common.
- **Beautify** - Beautifies current selection or buffer and closes the dialog box.
- **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.
- **Save Settings** - Saves beautify options in the `uformat.ini` file. These settings are used by the **ada\_beautify** command.

The tabs on the Ada Beautifier are described in the sections below.

### Indent Tab

The following settings are available:

- **Indent with tabs** - When checked, tab characters are used for leading indent of lines. This value defaults to the **Tabs** text box on the [Indent Tab](#) of the Extension Options dialog box (**Tools > Options > File Extension Setup**).
- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level. The words “Syntax indent” are in parenthesis to help indicate that this field has the same meaning as

the **Syntax indent** text box on the [Indent Tab](#) of the Extension Options dialog box (**Tools > Options > File Extension Setup**). By default, this text box is initialized with the current extension setup setting.

- **Tab size** - Specifies output tab size. The output tab size is only used if the **Indent with tabs** check box is selected on the [Indent Tab](#) of the Extension Options dialog box (**Tools > Options > File Extension Setup**). This value defaults to the **Syntax indent** text box on the [Indent Tab](#) of the Extension Options dialog box.
- **Original tab size** - Specifies what the original file's tab expansion size was. It is necessary to know the tab expansion size of your original file to handle reusing indent amounts from your original file. Currently the beautifier only reuses the original source file's indenting for comments. This option has no effect if the original file has no tab characters.
- **Continued Lines**
  - **Max line length** - Specifies the maximum length a statement line can be before it is wrapped to a new line. Set this value to 0 to preserve line breaks.
  - **Continuation indent** - Specifies how much to indent lines of statements which continue to the next line. This has no effect on assignment statements or parenthesized expressions. Lines which are a continuation of an assignment statement are indented after the first equal sign (=). Lines which are a continuation of a parenthesized expression are indented after the open paren.
- **Operator position** - Specify where the operator should be positioned when breaking a statement across multiple lines. For example, given the statement:

```
Seconds := Days * Hours_Per_Day * Minutes_Per_Hour * Seconds_Per_Minute ;
```

An operator position setting of "End of same line" would result in:

```
Seconds := Days *
Hours_Per_Day *
Minutes_Per_Hour *
Seconds_Per_Minute ;
```

An operator position setting of "Beginning of next line" would result in:

```
Seconds := Days
* Hours_Per_Day
* Minutes_Per_Hour
* Seconds_Per_Minute ;
```

## Statements/Declarations Tab

The following options are available on the Statements/Declarations tab:

- **Reserved word case** - Specifies the case for reserved words. For example, if you choose **UPPER**, then the Ada reserved word "procedure" would be beautified to "PROCEDURE".
- **One statement per line** - When checked, only one statement is allowed per line of code.
- **One declaration per line** - When checked, only one declaration is allowed per line of code.
- **One parameter per line** - When checked, only one parameter is allowed per line of code in a formal parameter list of a subprogram specification.
- **One enumeration per line** - When checked, only one enumeration is allowed per line of code in an enumerated type definition.

## Horizontal Spacing Tab

This tab allows you to specify how certain operators and separators are padded. The following options are available:

- **Item** - Syntactic item to which padding settings get applied.

**NOTE** The “Binary operators” item includes: + - \* / \*\* := = /= => <= >= < >

- **Padding Before** - When checked, one space is placed before the item.
- **Padding After** - When checked, one space is placed after the item.
- **Padding Preserve** - When checked, the original padding (or lack of padding) around the item is preserved.

## Vertical Alignment Tab

The following options are available on the Vertical Alignment tab:

- **Align on declaration colon** - When checked, adjacent declaration lines (including parameter specifications) have their colons vertically aligned. For example, before beautify:

```
procedure foo ( A_Var : Boolean ;
               Another_Var : Boolean ) ;
```

After beautify:

```
procedure foo ( A_Var      : Boolean ;
               Another_Var : Boolean ) ;
```

- **Align on declaration in-out** - When checked, the modes of parameter specifications in the formal part of a subprogram declaration are vertically aligned. For example, before beautify:

```
procedure foo ( A_Var : in Boolean ;
               Another_Var : in out Boolean );
```

After beautify:

```
procedure foo ( A_Var      : in      Boolean ;
               Another_Var : in out Boolean ) ;
```

## Blank Lines Tab

The following options are available on the Blank Lines tab:

- **Item** - Syntactic item to which blank lines settings get applied.
  - **Subprogram declaration** - Procedure or Function declaration.
  - **Subprogram body** - Procedure or Function body.
  - **Type declaration** - Any declaration that begins with the reserved word “TYPE”.
  - **for...use** - Aspect clause. For example:

```
for Medium'Size use 2*Byte;
```

- **Subunit comment header** - The comment block that appears just before a subunit (e.g. Procedure body, etc.).
- **begin/end** - Any line that starts with the reserved words “begin” or “end.”

- **if/elsif/else** - The **if**, **elsif**, and **else** parts of an **if** statement.
- **return** - Any line that starts with the reserved word "return."
- **Loops** - Loop statements (e.g. **loop**, **while**, **for**).
- **Nested paren list item** - A parenthesized item that is itself enclosed in a larger parenthesized list. For example, before beautify:

```
Default_Data : constant Data_Type :=
  ( A_Set => ( others => ( Item1 => false ,
                        Item2 => false ,
                        Item3 => false ) ) , -- Paren'd item enclosed in larger
paren'd list
    B_Set => ( others => ( Item1 => false ,
                        Item2 => false ,
                        Item3 => false ) ) ) ;
```

After beautify:

```
Default_Data : constant Data_Type :=
  ( A_Set => ( others => ( Item1 => false ,
                        Item2 => false ,
                        Item3 => false ) ) , -- Paren'd item enclosed
in larger paren'd list
    B_Set => ( others => ( Item1 => false ,
                        Item2 => false ,
                        Item3 => false ) ) ) ;
```

- **Before** - Specify how many blank lines are inserted before item.
- **After** - Specify how many blank lines are inserted after item.
- **Between** - Specify how many blank lines are inserted between like items.

## Comments Tab

The following options are available on the Comments tab:

- **Comment lines immediately below a type declaration indented by** - The amount to indent a comment appearing immediately below a TYPE declaration.
- **Trailing comments** - Trailing comments appear at the end of lines which contain statements or declarations. For example:

```
A := B + C ; -- This is a trailing comment
-- This is not a trailing comment
procedure foo ( A_Var : Boolean ) ;
```

- **Specific column** - When selected, trailing comments are placed at the specified column.
- **Indent by** - When selected, trailing comments are indented by the specified number of columns after the last character of the end of the statement or declaration.
- **Original relative indent** - When selected, trailing comments are indented by reusing the indent after the last character of the end of the statement or declaration of the original source file.
- **Force type declaration comments to next line** - When selected, trailing comments appearing at the end of a TYPE declaration line are forced onto the next line.

## Advanced Tab

The following options are available on the Advanced tab:

- **if-then-else continued lines** - Use these advanced options to customize how multi-line conditional expressions of an **if-then-else** statement are indented.
  - **Force a linebreak on logical operators** - A line break is forced before/after (depending on your Operator position setting) every logical operator in the condition of an **if/elseif**. For example, before beautify:

```
-- Indent per level = 3
-- Operator position = Beginning of next line

if A = B and C = D then
    null ;
end if ;
```

After beautify:

```
if A = B
    and C = D then
    null ;
end if ;
```

- **Additional indent for logical operator** - Additional indent amount for a line broken on a logical operator. This amount is in addition to the current indent level. For example, before beautify (Indent per level = 3; Operator position = Beginning of next line; Additional indent for logical operator = 3):

```
-- Indent per level = 3
-- Operator position = Beginning of next line
-- Additional indent for logical operator = 3

if A = B and C = D then
    null ;
end if ;
```

After beautify:

```
if A = B
    and C = D then
    null ;
end if ;
```

- **Additional indent for logical operator when followed by another line that begins with logical operator** - Additional indent amount for a line broken on a logical operator that is followed by another line that also is broken on a logical operator that is different. This amount is in addition to the current indent level, and in addition to the **Additional indent for logical operator** setting.

For example, before beautify (Indent per level = 3; Additional indent for logical operator = 3; Additional indent for logical operator when followed by another line that begins with different logical operator = 3):

```
-- Indent per level = 3
-- Operator position = Beginning of next line
-- Additional indent for logical operator = 3
-- Additional indent for logical operator when
--   followed by another line that begins with different logical operator
= 3

if A = B and then C = D or else E = F then
    null ;
end if ;
```

After beautify:

```
if A = B
    and then C = D
    or else E = F then
    null ;
end if ;
```

## Schemes Tab

To define a new scheme, set the various beautify options then click the **Save Scheme** button. User-defined schemes are stored in `uformat.ini`.



# COBOL

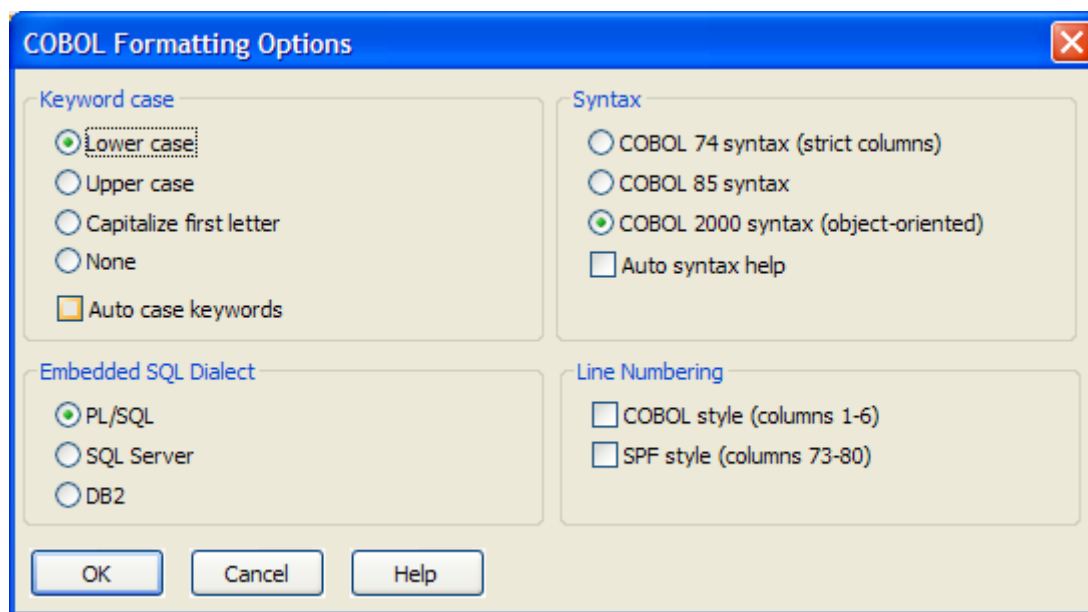
This section describes some of the advanced options that are available for COBOL.

## COBOL Formatting Options

Options are available for the COBOL language file extension, for changing smart indenting and styles for template editing. To access these options, from the main menu, choose **Tools > Options > File Extension Setup**. Choose the language extension you wish to work with from the **Extension** drop-down list, then click the **Options** button. The Formatting Options dialog specific to the file extension you have selected will be displayed.

**NOTE** Languages similar to COBOL have similar Formatting Options dialogs which are not specifically documented.

The COBOL Formatting Options dialog is pictured below.



The following options are available:

- **Keyword case** - Specifies the case of keywords used by template editing. If **Auto case keywords** is selected, the case of keywords are changed to the keyword case specified when you type them. For example, when you type the word “procedure” and the **Keyword case** is set to **Upper case**, the editor changes “procedure” to “PROCEDURE”.
- **Syntax** - Select the type of syntax to use. COBOL 74 and COBOL 2000 syntax are mutually exclusive options.
- **Embedded SQL Dialect** - Specifies the specific type of SQL that is embedded in your COBOL source. This affects embedded SQL language color coding.
- **Line Numbering** - Choose the line numbering style from the following options:

- **COBOL style line numbering** - When selected, expect line numbers in columns one through six when renumbering lines.
- **SPF style line numbering** - When selected, expect line numbers in columns 73 through 80 when renumbering lines.

# Pascal

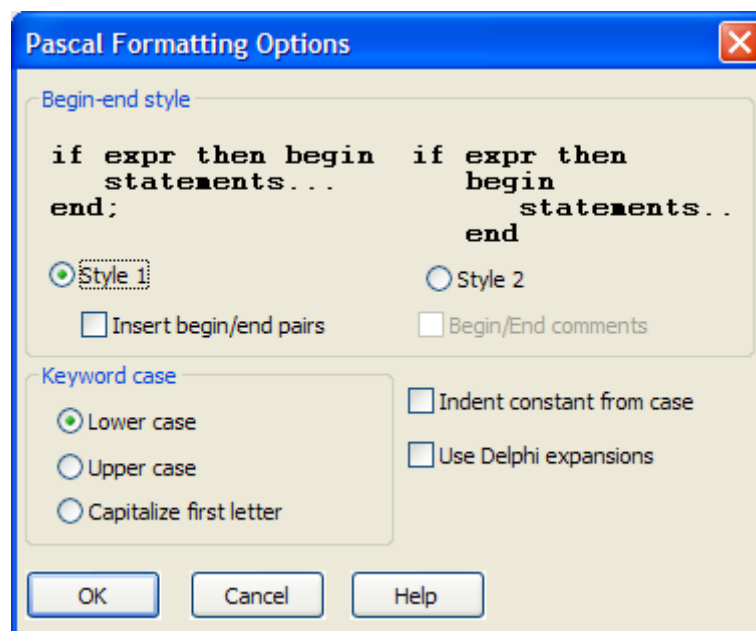
This section describes some of the advanced options that are available for Pascal.

## Pascal Formatting Options

Options are available for the Pascal language file extension, for changing smart indenting and styles for template editing. To access these options, from the main menu, choose **Tools > Options > File Extension Setup**. Choose the language extension you wish to work with from the **Extension** drop-down list, then click the **Options** button. The Formatting Options dialog specific to the file extension you have selected will be displayed.

**NOTE** Languages similar to Pascal have similar Formatting Options dialogs which are not specifically documented.

The Pascal Formatting Options dialog is pictured below.



The following options are available:

- **Begin-end style** - Specify the begin/end style used by template editing and smart indenting. For each style, select from the following options:
  - **Insert begin/end pairs** - Specifies whether template should be inserted with **begin** and **end**.
  - **Begin/End comments** - Specifies whether a comment is appended after the **end** keyword to indicate the type of loop or case it terminates. In addition the **begin** and **end** for procedures and functions are commented. No comment is appended to the **begin/end** pair of an **if** statement.
- **Keyword case** - Specifies the case of keywords used by template editing.

- **Indent constant from case** - Specifies whether constants of a case statement are indented or aligned to the case keyword.
- **Use Delphi expansions** - Specify whether Delphi®-style expansions should be used.

## PL/I

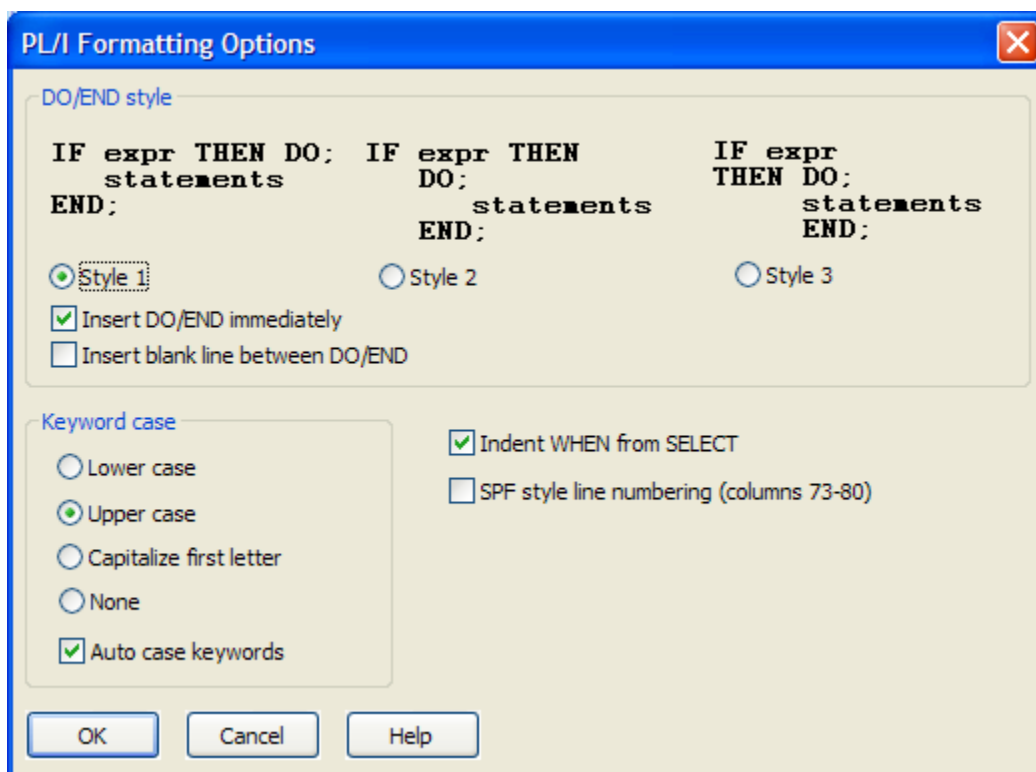
This section describes some of the advanced options that are available for the PL/I language.

## PL/I Formatting Options

Options are available for the PL/I language file extension, for changing smart indenting and styles for template editing. To access these options, from the main menu, choose **Tools > Options > File Extension Setup**. Choose the language extension you wish to work with from the **Extension** drop-down list, then click the **Options** button. The Formatting Options dialog specific to the file extension you have selected will be displayed.

**NOTE** Languages similar to PL/I have similar Formatting Options dialogs which are not specifically documented.

The PL/I Formatting Options dialog is pictured below.



The following options are available:

- **DO/END style** - Select the syntax expansion style that indicates whether syntax expansion should place the DO on a separate line, then select from the following options:
  - **Insert DO/END immediately** - Indicates whether syntax expansion should automatically add a DO/END block.

- **Insert blank line between DO/END** - Indicates whether syntax expansion should insert a blank line when a DO/END block is inserted.
- **Keyword case** - Specifies the case of keywords used by template editing. If **Auto case keywords** is selected, the case of keywords are changed to the keyword case specified when you type them. For example, when you type the word “procedure” and the **Keyword case** is set to **Upper case**, the editor changes “procedure” to “PROCEDURE”.
- **Indent WHEN from SELECT** - Indicates whether the WHEN clause inside a SELECT statement should be indented.
- **SPF style line numbering (columns 73-80)** - When selected, expect line numbers in columns 73 through 80 when renumbering lines.

# Python

---

This section describes some of the advanced features that are available for the Python language.

## Begin/End Structure Matching for Python

Begin/End Structure Matching moves the cursor from the beginning of a code structure to the end, or vice versa.

To place the cursor on the opposite end of the structure when the cursor is on a begin or end keyword pair, press Ctrl+] (**find\_matching\_paren** command or from the menu choose **Search > Go to Matching Parenthesis**). The **find\_matching\_paren** command supports matching parenthesis pairs {}, [] and ().

For Python, SlickEdit® supports the matching of the colon (:) token and the end of context.

Note the the cursor location in the code block below:

```
def function_foo(arg):| <- cursor
    ....
    return 0| <- destination
```

Executing **find\_matching\_paren** will move the cursor to the end of line containing the **return 0** statement. Executing it while the cursor is at the end of the **return 0** statement will bring the cursor back to the colon (:) position of the function signature line (**def function\_foo(arg):**).

This works on **function**, **class**, **for**, **while**, **if**, and **try** statements.

There is one limitation of this feature. Note the following code block:

```
for i in xrange(0, 10):| <- A
    for j in xrange(0, 10):| <- B
        for k in xrange(0, 10):| <- C
            print i, j, k| <- D
```

Invoking **find\_matching\_paren** at position A, B, or C will move the cursor to D, but doing so while the cursor is at D will only move the cursor back to C (not A nor B). This is because the Python language doesn't have the notion of end-of-scope token (such as } in C/C++, Java, etc.), so it's impossible to determine the correct destination when jumping from D. Therefore we pick the nearest possible destination in this scenario.

See [Begin/End Structure Matching](#) for more information about this feature.





# Tools and Utilities

This chapter contains the following topics:

- [Comparing and Merging](#)
- [Version Control](#)
- [Spell Checking](#)
- [FTP](#)
- [The Regex Evaluator](#)
- [Using the Calculator and Math Commands](#)
- [OS File Browser](#)



# Comparing and Merging

---

SlickEdit® provides two powerful ways to compare and merge files: [DIFFzilla®](#) and [3-Way Merge](#).

## DIFFzilla®

DIFFzilla provides powerful differencing capabilities that let you compare files or directories and view the differences side-by-side. You can make edits, merge changes, and save modified files easily within the results windows. As edits are made, the diff view is updated as you type, so you don't have to re-run the comparison. And, switching from a directory comparison to an individual file difference is as simple as a mouse click.

With DIFFzilla, you can:

- View differences between two files. See [Comparing Two Files](#).
- View differences between symbols and parts of files. See [Comparing Symbols or Parts of Files](#).
- View differences between all of the symbols in two files. See [Comparing All Symbols of Two Files](#).
- View differences between source trees. See [Comparing Two Directories](#).
- See intra-line differences, color-coded as you type. See [Dynamic Difference Editing](#).
- Generate file lists. See [Generating File Lists](#).
- Specify automatic directory mapping. See [Automatic Directory Mapping](#).
- Save/restore multi-file results.
- Utilize dialog box history (wild cards, paths, file specifications).

## Using the DIFFzilla® Dialog

The following sections describe how to use DIFFzilla and the differencing features in SlickEdit®. For more details on the specific options available on the DIFFzilla dialog (**Tools > File Difference** or **diff** command), see [DIFFzilla® Dialog](#).

## Dynamic Difference Editing

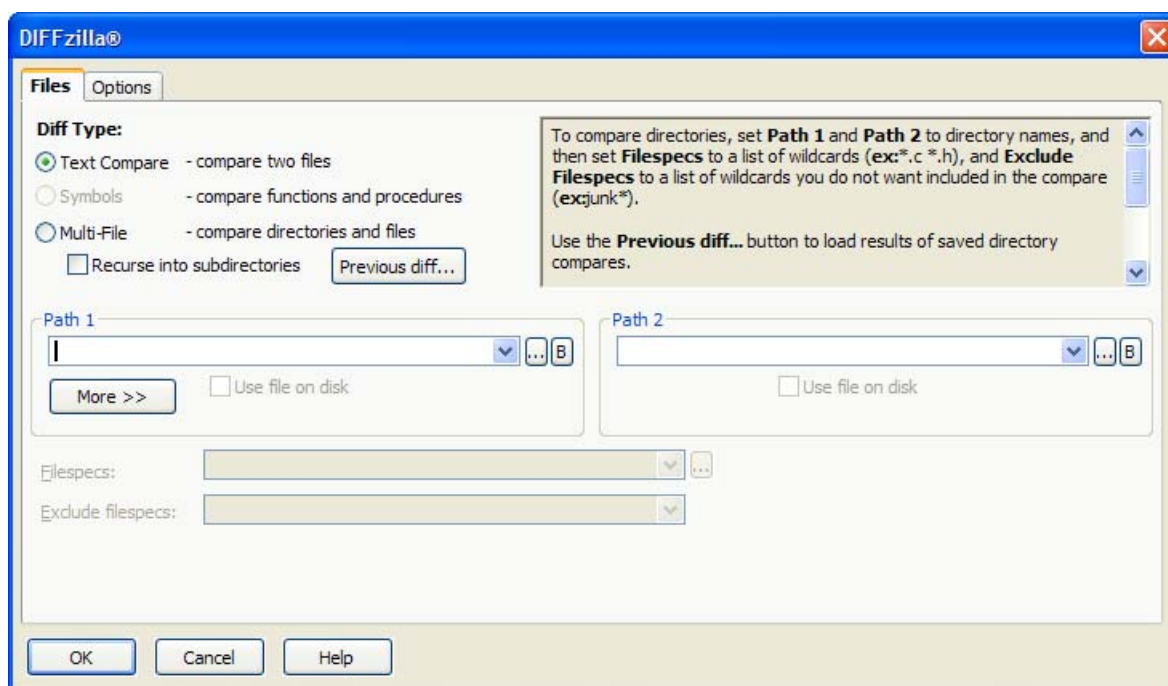
DIFFzilla® allows you to *diff*, or compare, files, and provides the ability to view the differences side-by-side or interleaved (one on top of the other). The output is color-coded. The side-by-side output differences can be merged from one file to the other, and you can edit the files directly inside the DIFFzilla dialog—this is called Dynamic Difference Editing.

Undo, copy/paste, Syntax Expansion/indenting, SmartPaste®, Auto List Members, Auto Parameter Info and many emulation key mappings work when editing in the DIFFzilla dialog box. When you type or make any edit, lines are re-diffed (compared again) so that you can view the new intra-line differences easily.

## Comparing Two Files

To diff two source files, complete the following steps:

1. From the main menu click **Tools > File Difference**, or use the **diff** command. The DIFFzilla® dialog appears, as pictured below.



2. Under **Diff Type**, select the **Text Compare** option.
3. Enter the name of the first file to compare in the **Path 1** text box. Enter the name of the second file in the **Path 2** text box. If the file names only differ by path, you only need to specify the path for Path 2.
4. Click **OK**.

## Comparing Symbols or Parts of Files

DIFFzilla® provides the ability to diff (compare) a selected range of lines from two files or the same file. This is very useful for comparing a piece of code that has been moved into a different part of a different file.

**NOTE** You can only use the interactive dialog output style when diffing a selected range of lines. Therefore, the option **Instead of an interactive dialog, output one buffer with the differences labeled**, on the DIFFzilla dialog Options tab, will have no effect.

To compare symbols, select the **Symbols** option under **Diff Type** on the DIFFzilla dialog, and all symbols from Path 1 will be diffed against all symbols from Path 2. If **Multi-File** is selected as the **Diff Type**, it always allows you to diff all symbols. Be sure to be careful when diffing all symbols. Some symbol blocks

are not yet picked up correctly. For example, for C++, C#, and Java, variable initializations are not yet handled correctly, as shown in the code below:

```
struct MYSTRUCT {
    int x;
    int y;
};

MYSTRUCT VariableDefinition= { // symbol definition stops here
    0,1
}; // really should end here
```

To diff a selected range of lines from two source files, complete the following steps:

1. From the main menu, select **Tools > File Difference**.
2. Select the **Multi-File** diff type.
3. Type the name of the first file in the **Path 1** text box.
4. Click **More**, and type the start and end line numbers next to **Line range** label.
5. Type the name of the second file in the **Path 2** text box and type the start and end line numbers next to **Line range** label. If the file names only differ by path, you need to specify the path for Path 2 only.
6. When differencing a symbol definition, click **Symbols** to enter the line number range.

## Comparing All Symbols of Two Files

DIFFzilla® allows you to diff all the symbols of two different files. This feature is most useful for diffing files with symbols that have been moved around. To diff all the symbols of two source files, complete the following steps.

1. From the main menu, select **Tools > File Difference** (or use the **diff** command).
2. Enter the first file in the **Path 1** text box.
3. Enter the second file in the **Path 2** text box. If the file names only differ by path, you need to specify the path for Path 2 only.
4. In the **Diff type**, select **Symbols** and click **OK**. You do not need to turn on the **Diff all symbols** check box when performing a multi-file diff because mismatching files will have a plus sign (+) in front of them so that you can diff all of the symbols.

## Comparing Two Directories

You can differences two source trees to determine what files have been added or removed and generate a list of filenames. When the source tree difference is complete, click **Save** to generate a list file. To diff two source trees, complete the following steps.

1. From the main menu, select **Tools > File Difference**, or use the **diff** command.
2. Mark the **Recurse into subdirectories** check box to compare subdirectories.
3. Enter the two directories in the **Path 1** and **Path 2** text boxes.
4. Fill in the **Filespecs** text box with the files that you want processed.
5. Click **OK**. The Multi-File Diff Output dialog is displayed.

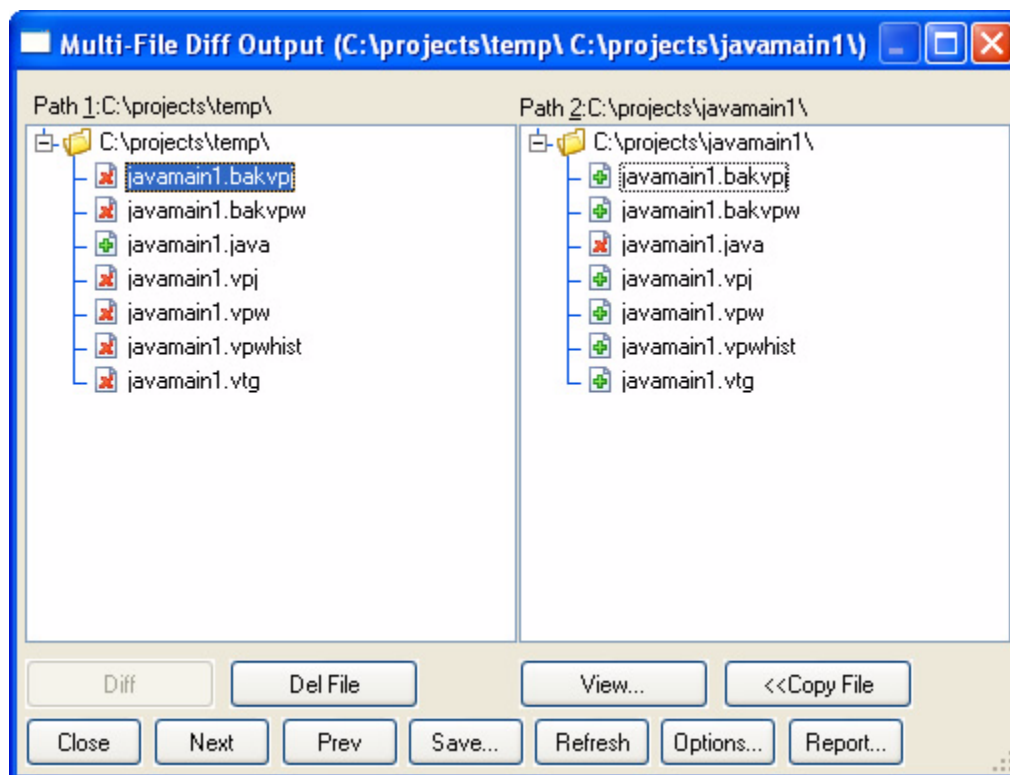
If a file exists in one tree but not the other, a plus sign (+) is displayed in the one tree and a minus sign (-) in the other. You can customize the files to view with the context menu. To display the context menu, right-click in the left or right tree. If you move the mouse over the plus or minus bitmap next to the item in the tool tree, a tooltip is displayed indicating what the bitmap means.

For descriptions of the buttons on the Multi-File Diff Output dialog, see [Multi-File Diff Output Dialog](#).

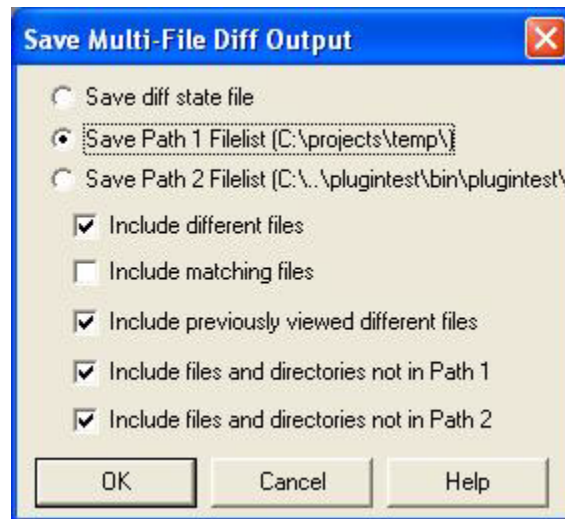
## Generating File Lists

DIFFzilla® can be used to find only the files that have been changed, and can generate file lists. The **Save** button in the Multi-File Diff Output dialog can create a list of files that includes different files, matching files, and files that do not exist in the other tree. Use the DIFFzilla dialog box to compare the new source tree with the original source tree.

1. From the main menu, select **Tools > File Difference**, or use the **diff** command.
2. On the **Files** tab, select **Multi-File**.
3. Enter the first file in the **Path 1** text box.
4. Enter the second file in the **Path 2** text box. If the file names only differ by path, you only need to specify the path for Path 2.
5. Click **OK**. The [Multi-File Diff Output Dialog](#) box opens.



- Click **Save**. The Save Multi-File Output dialog box opens.



- Select **Save Path 1 Filelist**, **Include different files**, and **Include files not in Path2**. All other check boxes should be clear.
- Click **OK** and select an output file for the list. The file you save will have a ".lst" appended to the output file name.
- Zip the files if you want.

## Automatic Directory Mapping

The DIFFzilla® dialog box automatically updates the **Path 2** text box with a directory, based on file paths that you previously typed in this field. For example, if you previously typed **f:\slick12\bitmaps\** into the Path 1 text box and **\\server\user\slick12\bitmaps\** into the Path 2 text box, then **f:\slick12\** is mapped to **\\server\user\slick12\**. The next time that you type **f:\slick12\macros\** in the Path 1 text box, **\\server\user\slick12\macros\** is automatically entered into the Path 2 text box.

To turn this option off, complete the following steps.

- From the main menu, select **Tools > File Difference**, or use the **diff** command.
- Select the **Options** tab.
- Click **Dialog Setup**.
- Clear the **Automatic directory mapping** check box.

## Diffing File History

The Backup History feature is available for viewing and comparing the differences between the current and previous versions of an open file. It utilizes the DIFFzilla® dialog for diffs (see [Using the DIFFzilla® Dialog](#)). For more information about this working with Backup History, see [File History and Backups](#).

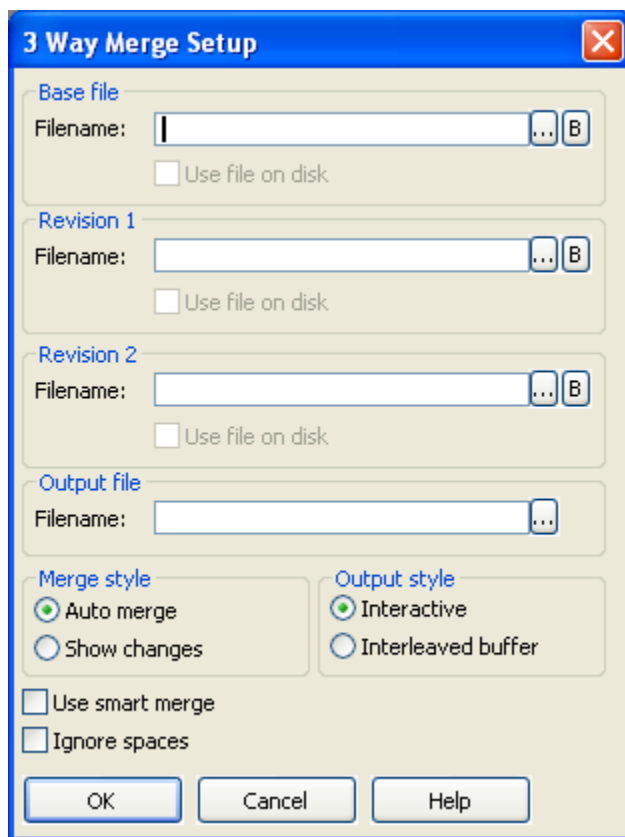
## 3-Way Merge

The 3-Way Merge editing feature can be used after two people make a local copy of the same source file, and each makes modifications to their local copy. The 3-Way Merge takes both sets of changes and creates a new source file. If there are any differences, a dialog box is displayed that lets you select the changes that you want in the output file. The output file can be viewed side-by-side or interleaved.

## Performing a Three-Way Merge

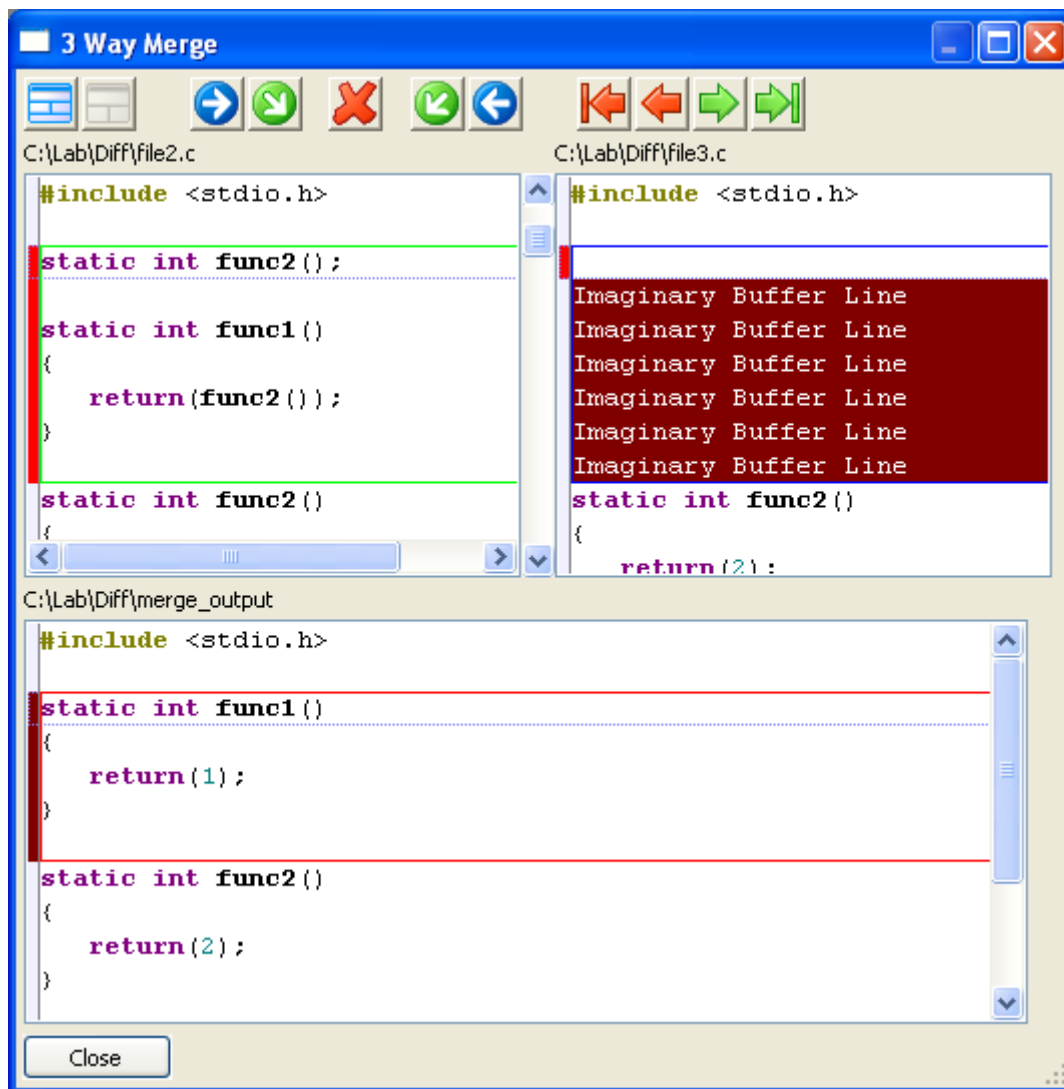
To perform a three-way merge, complete the following steps:

1. From the main menu, choose **Tools > File Merge** (or use the **merge** command). The 3-Way Merge Setup dialog is displayed.



2. In the **Filename** text box, enter the baseline (original) file name. Click the **Ellipses** icon to the right of the text box to select files. Click the **B** icon to select from the open buffers.
3. Enter the other names of the files to be merged in the **Revision 1** and **2** text boxes.
4. In the **Output file Filename** text box, enter the name of the output file, or click the dotted icon to select from an existing file.
5. Select any **Merge style** or **Output style** that you want.
6. Click **OK**. The following dialog box is displayed with the results of the 3-Way Merge:





## 3-Way Merge Settings

For descriptions of the options on the 3-Way Merge Setup dialog, see [3-Way Merge Dialog](#).

## The compare Command

The **compare** command compares two buffers in two tiled windows starting from the current cursor position of each window. If the current window is not one of two tiled windows, you will be prompted for the files/buffers you want to compare and two tiled windows will be set up for you.

**TIP** The functionality of the **compare** command has been replaced with DIFFzilla®. See [DIFFzilla®](#) for more information.

You can perform the following steps to manually set up two tiled windows before invoking the **compare** command:

1. Open (Ctrl+O) both files you wish to compare
2. Make current one of the files you wish to compare.
3. Zoom the current window by clicking on the **Maximize** button.
4. Use the **hsplit\_window** command (Ctrl+H or **Window > Hsplit**) to create two tiled windows.
5. Use the **link\_window** command (**Window > Link Window**) to display the other buffer in the newly created window.

After a compare mismatch, you can use the **resync** command to adjust the cursor in both windows to the next reasonable match. This command will be improved in the future to handle more sophisticated mismatches.

In ISPF emulation, this command is not called when invoked from the command line. Instead, **ispf\_compare** is called. Note that you cannot access the **compare** command when in ISPF emulation unless you bind it to a key.

## Setting Compare Options

The **compare\_options** command displays the Compare Options dialog box to set various compare options. The following settings are available:

- **Binary compare** - When selected, a stream compare (byte-by-byte) compare is performed. Typically a line-by-line compare is performed.
- **Expand tabs before compare** - When selected, tabs are expanded to the appropriate number of spaces, before lines from each file are compared.
- **Ignore leading spaces** - When selected, differences in leading spaces of lines are ignored.
- **Ignore trailing spaces** - When selected, differences in trailing spaces at the end of lines are ignored.
- **Ignore all spaces** - When selected, differences in spacing between characters in lines are ignored.
- **Ignore case** - When selected, differences in character casing is ignored.

# Version Control

---

## Overview of Version Control

Version control is accessed from the **Tools > Version Control** menu, by right-clicking within a file or buffer, or by right-clicking on an item in the Files list of the Open tool window.

The **History**, **Difference**, **Lock**, and **Properties** commands operate on the current buffer. For SCC version control systems, the SCC provider's function is called. For command line version control systems, the specified command is run, and the output is displayed.

The **Check In**, **Get**, **Check Out**, **Unlock**, **Add**, and **Remove** commands show a dialog box and operate on all files selected. This dialog will allow you to easily choose from files currently open, the files in the current project, or files in the current workspace. Note that CVS operations are different and designed to work especially well with CVS. Also, for SCC version control systems, you can select from files that are valid for that command. For example, when using the **Check In** command, you can click **Available** to view all files that can currently be checked in.

Use the **Manager** command to bring up the version control system's user interface. For many command line systems there will not be a program to call for this.

Menu items that appear grayed out for a command line version control system are blank. For SCC version control systems, **Lock** is always unavailable because the SCC interface does not make allowances for a lock command. It is possible to receive an "Operation not supported" message when running some commands if an SCC version control system does not implement an interface to that operation.

The following version control systems are supported:

- CCC/Harvest
- ClearCase
- ComponentSoftware RCS
- CVS
- MKS Source Integrity
- Perforce
- PVCS
- RCS
- SCC
- StarTeam
- Subversion
- TLIB
- Visual SourceSafe

## Using Version Control

Version control operations can be accessed from the main menu by choosing **Tools > Version Control**, and then choosing an operation. When you choose an operation, a dialog box for that operation is invoked. The left side of each dialog box contains a list of files determined by the options you have selected. You can click on individual files to select them, or you can select multiple files by using Ctrl+Click or Shift+Click. The following options are available for each operation:

- **Workspace** - When checked, all files that are in the current workspace are listed.
- **Project** - When checked, all files that are in the current project are listed.

- **Buffers** - When checked, all files that are currently open in the editor are listed.
- **Available** - When checked, all files that are available for the specified operation are listed. This option is only available for SCC version control systems.
- **Advanced button** - Click this button to configure the specified operation's options. This button is only available for SCC version control systems that support these options.
- **Check In** - To check files in to the version control system, from the main menu choose **Tools > Version Control > Check In**, or use the **vccheckin** command. Select the files in the list that you wish to check in, then click the **Checkin** button.

The Check In dialog provides an additional option to **Save if modified**. When this option is selected, any files that are modified are saved before check-in.

- **Get Files** - To retrieve files from the version control system, from the main menu choose **Tools > Version Control > Get**, or use the **vcget** command. Select the files in the list that you wish to get, then click the **Get** button.
- **Check Out** - To check files out of the version control system, from the main menu choose **Tools > Version Control > Check Out**, or use the **vccheckout** command. Select the files in the list that you wish to check out, then click the **Checkout** button.
- **Lock** - To lock files in the version control system, from the main menu choose **Tools > Version Control > Lock**, or use the **vclock** command. Select the files in the list that you wish to lock, then click the **Lock** button.
- **Unlock** - To unlock files in the version control system, from the main menu choose **Tools > Version Control > Unlock**, or use the **vcunlock** command. Select the files in the list that you wish to unlock, then click the **Unlock** button.
- **Add files** - To add files to the version control system, from the main menu choose **Tools > Version Control > Add**, or use the **vcadd** command. Select the files in the list that you wish to add, then click the **Add** button.

The Add Files dialog provides two additional buttons:

- The **Browse** button will invoke a dialog box that allows you to add other files to the list, so that you can select them to be added to the version control system.
- The **Remove** button will remove files from the list.
- **Remove files** - To remove files from the version control system, from the main menu choose **Tools > Version Control > Remove**, or use the **vcremove** command. Select the files in the list that you wish to remove, then click the **Remove** button.

## Configuring Version Control

Before using Version Control, you should configure your setup. Invoke the Version Control Setup dialog box from the main menu by choosing **Tools > Version Control > Setup** (**vcsetup** command).

This dialog box allows you to modify the command strings for a specific version control system. Command strings are stored in the file `vcsystem.slk` (UNIX: `uvcsys.slk`), and `uservc.slk` and are the same for all projects which use the version control system selected.

For a list and descriptions of the options on the Version Control Setup dialog, see [Version Control Setup Dialog](#).

## Advanced Setup Options

You can access advanced setup options for each version control system that supports them. From the Version Control Setup dialog box (**Tools > Version Control > Setup** or **vcsetup** command), select the version control system you wish to set up. Click on the **Setup** button. This will display the setup dialog box for the system you have selected. Click the **Advanced** button on this dialog to access the advanced options. The options are similar for each version control system. See [Version Control Advanced Setup Dialog](#) for a list and descriptions of the options.

## Setting Up Command Line Version Control Systems

If you are on a non-Windows platform, or are using a version control system that does not support SCC, the version control system must have a command line interface. To configure a command line version control system for a system that is using a command line driven operating system, complete the following steps:

1. From the main menu, select **Tools > Version Control > Setup** (or use the **vcsetup** command).
2. Select the **Command line systems** check box.
3. Select a **Version Control System**. Be sure that the executable file for each version control command is in your file path, and that the executable files that are included with the version control system can be found in the PATH environment variable. Depending on the system you are using, you might need to complete information in the **VCS Project** text box.
4. Click **Setup** and verify that each command matches the options that you want. (If you are not sure what any of the **%<character>** macros expand to, click on the arrow to the right of the text box to view a list.)
5. Click **OK**.

## Specific Version Control Support

### Source Code Control (SCC)

SCC is a Source Code Control interface specification that was designed by Microsoft. This interface allows for direct communication between a version control system and another software application. When you are using SCC, keep in mind the following information:

- If you are using a system that supports SCC, use the SCC support because it provides tighter integration.
- If your system does not have an SCC interface installed, contact the manufacturer to be sure that an SCC interface is not available.
- If your version control system has an SCC interface, but does not seem to behave properly, contact SlickEdit Product Support.

Version control systems that have support for an SCC interface are supported. If you are using PVCS, install the SCC interface support because it does not install SCC support by default. SourceSafe automatically installs SCC support.

The following list of version control systems have SCC interfaces. If any of these systems are not displayed in the SCC Providers list (or the SCC Providers list does not appear) dialog box, you might need to install the support separately:

- CCC/Harvest
- ClearCase

- ComponentSoftware RCS
- MKS Source Integrity
- Star Team

### Configuring SCC

SCC version control is available for systems that are using a Windows operating system only. To configure an SCC version control system, complete the following steps.

1. From the main menu, select **Tools > Version Control > Setup** (or use the **vcsetup** command).
2. Select the system that you want.
3. Click **Initialize Provider**.
4. Click **Open Project** and complete the Open Project dialog box.
5. Click **OK**. A project name is displayed in the list of **SCC Version Control Systems**. This version control system and project are now bound together.

### Opening an SCC Project

The SCC version control system is used to help you manage your files when you are working with projects. SCC is available for systems that are using a Windows operating system only.

To open an SCC version control project, complete the following steps.

1. From the main menu, select **Tools > Version Control > Setup** (or use the **vcsetup** command).
2. Be sure that **SCC providers** check box is marked. If there are no SCC providers installed, the check box is unavailable. To enable this feature, install an SCC provider. For more information, see [Configuring SCC](#) above.
3. Select the system that you want, and click **Initialize Provider**.
4. Click **Open Project**. The Open Project dialog box is displayed.

The following list contains additional information to assist you when using SCC with certain applications:

- **Source Integrity** - For the **SI Project Filename**, type the entire path for the `project.vpj` file. For **Sandbox Path**, type the path where the files are located on the local system.
- **Perforce** - For **Client Name**, type the name of the Perforce depot (for example, “//depot”). For **Local Path**, type the name of the path where the files are located on the local machine.
- **StarTeam** - For **StarTeam Project Name**, type the name of the StarTeam project. For **Local Path**, type the name of the path where the files are located on the local system. You are then prompted for additional information by StarTeam.

After you open an SCC version control project, it is bound to the currently active project. When you restart the project, or switch to this project from another project, this version control project is automatically activated.

## PVCS

If you are using PVCS, there is typically no need to switch version control projects since the source files are placed in the same directory as the archive files. In some PVCS configurations, you will want to set some environment variables when you switch projects. To set these environmental variables, complete the following steps.

1. From the main menu, select **Project > Project Properties**.

2. Click the **Open Command tab**, and type one or more **set** statements to set the environment variables.
3. Close and then reopen the project for the project macro to be executed.

## CVS

A graphical interface for CVS updates is provided. Before updating a directory, a dialog box is displayed that provides the status information for each file. You can then select the files that you want to update, commit, or add. Use DIFFzilla® to view differences between various versions of a file (see [DIFFzilla®](#)).

Move the mouse pointer over the icon to the left of the file to display a tooltip which indicates what the icon means. The file icon with a blue star means that the file is not up-to-date. A file icon with a red star means that you have modified the file.

A graphical history dialog for files checked out from CVS is also provided. This includes displaying the current state of the file, all tags for the file and a graphical display of the branches. You can also use DIFFzilla to view differences between the current version and any past version, or any two past versions with each other (see [DIFFzilla®](#)).

**Commit sets** are a way to group files checked out of CVS that need to be checked in at the same time. For example, when fixing a defect you may want to group all of the files that you modified. **Commit sets** allow you to do this, give a common comment for the group of files and then give an individual comment to each file. When you are done, you can review the commit set, which allows you to easily compare each file with the most up-to-date version using DIFFzilla (see [DIFFzilla®](#)). Then, you can commit all of the files at one time.

## Subversion

Subversion support provides easy convenient access to information about the files with which you are working, and also a GUI checkout dialog. To get started, go to **Tools > Version Control > Setup** (**vcsetup** command), and set the **Command line system** to **Subversion**. After you enable these settings, you can diff any file with the current version on its branch, view the history of the file, and update or commit the file. You can also select **Tools > Version Control > Compare Workspace with Subversion** to compare your local workspace with the files in the repository. The Subversion support mimics the existing SlickEdit® CVS support.





# Spell Checking

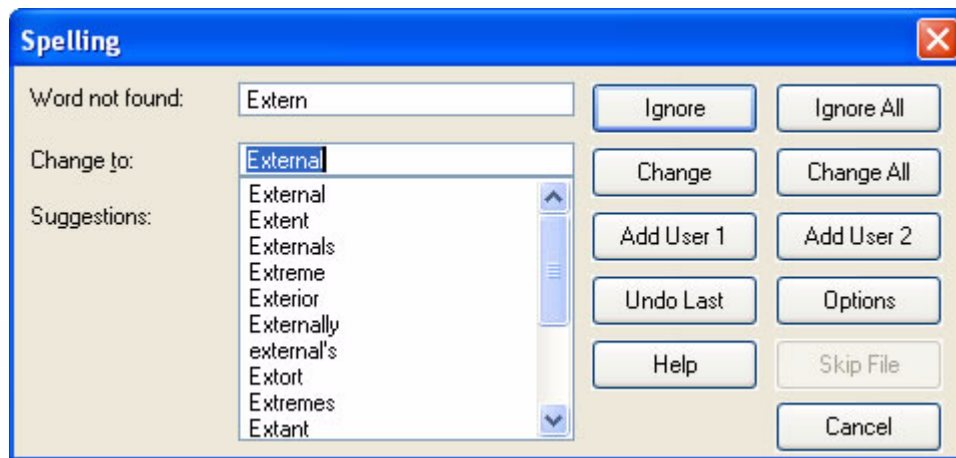
## Spell Check Operations

You can access spell checking operations from the main menu by choosing **Tools > Spell Check**. Select one of the following operations:

- **Check from Cursor** - Check spelling on the open file starting at the cursor's location. You can also use the **spell\_check** command to perform this operation.
- **Check Comments and Strings** - Check spelling only on comments and strings within the open file. Spell Check will start at the cursor's location. You can also use the **spell\_check\_source** command to perform this operation.
- **Check Selection** - Check spelling only on text that is currently selected. The **spell\_check** command also works for this operation.
- **Check Word at Cursor** - Check spelling only for the word currently under the cursor. You can also use the **spell\_check\_word** command to perform this operation.
- **Check Files** - Check spelling on multiple files. You can also use the **spell\_check\_files** command to perform this operation. This will invoke the Spell Check Files dialog, which allows you to specify the files for checking. See [Spell Checking Multiple Files](#).

## Running Spell Check

When Spell Check is running and a word is found that is not in the dictionary, the Spelling dialog appears, prompting you for action.



The dialog shows the word not found and gives suggestions for a word replacement. Use the buttons on the dialog to perform the following actions:

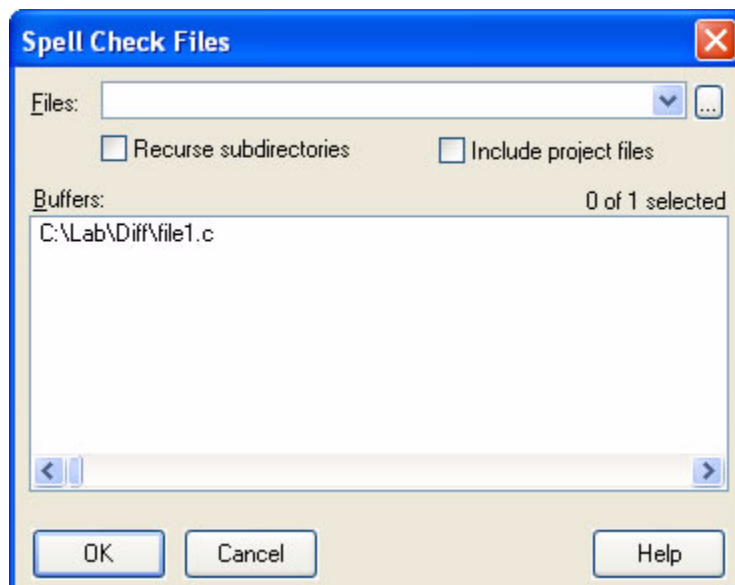
- **Ignore** - Disregard this word and continue spell checking.
- **Ignore all** - Disregard all instances of this word in the selected range and continue spell checking.

## SPELL CHECKING

- **Change** - Replace this word with the text in the **Change to** text box and continue spell checking. You can use the suggested word or type your own word.
- **Change All** - Replace all instances of this word in the selected range with the text in the **Change to** text box, and continue spell checking.
- **Add User 1 and 2** - Add the word not found to one of two custom dictionaries, after which spell checking continues. The first time new words are added to these lists, SlickEdit® creates new files in your configuration directory named `userdct1.lst` and `userdct2.lst`. See [Setting Spell Check Options](#) for more information on custom dictionary files.
- **Undo Last** - Undo the last spell checking operation. The focus is placed on the last word not found.
- **Options** - Displays the Spell Options dialog. See [Setting Spell Check Options](#).
- **Skip File** - When spell checking multiple files, use this button to skip checking in the current file.

## Spell Checking Multiple Files

The Spell Check Files dialog is used to specify multiple files for checking, and always does a language-sensitive spell check. For HTML, markup that is not literal text is ignored. For source languages where color coding is provided, only comments and strings are checked for spelling. To access the Spell Check dialog box, pictured below, from the main menu choose **Tools > Spell Check > Check Files** (or use the `spell_check_files` command).



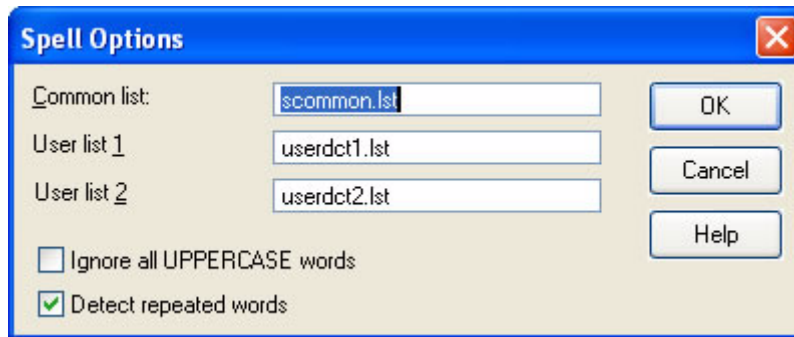
In the **Files** text box, enter one or more files separated by spaces. Wildcards may be used (for example, `*.html` or `*.c`). You can use the **Browse** button to the right of this text box to choose a directory. There are two file options available:

- **Recurse subdirectories** - If checked, wildcard file specifications in the **Files** text box will process subdirectories recursively.
- **Include project files** - If checked, all project files are checked for spelling.

The **Buffers** list box lists the open buffers that will be spell checked in addition to the directory specified.

## Setting Spell Check Options

To specify the dictionary word list to use for spell checking, use the Spell Options dialog box, pictured below. To access this dialog, from the main menu choose **Tools > Spell Check > Spell Options**, or use the command **spell\_options**.



Dictionary lists are text files that have the extension `.lst`. Words that are frequently misspelled are spelled correctly in these files for matching during spell checking. Each line of a dictionary list can contain only one word. The following list describes the fields and options on the Spell Options dialog:

- **Common list** - The default dictionary list is named `scommon.lst` and is located in the SlickEdit® installation directory.
- **User lists** - You can have up to two custom dictionaries. When a word is not found during a spell check and you add the word to **User list 1** or **2**, the word will be added to the file specified in the **User list 1 or 2** text box. By default, the first time words are added, SlickEdit will create files named `userdct1.lst` and `userdct2.lst` in your configuration directory, so these file names are filled in for you. If you choose to create your own files, be sure to place them in the default configuration directory and add the file names here.
- **Ignore all UPPERCASE words** - Applies to all Spell Check operations. By default, this option is not selected.
- **Detect repeated words** - Applies to all Spell Check operations. By default, this option is selected.



# FTP

---

FTP support within SlickEdit® includes a complete FTP/SFTP client and the ability to easily open and edit FTP files.

## Working with FTP

Before you can access FTP files, you must create an FTP profile, then start that connection. FTP operations can be accessed from FTP tool windows or by right-clicking on FTP files after a connection is active.

### FTP Tool Windows

There are two tool windows available for working with FTP: FTP and FTP Client.

The FTP tool window can be used to connect to FTP servers and open files. To access this tool window, from the main menu, choose **View > Toolbars > FTP**. Right-click on files to display a menu of FTP operations.

The FTP Client tool window can also be used to connect to FTP servers and transfer files. As with most FTP clients, local directories and files are displayed in the left section of the tool window, and the FTP server directories and files are on the right. To access this tool window, from the main menu, choose **View > Toolbars > FTP Client**. Right-click on files to display a menu of FTP operations.

### Creating a New FTP Profile

To create a new FTP connection profile, complete the following steps:

1. From the main menu, choose **File > FTP > Profile Manager** (`ftpprofilemanager` command). Alternatively, you can display the FTP tool window (**View > Toolbars > FTP**) and click the icon labeled **Start a New Session** (shown below).



The FTP Profile Manager dialog box is displayed, as pictured below.



2. Click **Add** to create a new profile. The Add FTP Profile dialog box is displayed.
3. Click **Edit** to Edit a profile. The Edit FTP Profile dialog box is displayed.

See [Setting FTP Options](#) for information about the options on the Add or Edit FTP Profile dialogs.

## Starting a Connection

To start a new connection, use the FTP or FTP Client tool windows described above, and complete the following steps:

1. Click the **FTP** icon to start a new session.



2. The FTP Profile Manager dialog box appears. From the Profiles list, select the profile name to connect to.
3. Click **Connect**. The FTP tool window displays the content of the remote directory.
4. Toggle the **ASCII Transfer mode** button to transfer text files. When in ASCII transfer mode, line ending characters may be translated.
5. Toggle the **Binary Transfer mode** button to transfer images and executables.



6. To stop the current operation, click stoplight.



## Stopping a Connection

To stop a connection, use the FTP or FTP Client tool windows, and complete the following steps:

1. Select the connection that you want from the drop-down list at the top of the tool window.
2. Click the **Disconnect Current Session** button.



## Opening FTP Files

Before you can open FTP files, you need to start a connection. See [Starting a Connection](#) above for more information. After your connection starts, from the FTP or FTP Client tool window, right-click on selected files to open them, to change the directory, or to access more options.

## Setting FTP Options

There are two types of settings available for working with FTP:

- **FTP connection profile options** - These options are used to add or edit new FTP connection profiles. The Add/Edit FTP Profile dialog box is used. To access this dialog, choose **File > FTP > Profile Manager**, then select **New** or **Edit** to edit an existing selected profile. See [Add/Edit FTP Profile Dialog](#) for a complete list of available options.
- **Default FTP options** - These are the general default FTP settings. To access these options, choose **File > FTP > Options**. The FTP Options dialog will be displayed. See [FTP Options Dialog](#) for a list of the available options.





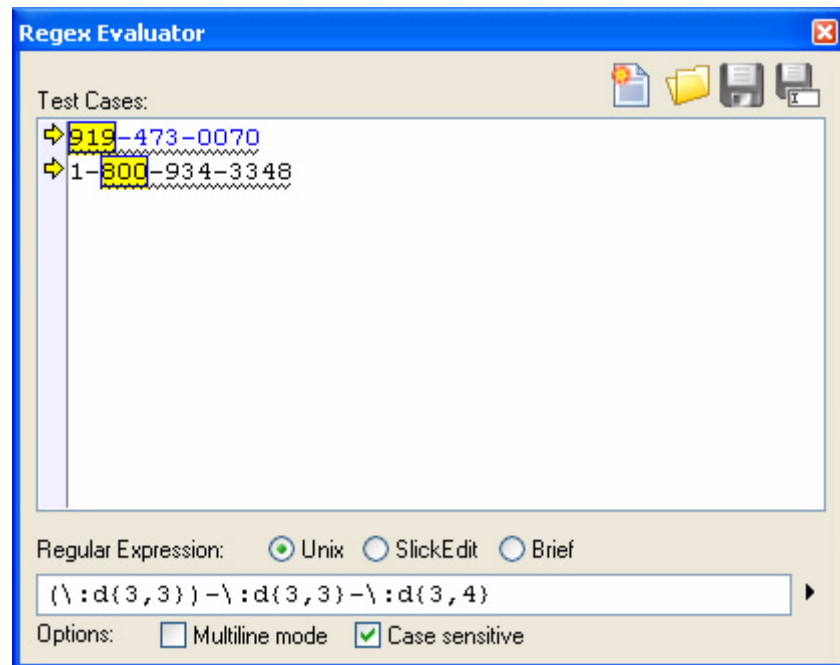
## The Regex Evaluator

Regular expressions are used to express text patterns for searching. The Regex Evaluator provides the capability to interactively create, save, and re-use tests of regular expressions.

To access the Regex Evaluator, select **Tools > Regex Evaluator** (or use the `activate_regex_evaluator` command). Like other tool windows in SlickEdit®, this tool window is dockable. Docking options can be accessed by right-clicking on the tool window's title bar.

## Using the Regex Evaluator

Type some samples of the text you are trying to match in the top portion of the tool window labeled **Test Cases**. Enter your regular expression pattern in the bottom field. The Regex Evaluator will highlight matched portions of your sample text and identify groups.



## Entering Test Cases

Type your test cases in the **Test Cases** text box. These test cases will be evaluated as you type your regular expression in the bottom field. A wavy underline will indicate the ranges of text that match the entire expression. Matches are also marked with a yellow arrow that appears in the gutter to the left of the test case. You can hover your mouse on this arrow to see a tooltip which displays the matched expression details. When groups (tagged expressions) are used in your regular expression pattern, the groups will be boxed and highlighted in yellow in the Test Cases section.

## Entering a Regular Expression

Enter the regular expression to test in the text field. Use the radio buttons to select the expression syntax that you wish to use: UNIX, SlickEdit®, or Brief. Click the arrow to the right of the regular expression field to pick from a menu of common syntax and operators.

## Regex Evaluator Options

The following options and buttons are available on the Regex Evaluator tool window:

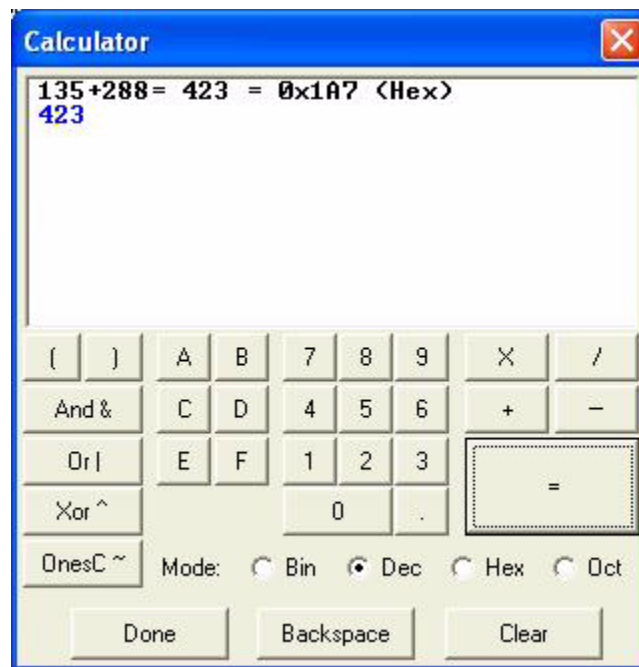
- **Multiline mode** - If **Multiline mode** is selected, rather than searching through the test cases line-by-line, regular expressions will be searched on all lines at once. This is useful for test cases that wrap to the next line. This works just as if you had entered `\om` on the SlickEdit® command line.
- **Case sensitive** - If **Case sensitive** is selected, the regular expression search will be case sensitive. This option is enabled by default.
- **New expression** icon - To clear the tool window of all entries in order to start a new evaluation, click the icon at the top of the tool window labeled **New expression**.
- **Open a saved expression** icon - To open an expression that you have already saved, click the folder icon at the top of the tool window labeled **Open a saved expression**.
- **Save the current expression** icon - To save the current expression, click the diskette icon at the top of the tool window labeled **Save the current expression**. Both the expression and the test cases will be saved to a file. The default extension is `.regx`.
- **Save as** icon - To save the current expression with a different file name than what has previously been saved, click the icon at the top of the tool window labeled **Save the current expression as**.

## Using the Calculator and Math Commands

---

### The Calculator

To access the calculator, select **Tools > Calculator**, or use the **calculator** command. The Calculator is displayed as shown below.



You can use the calculator in various ways. Type in mathematical expressions from the keyboard or by clicking buttons, including parentheses. Almost all the editing keys including undo, next word, and previous word are supported. The calculator uses a slightly enhanced C expression syntax. The Calculator supports specifying binary numbers and allows just an **x** prefix when specifying hexadecimal numbers.

For example, to add the decimal numbers 135 and 288, type **135+288=**. Press the **=** character to evaluate the expression and place the result on the next line. To see the result in a different base, click **Hex**, **Dec**, **Oct**, or **Bin**.

### Calculating Expressions with Mixed Bases

To add hex FF with octal 77 with binary 111 with decimal 99, complete the following steps:

1. Click the **Hex** button and type or click **FF**.
2. Click **+**.
3. Click **Octal**, and type or click **77**.
4. Click **+**.
5. Click **Bin**, and type or click **111**.
6. Click **+**.

7. Click **Dec**, and type or click **99**.
8. Select the output base by clicking one of the base buttons and type or click **=** to compute the result.

## Math Commands

Evaluate mathematical expressions by selecting expressions in a buffer and executing the **add** command or by executing one of the math commands on the command line followed by an expression.

These commands support the same expression input. The syntax of the **math** command is:

**math expression**

The **math** command evaluates the Slick-C® language expression given and places the results in the message line. You can specify octal numbers by prefixing the number with a zero and specify binary numbers by prefixing the number with the character **b**. If no operator is specified between two unary expressions, addition is assumed. The characters **\$** and comma (,) are stripped from the expression before it is evaluated. The **mathx**, **matho**, and **mathb** commands evaluate the Slick-C language expression given and places the result in the message line in hexadecimal, octal, and binary respectively. The *expression* can have the following unary operators:

- **~** bitwise complement
- **-** negation
- **+** no change

The available binary operators are listed below, from lowest to highest precedence. A comma after the operator indicates that the next operator is of the same precedence.

<b>&amp;,  </b>	bitwise AND, bitwise OR
<b>^</b>	xor
<b>+, blank(s), -</b>	addition, implied addition, subtraction
<b>*, /, %</b>	multiplication, division, remainder
<b>**</b>	power

Hexadecimal numbers are prefixed with the characters **0x** or just **x**. Octal numbers are prefixed with the character **O** or digit **0**.

**NOTE** Not all Slick-C language operators are supported.

## Math Command Examples

The following table shows some examples of math commands:

<code>math 2.5*2</code>	Multiplies 2.5 times 2
<code>math 5/2</code>	Divides 5 by 2
<code>mathx 255</code>	Converts 255 to hexadecimal

<code>math xFF</code>	Converts hexadecimal FF to decimal
<code>math o77</code>	Converts octal 77 to decimal
<code>matho 255</code>	Converts 255 to octal
<code>math 077+0xff+10</code>	Adds octal 77, hex FF, and 10

## Overflow/Underflow

If overflow or underflow occurs, the message “Numeric overflow or underflow” is displayed on the message line. Floating point numbers may have up to a 32-digit mantissa and a 9-digit exponent.

## Document Math

Type mathematical expressions into a buffer and evaluate them with the **add** command. This feature is called document math. The **add** command adds selected text and inserts the result below the last line of the selection. If no operator exists between two adjacent numbers on the same line, addition is assumed. The result of each adjacent line is added.

## Prime Numbers

Prime numbers are often useful for sizing hash tables. The **isprime** command (used from the command line) takes a decimal number as an argument and tells you if it is prime, and if not, its first divisor. The **nex-tpime** command takes a decimal number as an argument and finds the next greater prime number.



## OS File Browser

---

SlickEdit® provides a way to display the operating system's (OS) file manager/browser. For example, Windows Explorer is displayed on Windows, Finder on Mac OS X, Konquerer on Linux KDE desktop, etc.

To display the OS file browser, select **Tools > OS File Browser**, or use the **explore** or **finder** command (the **finder** command is the same as the **explore** command).

If you are editing a document, the file manager will be rooted in that file's directory, otherwise it will default to the current working directory. Using the - option after the command (for example, **explore -**) will ignore any file directory or working directory and go to the system root.





# Macros and Macro Programming

This chapter contains the following topics:

- [Recorded Macros](#)
- [Programmable Macros](#)



## Recorded Macros

---

There are two types of macros in SlickEdit®: macros that you record, described below, and macros that are available for programming (see [Programmable Macros](#)).

You can automate repetitive tasks by recording a series of SlickEdit® operations in a macro. After you create a macro, you can run it, save it, bind it to a key sequence, and/or modify the macro's source code.

Recording a macro generates Slick-C® code for performing the action being recorded. Therefore, recording a macro is also a useful way to discover and implement Slick-C code that controls SlickEdit's behavior. See [Using Macros to Discover and Control Options](#) for information.

## Recorded Macro Operations

Macros can be recorded, executed, and saved from the **Macro** menu, or you can use commands or predefined key bindings to perform macro operations:

- To start or end macro recording, from the main menu, click **Macro > Record Macro** or **Macro > Stop Recording Macro**, respectively. Alternately, you can toggle recording on and off with one of the following methods:
  - Click the recording indicator **REC**, located along the bottom edge of the editor. When a macro is being recorded, the recording indicator is active (not dimmed).
  - In CUA emulation, press **Ctrl+F11** (the key binding associated with the `record_macro_toggle` command).
  - On the SlickEdit command line, type `record_macro_toggle`.

See [Recording a Macro](#) for more information.

- To run the last macro that you recorded, click **Macro > Execute last-macro**, press **Ctrl+F12**, or use the `record_macro_end_execute` command. See [Running a Recorded Macro](#) for more information.
- To display a list of your recorded macros, from which you can edit, run, delete, or bind to a key sequence, click **Macro > List Macros**, or use the `list_macros` command.

**NOTE** List Macros only shows your “saved” macros, not your last recorded macro or macros created using `execute_last_macro_key`.

## Recording a Macro

To record a macro, simply start the recording, enter the keystrokes you want to record, then end the recording. The instructions below outline the steps.

1. From the main menu, click **Macro > Record Macro** (or use one of the toggle methods to start recording, as described under [Recorded Macro Operations](#) above).
2. Enter the keystrokes that you want to record. For example, to record a macro of the cursor moving three spaces to the right, press the right arrow key three times. You can also change a configuration option, view settings, or expand a code template during macro recording.

3. Select **Macro > Stop Recording Macro** (or the same toggle you used in Step 1) to end recording. The [Save Macro Dialog](#) is displayed.

**TIP** For recorded macros you don't need to track, perhaps for immediate or one-time use, SlickEdit provides a way to stop macro recording and instantly bind the macro to a key sequence. This allows you to keep a set of recent, unnamed macro recordings instead of having just one "last recorded macro". See [Binding Macros Using execute\\_last\\_macro\\_key](#) for more information.

4. The next step depends on the purpose of your recorded macro. If you plan to run the macro, continue with the steps below. If you're just recording it to discover Slick-C® code (see [Using Macros to Discover and Control Options](#)), and don't care about naming or saving it, click **Edit** (or press **Alt+E**) at this time to view the source code. However, you will not be prompted to save the macro and bind it to a key sequence. In order to do that, you will need to use **Macro > Save last-macro** prior to recording a new macro or exiting the editor. See [Saving and Editing Recorded Macros](#) for more details.
5. Specify the name for the macro in the **Macro Name** text box.
6. Select the options that you want from the following, or leave the defaults if you aren't sure:
  - **Requires editor control** - Check this box if your macro can only operate if the target is an editor control.
  - **Allow in read only mode** - Check this box if your macro does not modify the current buffer.
  - **Allow when window is iconized** - You will probably NOT want this box checked if your macro modifies the current buffer. Whether to check this box is more a matter of personal taste.
  - **Allow in non-MDI editor control** - Check this box if your macro should be allowed in a non-MDI editor control. This is typical for commands which require an editor control but do not open or close editor windows/buffers.
7. Click **Save**. The [List Macros Dialog](#) is displayed, from which you can run the macro, edit the source, delete it, or choose to bind it to a key sequence. If you plan to use the macro often, it's best to go ahead and create a key binding for it now. See [Binding Recorded Macros to Keys](#) for more information.

## Binding Recorded Macros to Keys

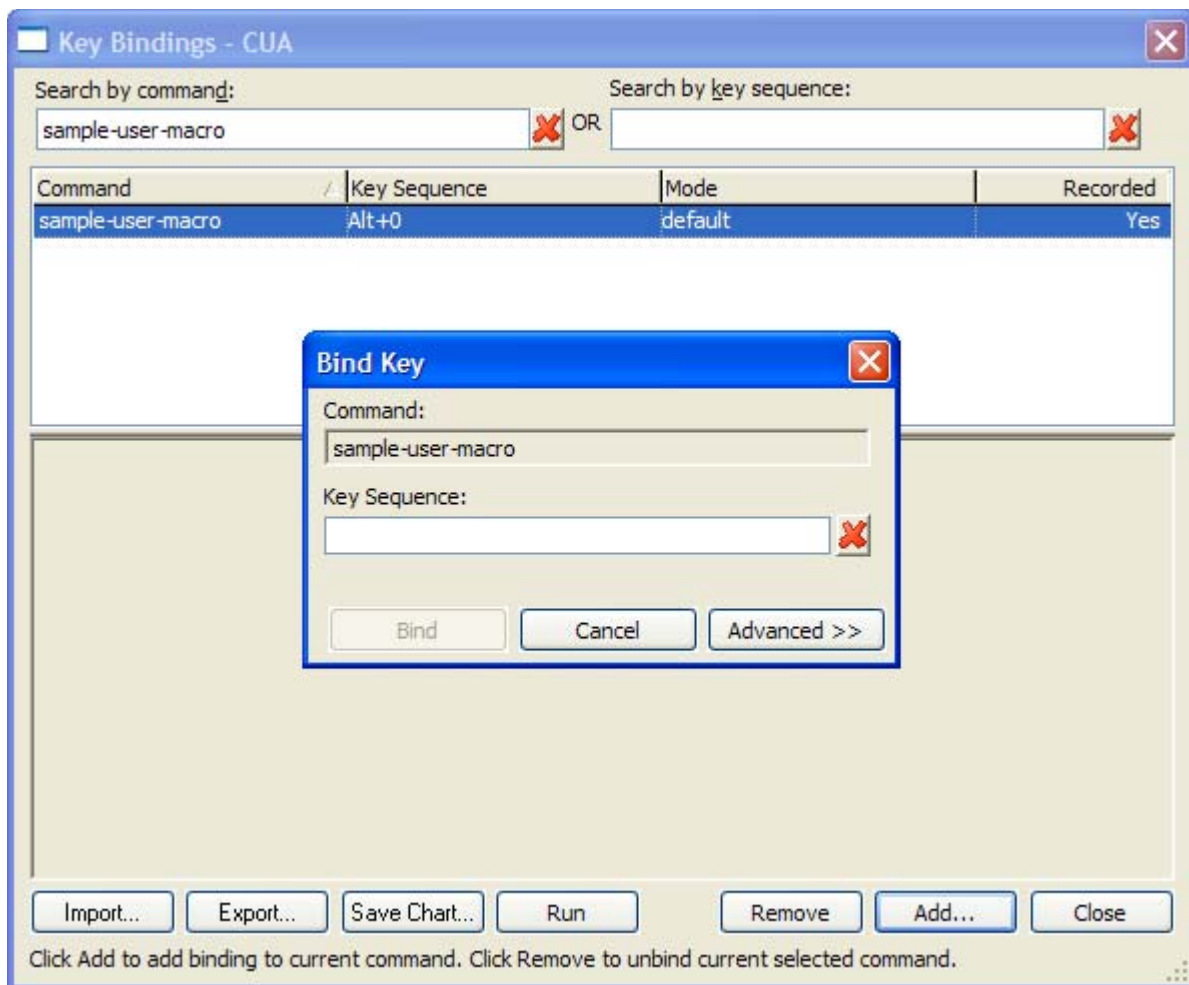
To use recorded macros most effectively, create key bindings for them so they can be executed quickly when you want to use them. Macros can be bound through the Key Bindings dialog (see [Binding Macros Using the Key Bindings Dialog](#)), or by using the instant "stop recording and bind" method associated with the `execute_last_macro_key` command (see [Binding Macros Using execute\\_last\\_macro\\_key](#)).

### Binding Macros Using the Key Bindings Dialog

After recording a new macro, the [List Macros Dialog](#) is automatically displayed. You can access the List Macros dialog any time from the main menu by clicking **Macro > List Macros**, or by using the `list_macros`

command. Click **Bind to Key** to open the Key Bindings dialog, showing a listing of only your recorded macros.

**NOTE** You can also display the Key Bindings dialog by clicking **Tools > Options > Key Bindings**, or by using the **gui\_keybindings** command. However, if you display the dialog in this manner, it will show a list of all commands and user-recorded macros. To view your recorded macros, click on the **Recorded** column header to sort and display items with a “Yes” (which indicates these are recorded macros). A more convenient method is to use the **Bind to Key** button on the List Macros dialog to only show recorded macros in the Key Bindings dialog.



Creating bindings for recorded macros works the same as creating bindings for SlickEdit commands. Click **Add** to initiate the binding, then specify the key sequence or mouse event to use. See [Managing Bindings](#) for more information about creating, editing, and removing bindings.

### Binding Macros Using `execute_last_macro_key`

The **`execute_last_macro_key`** command provides functionality to stop macro recording and instantly bind the macro to a key sequence. This feature is convenient for recorded macros you want to use perhaps immediately or one-time only, and don't need to track. It allows you to keep a set of recent, unnamed

macro recordings instead of having just one “last recorded macro”, similar to a feature provided by early text editors that supported macro recording, such as the EVE and Edt editors on the Vax (VMS).

Unlike other SlickEdit commands we document, **execute\_last\_macro\_key** is not intended to be used on the command line—instead, you use a key binding that is automatically assigned when you press it to stop macro recording.

To bind a macro to a key sequence using this method, start recording the macro and enter the keystrokes you want to record. Then press **Ctrl+Shift+F12, key** where **key** stands for keys **0-9**, **A-Z**, or **F1-F12**, to stop recording the macro and instantly bind it to the key sequence you just pressed.

**NOTE** The prefix key sequence **Ctrl+Shift+F12** works in all emulations except SlickEdit text mode edition. In that emulation, the prefix key sequence is **Ctrl+Shift+T**.

Each macro that you record and bind using this feature is saved to a new file named `lastmac<key>.e`, located in your configuration directory, where `<key>` matches the **key** you used when creating the binding (keys **0-9**, **A-Z**, or **F1-F12**). These files can be helpful for determining what was recorded, because if you use this method to bind a recorded macro, you will not have an opportunity to name the macro or see a list of macros created with this method (they will not appear in the List Macros or Key Bindings dialogs).

## Running a Recorded Macro

If you have saved the macro and created a key binding for it, the easiest way to run it is to simply press the associated key sequence. You can also run it by:

- Typing the name of the macro in the SlickEdit® command line then pressing **Enter**.
- Using the [List Macros Dialog](#) (**Macro > List Macros** or `list_macros` command)—select the macro and click **Run**.

You can run the last macro that you recorded, whether it was saved or not, by clicking **Macro > Execute last-macro** (**Ctrl+F12** or `execute_last_macro` command).

## Saving and Editing Recorded Macros

When a recorded macro is saved, the source code of the macro is appended to the `vusrmacros.e` user macros file located in your configuration directory.

To edit a macro that has previously been recorded and saved, from the main menu, click **Macro > List Macros** (or use the `list_macros` command) to display the [List Macros Dialog](#). The list box on the left displays a list of your recorded macros. Select the macro you want to edit, then click **Edit**. The `vusrmacros.e` file opens in the editor. Save the file when you're done making edits.

If you are using recorded macros to discover variables (see [Using Macros to Discover and Control Options](#)), you can view/edit the source of a macro that you have just recorded but have not yet saved. After creating a new recorded macro, you are prompted with the [Save Macro Dialog](#). Instead of naming the macro and saving it, click **Edit** (or press **Alt+E**) to view the source. A new editor window named `lastmac.e`, which is the name of the file that contains the source of the the last macro that was recorded, is opened showing the macro's source code. If you make edits, you will need to save the changes by selecting **Macros > Save last-macro**. The Save Macro dialog is displayed where you can name the macro and then click **Save**, which then appends the new code to the user macros file (`vusrmacros.e`). To bind the macro to a key, use the Key Bindings dialog, which is not automatically displayed like it is when recording/saving a macro in the normal way (see [Binding Macros Using the Key Bindings Dialog](#)).

Each macro recorded and bound using **execute\_last\_macro\_key** is saved in a file named `lastmac<key>.e`, and the corresponding compiled byte code is saved in `lastmac<key>.ex`, where `<key>` matches the **key** you used when creating the binding (keys **0-9**, **A-Z**, or **F1-F12**). Both files are located in

your configuration directory. To edit a macro bound using this method, open the .e file for the macro you want to edit, make and save the changes, then from the main menu, click **Macro > Load Module (F12 or gui\_load** command). Find and select the .e file you just edited and click **Open**. The message **Module(s) loaded** appears on the message line, and SlickEdit will now honor the changes you made to the .e file when you use the corresponding key sequence.

## Deleting Recorded Macros

To delete a macro that has been recorded and saved, from the main menu, click **Macro > List Macros** (or use the **list\_macros** command). Select the macro you want to delete, and click **Delete**.

To delete a macro that you recorded and bound to a key sequence using **execute\_last\_macro\_key**, browse to your configuration directory and delete `lastmac<key>.e` and its corresponding `lastmac<key>.ex` file, where `<key>` matches the **key** you used when creating the binding (keys **0-9**, **A-Z**, or **F1-F12**).

## Using Macros to Discover and Control Options

Recording macros provides a good starting point for discovering variables in Slick-C® code that control the behavior of SlickEdit®.

Since responses to dialog boxes (such as when you select/deselect options) are recorded as Slick-C® source, you can use recorded macros to discover and change these variables quickly. For example, perhaps you frequently switch line insert styles. Instead of every time selecting **Tools > Options > General**, then clicking the **More tab**, then selecting the option, you can record those steps as a macro and bind it to a key sequence. Now you have an easy way to toggle a feature on and off.

You can also view the source of a recorded macro without naming or saving it, if you just want to see the code. After recording the macro, click **Edit** on the [Save Macro Dialog](#) to look at the macro's source to discover the other variables for options on that dialog. See [Saving and Editing Recorded Macros](#) for more information.

## Programmable Macros

---

Many of the actions performed using SlickEdit® are performed using Slick-C® macros. Slick-C functions are mapped to menus, buttons, and keys and perform the action behind an event. Use Slick-C to customize, modify, and bind functions to other shortcuts.

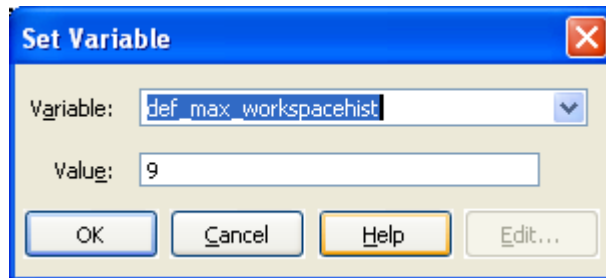
To learn more about macro functions, from the main menu choose **Help > Macro Functions by Category**. This will display the Help dialog, with a list of all macro functions organized into categories.

### Loading Macros

To load a Slick-C® macro file, from the main menu choose **Macro > Load Module**, or use the **gui\_load** command. The Open dialog box is displayed, prompting you for a file.

### Setting Macro Variables

You can set Slick-C® macro variables to specific values using the Set Variable dialog box (**Macro > Set Macro Variable** or **gui\_set\_var** command).

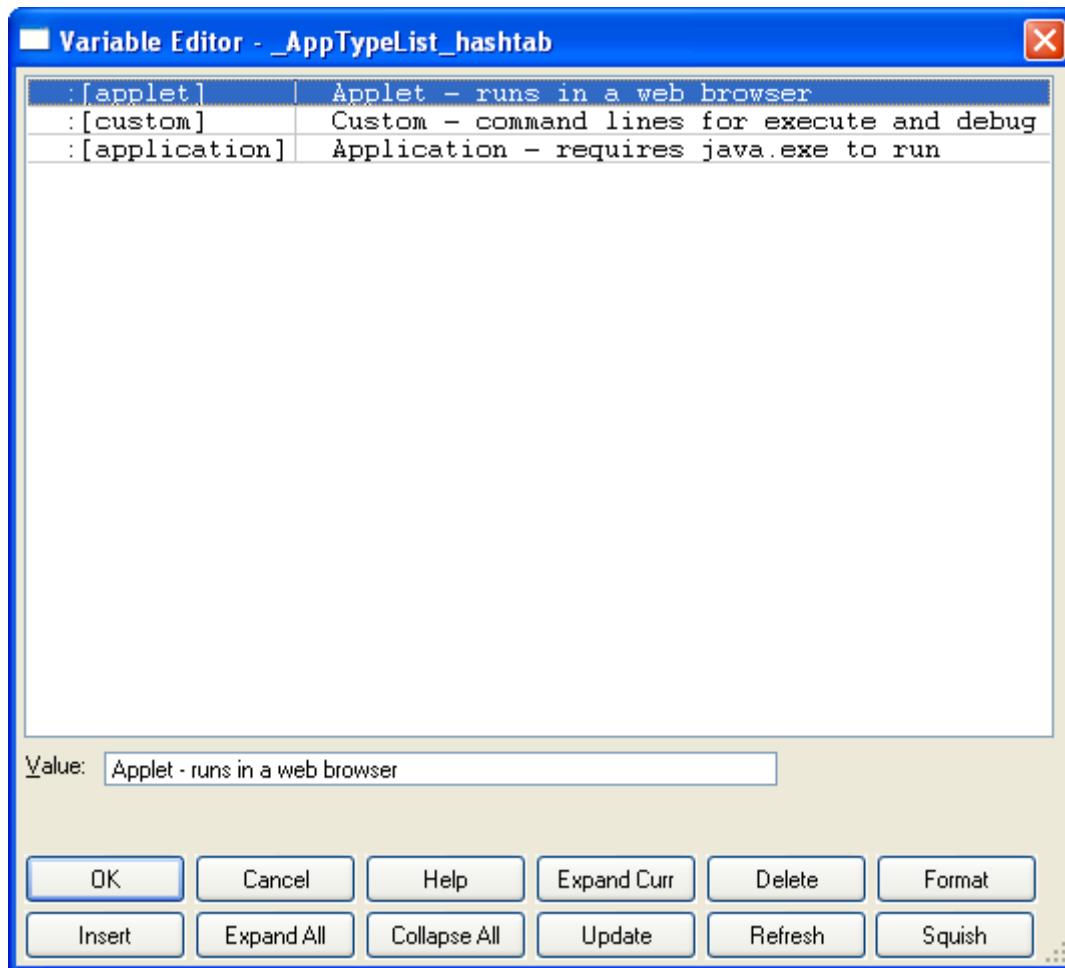


Enter the name of Slick-C global variable in the **Variable** text field. You may use the spacebar and “?” (completion) to assist you in entering the name. Click the drop-down arrow to select a variable from the list.

Enter the new value of the variable in the **Value** text box. Click **Edit** to display the Variable Editor, used for editing complex variables such as arrays, hash tables, structures, and unions.

Currently the Variable Editor does not have enough symbolic information to give you member names of structures or unions. Structures will appear as an array.





The data structure of the variable is displayed in the list box at the top of the dialog, and the value for each entry is displayed in the **Value** text box. For a list of all elements on this dialog, see [Variable Editor Dialog](#).



# Menus, Dialogs, and Tool Windows

This chapter contains the following topics:

- [File](#)
- [Edit](#)
- [Search](#)
- [View](#)
- [Project](#)
- [Build](#)
- [Debug](#)
- [Document](#)
- [Macro](#)
- [Tools](#)
- [Options](#)
- [Window](#)
- [Help](#)
- [Menu Editing](#)



# File

This section describes items on the **File** menu and associated dialogs and tool windows. See the chapter [Workspaces, Projects, and Files](#) for more details about file operations.

## File Menu

The **File** menu items are summarized in the table below.

File Menu Item	Description	Command
New	Displays the New dialog, which allows you to create an empty file to edit. This dialog also lets you create new projects and workspaces. See <a href="#">New Dialog</a> .	<b>new</b>
New Item from Template	Displays the Add New Item dialog, which allows you to create a new file from a template. See <a href="#">Code Templates</a> .	<b>add_item</b>
Open	Displays the Open dialog, which allows you to open a file for editing. See <a href="#">Open Dialog - Windows</a> and <a href="#">Open Dialog - Linux, UNIX, and Mac</a> .	<b>gui_open</b>
Open URL	Displays the Open URL dialog, allowing you to open an HTTP file. See <a href="#">Open URL Dialog</a> .	<b>open_url</b>
Close	Closes the current file.	<b>quit</b>
Close All	Closes all files.	<b>close_all</b>
Save	Saves the current file.	<b>save</b>
Save As	Displays the Save As dialog, which allows you to save the current file under a different name. See <a href="#">Save As Dialog</a> .	<b>gui_save_as</b>
Save All	Saves all modified files.	<b>save_all</b>
Revert	Revert current file to version on disk.	<b>revert</b>
Change Directory	Displays the Change Directory dialog, which lets you change the current working directory. See <a href="#">Change Directory Dialog</a> .	<b>gui_cd</b>
Backup History for	Displays the Backup History tool window, where previous versions are listed. Diffs can be run from here. See <a href="#">Using Backup History</a> .	<b>activate_deltasave</b>
FTP	Displays menu of FTP commands. See <a href="#">File FTP Menu</a> .	N/A

File Menu Item	Description	Command
Print	Displays the Print dialog (or Text Mode Print dialog for UNIX and Mac), which contains options to print the current file or selection and provides setup options. See <a href="#">Print Dialog - Windows</a> and <a href="#">Print Dialog - Linux, UNIX, and Mac</a> .	<b>gui_print</b>
Insert a File	Displays the Insert File dialog, which lets you insert a selected file at the cursor location. See <a href="#">Inserting Files into Buffers</a> .	<b>gui_insert_file</b>
Write Selection	Displays the Write Selection dialog, which lets you write or append selected text to a file you choose. See <a href="#">Creating a File from a Selection</a> .	<b>gui_write_selection</b>
Template Manager	Create, edit, and delete your templates. See <a href="#">Code Templates</a> .	<b>template_manager</b>
Export to HTML	Write file to HTML format.	<b>export_html</b>
File Manager	Displays menu of file manager commands. See <a href="#">File Manager Menu</a> .	N/A
Exit	Prompts you to save files if necessary and exits the editor. See <a href="#">Exiting the Program</a> .	<b>safe_exit</b>

## File FTP Menu

The **File > FTP** menu, summarized in the table below, is available for performing FTP operations and changing FTP options. See [FTP](#) for more information about working with FTP features.

FTP Menu Item	Description	Command
Start New Connection	Activates FTP tool window and starts a new connection.	<b>ftpOpen 1</b>
Activate FTP	Activates FTP tool window.	<b>activate_ftp</b>
Upload	Uploads the current FTP file.	<b>ftpUpload</b>
Client	Activates FTP Client toolbar.	<b>ftpClient</b>
Profile Manager	Display FTP Profile Manager dialog box. See <a href="#">Add/Edit FTP Profile Dialog</a> .	<b>ftpProfileManager</b>
Options	Displays the FTP Options dialog box. See <a href="#">FTP Options Dialog</a> .	<b>show_ftpOptions_form</b>

## File Manager Menu

The **File > File Manager** menu, summarized in the table below, contains options for working with the SlickEdit® File Manager, which offers a rich set of file listing, selecting, and operating capabilities. See [Using the File Manager](#) for detailed information about this feature.

File Manager Menu Item	Description	Command
New File List	Displays a directory of files you choose.	<b>fileman</b>
Append Files	Appends files to current list.	<b>fileman append</b>
Sort	Sorts file list.	<b>fsort</b>
Backup	Copies selected files and preserves directory structure.	<b>fileman_backup</b>
Copy	Copies selected files to a directory you choose.	<b>fileman_copy</b>
Move	Moves selected files to a directory you choose.	<b>fileman_move</b>
Delete	Delete selected files.	<b>fileman_delete</b>
Edit	Edits selected files.	<b>fileman_edit</b>
Select	Displays menu of file manager select commands. See <a href="#">File Manager Select Menu</a> .	N/A
Files	Displays menu of file manager listing commands. See <a href="#">File Manager Files Menu</a> .	N/A
Attribute	Sets the Read Only, Hidden, System, and Archive attributes of the selected files.	<b>fileman_attr</b>
Repeat Command	Runs internal or external command on selected files.	<b>for_select</b>
Global Replace	Performs search and replace on selected files.	<b>fileman_replace</b>
Global Find	Performs search on selected files.	<b>fileman_find</b>

## File Manager Select Menu

The **File > File Manager > Select** menu contains File Manager selection operations.

File Manager Select Menu Item	Description	Command
All	Selects all files.	<b>fileman_select_all</b>
Deselect All	Deselects all files.	<b>deselect_all</b>
Invert Select	Selects files which are not selected and deselects files which are selected.	<b>select_reverse</b>
Attribute	Selects files based on file attribute.	<b>select_attr</b>
Extension	Selects files based on file extension.	<b>gui_select_ext</b>
Highlight	Selects files which are highlighted.	<b>select_mark</b>
Deselect Highlight	Deselects files which are highlighted.	<b>deselect_mark</b>

## File Manager Files Menu

The **File > File Manager > Files** menu contains operations for working with files in the File Manager. Note that commands in this menu do NOT delete files on disk.

File Manager Files Menu Item	Description	Command
Unlist All	Removes all files from the list.	<b>unlist_all</b>
Unlist Select	Removes selected files from the list.	<b>unlist_select</b>
Unlist Extension	Removes files with a specific extension from the list.	<b>gui_unlist_ext</b>
Unlist Attribute	Removes files with a specific attribute from the list.	<b>unlist_attr</b>
Unlist Search	Removes lines which contain a particular search string.	<b>unlist_search</b>
Read List	Appends a list of files contained in a file.	<b>read_list</b>
Write List	Writes a file containing the currently selected files.	<b>write_list</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with **File** menu items.

### New Dialog

The New dialog (**File > New**) is used to create new files, projects, and workspaces. See also [Creating Files](#) and [Creating Projects](#).

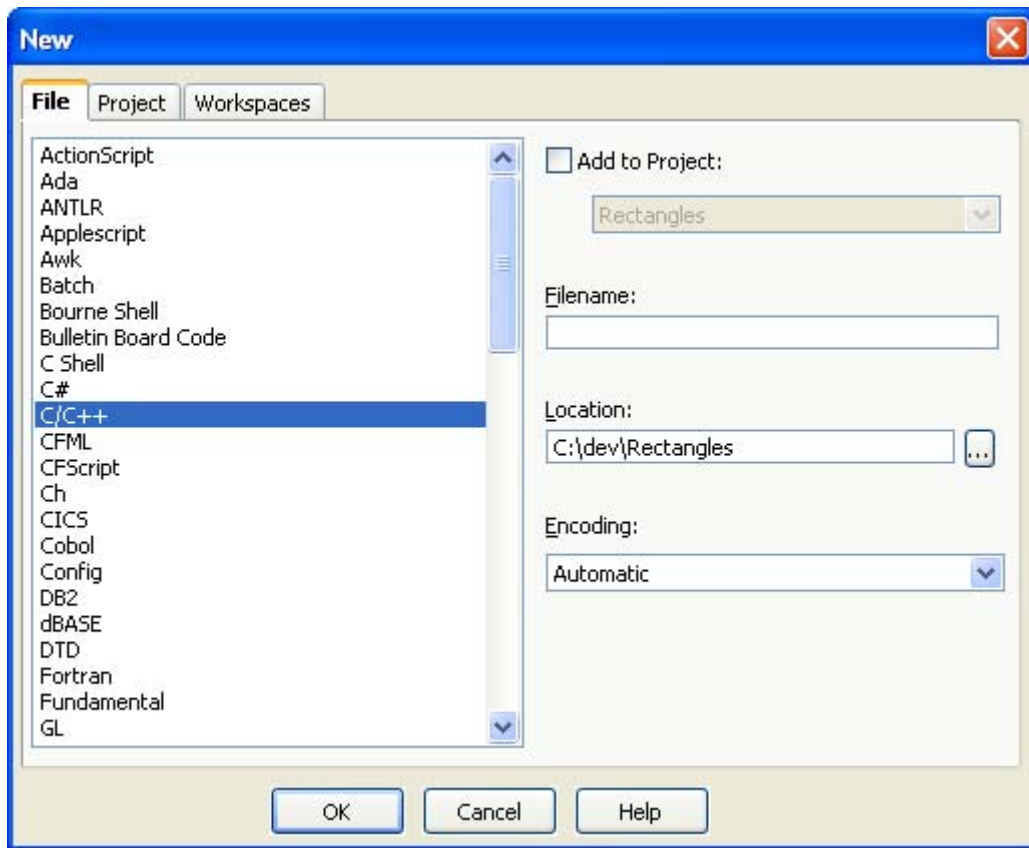
The options on this dialog are categorized into tabs:

- [File Tab](#)
- [Project Tab](#)
- [Workspaces Tab](#)

### File Tab

The File tab of the New dialog, shown below, allows you to create a new file.





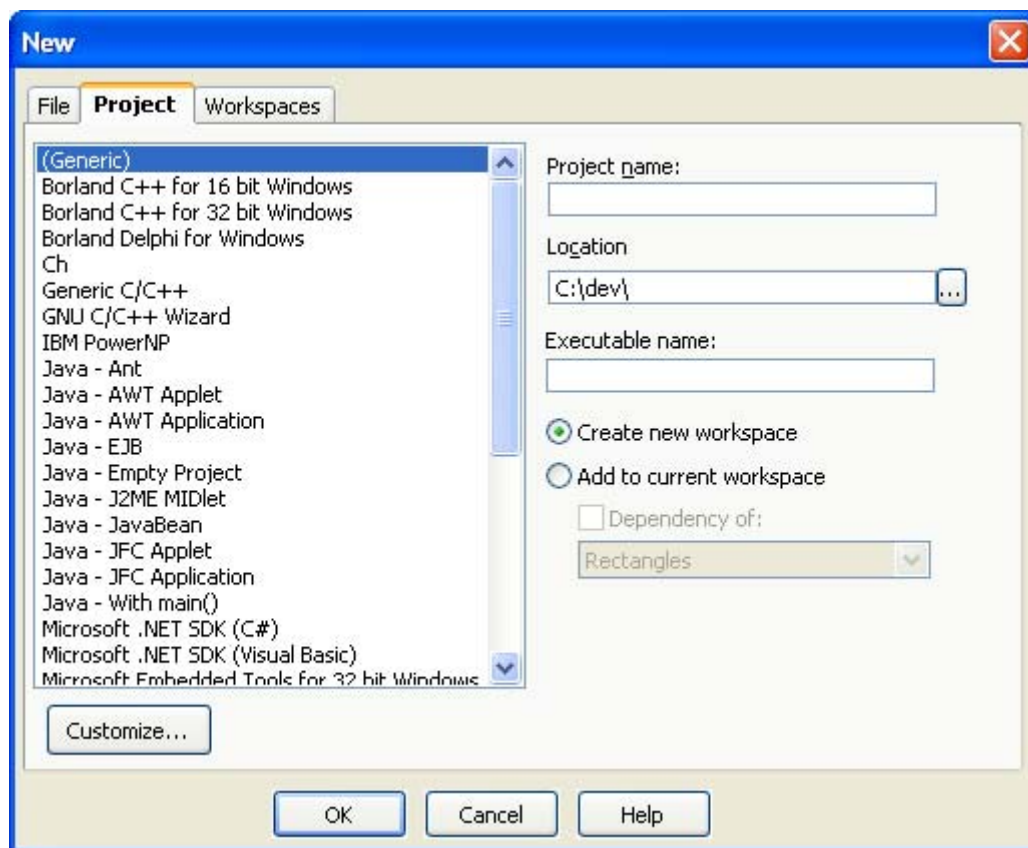
The following options are available:

- **Language Mode** - The language modes are listed on the left side of the dialog. Double-click on a language mode to instantly create an empty, untitled buffer set to that mode.
- **Add to Project** - If selected, the new file is added to the active project.
- **Filename** - Name (without path) of the file to create.
- **Location** - Specifies the directory location for the file.
- **Encoding** - The encoding specifies whether to convert a file to either SBCS/DBCS for the active code page or Unicode (more specifically UTF-8) data. By default, XML and Unicode files with signatures (UTF-16, UTF-32, and UTF-8) files are automatically loaded as Unicode UTF-8 data, while other more common program source files like .c, .java, and .cs source files are loaded as SBCS/DBCS active code page data.

To provide better support for editing Unicode and non-Unicode files, two modes of editing exist: Unicode and SBCS/DBCS mode. Files that contain Unicode, XML, or code page data not compatible with the active code page should be opened as Unicode files. See [Encoding](#) for more information working with encodings.

## Project Tab

The **Project** tab on the New dialog, shown below, allows you to create a new project. See [Creating Projects](#) for more information.

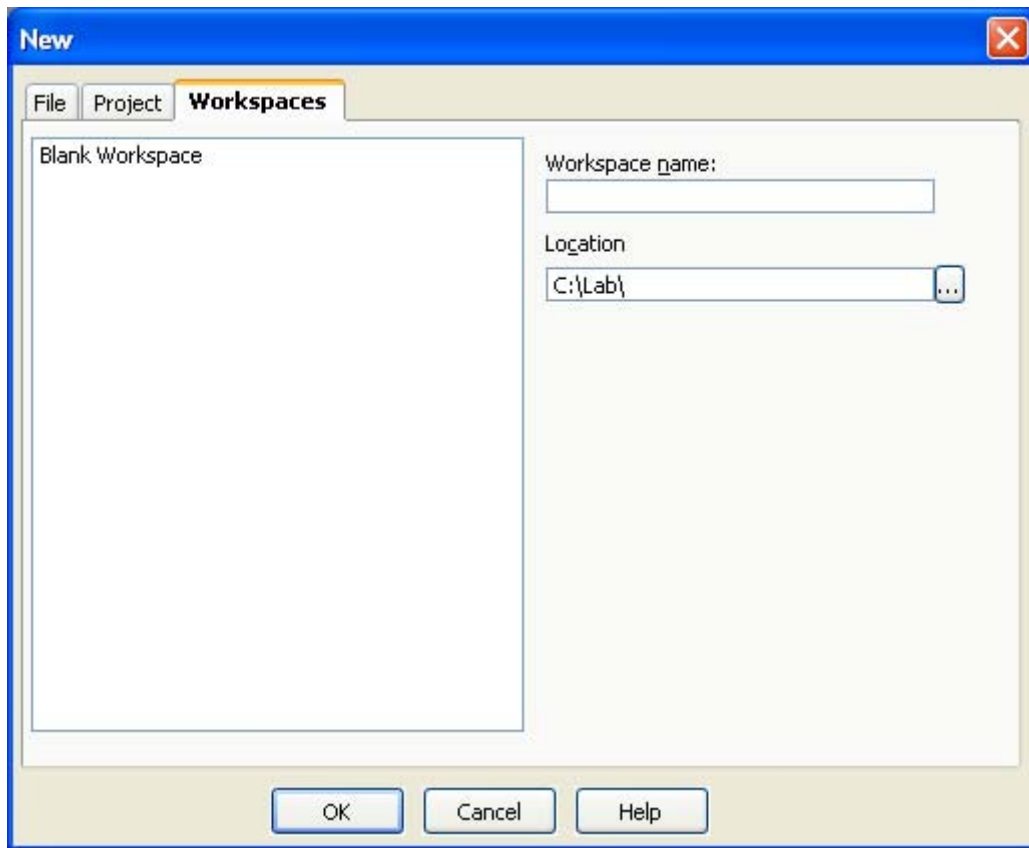


The following options are available:

- **Package List** - The project package types are listed on the left side of the dialog.
- **Customize** - Displays the Customize Packages dialog, which allows you to create, edit, and delete project packages from the list. System-defined packages are stored in `prjtemplates.vpt`. User-modified or additional packages are stored in `usrprjtemplates.vpt`.
- **Project name** - Specifies the name of the project.
- **Location** - Specifies the directory that the project should be created in. If the directory does not exist, you will be prompted to create it after you click **OK**.
- **Executable name** - Specifies the name of the executable or output file.
- **Create new workspace** - If selected, the project will be created in a new workspace.
- **Add to current workspace** - If selected, the project will be added to the current workspace.
- **Dependency of** - When adding the project to the current workspace, if **Dependency of** is selected, you can specify on what existing project the new project should depend. See also [Defining Project Dependencies](#).

## Workspaces Tab

The **Workspaces** tab of the New dialog, shown below, allows you to create a new workspace. See also [Creating Workspaces](#).



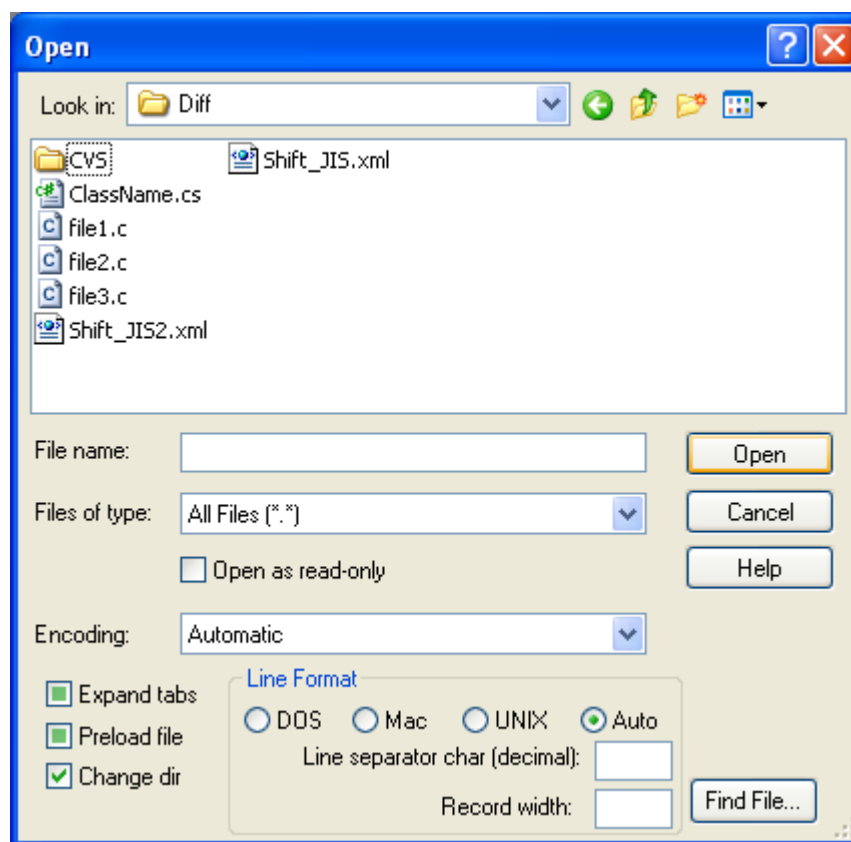
The following options are available:

- **Workspace name** - Specifies the name of the new workspace.
- **Location** - Specifies the directory location of the new workspace. If the directory does not exist, you will be prompted to create it when you click **OK**.

Choose **Blank Workspace** if you want to create an empty workspace to which you can add projects. The data for each workspace, solution, or project is stored in a text file with the `.vpw` extension.

## Open Dialog - Windows

The default Open dialog for Windows (**File > New** or `gui_open` command), shown below, is used for opening files.



The following options are available:

- **Look in** - Specify the directory for which to display files. Select multiple files with Ctrl+Click. Press Ctrl+A to select all files.
- **File name** - The name of the file you wish to edit is typed into the **File name** text box. Click the **OK** button to open the selected files. This text box supports alias expansion. Type the alias name and press Ctrl+Space to expand an alias. Aliases save time typing in long path names and wasting time clicking through the **Directories** list box. See [Directory Aliases](#) for more information.
- **Files of type** - This combo box lets you display files of particular extensions. Select a different file filter from this combo's list box to change the file list.
- **Files of type** - You can change the file specifications that appear in the **Files of Type** combo box at the bottom left of the Open and Save As dialog boxes. From the main menu, choose **Tools > Options > File Options**, then select the [File Filters Tab](#). The first wildcard pattern(s) specified in the File Filters tab is used to initialize the Open dialog box. The default is to list all files. You may want an initially smaller list which displays the source files you typically edit.
- **Open as read-only** - When opening a file, check this box if you wish to open a file but do not want to accidentally modify the file. Files are automatically opened with the read only attribute set as read only, regardless of the settings of this check box.
- **Encoding** - The encoding specifies whether to convert a file to either SBCS/DBCS for the active code page or Unicode (more specifically UTF-8) data. By default, XML and Unicode files with signatures (UTF-16, UTF-32, and UTF-8) files are automatically loaded as Unicode UTF-8 data, while other more common program source files like .c, .java, and .cs source files are loaded as SBCS/DBCS active code page data.

To provide better support for editing Unicode and non-Unicode files, two modes of editing exist: Unicode and SBCS/DBCS mode. Files that contain Unicode, XML, or code page data not compatible with the active code page should be opened as Unicode files. See [Encoding](#) for more information on the available encodings.

- **Expand tabs** - When this option is checked, tabs found in opened files are expanded to spaces in increments of eight.
- **Preload file** - This check box is used to force the entire contents of opened files to be read into memory (may spill to disk), count the number of lines in the file, and truncate the file when an EOF character is found.
- **Change dir** - (Not always present) Check this box if you want the current directory to be changed to the path where this file is located. To set the default value of this check box, from the main menu choose **Tools > Options > General**, select the [General Tab](#), and change the **Change directory** option.
- **Line format** - This option specifies the type of line ending to be used. Select DOS, UNIX, or MAC line endings, or select **Current line format** to use the line endings already in the current file.
- **Line separator char (decimal)** - This text box allows you to specify a decimal character which the editor will use as a single line separator character.
- **Record width** - This text box allows you to specify a decimal line width. Use this option to open ASCII record files or binary files.
- **Find File** - This button displays the Find File dialog box used to scan subdirectories for wildcard file names. Files found may be opened. See [Find File Dialog](#).

## Open Dialog - Linux, UNIX, and Mac

The default Open dialog for Linux, UNIX, and Mac (**File > New** or **gui\_open** command), is used for opening files. This is the same dialog that is also called "Windows 3.1 Style" Open dialog, which Windows users can select to use by choosing **Tools > Options > General**, selecting the [More Tab](#), then selecting the option **Windows 3.1 style open dialog**. This dialog is generally faster than the default Windows dialog.

This dialog contains the same options as the Windows dialog (see [Open Dialog - Windows](#)), with the following additions/differences:

- **File name** - The name of the file you wish to edit is typed into the **File name** combo box. Press the **OK** button to open the file. Use the **Open** button to open several files at once with a wildcard file specification such as `*.c`.

The syntax for specifying a sequential data set is `//dataset`. The syntax for specifying a partitioned data set member is `//dataset/member`. For example, type `//SYS1.PARMLIB/BPXPRM00` into the File name combo box and press Enter to edit the BPXPRM00 member of SYS1.PARMLIB.

The following characters have special meaning when entered as the last part of a file name:

Character	Meaning
*	Matches 0 or more characters.
?	Matches any character.
[ <i>char-set</i> ]	Matches any one of the characters specified by <i>char-set</i> . A "-" character may be used to specify ASCII (not EBCDIC) ranges. The expression [A-Z] matches any upper case letter. "\" may be used inside the square brackets to define literal characters or ASCII characters. For example, "\-" specifies a literal dash character.

File name wildcard examples:

Example	Meaning
//sys1.*	Match data sets starting with "sys1".
//p*a.b.*	Match data sets starting with "p" that have "a.b." in the middle.
//sys1.maclib/a*	Match all members that start with the letter "a".
//dataset/vs*xy	Match all members starting with "vs" and ending with "xy".
//dataset/a?c	Match member names that start with "a", end with "c", and are exactly three characters long. For example, this matches "abc" or "axc".
//dataset/a[a-c]c	Matches members names "aab", "abc", and "acc".

Like many combo boxes in SlickEdit®, the File name combo box has the following powerful features:

- **Combo box retrieval** - Press the Down arrow to retrieve the previous file name you entered here. The combo box list (Alt+Down arrow) displays all the previous file names entered. This retrieval information is saved across edit sessions.
- **Completion** - Press the spacebar to automatically type the rest or more of the partial file name for you. For example, type `c:\aut` and press Space. SlickEdit will attempt to expand this to `c:\autoexec.bat` (or to `c:\autoexec.` if you have a file like `c:\autoexec.bak`). Press `?` to list possible matches for what you have typed so far. For example, if you type `"a"` and press `?`, a list will be displayed of all files which start with the letter `"a"`. See [Completions](#) for more information.
- **Alias expansion** - Type the alias name and press Ctrl+Space to expand an alias. Aliases save you from typing long path names and wasting time clicking through the Directories list box. Many keys from your emulation are supported here such as "copy word to clipboard" (Ctrl+K), "cut line" (Ctrl+Backspace), and "cut word" (Ctrl+Shift+K). See [Directory Aliases](#) for more information.
- **Multiple clipboards** - Press Ctrl+Shift+V to view a list of all your clipboards.

**NOTE (UNIX)** Embed `~`, `~login name`, or `$envvar` in the File Name specification in the same way as at a UNIX shell prompt.

- **File list** - The **File List** is located below the **File Name** combo box. Click in this list box to change the file name displayed in the combo box. Multiple files may be selected with Ctrl+Click.
- **List files of type** - This combo box lets you display files of particular extensions. Select a different file filter from this combo's list box to change the file list.

**TIP** You can change the file specifications that appear in the **List Files of Type** combo box at the bottom left of the Open and Save As dialog boxes. From the main menu, choose **Tools > Options > File Options**, then select the [File Filters Tab](#). The first wildcard pattern(s) specified in the File Filters tab is used to initialize the Open dialog box. The default is to list all files. You may want an initially smaller list which displays the source files you typically edit.

- **Directories** - Double-click on files in this list box to change the directory. The File List is updated.
- **Read only** - When opening a file, check this box if you wish to open a file but do not want to accidentally modify the file. SlickEdit will automatically open files with the read only attribute set (UNIX:

read only permissions) as read only, regardless of the settings of this check box. This check box is also present when saving a file. It allows you to save a read-only file as read-only.

- **Show hidden files** - Dot files (files with names beginning with a dot character) are hidden in the Open dialog by default. To view dot files, select this option. The default state of this option is controlled by the option **Show files beginning with a dot** on the [General Tab](#) of the General Options dialog (**Tools > Options > General**). **Show files beginning with a dot** can be also set using the `def_filelist_show_dotfiles` configuration variable.
- **Open** - To open a wildcard file specification such as \*.c, type \*.c in the **File Name** text box, and click on the **Open** button.
- **Invert** - Click on the **Invert** button to select all files in the File List that are currently not selected and deselect all files which are selected. The **Invert** button appears on other dialog boxes which have list boxes which support selection of multiple files with Ctrl+Click. Since there are no files selected when the Open dialog box initially appears, you can click on the **Invert** button followed by the **OK** button to open all files displayed in the File List.
- **Advanced** - Expands the Open dialog box to display additional less-often used options.

You will notice that several check boxes initially appear grayed. A grayed check box indicates that a default action will be performed. SlickEdit allows many load options to be based on the drive from which the file you open resides. For more information, see [Load Tab](#).

- **File locking** - When on, ensures that a file handle is kept open to the file for locking purposes. This enables SlickEdit to detect when another user is editing the same file.
- **Binary** - When on, makes it safe to edit binary files by overriding some file load and save options. If you are using the default load and save options, you do not need to turn on this check box before editing a binary file.
- **New window** - Select this option to cause the file to be opened in a new editor window.
- **File format** - SlickEdit automatically detects DOS, Macintosh, and UNIX file formats by checking the first 8096 bytes of the file. If this logic is not working for a particular file you have, force a file in a specific format. Select the DOS, Mac, or UNIX radio button.

## Find File Dialog

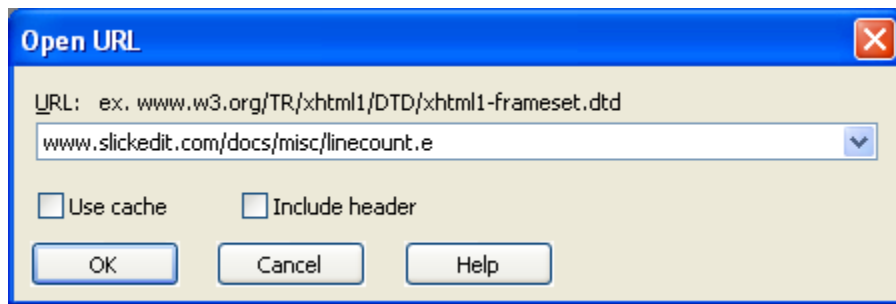
The Find File dialog is used to find a file to open. Choose **File > Open**, then click the **Find File** button. The following options are available:

- **File pattern** - File or file pattern to search for. Subdirectories are always scanned. Press the Search button to start the search.
- **Search** - Starts searching for the file pattern specified.
- **Files found** - List the files that match the last search. Use Ctrl+Click to select more than one file.
- **Read only** - When checked, the selected files are opened as read-only.
- **Open** - Opens the selected files in the Files Found list box.
- **Advanced** - These are the same options found on the Open dialog box. See [Open Dialog - Windows](#) or [Open Dialog - Linux, UNIX, and Mac](#).

## Open URL Dialog

The Open URL dialog (**File > Open URL**), shown below, allows you to specify an HTTP file to open.



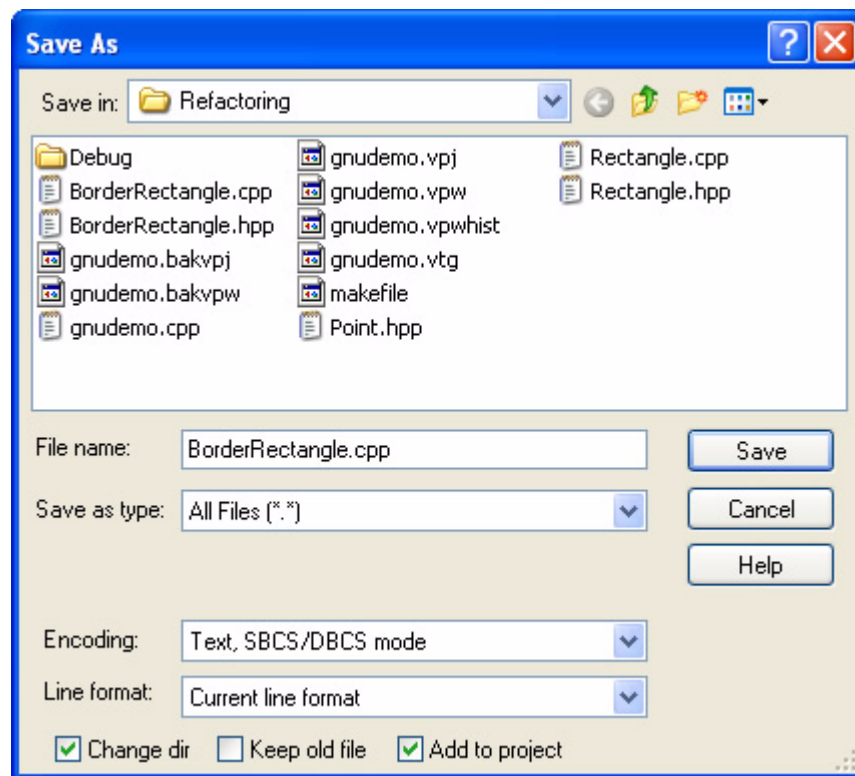


The following options are available:

- **URL** - Enter a URL to open in this field. You may use forward or backward slashes. The prefix "http://" is not required.
- **Use cache** - If enabled and the URL being opened already exists in the cache, the cached version is opened. This option is intended to save time required to download remote files. If the URL specified is already open, this option has no effect.
- **Include header** - If enabled, all of the URL header data is displayed. If the URL specified is already open, this option has no effect.

## Save As Dialog

The Save As dialog (**File > Save As**), shown below, is used to save the current file under a different name.



The following options are available:



- **Save as type** - This option filters the list of files displayed based on file extension. Changing this value will not change the extension of the file, unless the extension is not specified in the File name field.
- **Encoding** - The encoding specifies whether to convert a file to either SBCS/DBCS for the active code page or Unicode (more specifically UTF-8) data. By default, XML and Unicode files with signatures (UTF-16, UTF-32, and UTF-8) files are automatically loaded as Unicode UTF-8 data, while other more common program source files like `.c`, `.java`, and `.cs` source files are loaded as SBCS/DBCS active code page data. See [Encoding](#) for more information.
- **Line format** - This option specifies the type of line ending to be used. Select DOS, UNIX, or MAC line endings, or select **Current line format** to use the line endings already in the current file.
- **Change dir** - Check this box if you want the current directory to be changed to the path where this file is saved. To set the default value of this check box, from the main menu choose **Tools > Options > General**, select the [General Tab](#), and change the **Change directory** option.
- **Show hidden files** - (UNIX only) Dot files (files with names beginning with a dot character) are hidden in the Save As dialog by default. To view dot files, select this option. The default state of this option is controlled by the option **Show files beginning with a dot** on the [General Tab](#) of the General Options dialog (**Tools > Options > General**). **Show files beginning with a dot** can be also set using the `def_filelist_show_dotfiles` configuration variable.
- **Keep old file** - When checked, the file is saved under the name you specify but the buffer name is not changed. This check box is not always displayed.
- **Add to project** - When checked, the saved file will be added to the project that is currently open in the workspace. If no project is open, this option is disabled. To set the default value of this check box, from the main menu choose **Tools > Options > File Options**, select the [Save Tab](#), and change the option **Add file to project upon Save As**.

**TIP** This option is also available from the command line by using the option letter **+P** with the `save_as` command. For example, `save_as +P /path/to/file.cpp` will add `file.cpp` to the current project. Similarly, `save +P` will save the current file and add it to the current project.

## Save Failed Dialog

The Save Failed dialog appears if a save operation has failed. The following options are available:

- **Save as read only** - (UNIX only) This check box is enabled if your file does not have any write permissions (no “w” letter). Turn this check box on to have SlickEdit® temporarily change the permissions on the file to read/write. The resulting file will not have any write permissions (no “w” letter).
- **Save without creating a backup** - This check box is enabled if SlickEdit was unable to create a backup file when trying to save your file. This can happen when you do not have permissions to create the backup directory or when you are out of disk space. If you are editing files on a network drive, you may not have access rights for creating a backup directory on that drive.
- **Configure local backup directory** - (Non-UNIX only) If you are editing files on a network drive, you may not have access rights for creating a backup directory on that drive. Configuring a local backup directory guarantees that you always have write access. If the directory you specify does not exist, SlickEdit will create one for you.

## Exiting with Modified Buffers Dialog

If files have not been saved or closed upon exit, you will be prompted with this dialog to save or discard any changes. The buffer names in the list box are buffers which have not been saved.

The following options are available:

- **Save All** - Saves all buffers. If a buffer does not have a name, you are prompted to give a file-name.
- **Save Selected** - Saves only the buffers that are selected. Use Ctrl+Click to select more than one buffer.
- **Invert** - Reverses the selection status for all buffers. When no buffers are selected, this selects all buffers.
- **Save None** - Selects to save no buffers. Beware, if you are exiting the editor you will not be given another chance to save your files.

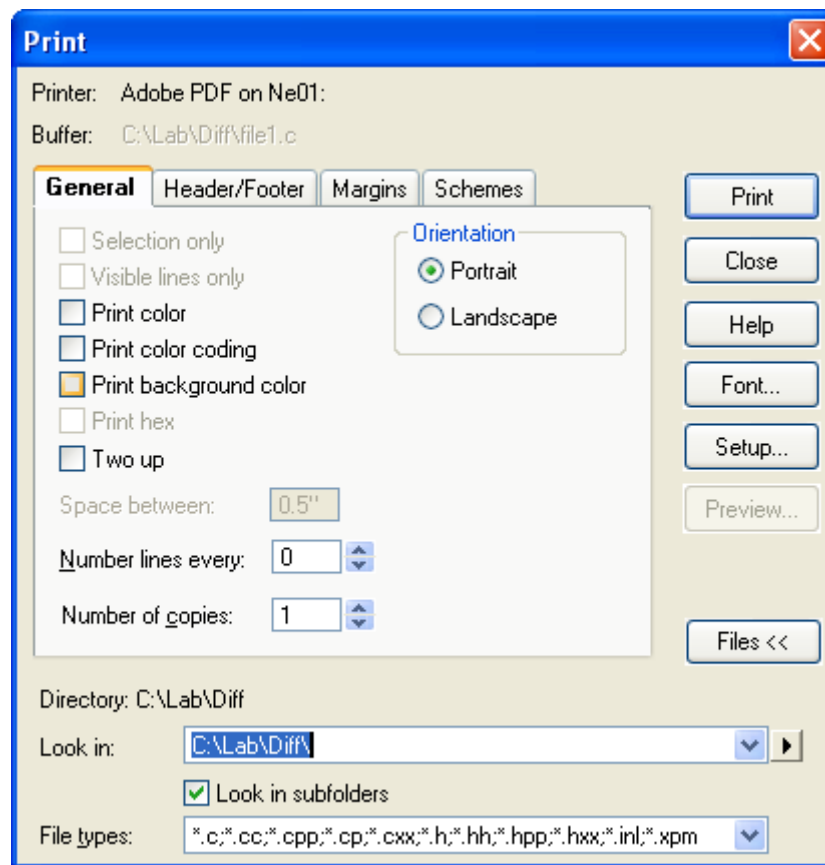
## Change Directory Dialog

The Change Directory dialog (**File > Change Directory**) can be used for changing the current working directory. It contains the following options:

- **Directory name** - Name of directory to change to make active.
- **Expand alias** - Check this box if you want to specify directory aliases in the **Directory name** text box. See [Aliases](#) for more information.
- **Process chdir** - Check this box if you want the current directory in the concurrent process buffer to be changed in addition to the editor's current directory.
- **Save Settings** - Save the settings of the **Expand alias** and **Process chdir** check box values. The next time this dialog box appears, these check boxes are set to the values last saved. In addition, these check box settings affect the **cd** command which is used to change directory using the command line.

## Print Dialog - Windows

The Print dialog (**File > Print** or **gui\_print** command), shown below, allows you to print files and contains options for printer setup. This is the default Print dialog for Windows. See [Printing](#) for more information about working with printing in SlickEdit®.



The Print dialog contains general settings that are described below (see [Print Dialog - General Settings](#)). The remaining options are categorized into the following tabs:

- [General Tab](#)
- [Header/Footer Tab](#)
- [Margins Tab](#)
- [Schemes Tab](#)

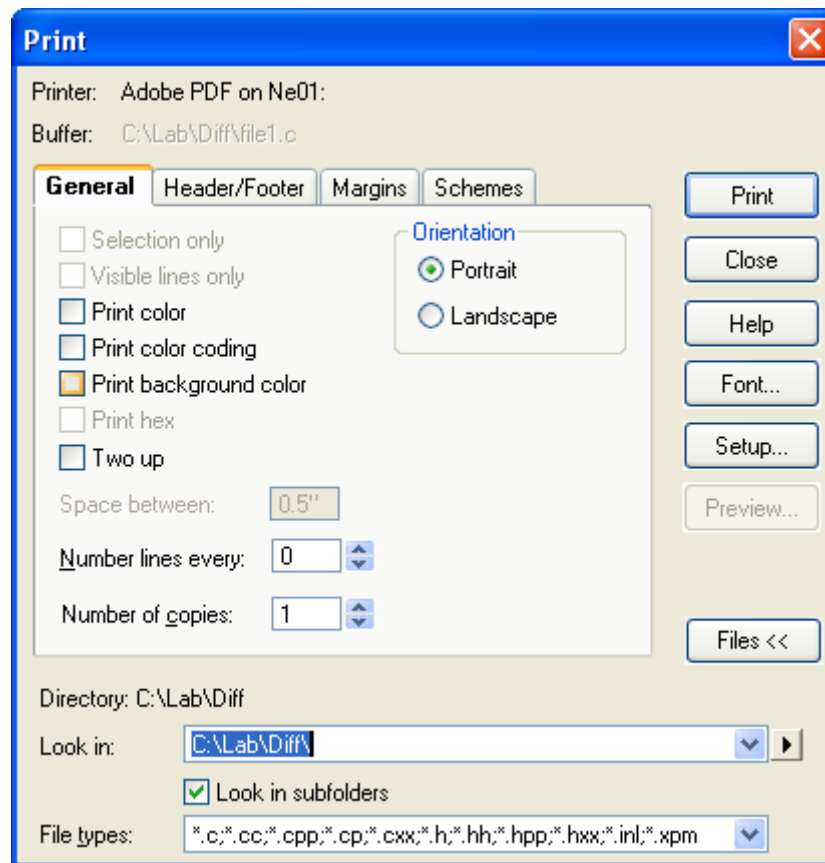
## Print Dialog - General Settings

The following buttons are available on the Print dialog:

- **Print** - Sends the selection or active buffer to the printer specified in the Print Setup dialog (**print** command).
- **Font** - Displays the Font dialog, which allows you to specify the font for the printed text. See [Font Configuration Dialog](#) for more information.
- **Setup** - Displays the Print Setup dialog, which allows you to specify the printer to use (**printer\_setup** command). The Print Setup dialog box is built into the operating system, and its contents might vary depending on the print driver that you are using.
- **Preview** - Displays a preview of what the printed file will look like (**print\_preview** command).
- **Files** - Expands the Print dialog to allow you to pick another file or multiple files for printing (as opposed to the active file).

## General Tab

The General tab on the Print dialog for Windows is shown below.



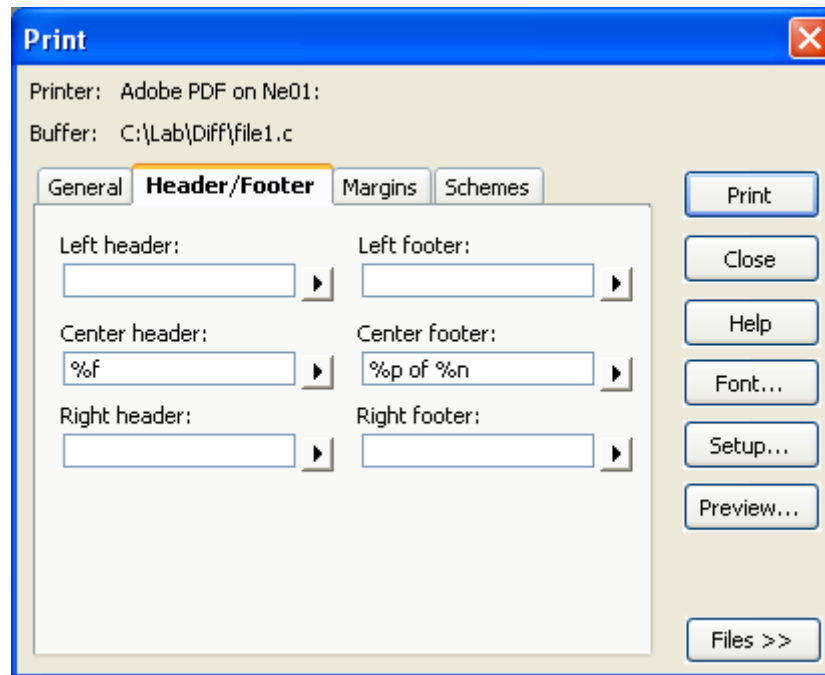
The following options are available:

- **Selection only** - When this option is selected, only the selection is printed. To print the selection immediately using the default configuration, use the **print\_selection** command.
- **Visible lines only** - When this option is selected, only visible lines are printed. This option allows you to print selective display.
- **Print color coding** - When this option is selected, language-specific color coding is printed using the same font attributes (bold, italic, underline).
- **Print color** - When this option is selected, language-specific color coding is printed using the same colors.
- **Print hex** - When this option is selected, printed output is displayed as if it were the hex display mode.
- **Two up** - When this option is selected, it specifies two columns where one page is printed in the left column and the next page is printed in the right column. This is useful when printing in landscape mode, especially when you are using a small font.
- **Orientation** - Specifies whether text is printed top to bottom (portrait) or left to right (landscape).
- **Space between** - This text specifies the width between columns, in inches. This text box is unavailable unless the **Two Up** check box is marked.

- **Number lines every** - When the value is not zero, lines at intervals of this value are printed with line numbers.
- **Number of copies** - This value specifies the number of copies to print.

## Header/Footer Tab

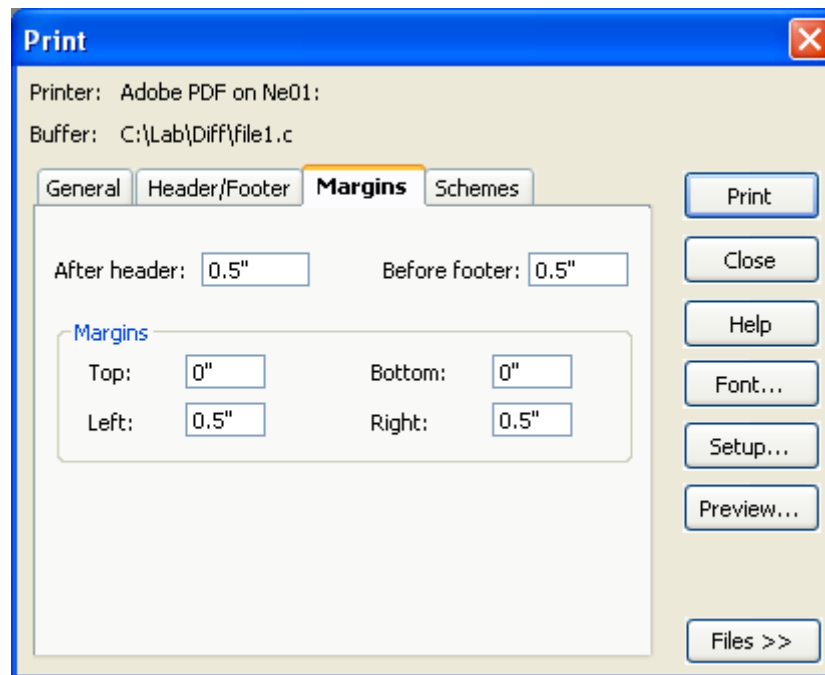
The Header/Footer tab on the Print dialog for Windows is shown below.



Type directly into the text boxes to specify text for the top left, center, and right headers and bottom left, center, and right footers. Click the arrow to the right of each text box pick from a list of escape sequences to be inserted. Escape sequences are values that are replaced with real data, such as **%f** (which will be replaced with the file name) and **%d** (which will be replaced with the date).

## Margins Tab

The Margins tab on the Print dialog for Windows is shown below.



The following options are available:

- **After header/Before footer** - Specifies the amount of spacing to come between the header and the first line on a page, and the amount of spacing to come between the last line on a page and the footer.
- **Margins** - The **Top**, **Bottom**, **Left**, and **Right** margin fields specifies the amount of spacing in inches to come between the outer edge of the paper and the printed text. To print the maximum amount of text, specify "0" for all margins.

## Schemes Tab

The Schemes tab of the Print dialog for Windows allows you to save the current printer settings as a scheme. See [Print Schemes](#) for more information.

## Print Dialog - Linux, UNIX, and Mac

The Text Mode Print dialog (**File > Print** or **gui\_print** command) is used for printing on Linux, UNIX, and Mac systems. See [Printing](#) for more information about working with printing in SlickEdit®.

The Text Mode Print dialog contains the following general settings:

- **OK** - Sends the selection or active buffer to the print device.
- **Save Settings** - Saves the settings that are configured in the Text Mode Print dialog. The next time a print command or this dialog is invoked, the previously saved settings are restored.
- **Files** - Expands the Text Mode Print dialog to allow you to pick another file or multiple files for printing (as opposed to the active file).

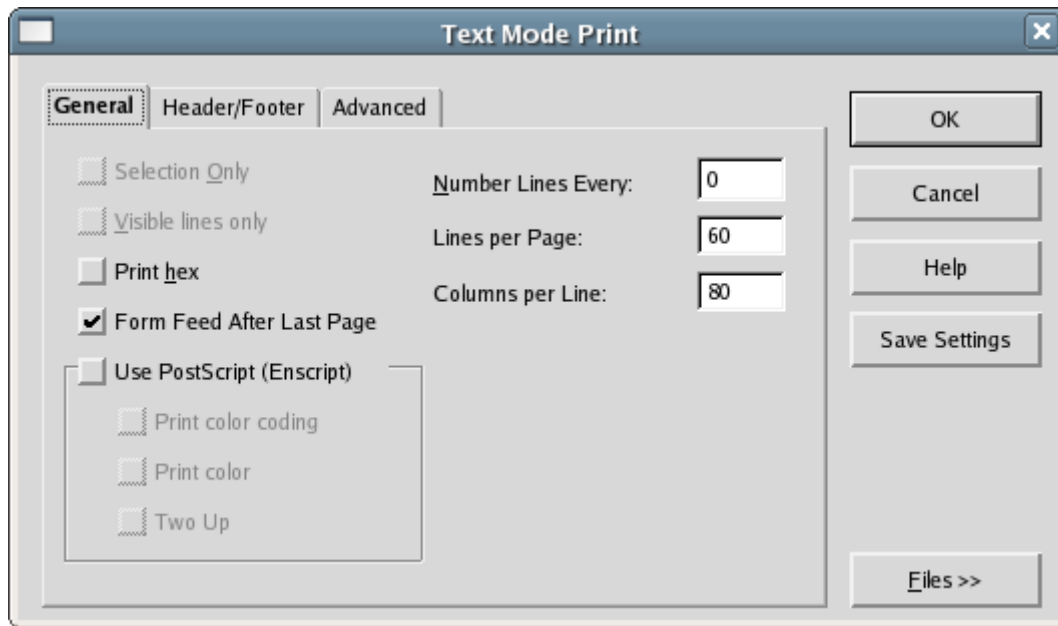
The remaining options are categorized into the following tabs:

- [General Tab](#)

- [Header/Footer Tab](#)
- [Advanced Tab](#)

## General Tab

The General tab of the Text Mode Print dialog is shown below.



The following options are available:

- **Selection Only** - When this option is selected, only the selection is printed. To print the selection immediately using the default configuration, use the **print\_selection** command.
- **Visible lines only** - When this option is selected, only visible lines are printed. This option allows you to print selective display.
- **Print hex** - When this option is selected, printed output is displayed as if it were the hex display mode.
- **Form feed after last page** - Determines whether the built-in print text formatting prints a form feed after the last page.
- **Use PostScript (Enscript)** - When this option is selected, the enscript program is used to print a source file. This allows for print formatting such as fonts, page layout, etc. To specify the options for enscript, see [Advanced Tab](#). Activating this option will make the following settings available:
  - **Print color coding** - When this option is selected, language-specific color coding is printed using the same font attributes (bold, italic, underline).
  - **Print color** - When this option is selected, language-specific color coding is printed using the same colors.
  - **Two Up** - When this option is selected, it specifies two columns where one page is printed in the left column and the next page is printed in the right column.
- **Number Lines Every** - When the value is not zero, lines at intervals of this value are printed with line numbers.

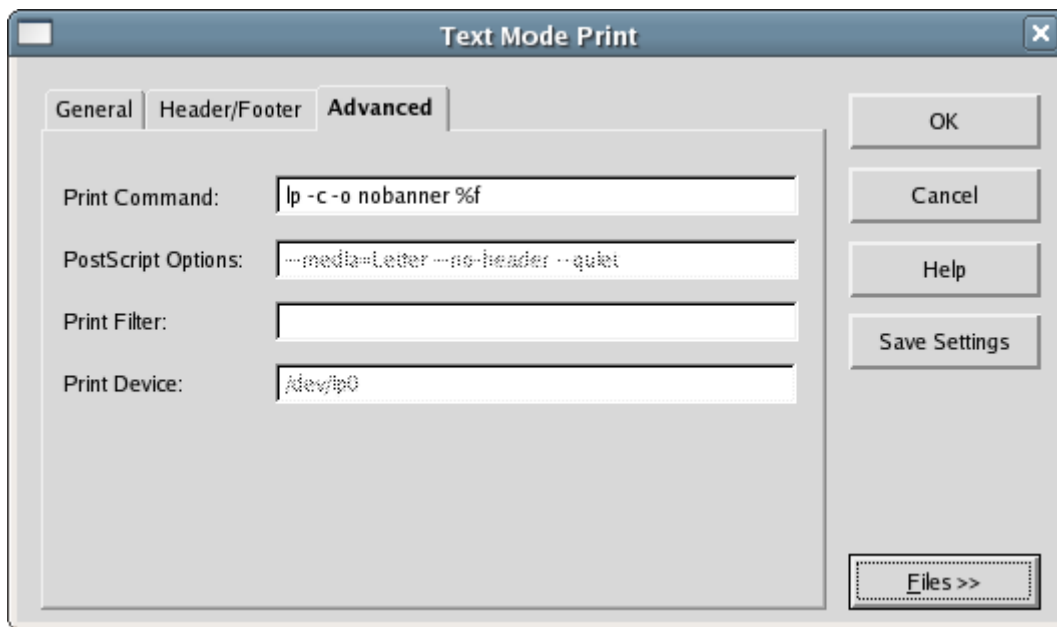
- **Lines per page** - Maximum number of lines that can be printed. This includes header, footer, and blank lines.
- **Columns per line** - Maximum number of columns that can be printed on a line.

## Header/Footer Tab

The Header/Footer tab on the Text Mode Print dialog is the same as the same tab on the Windows Print dialog. See [Header/Footer Tab](#).

## Advanced Tab

The Advanced tab on the Text Mode Print dialog is shown below.



The following options are available:

- **Print command** - Specifies the program that will be executed to print the formatted text. If no print command is specified, the text is written to the Print Device specified. The default print command is **lp -c -o nobanner %f**. The **%f** specifies the file to be printed. If the print program is not found, the command **lpr %f** is used instead.
- **PostScript Options** - To use PostScript options, you must first select the option **Use PostScript (Enscript)** on the General print settings tab. The default options are **--media=Letter --no-header --quiet**. For a list of all available enscript options, see the enscript man page.
- **Print filter** - Specifies an external program to format text before it is printed and overrides the built-in print text formatting. The program specified here must accept input from standard in and writes its output to standard out. A typical print filter command is **pr -1 -w76 -o0 -l50**.
- **Print device** - Specifies a printer device file name to write the formatted text to. This text box is ignored if a **Print Command** has been specified. The default print device is **/dev/lp0**.



## Add/Edit FTP Profile Dialog

The Add/Edit FTP Profile dialog is used to create or edit FTP profiles for FTP connections. To access this dialog, choose **File > FTP > Profile Manager**, then select **New** or **Edit** to edit an existing selected profile. The Add (or Edit) FTP Profile dialog is displayed, with options categorized into two tabs: [General Tab](#) and [Advanced Tab](#).

### General Tab

The General tab of the Add/Edit FTP Profile dialog is pictured below.

The screenshot shows the 'Add FTP Profile' dialog box with the 'General' tab selected. The dialog contains the following fields and options:

- Profile name:** A text input field.
- Host name:** A text input field.
- Server type:** A dropdown menu set to 'FTP'.
- Auth type:** A dropdown menu set to 'Auto'.
- Host type:** A dropdown menu set to 'Auto'.
- User ID:** A text input field.
- Password:** A text input field.
- Anonymous login:** An unchecked checkbox.
- Save password:** A checked checkbox.
- Transfer type:** Radio buttons for 'ASCII' and 'Binary' (selected).
- Initial directory:** Text input fields for 'Remote:' and 'Local:'.
- Filters:** Text input fields for 'Remote:' (containing '\*') and 'Local:' (containing '\*,\*').

Buttons for 'OK', 'Cancel', and 'Help' are located on the right side of the dialog.

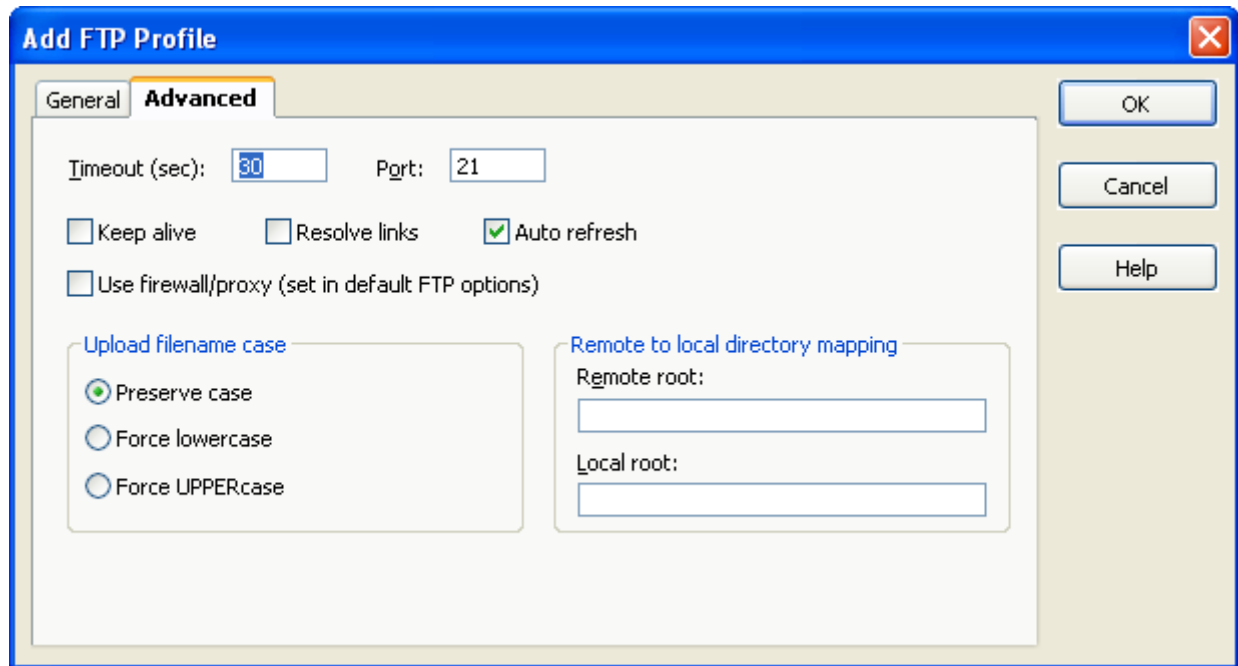
The following options are available:

- **Profile name** - Name displayed in connection combo box.
- **Host name** - Host name of the FTP server.
- **Server type** - If you are connecting to an FTP server, select FTP. If you are connecting to a Secure Shell (SSH) server that supports the SFTP subsystem, select SFTP/SSH.
- **Host type** - If the file listing is blank after connecting, select the host type from the list. Otherwise, leave it set to Auto. If the server type is SFTP/SSH, then this type is ignored.
- **User ID** - Logon user ID.
- **Password** - Logon password.
- **Anonymous login** - Specifies whether the FTP login uses the anonymous user ID.
- **Save password** - Specifies whether to save a password.
- **Transfer type** - Specifies the default file transfer method. Select ASCII if you want line breaks translated. Otherwise, specify binary.
- **Initial directory** - Specifies the initial remote and local directories after login.

- **Filters** - Specifies the initial remote and local file filters after login.

## Advanced Tab

The Advanced tab of the Add/Edit FTP Profile dialog is pictured below. It contains additional parameters that you can set for the FTP features when you are working with SlickEdit®.



The following settings are available:

- **Timeout** - The value that you type in this field specifies the amount of time that the application should wait for a reply from the FTP server.
- **Port** - FTP or SFTP/SSH port. By default, this is 21 for FTP server type, and 22 for SFTP/SSH.
- **Keep alive** - Keeps a connection alive even when idle. Disabled for SFTP/SSH server type.
- **Resolve links** - Resolves symbolic links on the remote host. Disabled for SFTP/SSH server type.
- **Auto refresh** - Determines whether the host directory listing is updated after an operation. Turn this off if the host is slow. Use the context menu (right mouse button) to temporarily turn Auto refresh on/off or force the directory list to be updated.
- **Use firewall/proxy** - This option is not available until a firewall is set up and enabled.
- **Upload filename case** - Indicates what file case should be used for the remote filename based on the local filename.
- **Remote to local directory mapping** - This specifies how a remote path maps to a local path and vice versa. For example, if Remote root is `/usr/ftp/www-slickedit` and local root is `c:\web\slickedit\`, then the remote path `/usr/ftp/www-slickedit/products/index.html` is mapped to the local path `c:\web\slickedit\products\index.html`. This option only affects the FTP tool windows.

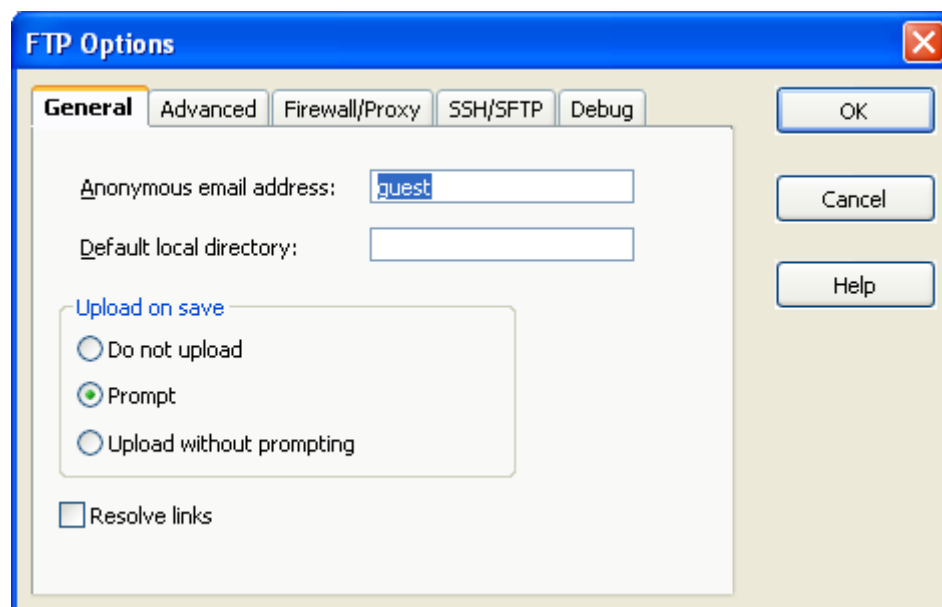
## FTP Options Dialog

The FTP Options dialog (**File > FTP > Options**) is used for setting default FTP options. The options are categorized into the following tabs:

- [General Tab](#)
- [Advanced Tab](#)
- [Firewall/Proxy Tab](#)
- [SSH/SFTP Tab](#)
- [Debug Tab](#)

### General Tab

The General tab, pictured below, contains general settings for the SlickEdit® FTP features.

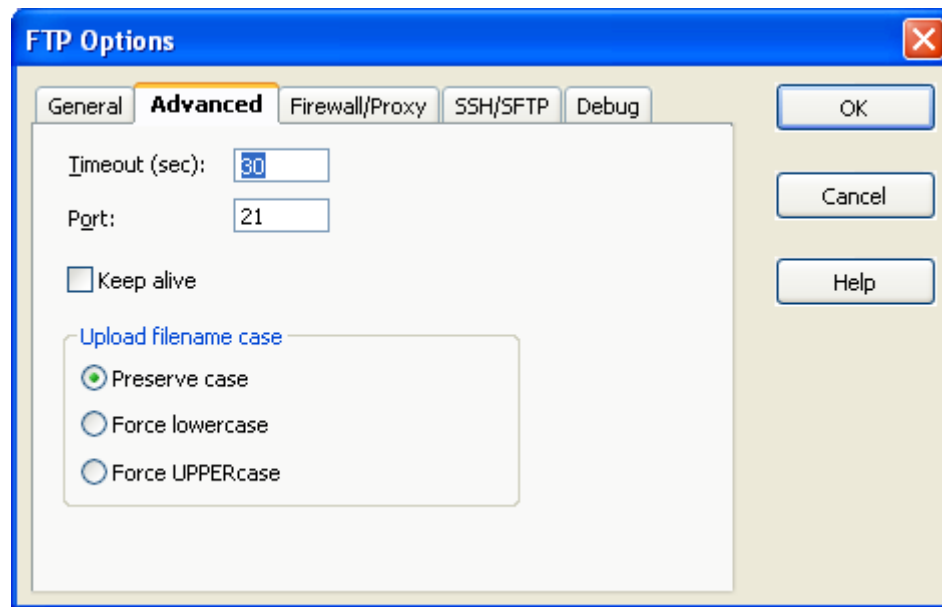


The following options are available:

- **Anonymous e-mail address** - Default password used for anonymous logins.
- **Default local directory** - Default used when adding a new connection profile. Specifies the initial local directory after login. The local directory only applies to the FTP Client toolbar.
- **Do not upload** - When on, saving an FTP file will not upload the file.
- **Prompt** - When on, a prompt appears to upload when an FTP file is saved to specify ASCII or Binary transfer type.
- **Upload without prompting** - When on, saving an FTP file will upload the file. The same transfer type used to open the file is used to upload the file.
- **Resolve links** - Default for adding a new connection profile. Resolves symbolic links on remote host.

### Advanced Tab

The Advanced tab, pictured below, provides the information for setting time parameters and for uploading the case sensitivity.

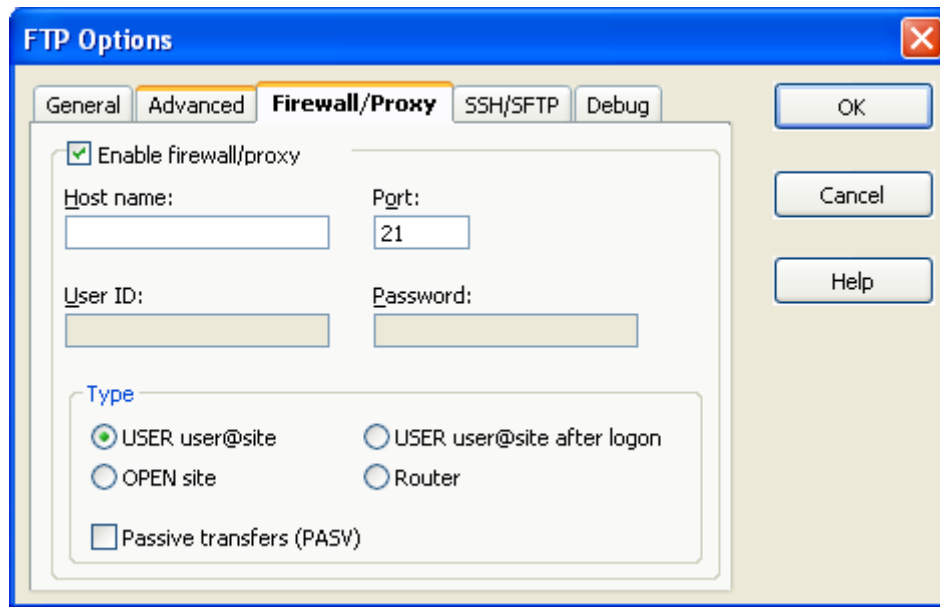


The following options are available:

- **Timeout (sec)** - Default used when adding a new connection profile. Specifies the wait time for a reply from the FTP server.
- **Port** - Default used when adding a new connection profile.
- **FTP port** - By default, this is 21.
- **Keep alive** - Default used when adding a new connection profile. Keeps a connection alive even when idle.
- **Upload filename case** - Default used when adding a new connection profile. Indicates what file case should be used for the remote filename based on the local filename.

## Firewall/Proxy Tab

The Firewall/Proxy tab, pictured below, provides the default host name and port information for working with SlickEdit® and the FTP features.

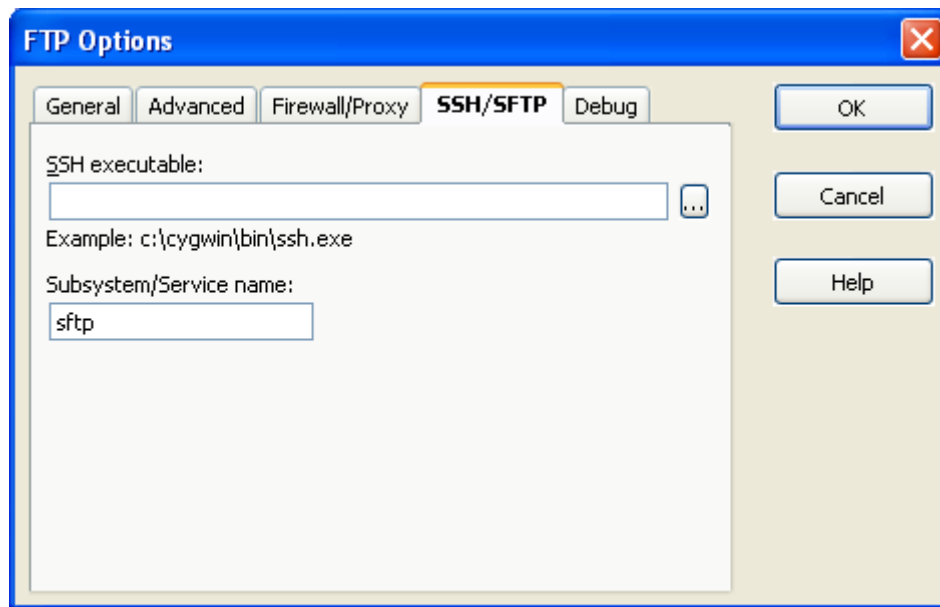


The following options are available:

- **Enable firewall/proxy** - When on, indicates you have a firewall or proxy. You need to turn this on to add a connection profile that uses a firewall.
- **Host name** - Host name of firewall.
- **Port** - Port number of firewall.
- **User ID** - User ID used when logging into firewall.
- **Password** - Password used when logging into firewall.
- **USER user@site** - When this option is selected, host and port are required. User id and password are ignored. USER @remote\_host is sent to the firewall when connecting.
- **USER user@site after logon** - When this option is selected, host, port, user id, and password are required. USER remote\_userid@remote\_host is sent to the firewall after logon.
- **OPEN site** - When this option is selected, host and port are required. User ID and password are ignored. OPEN remote\_host is sent to the firewall when connecting.
- **Router** - When this option is selected, host, port, user id, and password are ignored. Router based firewalls are transparent with the exception that connections can only be established one way (out through the firewall). Because incoming connections are not allowed, PASV is turned on automatically.
- **Passive transfers (PASV)** - When this option is selected, transfers are initiated by SlickEdit.

## SSH/SFTP Tab

The SSH/SFTP tab, pictured below, contains information that indicates the location of the client program that is used to establish the connection with the SSH server.

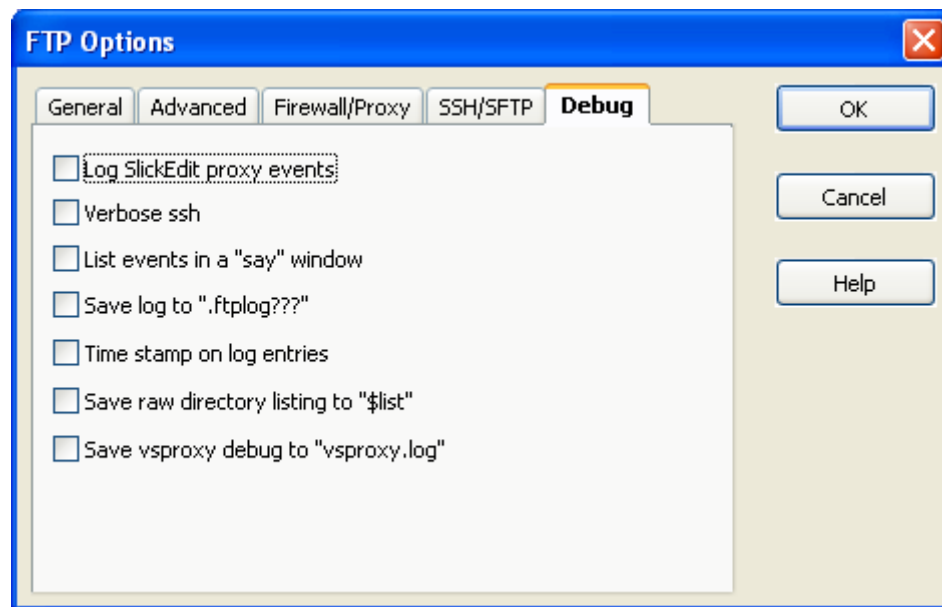


The following options are available:

- **SSH executable** - The location of the ssh client program that is used to establish the secure connection with the SSH server.  
  
SFTP support requires the OpenSSH client program to operate. Windows users can obtain the ssh client by downloading and installing the Cygwin package ([www.cygwin.com](http://www.cygwin.com)) and making sure to choose the **openssh** package during install.
- **Subsystem/Service name** - The name of the SFTP service being run by the SSH server. Defaults to **sftp**.

## Debug Tab

The Debug tab, pictured below, contains options that you can enable during the debug process.







## Edit

This section describes items on the **Edit** menu and associated dialogs and tool windows. See [Text Editing](#) for more details about editing operations.

### Edit Menu

The items on the **Edit** menu are summarized in the table below.

Edit Menu Item	Description	Command
Undo	Undoes the last edit operation.	<b>undo</b>
Redo	Undoes an undo operation.	<b>redo</b>
Multi-file Undo	Undoes the last multi-file operation (i.e.: Find and Replace in all files).	<b>mfundo</b>
Multi-file Redo	Undoes an multi-file undo operation.	<b>mfredo</b>
Cut	Deletes the selected text and copies it to the clipboard.	<b>cut</b>
Copy	Copies the selected text to the clipboard.	<b>copy_to_clipboard</b>
Paste	Inserts the clipboard into the current file.	<b>paste</b>
List Clipboards	Displays the Select Text to Paste dialog, which allows you to view and insert the selected clipboard. See <a href="#">Select Text to Paste Dialog</a> .	<b>list_clipboards</b>
Copy Word	Copies the current word to the clipboard.	<b>copy_word</b>
Append to Clipboard	Appends the selected text to the clipboard.	<b>append_to_clipboard</b>
Append Cut	Deletes the selected text and appends it to the clipboard.	<b>append_cut</b>
Insert Literal	Displays the Insert Literal dialog, which allows you to insert a specified character code. See <a href="#">Inserting Literal Characters</a> .	<b>insert-literal</b>
Select	Displays menu for selecting and deselecting text. See <a href="#">Edit Select Menu</a> .	N/A
Delete	Displays menu for deleting text. See <a href="#">Edit Delete Menu</a> .	N/A
Complete Previous Word	Retrieves previous word or variable matching word prefix at cursor.	<b>complete_prev</b>
Complete Next Word	Retrieves next word or variable matching word prefix at cursor.	<b>complete_next</b>
Fill	Displays the Fill Selection dialog, which lets you fill the selected text with a character you choose.	<b>gui_fill_selection</b>

Edit Menu Item	Description	Command
Indent	Indents the selected text based on the tabs or indent for each level.	<b>indent_selection</b>
Unindent	Unindents the selected text based on the tabs or indent for each level.	<b>unindent_selection</b>
Other	Displays menu containing more edit-related commands. See <a href="#">Edit Other Menu</a> .	N/A

## Edit Select Menu

The **Edit > Select** menu contains selection operations. See [Selections](#) for more information.

Edit Select Menu Item	Description	Command
Char	Starts or ends a character/stream selection.	<b>select_char</b>
Block	Starts or ends a block/column selection.	<b>select_block</b>
Line	Starts or ends a line selection.	<b>select_line</b>
Word	Selects the word under cursor.	<b>select-whole-word</b>
Code Block	Selects text in current code block (if/loop/switch block etc.).	<b>select_code_block</b>
Procedure	Selects text in current function including function heading.	<b>select_proc</b>
Deselect	Unhighlights selected text.	<b>deselect</b>
All	Selects all text in current buffer.	<b>select_all</b>

## Edit Delete Menu

The **Edit > Delete** menu contains text deletion operations. See [Cutting, Copying, and Moving Text](#) for more information.

Delete Menu Item	Description	Command
Word	Deletes text from the cursor to the end of the current word and copies it to the clipboard.	<b>cut_word</b>
Line	Deletes the current line and copies it to the clipboard.	<b>cut_line</b>
To End of Line	Deletes text from the cursor to the end of the line and copies it to the clipboard.	<b>cut_end_line</b>
Selection	Deletes the selected text.	<b>delete_selection</b>
All	Delete all text in current buffer.	<b>delete_all</b>

## Edit Other Menu

The **Edit > Other** menu contains miscellaneous editing operations.

Edit Other Menu Item	Description	Command
Lowcase	Translates the characters in the selected text to lower case.	<b>lowcase_selection</b>
Uppcase	Translates the characters in the selected text to upper case.	<b>upcase_selection</b>
Shift Left	Deletes the first column of text in each line of the selected text.	<b>shift_selection_left</b>
Shift Right	Inserts a space at the first column of each line of the selected text.	<b>shift_selection_right</b>
Overlay Block	Overwrites selected block/column of text at the cursor.	<b>overlay_block_selection</b>
Adjust Block	Overlays the selected text at the cursor and fills the original selected text with spaces.	<b>adjust_block_selection</b>
Enumerate	Displays the Enumerate dialog, which allows you to add incrementing numbers to a selection. See <a href="#">Enumerate Dialog</a> .	<b>gui_enumerate</b>
Filter Selection	Displays a Command dialog, which allows you to filter selected text through an external command. See <a href="#">Filter Selection: Command Dialog</a> .	<b>filter_selection</b>
Copy UCN As Unicode	Copies selected UCN to the clipboard as Unicode. See <a href="#">Using Unicode</a> .	<b>copy_ucn_as_unicode</b>
Copy Unicode As	Displays menu containing commands for copying Unicode as UCN. See <a href="#">Copy Unicode As Menu</a> .	N/A
Tabs to Spaces	Converts tabs to appropriate number of spaces. If there is no selection the entire buffer is converted.	<b>convert_tabs2spaces</b>
Spaces to Tabs	Converts leading spaces to tabs. If there is no selection the entire buffer is converted.	<b>convert_spaces2tabs</b>
Block Insert Mode	Allows you to insert/delete text for an entire block/column selection.	<b>block_insert_mode</b>

## Copy Unicode As Menu

The **Edit > Other > Copy Unicode As** menu contains operations for copying Unicode characters. See [Using Unicode](#) for more information.

Copy Unicode As Menu Item	Description	Command
C++ (UTF-16 \xHHHH)	Copies unicode characters in selection as C++ UTF-16 \xHHHH notation.	<b>copy_unicode_as_c</b>

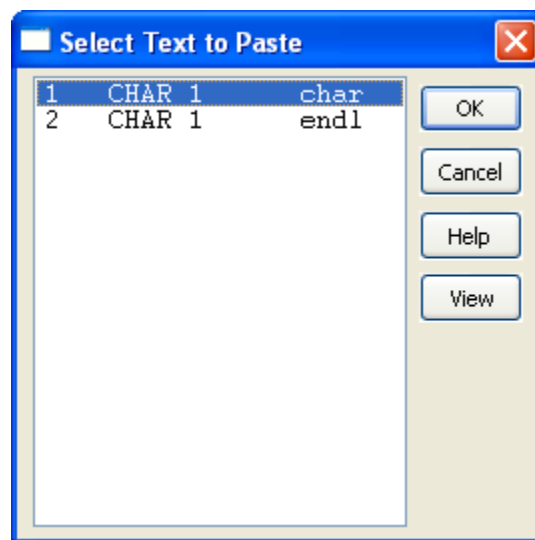
Copy Unicode As Menu Item	Description	Command
Regex (UTF-32 \x{HHHH})	Copies unicode characters in selection as Regex UTF-32 \x{HHHH} notation.	<b>copy_unicode_as_regex</b>
Java/C# (UTF-16 \uHHHH)	Copies unicode characters in selection as Java/C# UTF-16 \uHHHH notation.	<b>copy_unicode_as_java</b>
UCN (UTF-32 \uHHHH and \UHHHHHHHH)	Copies unicode characters in selection as UCN UTF-32 \uHHHH and \UHHHHHHHH notation.	<b>copy_unicode_as_ucn</b>
SGML/XML hexadecimal (UTF-32 &xHHHH;)	Copies unicode characters in selection as SGML/XML hexadecimal UTF-32 &xHHHH; notation.	<b>copy_unicode_as_xml</b>
SGML/XML decimal (UTF-32 &#DDDD;)	Copies unicode characters in selection as SGML/XML decimal UTF-32 &#DDDD; notation.	<b>copy_unicode_as_xmldec</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with **Edit** menu items.

### Select Text to Paste Dialog

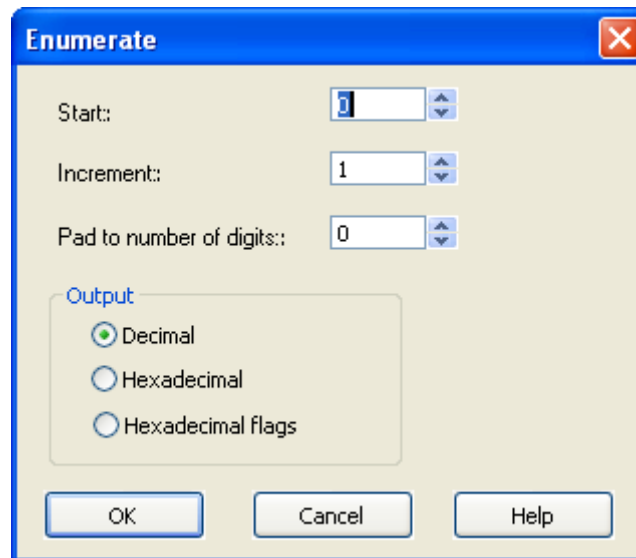
The Select Text to Paste dialog, shown below, is used for viewing and inserting recently used clipboards. To access this dialog, choose **Edit > List Clipboards**, or use the **list\_clipboards** command (Ctrl+Shift+V).



The numbers in the first column of the list box are used to help you move the selection cursor. The second column indicates the clipboard type. The third column shows all or a portion of the clipboard text (depending on the length). Click **OK** to insert the selected clipboard at the cursor location. Click **View** to see the complete text in the View Clipboard window. From here you can copy all or part of the text to the operating system clipboard.

## Enumerate Dialog

The Enumerate dialog, shown below, contains options for adding incrementing numbers to a selection. Choose **Edit > Other > Enumerate**. After options have been configured, you can use the **enumerate** command. See the Help system for command syntax.



The following options are available:

- **Start** - C syntax expression which evaluates to the number used for first line of selection. However, when the Hexadecimal flags output style is selected, the start must be an integer bit position or the first hexadecimal number with which to start.
- **Increment** - C syntax expression which evaluates to the amount to increment for each line in the selection. However, when the Hexadecimal flags output style is selected, this specifies the number of bit positions by which to increment.
- **Pad to number of digits** - Specifies the digit width for each number. Number is padded to at least this number of digits by adding leading zeros.
- **Output** - Both the Hexadecimal and Hexadecimal flags options specify hexadecimal syntax output based on the buffers extension. We determine the hexadecimal syntax based on the color coding which supports **0xhhhh** (C syntax), **&Hdddd** (Basic), **hhhhhH** (Intel assembler), and **\$hhhh** (Motorola assembler). If the buffer's extension has no color coding, the hex numbers are prefixed with **0x**.

## Filter Selection: Command Dialog

The Command dialog allows you to specify a command to run against the selected text. Choose **Edit > Other > Filter Selection** or use the **filter\_selection** command.

Enter the command in the **Command** text box. The selected text will be used as input to the command, and the output from the command will replace the selected text. Use the drop-down arrow to the right of the **Command** text box to select from a history of previously entered commands.

*EDIT*

## Search

This section describes items on the **Search** menu and associated dialogs and tool windows. For more information about using search and replace operations, see [Find and Replace](#).

### Search Menu

The **Search** menu contains items pertaining to search and replace, navigation, and bookmarks. The table below contains a summary of these items.

Search Menu Item	Description	Command
Find	Displays the Find and Replace tool window, open to the Find tab, which allows you to search for a specified string. See <a href="#">Find and Replace Tool Window</a> .	<b>gui_find</b>
Find in Files	Displays the Find and Replace tool window open to the Find in Files tab, which lets you search for a string in files. See <a href="#">Find in Files Tab</a> .	<b>find-in-files</b>
Next Occurrence	Searches for the next occurrence of the last string you searched for.	<b>find_next</b>
Previous Occurrence	Searches for the previous occurrence of the last string you searched for.	<b>find_prev</b>
Replace	Displays the Find and Replace tool window, open to the Replace tab, which allows you to search for a string and replace it with another string. See <a href="#">Replace Tab</a> .	<b>gui_replace</b>
Replace in Files	Displays the Find and Replace tool window, open to the Replace in Files tab, which allows you to search for a string and replace it with another string in files. See <a href="#">Replace in Files Tab</a> .	<b>replace-in-files</b>
Incremental Search	Searches for match incrementally. See <a href="#">Incremental Searching</a> .	<b>i-search</b>
Find File	Displays the Find File dialog, which lets you search for files on disk.	<b>find-file</b>
Go to Line	Places the cursor on a line you specify. See <a href="#">Navigating to a Specific Line</a> .	<b>gui_goto_line</b>
Go to Offset	Places the cursor on a byte/character offset in the current file. See <a href="#">Navigating to an Offset</a> .	<b>gui_seek</b>
Go to Matching Parenthesis	Finds the matching parenthesis or begin/end structure pair. See <a href="#">Begin/End Structure Matching</a> .	<b>find-matching-paren</b>

Search Menu Item	Description	Command
Go to Definition	Pushes a bookmark at the cursor and places your cursor on a symbol definition. See <a href="#">Find Symbol Tool Window</a> and <a href="#">Symbol Navigation</a> .	<b>gui_push_tag</b>
Go to Reference	Searches for references to the symbol under the cursor. See <a href="#">Symbol Navigation</a> .	<b>push_ref</b>
Find Symbol	Searches Context Tagging® databases for a symbol you specify. See <a href="#">Find Symbol Tool Window</a> .	<b>gui_push_tag</b>
Push Bookmark	Pushes a bookmark at the cursor. See <a href="#">Bookmarks</a> .	<b>push_bookmark</b>
Pop Bookmark	Pops the last bookmark. See <a href="#">Bookmarks</a> .	<b>pop_bookmark</b>
Set Bookmarks	Sets a persistent bookmark on the current line. See <a href="#">Bookmarks</a> .	<b>set-bookmark</b>
Go to Bookmark	Displays the Go to Bookmark dialog from which you can select a bookmark to navigate to. See <a href="#">Navigating Named Bookmarks</a> .	<b>gotot_bookmark</b>
Toggle Bookmark	Toggles setting a bookmark on the current line. See <a href="#">Bookmarks</a> .	<b>toggle-bookmark</b>
Bookmarks	Lists bookmarks and allows you to add and delete bookmarks. See <a href="#">Bookmarks</a> .	<b>activate-bookmarks</b>
Next Bookmark	Go to next bookmark. See <a href="#">Bookmarks</a> .	<b>next-bookmark</b>
Previous Bookmark	Go to previous bookmark. See <a href="#">Bookmarks</a> .	<b>prev-bookmark</b>
Last Find/Grep List	Displays list of Files/Buffers generated by Find commands.	<b>grep_last</b>

## Dialogs and Tool Windows

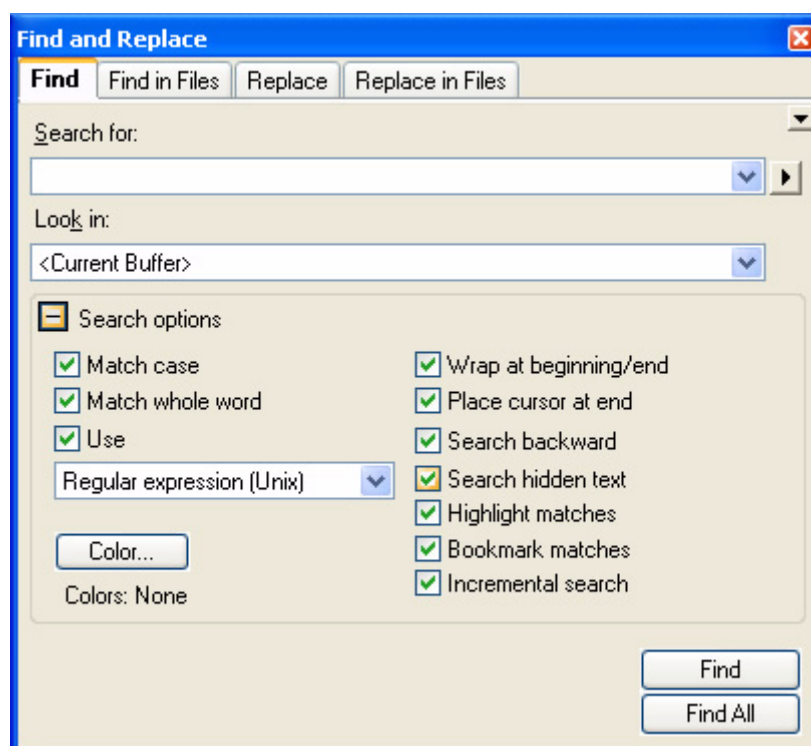
This section describes the dialogs and tool windows that are associated with the **Search** menu items. Note that there is an additional dialog not listed here that contains search options—the Search tab of the General Options dialog (**Tools > Options > General**). See [Search Tab](#) for a description of these settings.

### Find and Replace Tool Window

See [Find and Replace](#) for information about searching and replacing in SlickEdit®.

To access the Find and Replace tool window, press **Ctrl+F**, choose **Search > Find**, or choose **View > Toolbars > Find and Replace**.





The Find and Replace tool window contains a right-click menu of options, which are described below (see [Right-Click Context Menu](#)). The other options on the Find and Replace tool window are categorized into the following tabs—click on the links to go to that section in the documentation:

- [Find Tab](#)
- [Find in Files Tab](#)
- [Replace Tab](#)
- [Replace in Files Tab](#)

## Right-Click Context Menu

Right-click in the background of the Find and Replace tool window to access the following options:

- **Saved Search Expressions** - See [Saving Search and Replace Values](#).
- **Configure Options** - Displays the Search tab of the General Options dialog, from which you can set the default search options that the tool window should use. See [Search Tab](#).
- **Use Default Options** - If selected, the options specified in the Search tab of the General Options dialog are used instead of the options selected in the Find and Replace tool window. See [Search Tab](#).
- **Clear All Options** - Clears all options that are selected in the Find and Replace tool window.
- **Set Current Options as Default** - If selected, the options that are selected on the tool window replace the settings in the [Search Tab](#) of the General Options dialog.
- **Hide/Show Tabs** - Toggles the display of the tabs on the Find and Replace tool window.
- **Clear Highlights** - Removes all highlighting from text that was highlighted during a search or replace operation.

- **Allow Docking** - If selected, the Find and Replace tool window can be docked.

### Find Tab

The Find tab provides the following fields and options for searching and finding text or regular expressions:

- **Search for** - Enter the string you want to search for here. You can retrieve previous search strings by clicking the drop-down list button. Strings may be text or regular expressions and can include wildcards. Note that ISPF search expressions cannot be used here.

Click the right-pointing arrow button to the right of the **Search for** field to display a menu containing specific search syntax options such as **Character in Range**, **Beginning of Line**, and **Decimal Digit**.

- **Look in** - This field allows you to specify a range for your search to the current selection, current procedure, current buffer or all buffers.
- **Search options** - Click this button to expand or contract the search options section of the tool window. When contracted, the options that are enabled are summarized in this area.
- **Match case** - If selected, a case-sensitive search is performed.
- **Match whole word** - If selected, a word search is performed. Before a search is considered successful, the characters to the left and right of the occurrence of the search string found are checked to be non-word characters.

The default word characters are [A-Za-z0-9\_\$] and can be changed. To change these, from the main menu choose **Tools > Options > File Extension Setup**. Select the [Advanced Tab](#), and enter your desired characters in the **Word chars** field.

- **Use** - Set this option to select one of the following types of search syntax from the drop-down list:
  - Regular expression (UNIX)
  - Regular expression (Brief)
  - Regular expression (SlickEdit®)
  - Wildcards (\*,?)

See [Find and Replace with Regular Expressions](#) for more information.

- **Color** - Displays the Color Coding Search Options dialog. This dialog lets you pick various syntactic elements to filter a search. These are the same elements used by the Color Coding engine. Using these filters helps to reduce the number of false positives you find in a search. Each check box has three states:
  - **Neutral (the default)** - All check boxes start in the neutral state. These elements will be used in a search until deselected or until one or more other elements are selected. Putting a check in any check box essentially deselects all non-checked boxes.
  - **Selected** - If the check box is selected, the search will be restricted to this element and any other selected elements. There is no need to deselect any other elements if any elements are selected. If any elements are selected, only selected elements will be searched. For example, to search for the word “result” only in comments, put a check only in the **Comment** check box. All other syntactic elements will be ignored as part of this search.
  - **Deselected** - If the check box is clear, these elements will not be searched. For example, if you want to find the word “result” anywhere in your code except for in comments, clear the **Comment** check box.

Click the **Reset** button to mark all items as neutral.

**NOTE** Not all languages have all color coding elements defined. For example, dBase and Pascal do not have preprocessing. Only C++ and Java have function color defined. Only HTML has attributes (i.e. <img src=.>).

- **Wrap at beginning/end** - If selected, the search will always be performed on the entire buffer, starting from the cursor.
- **Place cursor at end** - If selected, the cursor is placed at the end of the occurrence found.
- **Search backward** - Select this option to have the search performed from the end to the beginning.
- **Search hidden text** - Select this option to search for text hidden by Selective Display. Matches found that were set to be hidden by Selective Display will be revealed. To enable Selective Display options, from the main menu choose **View > Selective Display**. See [Selective Display](#) for more information.
- **Highlight matches** - Select this option to highlight all matched patterns in the current search range. Highlight colors for these matches are customizable. To set this color, choose from the main menu, **Tools > Options > Color** and select **Highlight** from the **Screen element** list. Choose your desired color settings and click **OK**. See [Colors](#) for more information.

To clear all highlighted text in all buffers, deselect the **Highlight matches** option or simply close the Find and Replace tool window.

- **Bookmark matches** - Select this option to bookmark lines with matching patterns and display the Bookmarks tool window when a match is bookmarked.
- **Incremental search** - Select this option to search incrementally on patterns being typed into the **Search for** field, showing the location of the match at the cursor. See [Incremental Searching](#) for more information on this method of searching.
- **Find button** - Click this button when you have entered all desired search options and are ready to initiate a search. If no matches are found, the **Search for** field will turn red, and the text "String not found" will be displayed in the status area of the editor.
- **Find All button** - Click this button to mark all matches in the current search range which have any of the following options selected: **Search hidden text**, **Highlight matches**, and **Bookmark matches**. If no matches are found, the **Search for** field will turn red, and the text "String not found" will be displayed in the status area of the editor.

## Find in Files Tab

The Find in Files tab provides the same functionality as the Find tab (see [Find Tab](#)), with the added ability to conduct multi-file searches. Additional options are described below.

- **Look in** - This field allows you to specify a range for your search to the current selection, current procedure, current buffer or all buffers.

Click the right-pointing arrow button to the right of the **Look in** field to display a menu containing more specific range options such as **Directory**, **Project**, and **All Buffers**. From this sub-menu, you may also select **Append** and choose an item for which to have the search results appended.

- **File types** - Specifies one or more file types (extensions) to search for. Type in this field or use the drop-down list to select the extensions desired. When a file title is specified in the **Look in** field, the file types wildcards are ignored.

- **Exclude** - Paths, files, or file types can be excluded from a multi-file search by specifying them with wildcards or full path names. For examples of exclude patterns, see [Exclusion Examples](#). No files are searched in a path that is excluded, including any files in sub-directories beneath.
- **Look in subfolders** - Select this option to expand the search to sub-directories of the folder specified in the **Look in** field.
- **Results options** - Click this button to expand or contract the **Results** options section of the tool window. When contracted, the options that are set are summarized in this area.
- **Search Results Window** - This field allows you to send the search results to a specific Search Results window. The window to be used can be selected from the drop-down list, and these are labeled starting at **Search<0>**. A new results window can be added with the **<New>** option up to a pre-set limit of open Search Results windows. If **<Auto Increment>** is selected, the search results will cycle through all of the open Search Results tabs in the Search Results tool window with each new search. See [Search Results Output](#) for more information.

Right-click in the Search Results window to access the following options:

- **Quick Search** - Finds the next occurrence of the text selected.
- **Filter Search Results** - Select this option to display the Filter Search Results dialog. From here, if a match is found, you can choose to keep or delete lines with additional searches, match case, limit to current default regular expression syntax and/or remove matches found on the same line number in the same file (this can also be accomplished by selecting **List matching lines** only from the Find in Files tab).
- **Open as Editor window** - Opens current search results in a new editor window.
- **Go to Line** - Goes to the file/line number of the current line in the Search Results window.
- **Bookmark Line** - Places a bookmark at the line in the file where the result was found.
- **Clear Window** - Clears all results in the current Search Results window.
- **Align Columns** - Aligns the line numbers and column numbers for all search results.
- **Collapse All** - Collapses all Selective Display levels. See [Selective Display](#) for more information.
- **Expand All** - Expands all Selective Display levels. See [Selective Display](#) for more information.
- **Output to editor window** - If selected, search results are sent to an editor window.
- **Append to output** - Select this option to append search results to the search results window that is in focus.
- **List filenames only** - If selected, only file names and not occurrences are listed in the search output.
- **List matching lines only** - Selecting this option will display only one line in the search results window for each line containing one or more matching patterns on the same line, and will highlight all matching patterns.
- **Foreground search** - If selected, activates the three range options listed below. This option offers slightly better performance than a background search, but prevents you from continuing to work while the search is being performed. The default search for SlickEdit® is background searching unless this option is selected.
  - **Prompted** - When this option is selected, you are prompted whether to continue searching when an occurrence is found.
  - **Single** - When this option is selected, your cursor is placed on the first occurrence found, but the remaining files are not searched.

- **Global** - When this option is selected, all files are searched for occurrences without prompting.
- **Stop button** - Click **Stop** to terminate a multi-file, background search. Press **Esc** to terminate a long foreground search.

## Replace Tab

The Replace tab provides options for searching and replacing text or regular expressions. The same search options from the Find tab are available (see [Find Tab](#)), as well as the additional replace options described below.

- **Replace with** - Enter the text or regular expression for which to replace the item that is searched. You can retrieve previous replacement text or regular expressions by clicking the drop-down list button.  
  
Click the right-pointing arrow button to the right of the **Replace with** field to display a menu containing tagged expressions. See [Using Tagged Search Expressions](#) for more information.
- **Preserve case** - Select this option to perform a case-sensitive search and replace operation.
- **Highlight replaced text** - Select this option to highlight all instances of the text that was replaced.
- **Replace button** - Click to replace the first instance of the item.
- **Replace All button** - Click to replace every instance of the item.
- **Preview All button** - Click to show a side-by-side comparison of the original file and the file with replacements made. This lets you see the changes and confirm them before committing the changes to the file.

**TIP** You can use the menu items **Edit > Undo** and **Edit > Redo** to undo/redo replacements.

## Replace in Files Tab

The Replace in Files tab provides the same functionality as the Replace tab (see [Replace Tab](#)), with the added ability to conduct multi-file replacements. It contains the following additional option:

- **Leave modified files open** - Select this option to open all of the files on which a replace has been performed.

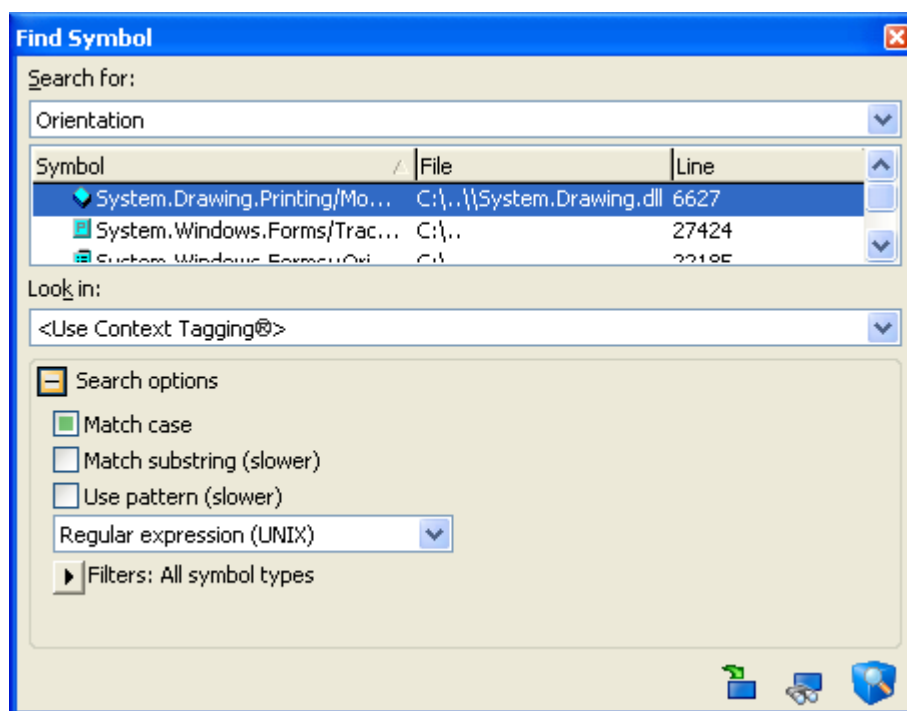
**TIP** You can use the menu items **Edit > Multi-File Undo** and **Edit > Multi-File Redo** to undo/redo replacements in multiple files.

## Find Symbol Tool Window

For more information on finding symbols, see [Symbol Browsing](#).

The Find Symbol tool window (**Search > Find Symbol** or **View > Toolbars > Find Symbol**) is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match.

Searching for a symbol is faster than a normal text search because it is executed against the Context Tagging® database, rather than searching through your source files. Find Symbol also avoids false hits in comments or string literals. Though [Syntax-Driven Searching](#) in the regular [Find and Replace Tool Window](#) provides this same capability, it cannot match the speed of Find Symbol.



- **Search for** - Enter the name of the symbol to find. If you select the option **Use pattern**, you can enter regular expressions or wildcards in the search field. If you specify **<Use Context Tagging@>** for the **Look in** field, then you can enter language-specific expressions, such as "this->get" to find getters in your current class. SlickEdit® displays a progress bar at the top of this tool window while a search is in progress.

Incremental matches are displayed with each character you type, and the first element in the list is selected. Press **Tab** to put focus into the list of matches. Press **Enter** to navigate to the first match. Press **Down** to select the next match. Press **Escape** to stop the search.

- **Symbol List** - The list of search results are refreshed as you type the search string. They include the symbol name, the file that contains it, and the line number. You can sort by any of the three columns.

The selected match is highlighted and is displayed in the Preview tool window. Single-click or use the arrow keys to select a match. Double-click or press **Enter** to navigate to that match.

- **Look in** - Use this control to specify the scope of the symbol search. The options are:
  - **<Use Context Tagging@>** - This is the default setting. It uses Context Tagging to intelligently determine which tag files to search.
  - **<Current File>** - Select this setting to only search the tags in the current file, including local variables in the current function scope.
  - **<Current Project>** - Select this setting to only search in files that are in the current project.
  - **<Current Workspace>** - Select this setting to only search in files that are in the current workspace.
  - **<Extension Tag Files>** - Select this setting to search all extension-specific tag files for the indicated extension. This may also include your workspace tag file.
  - **Specific tag files** - Select one of the specific tag files listed to limit search to that file.

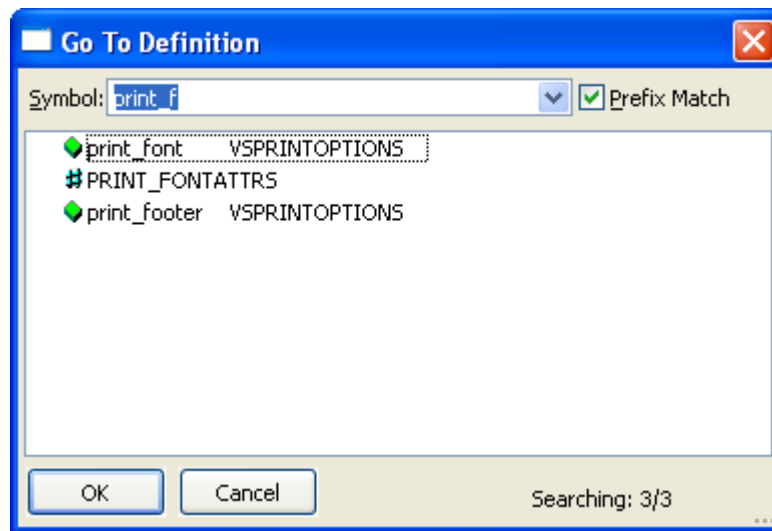
- **<All Tag Files>** - Select this setting to search all tag files for all languages.
- **Search Options** - The search options can be expanded or collapsed to save space.
  - **Match case** - When selected, SlickEdit uses a case-sensitive search to find symbol matches. When this option is not selected, SlickEdit uses a case-insensitive search. When this option is in the neutral (mixed) state, SlickEdit first searches for case-sensitive matches, and if none are found, attempts to perform a case-insensitive search. Note that for case-insensitive languages, this may have no effect.
  - **Match substrings (slower)** - When selected, SlickEdit searches for the specified string within the available symbols. For example, finding all symbols containing the word "order," not just those that begin with "order." Selecting this option causes the search to execute more slowly.
  - **Use pattern (slower)** - When selected, SlickEdit interprets the search string as a regular expression or wildcard expression. This can result in slower search times, since SlickEdit must test every symbol in the tag file against the regular expression.
  - **Filters** - Use filters to restrict the search to certain types of symbols. The filters are the same as the ones available on the Definitions tool window. See [Defs Tool Window](#) for more information.
- **Icons** - The following icons are located at the bottom of the tool window:
  - **Go to definition** - Navigates to the definition of this symbol in the editor window. If the programming language allows for separate declaration and definition, you can control which is selected by using the Extension Options dialog (**Tools > Options > File Extension Setup**). Select the appropriate language from the **Extension** drop-down list, then select the [Context Tagging® Tab](#). Select either **Go to Definition navigates to symbol definition (proc)** or **Go to Definition navigates to symbol declaration (proto)**. See [Code Navigation](#) for more information.
  - **Go to reference** - Displays a list of references for the selected symbol in the [References Tool Window](#) and, optionally, navigates to the first reference. Select **Tools > Options > Context Tagging® Options** and check the option **Go to Reference only lists references** if you just want to build the list of references. See [Code Navigation](#) for more information.
  - **Show in symbol browser** - Displays the selected symbol in the [Symbols Tool Window](#). Note that this feature does not work for local variables or symbols from the current file that are not in a tag file.
  - **Manage tag files** - Displays the [Context Tagging® - Tag Files Dialog](#), which can be used to update your tag files.

## Go to Definition Dialog

For more information about navigating between symbols, see [Code Navigation](#).

The Go to Definition dialog (**gui\_find\_proc** command) can be used to navigate to symbols. This dialog box lists all tags that match the prefix you have typed so far.

**TIP** In SlickEdit® 2007, the functionality of this dialog has been replaced with the [Find Symbol Tool Window](#).



The following options are available:

- **Symbol** - Specifies the symbol to search for.
- **Prefix Match** - Deselect to match tags using a regular expression or a substring search.



## View

This section describes items on the **View** menu and associated dialogs and tool windows. For more information, see [Viewing and Displaying](#).

### View Menu

The **View** menu contains options that pertain to viewing and displaying special characters, code, and toolbars. The table below describes each item and its corresponding command.

View Menu Item	Description	Command
Hex	Toggles hex/ASCII display. See <a href="#">Hexadecimal View and Edit Mode</a> .	<b>hex</b>
Line Hex	Toggles line hex/ASCII display. See <a href="#">Hexadecimal View and Edit Mode</a> .	<b>linehex</b>
Special Chars	Toggles viewing of tabs, spaces, and new line character(s) on/off. See <a href="#">Viewing Special Characters</a> .	<b>view_specialchars_toggle</b>
New Line Chars	Toggles viewing of new line character(s) on/off. See <a href="#">Viewing Special Characters</a> .	<b>view_nlchars_toggle</b>
Tab Chars	Toggles viewing of tab character(s) on/off. See <a href="#">Viewing Special Characters</a> .	<b>view-tabs-toggle</b>
Spaces	Toggles viewing of space character(s) on/off. See <a href="#">Viewing Special Characters</a> .	<b>view_spaces_toggle</b>
Line Numbers	Toggles the display of line numbers on/off for the current document. See <a href="#">Viewing Line Numbers</a> .	<b>view_line_numbers_toggle</b>
SoftWrap	Toggles wrapping of long lines to window width.	<b>softwrap-toggle</b>
Toolbars	Show, hide, or customize a toolbar or tool window. See <a href="#">Toolbars and Tool Windows</a> .	<b>toolbars</b>
Fullscreen	Toggles full screen editing mode. See <a href="#">Full Screen Mode</a> .	<b>fullscreen</b>
Selective Display	Displays the Selective Display dialog, which allows you to hide lines and create an outline. See <a href="#">Selective Display</a> .	<b>selective-display</b>
Hide All Comments	Hides all lines that only contain a comment.	<b>hide_all_comments</b>
Hide Code Block	Hides lines inside current code block. See <a href="#">Expanding/Collapsing Code Blocks</a> .	<b>hide_code_block</b>
Hide Selection	Hides selected lines.	<b>hide_selection</b>

View Menu Item	Description	Command
Hide #region Blocks	Hides .NET #region blocks.	<b>hide-dotnet-regions</b>
Function Headings	Collapses all function code blocks in the current file. See <a href="#">Selective Display</a> .	<b>show-procs</b>
Expand/Collapse Block	Toggles between hiding and showing the code block under the cursor. See <a href="#">Expanding/Collapsing Code Blocks</a> .	<b>plusminus</b>
Copy Visible	Copies text not hidden by Selective Display. See <a href="#">Selective Display</a> .	<b>copy-selective-display</b>
Show All	Ends selective display. All lines are displayed and outline bitmaps are removed. See <a href="#">Selective Display</a> .	<b>show_all</b>

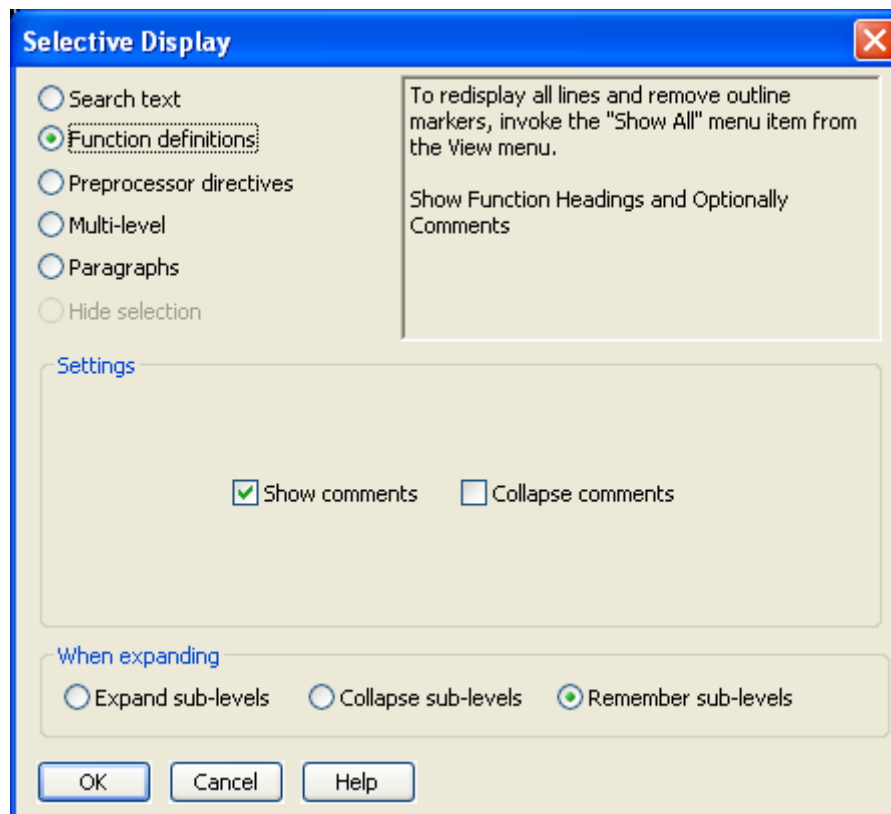
## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with **View** menu items.

### Selective Display Dialog

The Selective Display dialog (**View > Selective Display** or **selective\_display** command) allows you to enable Selective Display and choose the regions in your code that you want to display or hide. Each region contains settings that are specific to that region. The dialog also contains static options for expanding.

See [Selective Display](#) for more information about working with this feature.



## Search Text

Select **Search text** to specify a search string and display lines containing the search string specified or lines not containing the search string specified. Click the right-pointing arrow button to the right of the field to display a menu containing specific search syntax options such as **Character in Range**, **Beginning of Line**, and **Decimal Digit**. The following settings are available:

- **Match case** - When checked, a case sensitive search is performed.
- **Match whole word** - When checked, a word search is performed. Before a search is considered successful, the characters to the left and right of the occurrence of the search string found are checked to be non-word characters. The default word characters are [A-Za-z0-9\_.\$] and may be changed by the Extension Options dialog box (from the main menu choose **Tools > Options > File Extension Setup**, then select the [Advanced Tab](#)).
- **Regular expression** - When checked, a regular expression search is performed. See [Find and Replace with Regular Expressions](#) for more information.
- **Reset selective display** - When checked, all lines are made visible and plus and/or minus icons are removed before a search is performed.
- **Hide matched lines** - When checked, lines containing the search pattern are hidden.

## Function Definitions

Select **Function definitions** to display only function headings and optional function heading comments. The following settings affect how comments before function definitions are handled:

- **Show comments** - When checked, comments above function definitions are displayed as if they were part of the function definition.
- **Collapse comments** - When checked, comments above function definitions are visible but multi-line comments will require that you expand them to see all comments.

When both check boxes are off, comments will not be visible at all, making it difficult to copy or move functions and comments.

## Preprocessor Directives

Select **Preprocessor directives** to display a source file as if it were preprocessed according to the define values you specify. If you do not remember your defines, use the **Scan for Defines** button. The following settings are available:

- **Defines** - Specifies defines and optional values used when you select the **Preprocessor Directives** option on the Selective Display dialog box. The syntax is:

***name1[=value1] name2[=value2]***

For example:

**WIN32S VERSION=4**

- **Warning if Not Defined** - If on when you preprocess your source, a message box is displayed for each define found in an expression which does not have a value.
- **Scan for Defines** - Searches for define variables in the current source file and lets you specify values. Resulting values are placed in the **Defines** combo box.

## Multi-Level

Select **Multi-level** to set multiple levels of selective display based on braces or indent. The following settings are available:

- **Braces** - When on, multiple levels of selective display are set to correspond to curly brace nesting levels.
- **Indentation** - When on, multiple levels of selective display are set to correspond to indentation levels.
- **Limit levels** - When too many nested levels of selective display confuse you, place a limit on the maximum number of nested levels. Nesting deeper than this specified level is ignored.

## Paragraphs

Select **Paragraphs** to display the first line of each paragraph. A paragraph is defined by a group of lines followed by one or more blank lines.

## Hide Selection

Select **Hide selection** to hide the lines in the current selection.

## Expansion Options

The following expansion options can be applied for each region:

- **Expand sub-levels** - When on, expanding hidden lines expands all nested hidden lines.
- **Collapse sub-levels** - When on, expanding hidden lines collapses all nesting hidden lines.

- **Remember sub-levels** - When on, expanding hidden lines displays nested hidden lines the way they were last displayed.

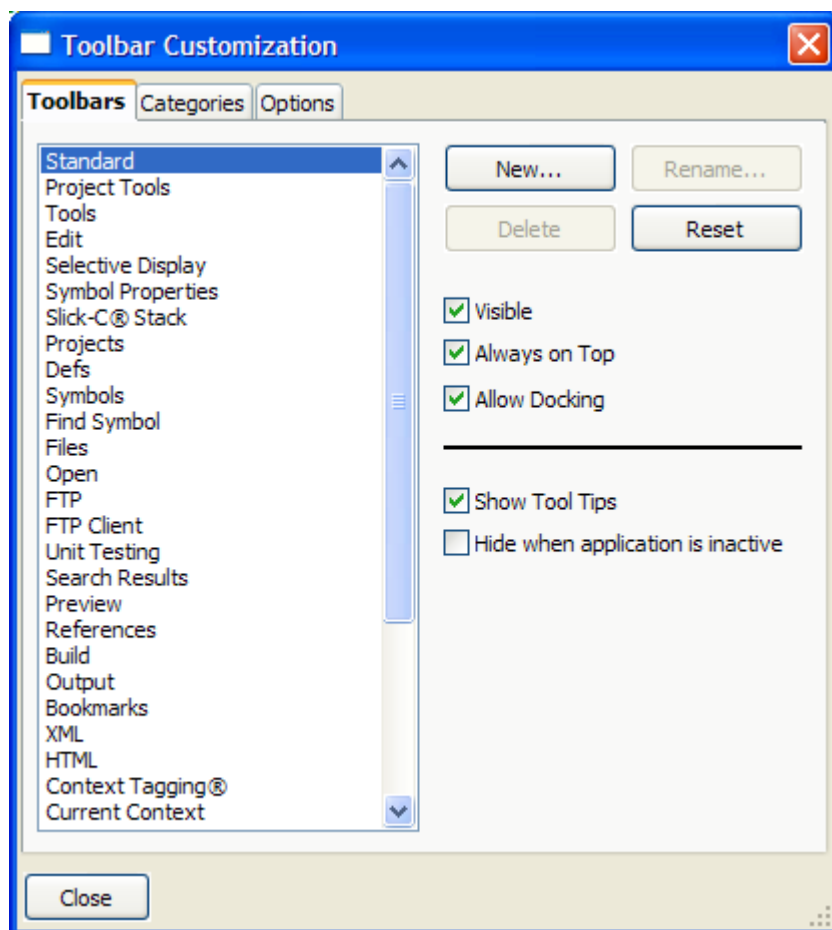
## Toolbar Customization Dialog

The Toolbar Customization dialog contains settings for creating, editing, and deleting toolbars. To access this dialog, choose **View > Toolbars > Customize**, or right-click on any tool window's title bar on Windows (right-click on the tool window's background for UNIX/Mac) and select **Customize**. The options on this dialog are categorized into three tabs:

- [Toolbars Tab](#)
- [Categories Tab](#)
- [Options Tab](#)

### Toolbars Tab

The **Toolbars tab** of the Toolbar Customization dialog, pictured below, contains a list of the default tool windows within SlickEdit®, and settings made here are on a per-tool window basis.



The following options are available:

- **New** - Creates a new, empty tool window.
- **Rename** - Renames the selected tool window.

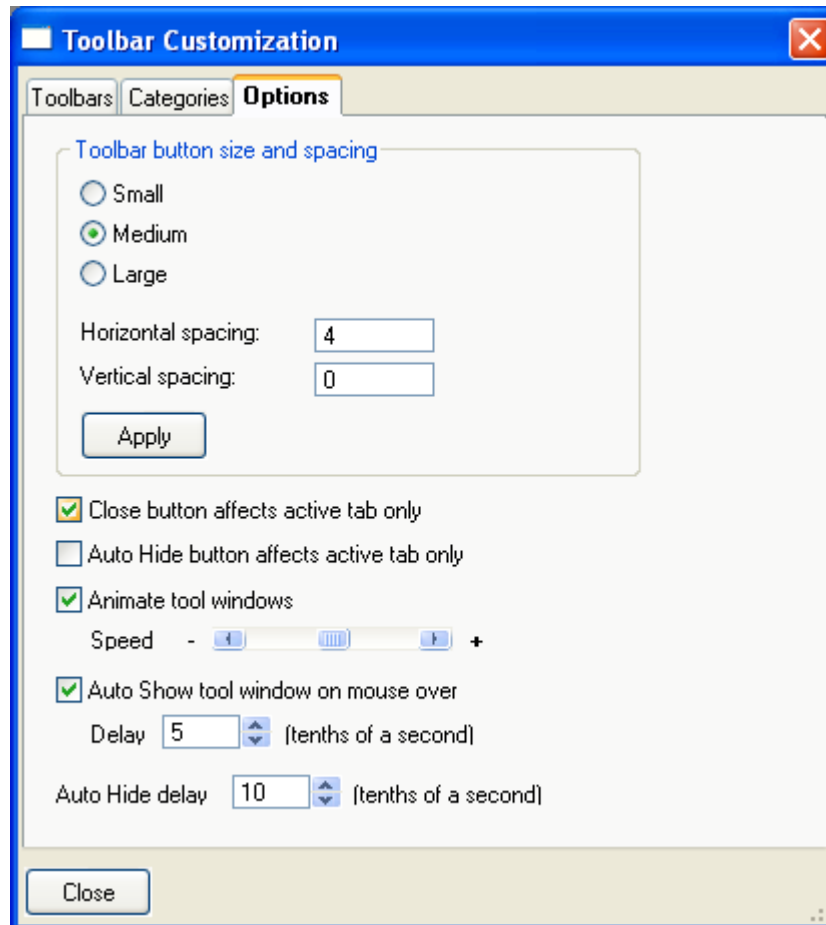
- **Delete** - Deletes the selected tool window.
- **Reset** - Restores the default buttons to the selected standard tool window (not a user-created tool window).
- **Visible** - When checked, the selected tool window is displayed, if it is not already displayed. When unchecked, the selected tool window is closed.
- **Always on Top** - When checked, the selected tool window, when non-docked, will remain on top of the editor window.
- **Allow Docking** - When unchecked, the selected tool window, when non-docked, cannot accidentally be docked.
- **Show Tool Tips** - Indicates whether all tool windows should display tool tips. This option is global to all tool windows.
- **Hide when application is inactive** - When checked, non-docked tool windows are hidden when you switch to another application. When you switch back to SlickEdit, the tool window is made visible again. This option is global to all tool windows.

### Categories Tab

The **Categories tab** of the Toolbar Customization dialog categorizes the toolbar controls (buttons) and allows you to drag and drop them onto existing tool windows. Select a category from the **Categories** list. The associated controls will be displayed in the **Controls** box. Click on the control you wish to add, and drag it onto a tool window. See also [Toolbar Control Properties Dialog](#).

## Options Tab

The **Options tab** of the Toolbar Customization dialog contains settings that apply to all tool windows.



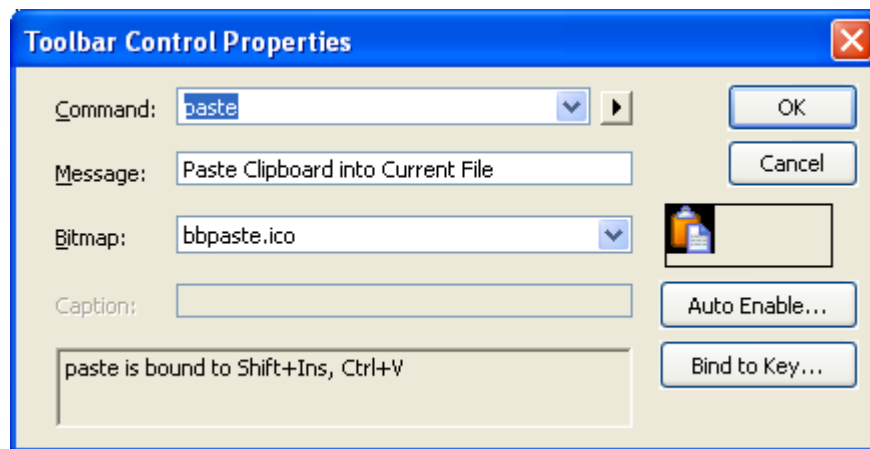
The following settings are available on the Options tab:

- **Toolbar button size and spacing** - Specify the size of buttons and the amount of vertical and horizontal spacing between buttons on a toolbar. These options affect both vertically and horizontally docked toolbars. Click **Apply** to save changes.
- **Close button affects active tab only** - When checked, the **Close** button on a tool window will close only that tool window. When unchecked, and the tool window is tab-linked with other tool windows, all tab-linked tool windows are closed. By default, this option is selected.
- **Auto Hide button affects active tab only** - When checked, the **Auto Hide** button (the **Pushpin** icon) on a tool window will auto-hide only that tool window. When unchecked, and the tool window is tab linked with other tool windows, all tab-linked tool windows are auto-hidden. Auto-hidden tool windows are displayed in the dock channel on the side of the editor where they were auto-hidden. By default, this option is not selected.
- **Animate tool windows** - When checked, auto-hide and auto-show of tool windows is animated. When unchecked, tool windows are shown and hidden immediately. By default, this option is selected.
- **Speed** - When **Animate tool windows** is checked, a slider is used to adjust the speed of animation during auto-hide and auto-show. Adjust positive (+) to increase speed and negative (-) to decrease speed. By default, the speed is set to medium.

- **Auto Show tool window on mouse over** - When checked, auto-hidden tool windows are auto-displayed when the mouse is over the item in the dock channel. **Delay** specifies how long to wait (in tenths of a second) before the mouse triggers an auto-hidden tool window to auto-display. By default, the delay is set to 5.
- **Auto Hide delay** - Specifies how long to wait (in tenths of a second) before an auto-displayed tool window auto-hides itself. By default, this delay is set to 10.

## Toolbar Control Properties Dialog

To change a button's command binding, on the actual toolbar or tool window, right-click on any control and select **Properties**. This will display the Toolbar Control Properties dialog, shown below. Note that the Properties option is only available for controls that can be modified.



The following options are available:

- **Command** - Specifies the command that the button is bound to. Use the drop-down arrow to pick from a list of commands. Use the right-pointing arrow to insert special escape sequences for a file name, line number, or word.
- **Message** - Use this text box to enter the tooltip message that should appear when hovering the mouse over the button.
- **Bitmap** - Specifies the bitmap that will be used for the button. Click the drop-down arrow to display an Open dialog in order to specify an alternate bitmap. This option is only available for graphical controls.
- **Caption** - Specifies the text that appears on the button for text-only controls (like the Sample Button).
- **Auto Enable** - Displays the Auto Enable Properties dialog, which allows you to enable/disable predefined attributes. See [Auto Enable Properties Dialog](#) for more information.
- **Bind to Key** - Displays the [Key Bindings Dialog](#) so that you can create a key binding for this command.



# Project

This section describes items on the **Project** menu and associated dialogs and tool windows. For more information, see [Workspaces and Projects](#).

## Project Menu

The **Project** menu contains operations and options for working with projects and workspaces. The table below contains a summary of these items.

Project Menu Item	Description	Command
New	Allows you to create a workspace and/or project.	<b>workspace_new</b>
Open Workspace	Opens a workspace.	<b>workspace_open</b>
Open Other Workspace	Displays menu for open projects, workspaces, or makefiles from other tools. See <a href="#">Open Other Workspace Menu</a> .	N/A
Close Workspace	Closes the current workspace.	<b>workspace_close</b>
Organize All Workspaces	Allows you to organize your workspaces which appear in the All Workspaces menu.	<b>workspace_organize</b>
Workspace Properties	Displays the Workspace Properties dialog, which allows you to add/remove projects from the current workspace. See <a href="#">Workspace Properties Dialog</a> .	<b>workspace_properties</b>
Add New Item from Template	Adds new template file to the existing project.	<b>project-add-item</b>
Open Files from Project	Allows you to open files from the current project.	<b>project-load -p</b>
Open Files from Workspace	Allows you to open files from the current workspace.	<b>project-load</b>
Insert Project into Workspace	Adds an existing project to the current workspace. Use the Workspace Properties dialog box to remove a project from the current workspace.	<b>workspace_insert</b>
Dependencies	Displays the Project Properties dialog open to the Dependencies tab, which lets you set the dependencies for the active project. See <a href="#">Dependencies Tab</a> .	<b>workspace_dependencies</b>
Set Active Project	Allows you to set the active project	<b>projecttbSetCurProject</b>
Project Properties	Displays the Project Properties dialog, which is used to edit settings for the current project. See <a href="#">Project Properties Dialog</a>	<b>project_edit</b>

## Open Other Workspace Menu

The **Project > Open Other Workspace** menu contains options for opening projects, workspaces, or makefiles from other tools. The table below contains a summary of these items.

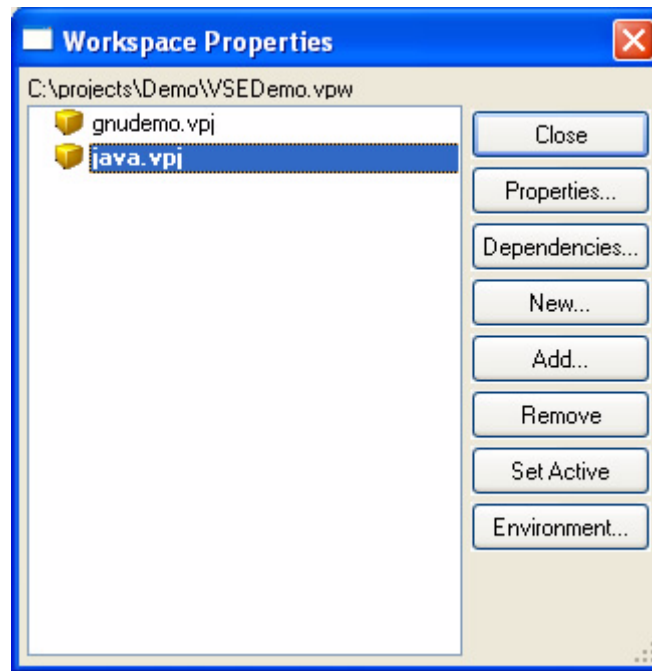
Open Other Workspace Menu Item	Description	Command
Visual Studio .NET Solution	Open a Visual Studio .NET Solution.	<b>workspace_open_visualstudio</b>
Visual C++ Workspace	Open a Visual C++ Workspace.	<b>workspace_open_visualcpp</b>
Visual C++ Embedded Workspace	Open a Visual C++ Embedded Workspace.	<b>workspace_open_visualcppembedded</b>
Tornado Workspace	Open a Tornado Workspace.	<b>workspace_open_tornado</b>
Ant XML Build File	Open an Ant XML Build File.	<b>workspace_open_ant</b>
JBuilder Project	Open a JBuilder® Project.	<b>workspace_open_jbuilder</b>
Xcode Project	Open a Xcode Project.	<b>workspace_open_xcode</b>
Workspace from CVS	Checkout and open a workspace from CVS.	<b>cvs-open-workspace</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Project** menu items.

### Workspace Properties Dialog

To list projects in the current workspace, add or remove projects from the current workspace, or to set the active project, use the Workspace Properties dialog box. The dialog, pictured below, can be accessed from the main menu by choosing **Project > Workspace Properties**.

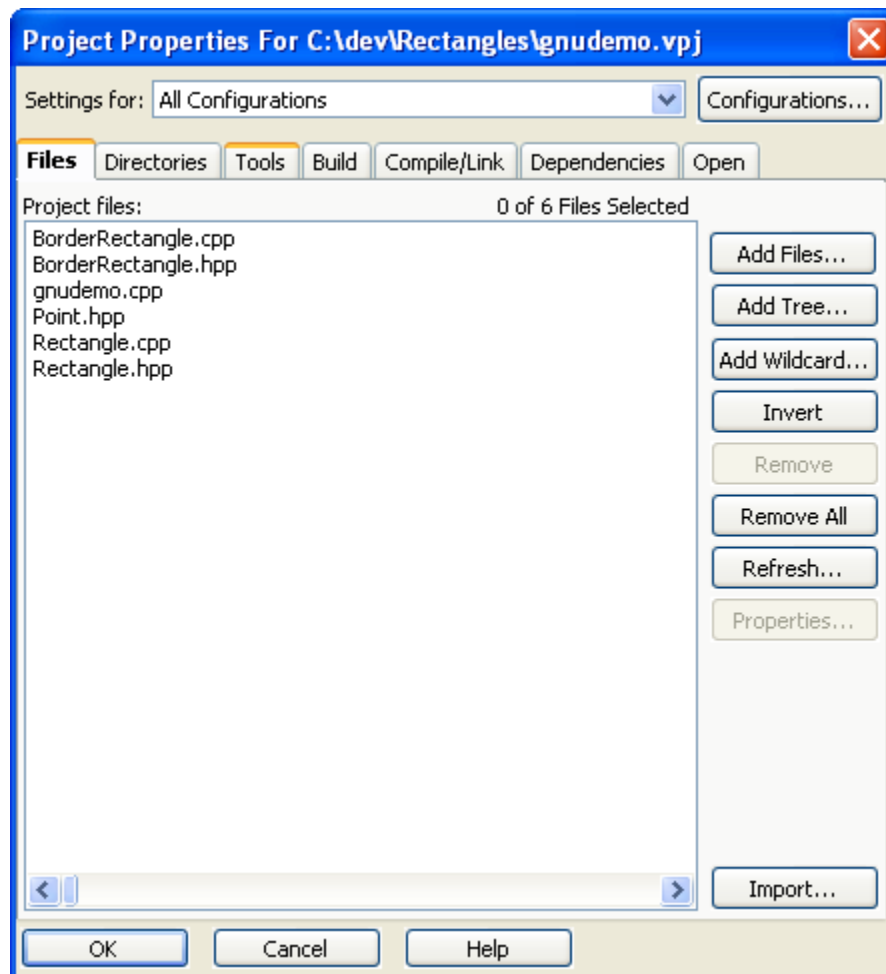


The following options are available:

- **Properties** - Displays project properties for the selected project.
- **New** - Allows you to add a new project to the current workspace.
- **Add** - Allows you to add an existing project to the current workspace.
- **Remove** - Removes the selected project from the workspace.
- **Set Active** - Sets the selected project active.
- **Environment** - Displays the Workspace Environment Options dialog box, allowing you to set environment variables. For more information on setting environment variables, see [Environment Variables](#).

## Project Properties Dialog

The Project Properties dialog, shown below, is used to manage and edit many settings for the current project. To access this dialog, choose **Project > Project Properties** or use the **project\_edit** command. You can also right-click within the Projects tool window and select **Project Properties**.



Click and drag the dialog box's edges to resize it. Both the size and position of the dialog are remembered between editing sessions. The buttons on the Project Properties dialog are described below (see [Project Properties Dialog - General Options](#)). Other options are categorized into the following tabs. Click on an item to go to that section in the documentation.

- [Files Tab](#)
- [Directories Tab](#)
- [Tools Tab](#)
- [Build Tab](#)
- [Compile/Link Tab](#)
- [Dependencies Tab](#)
- [Open Tab](#)

## Project Properties Dialog - General Options

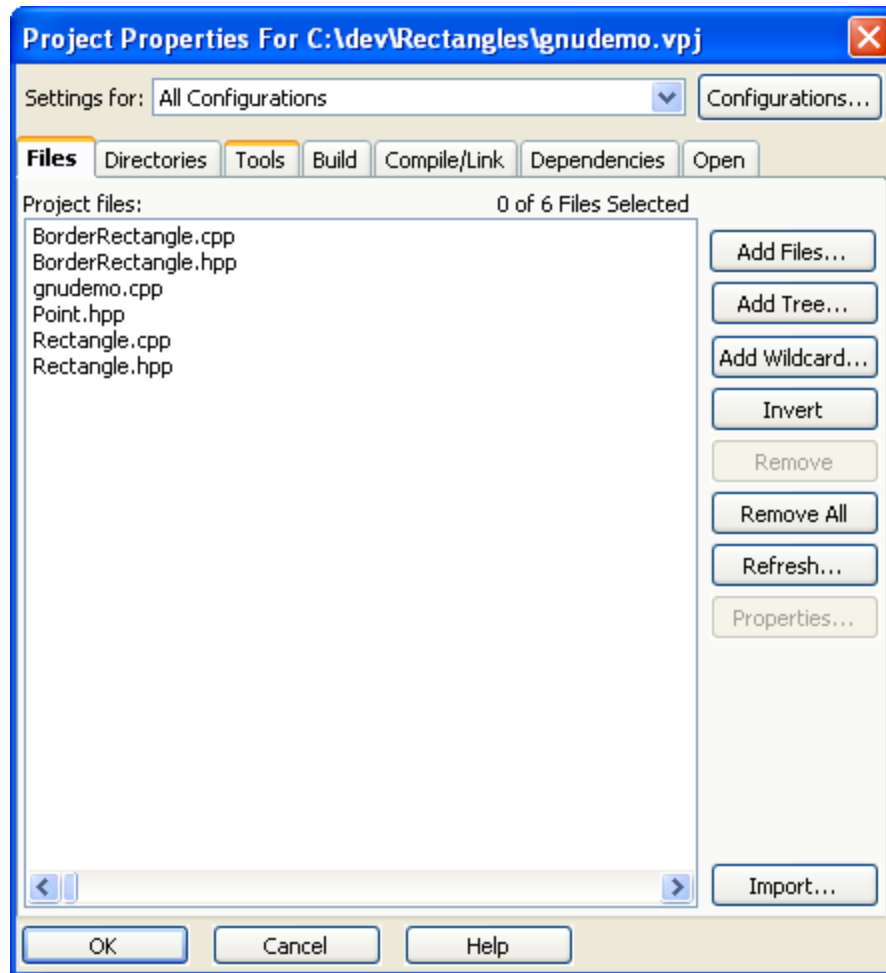
The following options are available at the top of the dialog:

- **Settings for** - Allows you to select which configuration to modify. The **All Configurations** option allows you to change the settings for all the configurations. All settings in the Project Properties dialog are per configuration except the working directory, open command, and filters.

- **Configurations** - Click this button to view, add, or delete configurations. See [Project Configurations](#) for more information.

## Files Tab

The **Files** tab of the Project Properties dialog (**Project > Project Properties**) is shown below, and displays a list of the files in the current project and also allows you to remove files from projects.



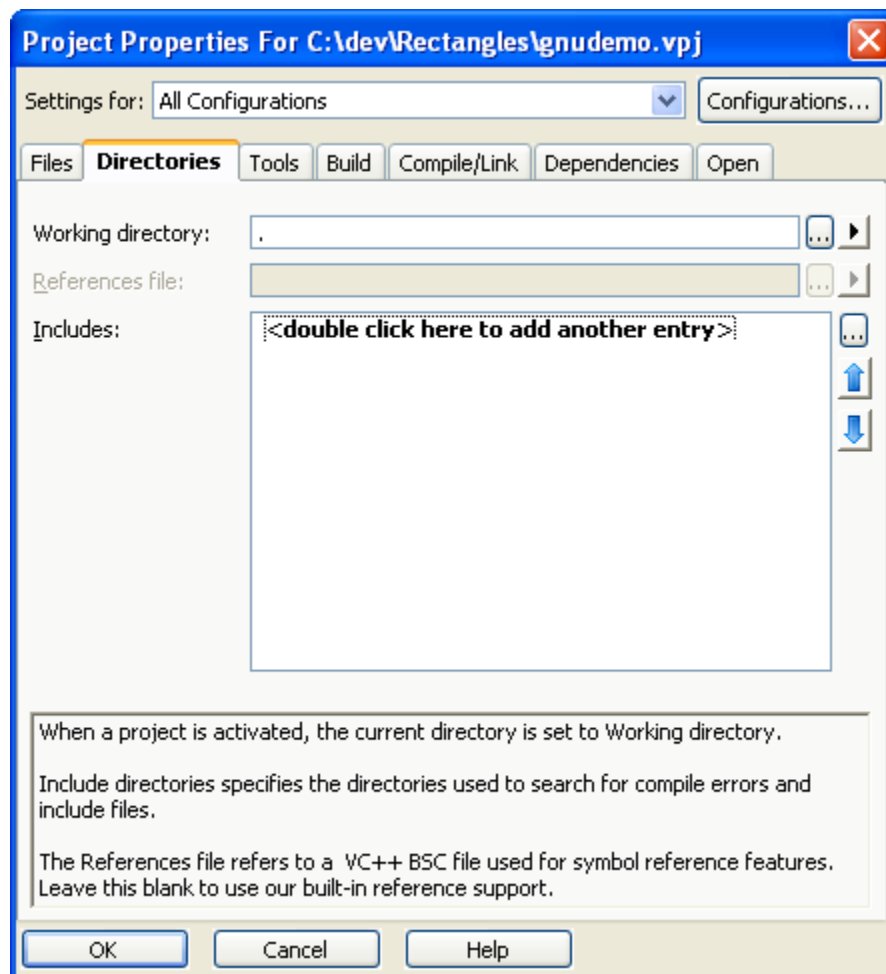
The following buttons are available:

- **Add Files** - Adds one or more existing files from a single directory to the project.
- **Add Tree** - Prompts for one or more wildcard file specifications separated with semicolons (\*.c;\*.cpp;\*.h) and searches through a directory tree adding the files that are found to the project. To search directories recursively, select the **Recursive** option.
- **Add Wildcard** - Adds one wildcard filespec to the project. For example, you could add a wildcard such as \*.java to your project. You will get duplicate entries if you repeat wildcards or add files that are already included by a wildcard.
- **Invert** - Inverts the selected items in the **Project files** list box.
- **Remove** - Removes selected items from the project.

- **Remove All** - Removes all files from the project.
- **Refresh** - Provides an easy way to remove files that do not exist from the project.
- **Properties** - Only checked when the current file has wildcard characters. Displays various supported options for a wildcard specification.

## Directories Tab

The **Directories** tab of the Project Properties dialog box (**Project > Project Properties**), shown below, allows you to set the working directory, references file, and include file search directories for the current project.



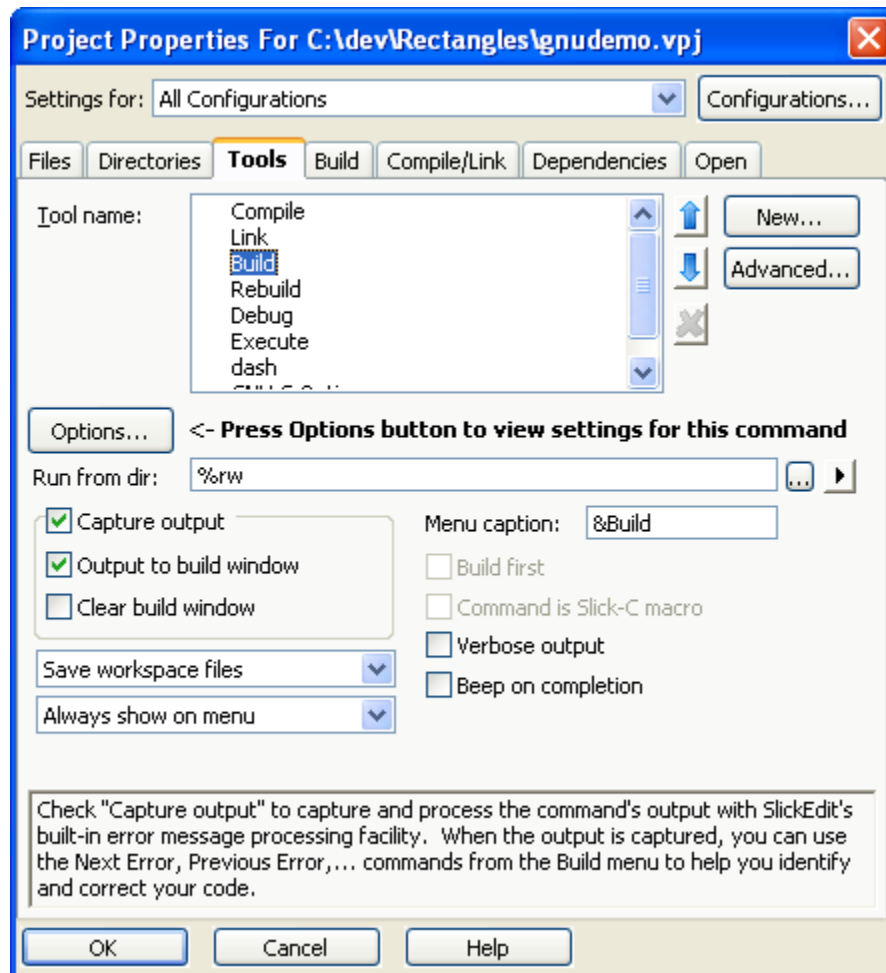
The following information describes the available fields and settings:

- **Working directory** - When a project is set active, the current directory is set to the working directory (if specified). This information is stored per project and not per configuration. Click the button to the right of this field to browse for and specify an alternate working directory.
- **References file** - You only need to complete this text box if you are using Microsoft Visual C++ and you prefer to use a Visual C++ .bsc database file instead of the Context Tagging® database when viewing references. If you open a Visual C++ v5.0 or later workspace and you have configured Visual C++ to generate a .bsc database file, this field is automatically configured.

- **Includes** - Specifies the directories the **cursor\_error** (Alt+1) and **next\_error** (Ctrl+Shift+Down) commands will search when trying to open a file. For COBOL and High Level Assembler, this list of directories is used to find copy books or macros. You might want to add some of your own include directories here before the compiler's include directories. You can specify environment variables with the syntax **%(EnvVarName)** (see [Environment Variables](#)). Click the button to the right of this field to browse for an include directory to specify. Use the up and down arrows to move the includes up or down in the list.

## Tools Tab

The **Tools** tab of the Project Properties dialog box (**Project > Project Properties**) is used to change project commands and their properties.



The following options are available:

- **Tool name** - Contains a list of the tools/commands that can be used for projects in SlickEdit®. You can have different tools for different projects. The options on the Tools tab vary, depending on the tool name that is selected in the **Tool name** text box.

Use the **Up** and **Down** arrows to move the tools up and down in the list. This order corresponds to the order in which the tool appears on the Build menu. Click the **Delete** icon (displayed as a red "X") to remove a user-defined tool (default tools cannot be deleted).

- **New** - Click the **New** button to add a tool.
- **Advanced** - Click the **Advanced** button to change environment variables (see [Environment Variables](#)).
- **Options** - Displays the Extension Options dialog box specific to the language extension with which you are currently working. This button is only available for selected tools that support the extension-specific options. For more information on changing extension options, see the topic for your language in the chapter [Language-Specific Editing](#).
- **Command line** - Defines the command line that is set to be executed for the selected tool in the Tool name combo box. This text box is only available (and visible) for selected tools that support a command line execution. Click the buttons to the right of this text box to insert files and escape sequences (such as **%f** which inserts the current buffer name) that you can use to build your command line.
- **Run from dir** - Specifies the directory from which to run selected tool command. By default, all of the tools are run from the working directory that is specified using the **%rw** or **%rp** escape sequences, which indicate the working directory or project directory, respectively. When running programs like **ant** or **make**, this is typically set to the directory containing the makefile.
- **Capture output** - Captures and processes the output of the command with SlickEdit's built-in error message processing facility. When the output is captured, the commands **next\_error** (Ctrl+Shift+Down or **Build > Next Error**) and **prev\_error** (Ctrl+Shift+Up or **Build > Previous Error**) are used to go to the next and previous compilation error positions respectively.

**NOTE** (UNIX only) Output of text mode programs that are executed using **xterm** cannot be captured. To see the output, uncheck the Output options **Capture output** and **Output to build window**, then prefix the program name in the **Command line** field with **xterm -e** or **dos -w** (this waits for a key press).

- **Output to build window** - Specifies that the output of the command be run in the concurrent build window. The concurrent build window has the following limitations:
  - You cannot run graphical applications in the concurrent build window.
  - Only programs that use standard in and standard out to read and write data can run in the concurrent build window. This is true for most compilers.
  - Some programs which use standard in and standard out will not run properly in the concurrent build window because they use ANSI escape sequences or do not flush standard output data before prompting for input.
  - Windows 2000/NT: Alternate command shells are not supported. Only **cmd.exe** is supported.
- **Clear build window** - Clears the build window output before the command is executed. The Output tool window displays the results of the processes that are run.
- **Save none** - Saves no files before the command is executed.
- **Save current file** - Saves the current file before the command is executed.
- **Save all files** - Saves all files before the command is executed.
- **List modified files** - Displays a selection list of modified files, which allows you to choose files to save before the command is executed.
- **Save workspace files** - Saves modified workspace files before the command is executed.
- **Always show on menu** - Always shows the command on the Build menu.



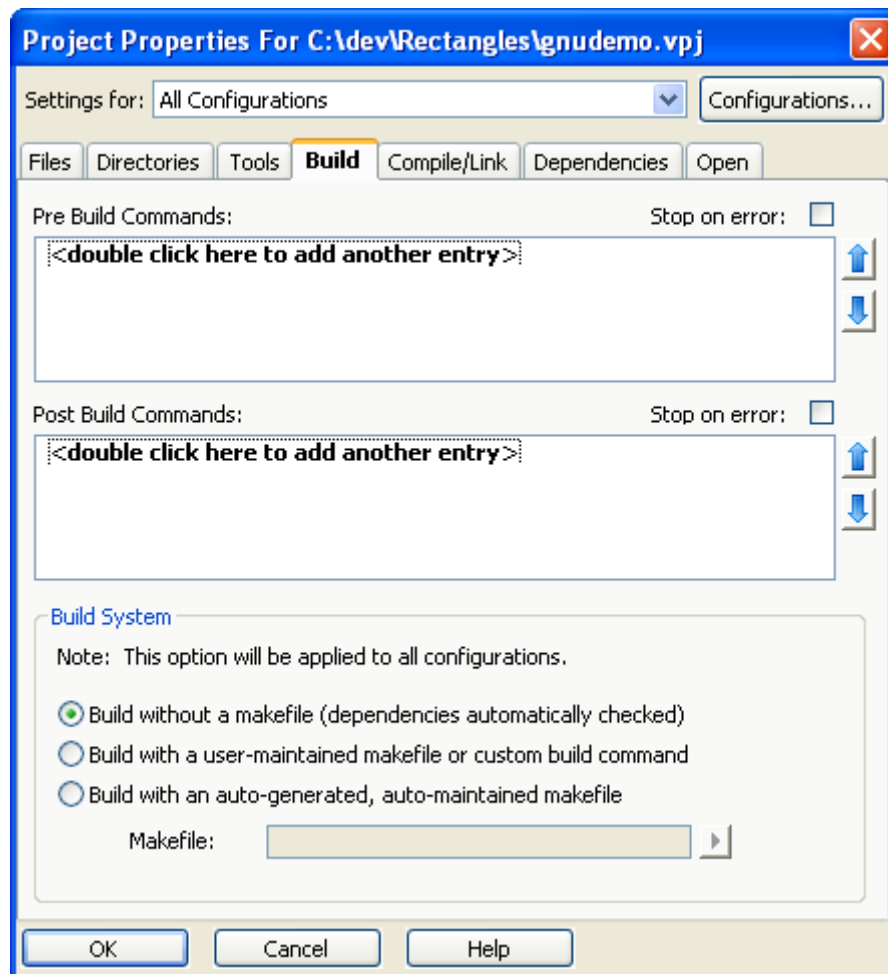
- **Hide if no command line** - Hides the menu item if the command line is blank. This is useful for saving space on the Build menu for blank command lines.
- **Never show on menu** - Never shows the command on the Build menu.
- **Application type** - Used to indicate the type of application as which a Java project should be executed. This primarily affects the Execute and Debug commands for Java applets and j2me projects.
- **Menu caption** - Defines the menu item text which appears on the Build menu. Prefix the selection character with an ampersand (&) to choose the selection character. If you have run out of selection letters, try using numbers. For example, "&1MyTool" picks "1" as the selection character.
- **Build first** - Executes the build command before the selected tool/command. If the build completes with a non-zero return code, this command is not executed. This option requires the use of the **vsbuild** utility (see [Using VSBUILD to Compile](#)) and will not work for build commands that do not return a valid return code. If the build command returns a zero return code, the command is executed even if the build actually failed.

#### NOTES

- **Windows:** Under Windows NT, batch programs return the return code of the last command executed. For commands which will not execute in the concurrent build window, prefix the command with **start** (for example: **start debug\myprogram.exe**).
- **UNIX:** If the build command is a shell script, make sure it returns a zero return code for a successful build and a non-zero return code for an unsuccessful build. For commands which will not execute in the concurrent build window, prefix the command with **xterm -e** (for example: **xterm -e debug/myprogram**).
- **Command is Slick-C macro** - Specifies that the command line is a Slick-C® macro as opposed to an external program.
- **Verbose output** - Specifies that the **vsbuild** utility is used to build the projects (see [Using VSBUILD to Compile](#)). Detailed information about the commands that are executed will be output during the build.
- **Beep on completion** - Specifies that the **vsbuild** utility is used to build the projects and to sound a status beep upon completion (see [Using VSBUILD to Compile](#)). A single beep indicates a successful build. Two beeps indicate an error occurred.
- **Run in an X terminal** - (UNIX only) Runs the command in an X terminal. This is useful for running programs which are full screen console applications such as vi.

## Build Tab

The **Build tab** of the Project Properties dialog (**Project > Project Properties**), pictured below, allows you to run programs and/or execute commands before or after a build. You can run different programs and commands for different projects as the information is stored per-configuration. The contents of this tab are disabled for extension-based projects.



The list below describes the settings that are available. For more in-depth information, see [Build System Options](#).

- **Pre- and Post-Build Commands** - Each line can contain a program to execute a command. For example, the **set** command could be used to set environment variables. Double-click on the text as indicated in the text boxes to add commands. Use the **Up** and **Down** arrows to the right of the text boxes to move the commands up and down in the list. The order corresponds to the order in which the command will be run.
- **Stop on error** - When this option is checked and the current project depends on other projects, the **vsbuild** utility (see [Using VSBUILD to Compile](#)) will be used to build the projects and check for error codes. When the **vsbuild** program detects an error, it does not continue building other dependencies.

**NOTE** (Windows only) Under Windows 95 or later, **vsbuild** cannot detect error codes returned from a batch program.

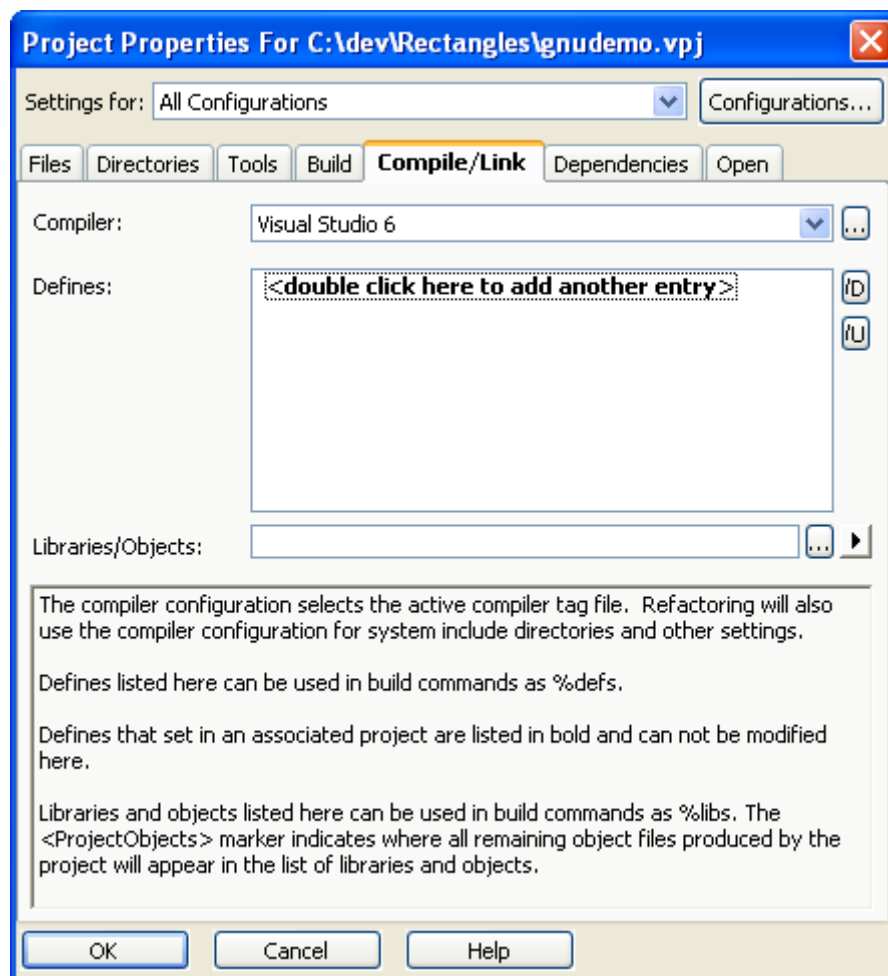
- **Build System Options** - These build methods apply to GNU C/C++ projects only and affect all configurations. With these options, you will not need to convert the current build methods to use the GNU debugger; you can select one of these methods when you create a new GNU C/C++ Wizard project.

- **Build without a makefile (dependencies automatically checked)** - Automatically checks dependencies and does not generate a makefile. Instead, the **vsbuild** utility (see [Using VSBUILD to Compile](#)) determines what should be compiled dynamically. This option is useful when you are not concerned with how the build gets done. Make sure the project include directories are set up correctly (**Project > Project Properties**, select the [Directories Tab](#)) so include files may be found.
- **Build with a user-maintained makefile or custom build command** - Sets the build command to **make** and does not generate a makefile. The build command can be changed from the Tools tab of the Project Properties dialog box (see [Tools Tab](#)). Select this option when you already have your own method for building the source.
- **Build with an auto-generated, auto-maintained makefile** - Automatically generates a makefile and updates when files are added to the project. This option is useful when you need a makefile and do not want to use the built-in **vsbuild** utility (see [Using VSBUILD to Compile](#)). Specify the path to the makefile in the **Makefile** field. Make sure the project include directories are set up correctly (**Project > Project Properties**, select the [Directories Tab](#)) so include files may be found.

To start a build from outside the application, execute the following command where **make** is the name of the make program, **Makefile** is the name of the makefile, and **ConfigName** is the name of the configuration: **make -f Makefile CFG=ConfigName**.

## Compile/Link Tab

The Compile/Link tab of the Project Properties dialog (**Project > Project Properties**), shown below, is used to specify project compilation and linking options.



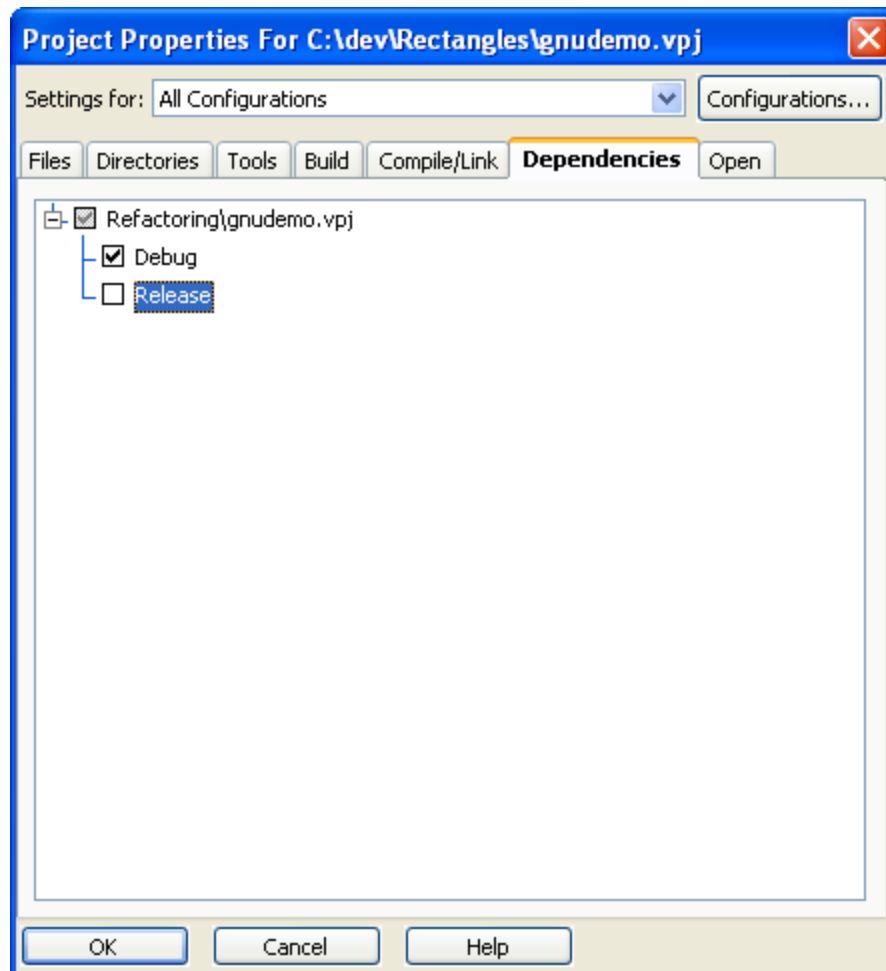
The following settings are available:

- **Compiler** - By default, the compiler configuration is set to the active compiler tag file based on the active project. To change the compiler or its configuration and properties, click the arrows to the right of the **Compiler** field. See [C/C++ Compiler Settings](#) for more information.
- **Defines** - Defines listed here can be used in build commands as **%defs**. Double-click as indicated to add a define. Click the **/D** icon to enter a macro. Click the **/U** icon to un-define a macro. Defines that set in an associated project are listed in bold and can not be modified here.
- **Libraries/Objects** - Libraries and objects listed here can be used in build commands as **%libs**. To add libraries and objects, click the button to the right of this field. This displays the Link Order dialog box. The **<ProjectObjects>** marker indicates where all remaining object files produced by the project will appear in the list of libraries and objects. Use the arrows to the right of the text box to move the libraries/objects up and down in the list, or use the red "X" icon to remove a library or object.

## Dependencies Tab

The Dependencies tab on the Project Properties dialog (**Tools > Project Properties**), pictured below, allows you to define a relationship between two projects, causing the dependent project to be built after the

projects it depends on. This ensures that elements in a depended-on project are up-to-date prior to building the dependent project. See [Defining Project Dependencies](#) for more information.



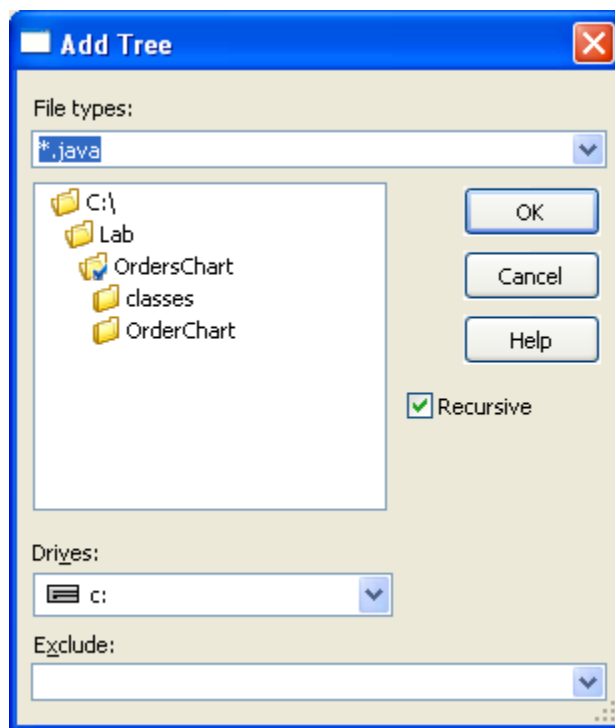
## Open Tab

The Open tab of the Project Properties dialog (**Project > Project Properties**) lets you enter commands that are executed when the project is activated. This information is stored per project, not per configuration. This tab is disabled for extension-based projects. For instructions on entering commands on this tab, see [Specifying Open Commands](#).

## Add Tree Dialog

The Add Tree dialog is used to add files in a directory or directory tree to a tag file or a project. It also gives you the ability to use wildcards so that you can add only files with certain extensions.

This dialog is displayed when you click the **Add Tree** button on the [Context Tagging® - Tag Files Dialog](#) or the [Files Tab](#) of the Project Properties dialog.



The dialog contains the following elements:

- **File types** - The **File types** combo box lets you select from predefined wildcard specifications or you can type your own. Each file spec should be separated with semicolons (\*.c;\*.cpp;\*.h). For example, to include only Java files, select **\*.java** from the predefined list. To include all files in a directory, type the wildcard **\***.
- **Directory list** - The directory list box lets you pick the directory from which to include files.
- **Drives** - Select the disk drive to look in.
- **Exclude** - Use this combo box to exclude paths, files, or file types from the specified directory using wildcards. For examples, see [Exclusion Examples](#) below.
- **Recursive** - If checked, the selected directory will be searched recursively.

## Exclusion Examples

The table below shows some examples of filespec exclude patterns that you can use in various **Exclude** combo boxes within SlickEdit®.

Example	Description
*math*.cpp	Exclude any .cpp with "math" in the file name.
readme.txt	Exclude all files named readme.txt.
*.a	Exclude any file with extension .a.
*\CVS\	Exclude any files in paths named "CVS".
/home/build/debug	Exclude all files in this path name.
*demo*	Exclude any file or path with "demo" in the name.







## Build

---

This section describes items on the **Build** menu. Currently, the section [Building and Compiling](#) contains all of the information about building and the dialogs and options that are available.

### Build Menu

The **Build** menu is language-specific and can have alternate options depending on the language in which the project is written.

Build Menu Item	Description	Command
Next Error	Processes the next compiler error message.	<b>next-error</b>
Previous Error	Processes the previous compiler error message.	<b>prev-error</b>
Go to Error or Include	Parses the error message or filename at the cursor and places cursor in file.	<b>cursor-error</b>
Clear All Error Markers	Removes all error markers in all files.	<b>clear-all-error-markers</b>
Configure Error Parsing	Configures regular expressions used to search for compiler messages.	<b>configure-error-regex</b>
Stop Build	Sends break signal to the build tab.	<b>stop-process</b>
Show Build	Starts or activates the build tab.	<b>start-process</b>
Build Automatically on Save	When enabled the project is built each time the workspace is saved.	<b>project-toggle-auto-build</b>



## Debug

This section describes items on the **Debug** menu and associated dialogs and tool windows. For more information about debugging, see [Running and Debugging](#).

### Debug Menu

The **Debug** menu contains debugging-related operations and options. The table below summarizes these items.

Debug Menu Item	Description	Command
Windows	Displays debug window toolbars. See <a href="#">Debug Windows Menu</a> .	N/A
Start	Starts debugger.	<b>project_debug</b>
Suspend	Suspends execution.	<b>debug_suspend</b>
Stop Debugging	Stops debugging the program.	<b>debug_stop</b>
Restart	Restarts the program.	<b>debug_restart</b>
Attach Debugger	Attach debugger to a process or remote server.	N/A
Detach	Detach from target process and allow application to continue running.	<b>debug_detach</b>
Step Into	Steps into the next statement.	<b>debug_step_into</b>
Step Over	Steps over the next statement.	<b>debug_step_over</b>
Step Out	Steps out of the current function.	<b>debug_step_out</b>
Step Instruction	Steps one instruction at a time.	<b>debug_step_instr</b>
Run to Cursor	Runs the program to the line containing the cursor.	<b>debug_run_to_cursor</b>
Show Next Statement	Displays the source line for the instruction pointer.	<b>debug_show_next_statement</b>
Set Instruction Pointer	Set the instruction pointer to the current line.	<b>debug_set_instruction_pointer</b>
Show Disassembly	Toggle display of disassembly.	<b>debug_toggle_disassembly</b>
Toggle Breakpoint	Toggles a breakpoint at the current line.	<b>debug_toggle_breakpoint</b>
Delete All Breakpoints	Deletes all debugger breakpoints.	<b>debug_clear_all_breakpoints</b>
Disable All Breakpoints	Disables all debugger breakpoints.	<b>debug_disable_all_breakpoints</b>
Add Watch	Add a watch on the variable under the cursor.	<b>debug_add_watch</b>
Set Watchpoint	Set a watchpoint on the variable under the cursor.	<b>debug_add_watchpoint</b>
Debugger Options	Displays the Debugger Options dialog. See <a href="#">Setting Debug Options</a> for detailed information.	<b>debug_props</b>

## Debug Windows Menu

The **Debug > Windows** menu items activate the debugging tool windows. The table below summarizes these items. See also [Debugger Tool Windows](#) for more information.

Debug Windows Menu Item	Description	Command
Call Stack	Activates the Call Stack window.	<b>activate_call_stack</b>
Locals	Activates the Locals window.	<b>activate-locals</b>
Members	Activates the window which displays member variables.	<b>activate-members</b>
Autos	Activates the Autos window.	<b>activate-autos</b>
Watch	Activates the Watch window.	<b>activate-watch</b>
Threads	Activates the Threads window.	<b>activate-threads</b>
Breakpoints	Activates the Breakpoints window.	<b>activate-breakpoints</b>
Registers	Activates the Registers window.	<b>activate-registers</b>
Memory	Activates the Memory window.	<b>activate-memory</b>
Classes	Activates the Classes window.	<b>activate-classes</b>

## Attach Debugger Menu

The **Debug > Attach Debugger** menu items are summarized in the table below. See [Multiple Session Debugging](#) for more information.

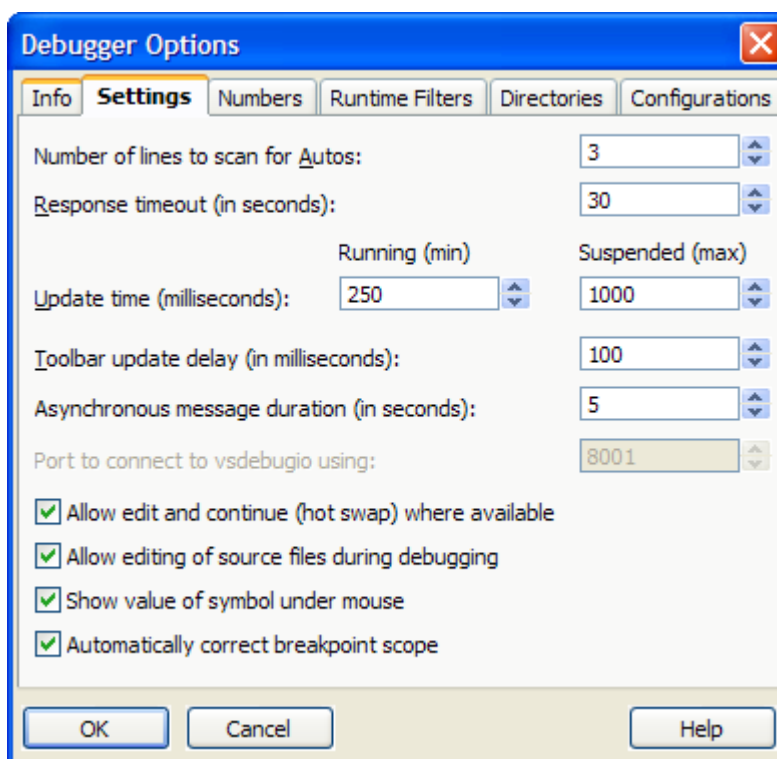
Attach Debugger Menu Item	Description	Command
Attach to Running Process using GDB	Attach debugger to a running process using GDB.	<b>debug_attach gdb</b>
Attach to Running Process using .NET	Attach debugger to a running process using .NET debugger.	<b>debug_attach dotnet</b>
Attach to Remote Process using GDB	Attach debugger to a remote GDB server or executable with GDB stub.	<b>debug_remote gdb</b>
Attach to Java Virtual Machine	Attach to a Java virtual machine executing remotely.	<b>debug_attach jdwp</b>
Debug Other Executable	Step into a program using GDB.	<b>debug_executable gdb</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Debug** menu items.

## Debugger Options - Settings Tab

The Settings tab on the Debugger Options dialog, shown below, is designed to increase debugger performance and is intended for advanced users. To access this dialog, choose **Debug > Debugger Options**, then select the **Settings** tab.



The following options are available:

- **Number of lines to scan for Autos** - This is the number of lines, starting with the current line, to scan for symbols to evaluate and display in the Autos tab of the Debug Variables toolbar. Default is 3.
- **Response time out (in seconds)** - Number of seconds to wait for a connection or response from the underlying debugger system before giving up.
- **Update time (milliseconds)** - The minimum and maximum amounts of time (in milliseconds) to spend polling for asynchronous events coming from the underlying debugger system. Decreasing these times will make the editor more responsive in some cases during debugging, however, increasing these times can improve overall performance. These values are also used to determine the frequency with which to poll for events. The minimum time (default 250 ms) is used when the application is running, and the maximum setting (default 1000 ms) is used when the application is suspended.
- **Toolbar update delay (in milliseconds)** - The amount of time to wait before updating the debugger toolbars. This is done to improve debugger performance and decrease overhead and redraws when single stepping through code. Set this value to 0 to force an immediate update.
- **Asynchronous message duration (in seconds)** - Certain informative messages caused by asynchronous events (such as loading Classes in Java) are displayed for this period of time, then erased.

## DEBUG

- **Port to connect to vsdebugio using** - This is the TCP/IP port to connect to vsdebugio at running locally on the same machine.

For detailed information regarding the other tabs on the Debugger Options dialog, see [Setting Debug Options](#).

# Document

---

This section describes items on the **Document** menu and associated dialogs and tool windows.

## Document Menu

The **Document** menu contains items pertaining to editor windows and the current document. The table below lists a summary of these items.

Document Menu Item	Description	Command
Next Buffer	Switches to the next buffer. See <a href="#">Buffers and Editor Windows</a> .	<b>next_buffer</b>
Previous Buffer	Switches to the previous buffer. See <a href="#">Buffers and Editor Windows</a> .	<b>prev_buffer</b>
Close Buffer	Closes the current buffer. See <a href="#">Buffers and Editor Windows</a> .	<b>close_buffer</b>
List Open Files	Displays the Files tool window, which lists all buffers and allows you to activate one. See <a href="#">Files Tool Window</a> .	<b>list_buffers</b>
Select Mode	List all modes and lets you select one. See <a href="#">Language Editing Modes</a> .	<b>select_mode</b>
Tabs	Sets tab stops.	<b>gui_tabs</b>
Margins	Sets margins.	<b>gui_margins</b>
Format Paragraph	Reflows the text in the current paragraph according to the margins.	<b>reflow_paragraph</b>
Format Selection	Reflows the selected text according to the margins.	<b>reflow_selection</b>
Format Columns	Format columns according to words.	<b>format_columns</b>
Edit Javadoc Comment	Edits Javadoc comments for the current source file.	<b>javadoc_editor</b>
Comment Block	Converts selected text into block comment using box comment setup characters. See <a href="#">Commenting</a> .	<b>box</b>
Comment Lines	Converts selected lines into line comments using the line comment setup. See <a href="#">Commenting</a> .	<b>comment</b>
Uncomment Lines	Uncomments any commented lines and ignores any that isn't commented. See <a href="#">Commenting</a> .	<b>comment_erase</b>
Reflow Comment	Reflows and reformats the current block comment. See <a href="#">Reflow Comment Dialog</a> .	<b>gui_reflow_comment</b>

Document Menu Item	Description	Command
Comment Setup	Displays the Extension Options dialog open to the Comments tab, which contains settings for box and line comments. See <a href="#">Comments Tab</a> .	<b>comment_setup</b>
Comment Wrap	Toggles comment wrap on/off.	<b>comment_wrap_toggle</b>
XML/HTML Formatting	Displays the XML/HTML Formatting menu. See <a href="#">XML/HTML Formatting Menu</a> .	N/A
Indent with Tabs	Toggles indenting with tabs on/off. See <a href="#">Syntax Indent</a> .	<b>indent_with_tabs_toggle</b>
Word Wrap	Toggles word wrap on/off.	<b>word_wrap_toggle</b>
Justify	Sets/displays paragraph justification style.	<b>gui_justify</b>
Read Only Mode	Toggles read-only mode on/off.	<b>read_only_mode_toggle.</b>

## XML/HTML Formatting Menu

The **Document > XML/HTML Formatting** menu is used to enable these features in SlickEdit®. You can enable and/or disable Tag Layout and/or Content Wrap for either file type on a global basis or on a per document basis. See [XML/HTML Formatting](#) for more information.

XML/HTML Formatting Menu Item	Description	Command
Current Document Options	Displays the <a href="#">Current Document Options Dialog</a> .	<b>xml_html_document_options</b>
Enable XML Formatting	Toggles XML Formatting on/off.	<b>xml_formatting_toggle</b>
(XML) Content Wrap	Toggles XML Content Wrap on/off.	<b>xml_contentwrap_toggle</b>
(XML) Tag Layout	Toggles XML Tag Layout on/off.	<b>xml_taglayout_toggle</b>
Enable HTML Formatting	Toggles HTML Formatting on/off.	<b>html_formatting_toggle</b>
(HTML) Content Wrap	Toggles HTML Content Wrap on/off.	<b>html_contentwrap_toggle</b>
(HTML) Tag Layout	Toggles HTML Tag Layout on/off.	<b>html_taglayout_toggle</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Document** menu items.



## Files Tool Window

The Files tool window allows you to view open files, project files, and workspace files, sortable by file name or path. It includes a filter to narrow the list of files shown in the list, as well as shortcuts for basic file operations (Open, Save, etc.).

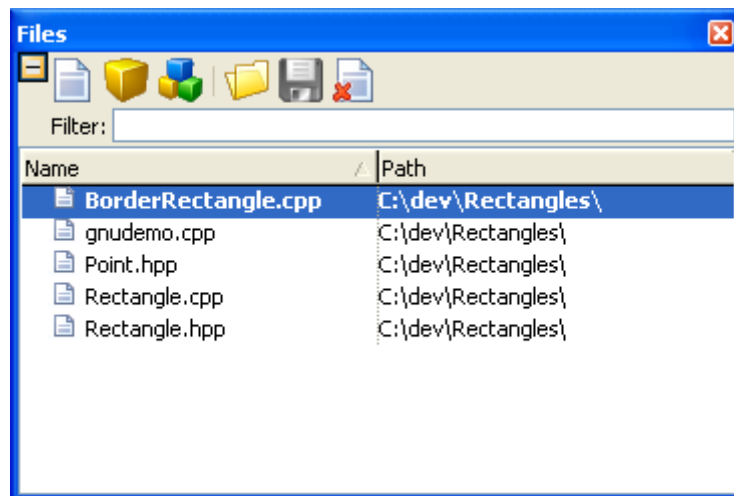
### NOTES

- The Files tool window replaces the Select a Buffer dialog found in previous versions of SlickEdit®.
- For documentation purposes, the word “files” generally includes both files and buffers.

## Accessing the Tool Window

There are several ways to access the Files tool window:

- Select **Document > List Open Files**, press **Ctrl+Shift+B**, or use the **list\_buffers** command.
- Use the **activate\_files** command.
- Toggle display of the tool window by selecting **View > Toolbars > Files**, or by using the **toggle\_files** command.



When the Files tool window is not docked, it can be dismissed by opening a file for editing or by pressing Esc. To make this dialog behave like other tool windows, right-click inside the Files list area and uncheck **Dismiss on select**.

**TIP** By docking this tool window, you have quick access for switching between files or opening other files.

## List Views

The Files tool window presents three available views for files. Use the **View** icons or the right-click menu to display:

- Files open for editing - Useful for selecting the file to edit when working on multiple files.
- Files in the active project - Useful for browsing and searching file names in the active project.

- Files in the open workspace - Useful for browsing and searching file names in the workspace.

## Working with the Files List

The bottom part of the tool window shows the Files list. The **Name** column displays, in alphabetical order, a list of file names or untitled buffers based on the selected view setting. The **Path** column displays the associated paths for the files listed. Click on either column header to sort by that column. When you click to sort, an arrow on the right side of the column header shows the ascending or descending order.

The **Filter** text box can be used to display matching file names. Files are removed from the list that do not contain the specified text. For example, if you type “ml,” the Files list is filtered to only show file names that contain the letters “ml,” as they appear in that order, anywhere in the file name.

To enable Fast Prefix Matching inside the Filter text box, right-click inside the Files list area and select **Prefix match**. When prefix matching is on, matching starts at the beginning of the word. For example, if you type “d,” the Files list is filtered to only show file names that begin with the letter “d.”

When the focus is not in the Filter text box, you can incrementally search the list of file names by typing the first few characters of the name. If you pause for a few seconds, the search is reset, and you can search for a different file name just by typing the first few characters again. You do not need to press Backspace, or reselect the first item in the list. Regardless of which item is selected, incremental search starts at the top of the list. For example, if items are sorted in descending alphabetical order, the incremental search starts at the top of the list, which would be the file that would appear last, if sorted in ascending alphabetical order.

**TIP** You can collapse display of the icons and the Filter text box by clicking the **Minus** icon in the top-left corner of the tool window. The collapsed area is replaced with a message that states your current view and filter settings. To view the icons and Filter text box again, click the **Plus** icon. Collapsing the icons and filter gives your window a cleaner look and more room for file listings.

## Opening Files for Editing

The name of the file that has current focus in the editor is displayed in a bold font style. The tool window provides several ways to a file for editing:

- Press **Enter** or **Alt+E**.
- Double-click on the file to be opened.
- Click the **Open a File for Editing** icon.
- Right-click and select **Open**.

**TIP** If there is no selection when you invoke an Open operation, the Open dialog is displayed from which you can specify a file to open.

## Saving Modified Files

Modified files are listed in a red font, and when selected, they have a red highlight. A Disk icon to the left of the file name acts as another visual indicator for modified files and allows for a quick save.

**NOTE** The Project and Workspace views do not display modified file indicators.

The Files tool window provides several ways to save modified files:

- Press **Ctrl+S**, **Alt+S**, or **Alt+W**.

- Click the **Disk** icon to the left of the file name.
- Click the **Save Selected File(s)** icon.
- Right-click and select **Save**.

#### TIPS

- When you invoke a Save operation, if a file is selected, it is simply saved. If an untitled buffer is selected, the Save As dialog is displayed from which you can save it with a specified name. If both a file and an untitled buffer are selected, the file will be saved, and the Save As dialog is displayed in order to save the untitled buffer.
- The red highlight color for modified files can be changed by specifying a different background color for the Modified Line screen element (**Tools > Options > Color**). Note that this is the same element that specifies coloring in tree controls such as DIFFzilla®, so change with caution. See [Setting Colors for Screen Elements](#) for more information.

## Closing Files

When you close a file, if you are using the option **One file per window** (**Tools > Options > General > General tab**), which is on by default, all windows displaying the buffer are closed as well. You are also prompted to save modified buffers.

The Files tool window provides several ways to close files:

- Press **Delete**, **Alt+C**, or **Alt+D**.
- Right-click and select **Close**.
- Click the **Close Selected File(s)** icon.

## Files Tool Window Interface

The elements on the Files tool window are described as follows, from left to right and top to bottom:

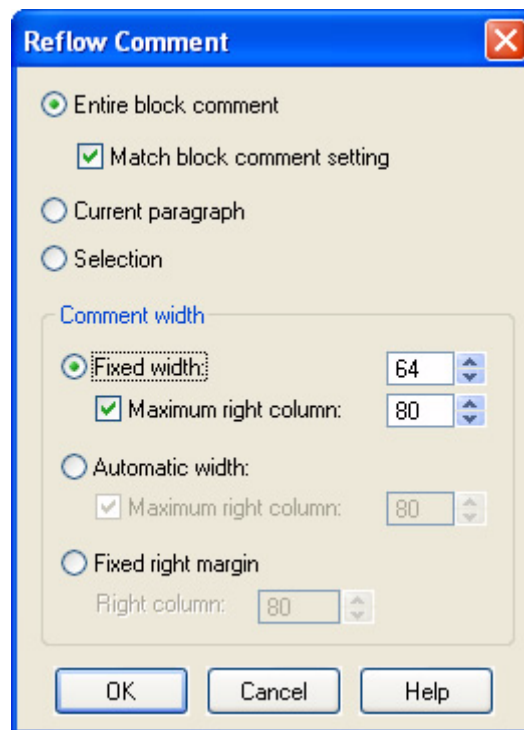
- **Minus/Plus icon** - The Minus icon collapses the display of the icons and Filter text box of the tool window. It changes to a Plus icon when the area is collapsed, and a message states your current view and filter settings.
- **View Open Files icon** - Displays all files and buffers that are currently open in the editor. This view can also be specified by using the right-click menu inside the Files list.
- **View Files in the Current Project icon** - Switches the view from the set of open files to the set of all files in the current project, regardless of whether they are open or not. This view does not show an indicator for modified files. This view can also be specified by using the right-click menu inside the Files list. See [Workspaces and Projects](#) for more information about working with projects.
- **View Files in the Current Workspace icon** - Switches the view to the set of all files in the current workspace. This view can also be specified by using the right-click menu inside the Files list. See [Workspaces and Projects](#) for information about workspaces.
- **Open a File for Editing icon** - Brings the selected file to focus in the editor. If a project or workspace file is selected that is not currently open in the editor, that file is opened for editing. If there is no selection, the Open dialog is displayed from which you can specify the file to open. This operation can also be specified by using the right-click menu inside the Files list. See [Opening Files for Editing](#) for more information.
- **Save Selected File(s) icon** - If a file is selected, it is simply saved. If an untitled buffer is selected, the Save As dialog is displayed from which you can save it with a specified name. If both a file and

an untitled buffer are selected, the file will be saved, and the Save As dialog is displayed in order to save the untitled buffer. This operation can also be specified by using the right-click menu inside the Files list. See [Saving Modified Files](#) for more information.

- **Close Selected File(s) icon** - Closes the selected file(s) in the editor, which in turn, removes the names from the Files window. If you are using the option **One file per window** (on by default), all windows displaying the buffer are closed as well. You are prompted to save modified buffers. This operation can also be specified by using the right-click menu inside the Files list. See [Closing Files](#) for more information.
- **Filter** - This text box can be used to display matching file names. Right-click inside the Files list area to enable **Prefix match** inside the Filter text box. When the focus is not in the Filter text box, you can incrementally search the list of file names by typing the first few characters of the name. See [Working with the Files List](#) for more details about filtering and searching the Files list.
- **Files list** - The Files list is divided into two columns:
  - **Name column** - Displays a list of file names or untitled buffers based on the selected view setting. Items are listed in alphabetical order. Click on the **Name** column header to sort by this column. When you click to sort, an arrow on the right side of the column header shows the ascending/descending order. The name of the file that has current focus in the editor is displayed in a bold font style. Modified files are listed in a red font, and when selected, they have a red highlight. A **Disk icon** to the left of the file name acts as another visual indicator for modified files and allows for a quick save.
  - **Path column** - Displays the corresponding paths to the files/buffers listed. Click on the **Path** column header to sort by this column. When you click to sort, an arrow on the right side of the column header shows the ascending/descending order.

## Reflow Comment Dialog

The Reflow Comment dialog (**Document > Reflow Comment**), shown below, is used to reflow block comments, paragraphs, or a selection of the current file.



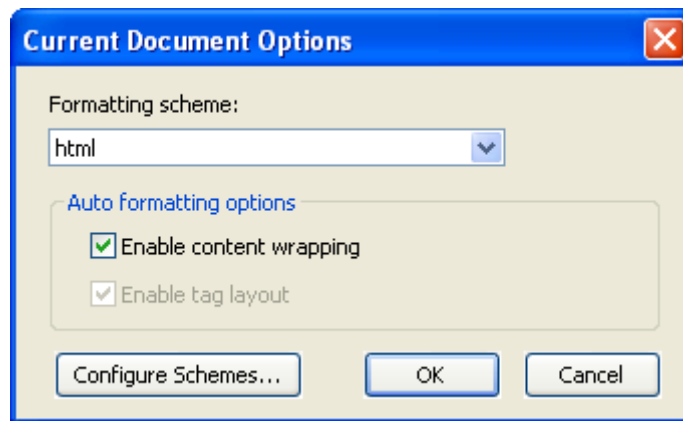
The following options are available:

- **Entire block comment** - If selected, reflows an entire block comment based on the current width and border settings for the block comment.
- **Match block comment setting** - If selected, forces the borders to conform to the comment settings (**Document > Comment Setup** - see [Comments Tab](#)).
- **Current paragraph** - If selected, reflows the current paragraph within the block comment.
- **Selection** - If selected, reflows a selection within a block comment paragraph based on current settings.
- **Comment width** - Select one of the width options to reflow a block comment to the margins or the width that you specify in these fields. See [Comment Wrap Tab](#) for information on these options.

For more information about comments, see [Commenting](#).

## Current Document Options Dialog

The Current Document Options dialog allows you to enable/disable aspects of XML or HTML Formatting for just the current document. It can be displayed by selecting **Document > XML/HTML Formatting > Current Document Options**, or by using the `xml_html_document_options` command.



The dialog contains the following:

- **Formatting scheme** - This drop-down specifies the formatting scheme applied to this document. Choose from the list of available schemes.
- **Auto formatting options** - These are the aspects of XML/HTML Formatting that can be enabled/disabled for the current document.
- **Configure Schemes button** - Allows you to modify or create a new scheme to apply to the current document.

For more detailed information, see [Enabling/Disabling for the Current Document](#).

## Macro

This section describes items on the **Macro** menu and associated dialogs and tool windows.

### Macro Menu

The table below describes each item on the **Macro** menu and its corresponding command. For more information about working with macros, see [Recorded Macros](#), [Programmable Macros](#), and the *Slick-C® Macro Programming Guide*.

Macro Menu Item	Description	Command
Load Module	Loads a macro source module.	<b>gui_load</b>
Unload Module	Unloads a Slick-C® macro file from the state file.	<b>gui_unload</b>
Record Macro	Starts recording Slick-C language macro based on the editor features you use.	<b>record_macro_toggle</b>
Execute last-macro	Runs last recorded macro.	<b>record-macro-end-execute</b>
Save last-macro	Saves the last recorded macro under a name you specify. See <a href="#">Save Macro Dialog</a> .	<b>gui_save_macro</b>
List Macros	Lists saved, recorded macros. See <a href="#">List Macros Dialog</a> .	<b>list_macros</b>
Set Macro Variable	Allows you to set global macro variables. See <a href="#">Variable Editor Dialog</a> .	<b>gui_set_var</b>
Go to Slick-C Definition	Opens a macro source file and places your cursor on the definition of a macro symbol.	<b>gui_find_proc</b>
Find Slick-C Error	Places your cursor on the macro source line which caused the last interpreter run-time error.	<b>find_error</b>
New Form	Opens a new form for editing with the Dialog editor.	<b>new_form</b>
Open Form	Opens an existing or new form for editing with the Dialog editor.	<b>open_form</b>
Selected Form	Displays edited form window currently selected.	<b>show-selected</b>
Load and Run Form	Loads form, loads Slick-C code, and runs the currently selected/edited form.	<b>run_selected</b>
Grid	Sets form grid settings. This affects the distance displayed between the dots on a form that is being edited. See <a href="#">Grid Settings Dialog</a> .	<b>gui_grid</b>
Menus	Lists all menus and allows you to edit, create, delete, or show menus. Provides access to the Menu Editor dialog box. See <a href="#">Menu Editor Dialog</a> .	<b>open_menu</b>

Macro Menu Item	Description	Command
Insert Form or Menu Source	Inserts source code into current file for a form or menu you specify.	<b>insert_object</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Macro** menu items.

### Save Macro Dialog

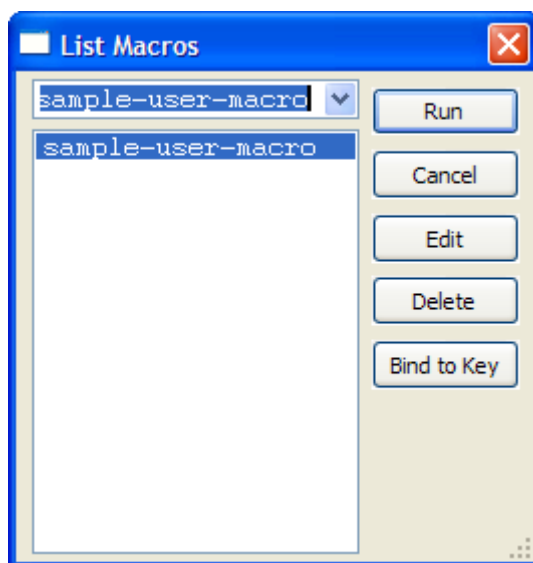
The Save Macro dialog appears when you end macro recording, or when you select **Macro > Save last-macro**. You can also display the dialog by using the **gui\_save\_macro** command. The following options are available:

- **Macro Name** - Specifies the name for the recorded macro.
- **Requires editor control** - Select this option if your macro can only operate if the target is an editor control.
- **Allow in read only mode** - Select this option if your macro does not modify the current buffer.
- **Allow when window is iconized** - You will probably NOT want this option selected if your macro modifies the current buffer. Whether to select this option is more a matter of personal taste.
- **Allow in non-MDI editor control** - Select this option if your macro should be allowed in a non-MDI editor control. This is typical for commands which require an editor control but do not open or close editor windows/buffers.
- **Save** - Saves the recorded macro and displays the Key Bindings dialog so you can bind the macro to a key sequence. See [Binding Recorded Macros to Keys](#) for more information.
- **Edit** - (Alt+E) Displays the macro source code in a new editor window. To save it, choose **Macros > Save last-macro**. The Key Bindings dialog will not appear automatically if you use this save operation. Instead, to bind the macro to a key, use the menu item **Macro > List Macros**. The **Edit** button is not available for saved macros. See [Saving and Editing Recorded Macros](#) for more information.
- **Delete** - Deletes the recorded macro.

### List Macros Dialog

The List Macros dialog is used to view and work with a list of macros you have recorded. It is accessed by clicking **Macro > List Macros** on the main menu, or by using the **list\_macros** command on the SlickEdit command line.



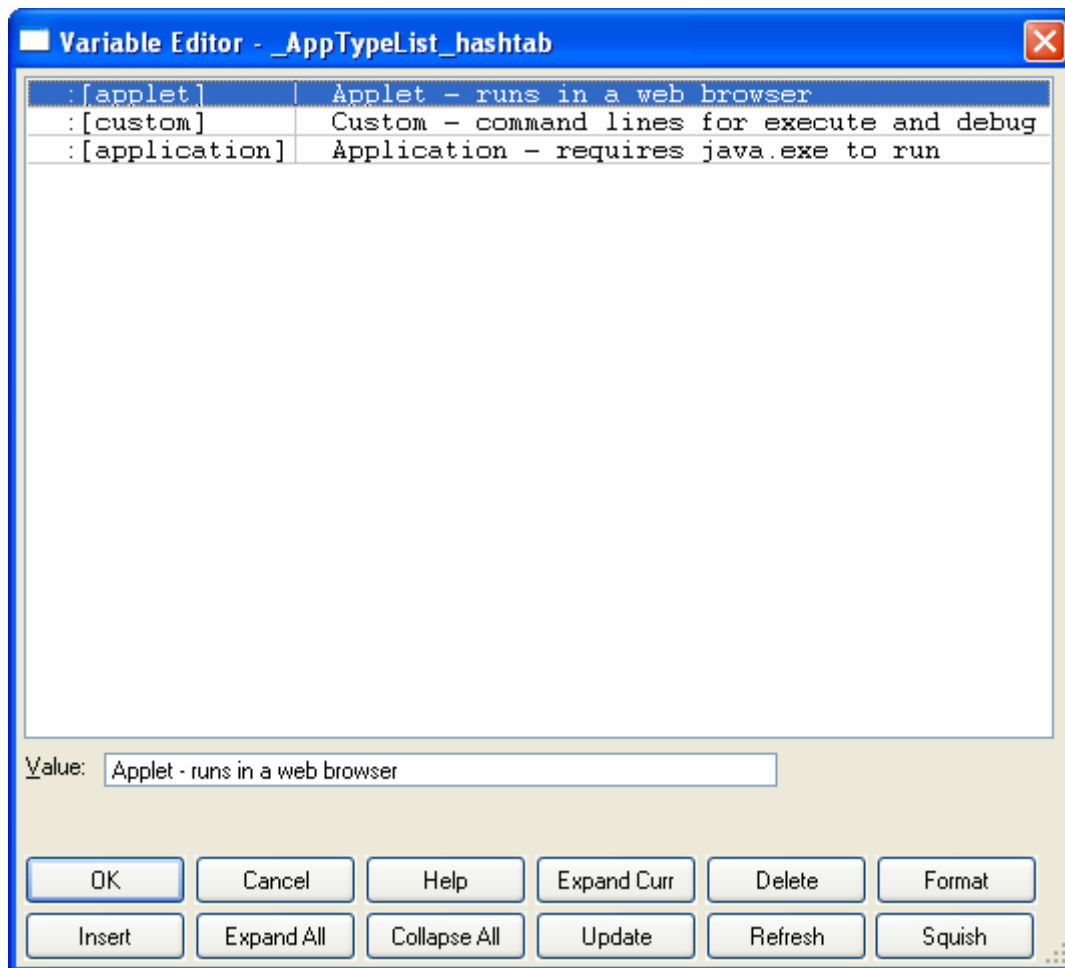


The dialog shows a list of all macros you have recorded. Use the buttons to perform the following operations:

- **Run** - Runs the selected macro. See [Running a Recorded Macro](#) for more information.
- **Cancel** - Closes the dialog.
- **Edit** - Opens the macro source for editing. See [Saving and Editing Recorded Macros](#) for more information.
- **Delete** - Deletes the selected macro. See [Deleting Recorded Macros](#) for more information.
- **Bind to Key** - Displays the Key Bindings dialog so you can assign a key or mouse shortcut to the macro. See [Binding Recorded Macros to Keys](#) for more information.

## Variable Editor Dialog

The Variable Editor dialog, shown below, is used to edit complex variables for macros. For more information about working with these programmable macros, see [Programmable Macros](#). To access the Variable Editor, choose **Macro > Set Macro Variable**, or use the `gui_set_var` command, select a variable to edit from the list, then click the **Edit** button.



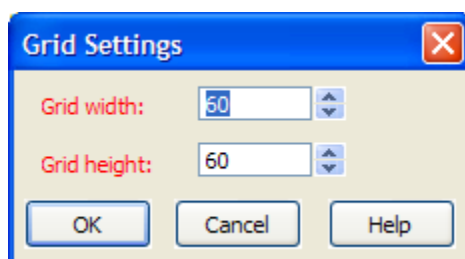
The data structure of the variable is displayed in the list box at the top of the dialog, and the value for each entry is displayed in the **Value** text box.

The following buttons are available:

- **Expand Curr** - Expands current item which has a plus sign (+) bitmap.
- **Delete** - Deletes current item.
- **Format** - Allows you to change the type of the current item.
- **Insert** - Inserts a new hash table or array element.
- **Expand All** - Expands all items so you can see the entire data structure.
- **Collapse All** - Display first level of variable with nothing expanded.
- **Update** - Sets the contents of the variable to what is currently displayed in the Variable Editor.
- **Refresh** - Cancels changes and displays current value of variable which is not necessarily the same as when this dialog box was originally displayed.
- **Squish** - Deletes array items which have the value `_notinit`.

## Grid Settings Dialog

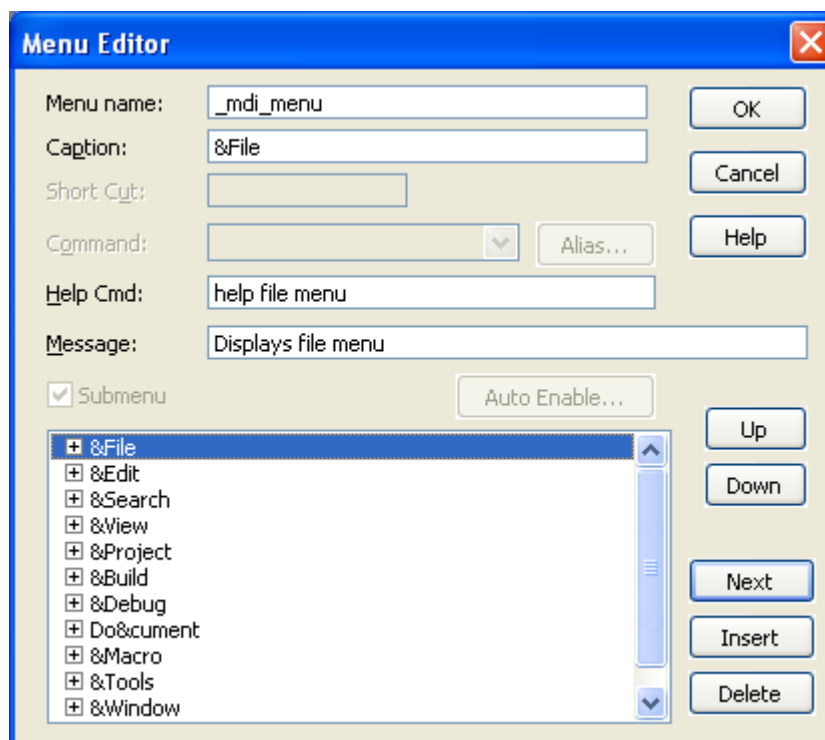
The Grid Settings dialog (**Macro > Grid** or **gui\_grid** command) is used to set the width and height of grid dots displayed on forms when you use the Dialog Editor. These settings affect the distance between the dots on a form that is being edited.



The width and height parameters are in twips (1440 twips equal one inch on the display).

## Menu Editor Dialog

The Menu Editor dialog, shown below, contains options for editing menus. To access this dialog, choose **Macro > Menus**, select the menu to edit from the list, then click **Open**.



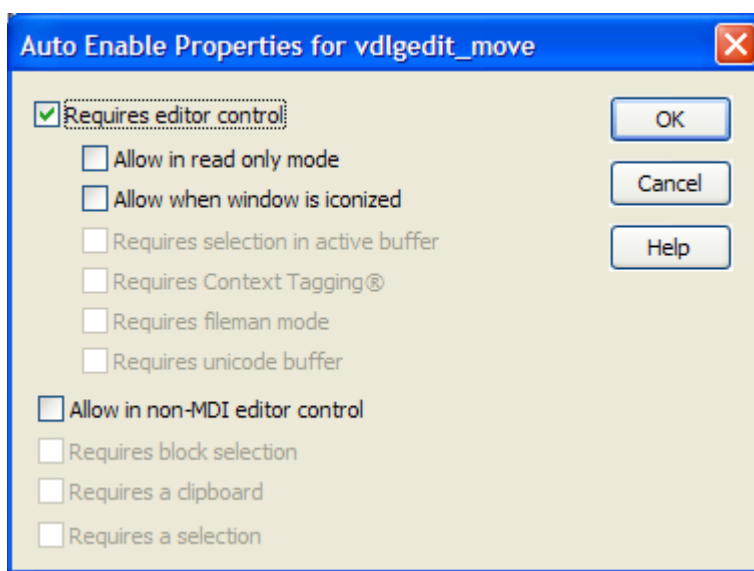
The following fields and settings are available:

- **Menu name** - Name of the current menu resource. You can define your own menu resource which is used instead of our menu bar WITHOUT changing the name of our default menu bar **\_mdi\_menu**. Use the **-m** invocation option (for example, **-m mymenu**) or set the **def\_mdi\_menu** macro variable to your menu name (see [Setting Macro Variables](#)).

- **Caption** - Title displayed for the menu item. For menu items, set the caption to “-” to specify a line separator.
- **Short Cut** - Key binding shortcut for the menu item.
- **Command** - Macro command executed when the menu item is selected. This may be an internal macro command or a command line for running an external program.
- **Alias** - Displays the Menu Item Alias dialog box to set an alias for the menu item. See [Defining Menu Item Aliases](#).
- **Help Cmd** - Macro command executed when F1 is pressed when the menu item is selected. Usually it is a **help** or **popup\_imessage** command. For example, if you specified **gui\_open** as the menu item command, specify “help open dialog box” as the help item. If you do not know the name of the dialog box displayed, search for help on the command. The help for each command should indicate the name of the dialog box displayed. Some commands do not display dialog boxes. For these commands, specify **help command** where *command* is name of the command this menu item executes or **help xxxx menu** where *xxxx* is the name of the drop-down menu this command is on.
- **Message** - Message text to be displayed when selection cursor is on this menu item. This message is currently only used when the menu is used as the SlickEdit® menu bar.
- **Submenu** - Check this box if you want to create a menu which contains other menu items.
- **Auto Enable** - Displays the Auto Enable Properties dialog box to set the properties for the menu item that should be automatically enabled. See [Enabling/Disabling Menu Items](#) and [Auto Enable Properties Dialog](#).
- **Up** - Moves the selected menu item above the previous menu item.
- **Down** - Moves the selected menu item below the next menu item.
- **Next** - Selects the menu item after the currently selected menu for editing. Use this button to insert a blank menu item after the last menu item in the list.
- **Insert** - Inserts a blank menu item before the selected menu item.
- **Delete** - Deletes the selected menu item.

## Auto Enable Properties Dialog

This dialog is used to set the auto-enable properties for a menu item. For example, the screen capture below shows the Auto Enable Properties dialog for **cut** on the **\_textbox\_menu**. For more information, see [Enabling/Disabling Menu Items](#). To access this dialog, click the **Auto Enable** button on the Menu Editor dialog.



The following settings are available:

- **Requires editor control** - Indicates that this command should be enabled only if operating on an editor control.
- **Allow in read only mode** - Indicates that this command should be enabled if the editor control is in strict read only mode.
- **Allow when window is iconized** - Indicates that this command should be enabled if the editor control is an editor window which is iconized.
- **Requires selection in active buffer** - Indicates that this command should be disabled if there is no selection in the active buffer.
- **Requires Context Tagging@** - Indicates that this command should be disabled if Context Tagging does not support the current buffer language type.
- **Requires fileman mode** - Indicates that this command should be disabled if the current buffer is not in Fileman mode.
- **Requires unicode buffer** - Indicates that this command should be disabled if the current buffer is not Unicode.
- **Allow in non-MDI editor control** - Indicates that this command should be allowed in a non-MDI editor control.
- **Requires block selection** - Indicates that this command should be disabled if there is no selection or the current selection is not a type of block or column.
- **Requires a clipboard** - Indicates that this command should be disabled if there is no editor control clipboard available.
- **Requires a selection** - Indicates that this command should be disabled if there is no selection.



## Tools

---

This section describes items on the **Tools** menu and associated dialogs and tool windows.

### Tools Menu

The table below describes each item on the **Tools** menu and its corresponding command.

Tools Menu Item	Description	Command
Options	Displays Options menu. See <a href="#">Options</a> .	N/A
Regex Evaluator	Shows the Regex Evaluator tool window. See <a href="#">The Regex Evaluator</a> .	<b>activate-regex-evaluator</b>
OS Shell	Runs operating system command shell (DOS).	<b>dos</b>
OS File Browser	Runs operating system file system browser. See <a href="#">OS File Browser</a> .	<b>explore</b> or <b>finder</b>
Calculator	Displays the Calculator, which allows you to evaluate mathematical expressions. See <a href="#">Using the Calculator and Math Commands</a> .	<b>calculator</b>
Add Selected Expr	Adds the result of evaluating each line in a selected area of text.	<b>add</b>
ASCII Table	Opens ASCII table file.	<b>ascii_table</b>
Version Control	Displays Version Control menu. See <a href="#">Version Control Menu</a> .	N/A
C++ Refactoring	Displays C++ Refactoring menu. See <a href="#">C++ Refactoring Menu</a> .	N/A
Quick Refactoring	Displays Quick Refactoring menu. See <a href="#">Quick Refactoring Menu</a> .	N/A
Imports	Displays Imports refactoring menu. See <a href="#">Imports Menu</a> .	N/A
Generate Debug	Generates debug code for symbol under the cursor.	<b>generate_debug</b>
Sort	Sorts current buffer or selected text. See <a href="#">Sorting Text</a> .	<b>gui_sort</b>
Beautify	Displays a language-specific beautifier dialog box used for setting options and beautifying source. See <a href="#">Beautifying Code</a> .	<b>gui_beautify</b>
File Merge	Displays the 3-Way Merge Setup dialog, which provides settings for merging two sets of changes made to a file. See <a href="#">3-Way Merge</a> and <a href="#">3-Way Merge Dialog</a> .	<b>merge</b>

Tools Menu Item	Description	Command
File Difference	Displays the DIFFzilla® dialog, which allows you to view and edit differences between files. See <a href="#">DIFFzilla®</a> and <a href="#">DIF-Fzilla® Dialog</a> .	<b>diff</b>
Spell Check	Displays menu of spell checking commands. See <a href="#">Spell Check Menu</a> .	N/A
Tag Files	Displays a dialog which allows you to build tag files for use by the Symbols tool window and other Context Tagging® features. See <a href="#">Creating Extension-Specific Tag Files</a> and <a href="#">Context Tagging® - Tag Files Dialog</a> .	<b>gui-make-tags</b>

## Version Control Menu

The table below describes each item on the **Tools > Version Control** menu and its corresponding command. For more information about working with Version Control, see [Version Control](#).

Version Control Menu Item	Description	Command
Check In	Checks in current file.	<b>vccheckin</b>
Get	Checks out current file read only.	<b>vcget</b>
Check Out	Checks out current file.	<b>vccheckout</b>
Lock	Locks the current file without checking out the file.	<b>vclock</b>
Unlock	Unlocks the current file without checking in the file.	<b>vcunlock</b>
Add	Adds current file to version control.	<b>vcadd</b>
Remove	Removes current file from version control.	<b>vcremove</b>
History	Views history for current file.	<b>vchistory</b>
Difference	Views differences of current file.	<b>vcdiff</b>
Properties	Views properties of current file.	<b>vcproperties</b>
Manager	Executes Version Control Manager.	<b>vcmanager</b>
Setup	Displays the Version Control Setup dialog, which allows you to choose and configure a Version Control System interface. See <a href="#">Version Control Setup Dialog</a> .	<b>vcsetup</b>



## C++ Refactoring Menu

The **Tools > C++ Refactoring** menu contains all of the C++ refactorings. For descriptions of each, see [C++ Refactoring](#). The table below describes the remaining items on the C++ Refactoring menu and each corresponding command. For more information about working with refactorings, see [Refactoring](#).

C++ Refactoring Menu Item	Description	Command
Test Parsing Configuration	Displays the C++ Refactoring Configuration Test dialog, which lets you view and test C++ Refactoring settings for the current file. See <a href="#">Test Parsing Configuration</a> .	<b>refactor_parse</b>
C/C++ Compiler Options	Displays the C/C++ Compiler Properties dialog, which allows you to configure C++ Refactoring compiler options. See <a href="#">C/C++ Compiler Settings</a> .	<b>refactor_options</b>

## Quick Refactoring Menu

The **Tools > Quick Refactoring** menu contains the Quick Refactorings that can be used for C++, C#, Java, and Slick-C®. These are summarized in the table below. For more information about working with these refactorings, see [Quick Refactoring](#).

Quick Refactoring Menu Item	Description	Command
Rename	Rename symbol.	<b>refactor_quick_rename</b>
Extract Method	Extract the selected code block into a new function.	<b>refactor_quick_extract_method</b>
Modify Parameter List	Modify the parameter list of a method.	<b>refactor_quick_modify_params</b>
Encapsulate Field	Encapsulate field using Context Tagging®.	<b>refactor_quick_encapsulate_field</b>
Replace Literal with Constant	Replace literal value with a declared constant.	<b>refactor_quick_replace_literal</b>

## Imports Menu

The **Tools > Imports** menu, summarized below, contains options for organizing Java imports. For more information, see [Organizing Java Imports](#).

Imports Menu Item	Description	Command
Organize Imports	Organize import statements in a Java file.	<b>jrefactor_organize_imports</b>
Add Import	Add import statement for symbol under cursor.	<b>jrefactor_add_import</b>
Options	Displays the Organize Imports Options dialog, which lets you view options for Organize imports operations. See <a href="#">Organize Imports Options Dialog</a> .	<b>jrefactor_organize_imports_options</b>

## Spell Check Menu

The **Tools > Spell Check** menu contains spell checking operations and access to options. For more information about working with Spell Check, see [Spell Checking](#). The table below contains a summary of the Spell Check menu items.

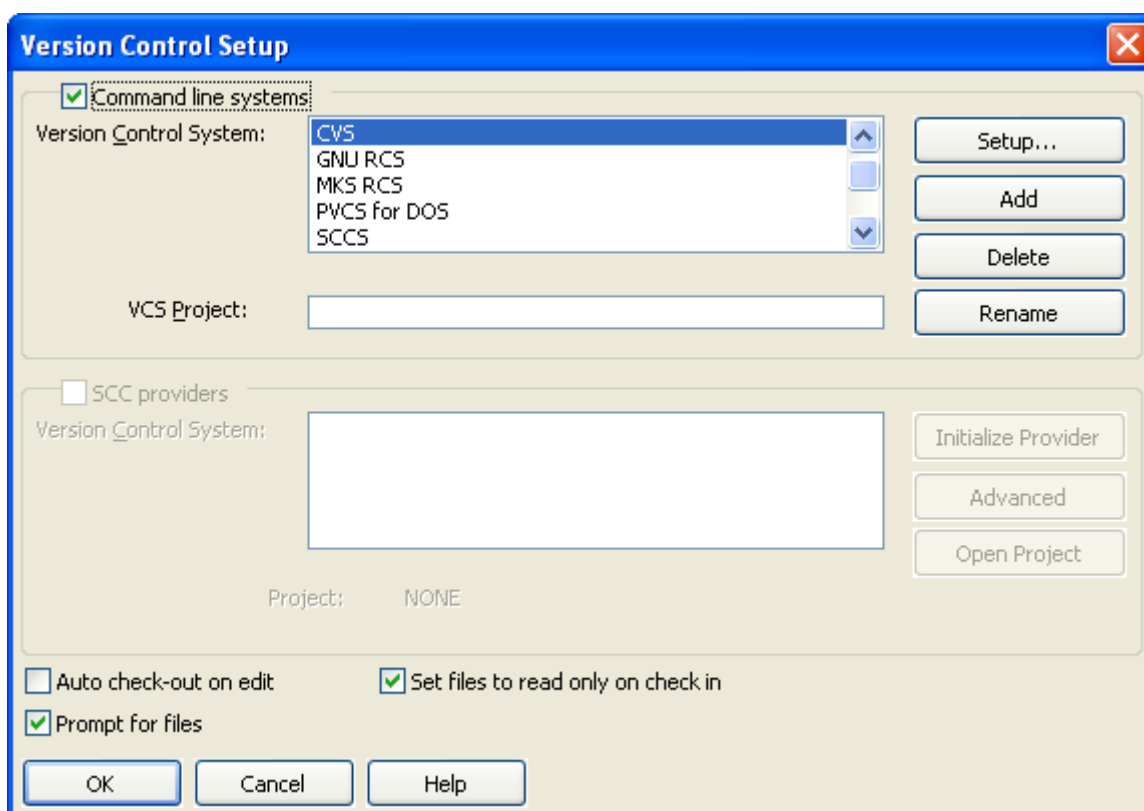
Spell Check Menu Item	Description	Command
Check from Cursor	Spell check starting from cursor.	<b>spell_check</b>
Check Comments and Strings	Spell check comments and strings starting from cursor.	<b>spell_check_source</b>
Check Selection	Spell check words in selection.	<b>spell-check-selection</b>
Check Word at Cursor	Spell check word at cursor.	<b>spell_check_word</b>
Check Files	Spell check multiple source files.	<b>spell-check-files</b>
Spell Options	Display/modify spell checker options.	<b>spell_options</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Tools** menu items.

### Version Control Setup Dialog

The Version Control Setup dialog, shown below, provides settings for specifying and configuring the version control system to use. For more information about working with Version Control, see [Version Control](#). To access the Version Control Setup dialog, choose **Tools > Version Control > Setup**.



The following settings and options are available:

- **Command line systems** - This is a list of command line version control systems that have built-in support. To select a version control system, first select this option and then select the system.
- **Setup button** - To change the individual commands that will be run for the selected system, click on the **Setup** button. You may need to fill in the **VCS Project** text box depending on your advanced version control setup.

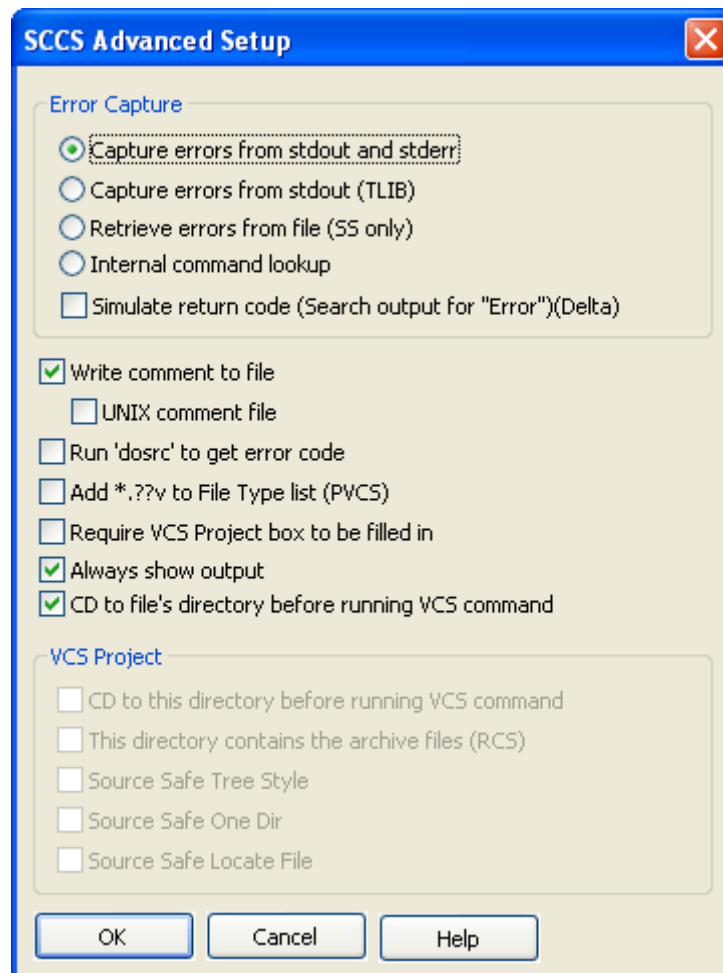
This **Setup** button invokes the setup dialog for the version control system you have selected. The following items appear on this setup dialog:

- **VC Command** - This is a list of commands that can be run from the editor.
  - **Command** - This is the command that is run by the operating system when the corresponding version control command is run. Click on the arrow to the right of the text box for a list of variables to be parsed in.
  - **Advanced button** - This button invokes the Advanced Setup dialog box for the selected version control system. See [Version Control Advanced Setup Dialog](#) for information on this dialog box.
- **Add button** - To add a new command line version control system, click the **Add** button and you will be prompted to fill in a name for the new version control system. Then the Version Control Command Setup dialog box will be launched.
- **SCC providers** - This section of the Version Control Setup dialog is for SCC version control systems that are registered.

- **Auto check-out on edit** - When selected, a prompt appears to check out a file when you open a file that does not exist or is read-only. This option is global and not local to the current project.
- **Set files to read only on check in** - When selected, after a file is checked in, the buffer in memory is set to read-only mode if the file is read-only. This option is global and not local to the current project.
- **Prompt for files** - When selected, the Checkin or Checkout Files dialog is displayed when checking a file in or out, respectively. This option is not available for CVS or Subversion. See [Checkin/Checkout Files Dialog](#) for more information.

## Version Control Advanced Setup Dialog

You can access advanced setup options for each version control system that supports them. From the Version Control Setup dialog box (choose from the main menu **Tools > Version Control > Setup**), select the version control system you wish to set up. Click the **Setup** button. This will display the setup dialog box for the system you have selected. Click the **Advanced** button on this dialog to access the advanced options. The options are similar for each version control system. For example, below is a screen capture of the SCCS Advanced Setup dialog. For more information about working with Version Control, see [Version Control](#).



The following options are available:

- **Capture errors from stdout and stderr** - What you want for most command line version control systems. Displays output from version control system if the return code from the version control executable is non-zero.
- **Capture errors from stdout (TLIB only)** - Capture errors from stdout only. Displays output from version control system if the return code from the version control executable is non-zero.
- **Retrieve errors from file (SS only)** - This option is for the command line version of Source Safe. It will open the error file and search for exit code. The error filename used is the default filename for directing errors into, and can be specified in the command line as %t. It also displays output from the version control system if the value after the colon is non-zero.
- **Simulate return code (Search output for “Error”)(Delta)** - Searches for Error or Warning. Displays output from the version control system if either are found.
- **Write comment to file** - Write comment to a temp file, the name of which can be parsed into the command line by putting in %c. If this option is not selected, and %c is in the command line, the comment can only be one line, and %c will be replaced with the comment itself.
- **UNIX comment file** - Writes comment file with UNIX end of line characters. This option is for Windows only.
- **Run ‘dosrc’ to get error code** - This option should usually be selected if you are running Windows.
- **Add \*.??v to File Type list (PVCS)** - Select this option for PVCS.
- **Require VCS Project box to be filled in** - Requires the **VCS Project** text box on the Version Control Setup dialog to be filled in.
- **Always show output** - Show output from the version control system regardless of return code.
- **CD to file’s directory before running VCS command** - Temporarily change the current directory to the path of the file being operated on while running the VCS command. The directory is changed back after the command is run.
- **CD to this directory before running VCS command** - Temporarily change the current directory to the path specified in the **VCS Project** text box on the Version Control Setup dialog while running the VCS command. The directory is changed back after the command is run.
- **This directory contains the archive files** - For command line versions of RCS, specify the directory that has the archive files for the **VCS Project** text box on the Version Control Setup dialog.
- **Source Safe Tree Style** - Use this style to map Source Safe projects to your actual disk hierarchy.

In the **VCS Project** text box on the Version Control Setup dialog, enter the Source Safe project name in square brackets followed by the root directory for files in the project. For example:

```
[ $/vslick15 ]
c:\vslick15
```

When using this style, the Source Project tree looks like the directory tree. If the name of the file you check in or check out is `c:\vslick15\clib\test.c`, this file will be placed in the project `$/vslick15/clib/test.c`. Only files at or below `c:\vslick15` directory may be checked in or out.

- **Source Safe One Dir** - When using this style, all source files are checked into or out of the Source Safe project directory specified in square brackets. Enter the project name in square brackets in the **VCS Project** text box on the Version Control Setup dialog. For example:

```
[ $/vslick15
]
```

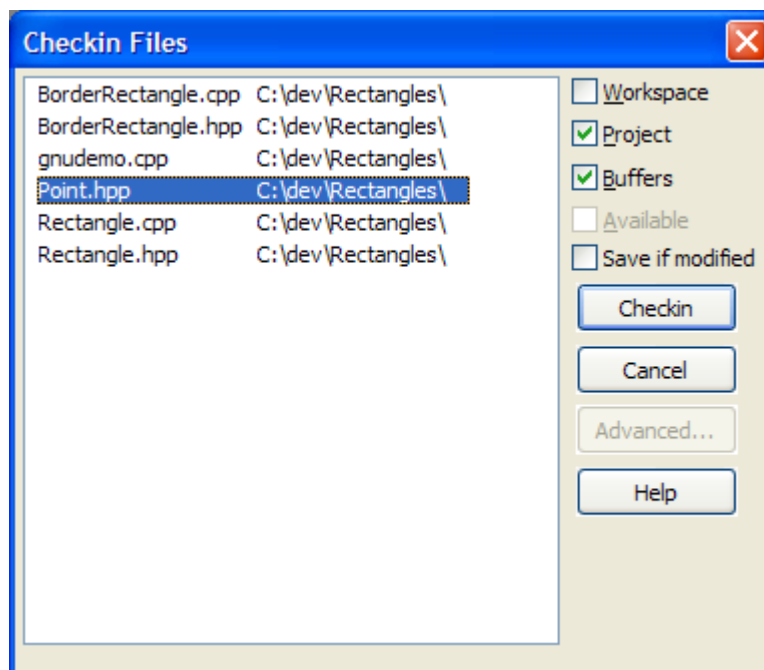
- **Source Safe Locate File** - When using this file, the **VCS Project** text box may be blank (even if the **vcp\_required** style is present). However, a Source Safe base project directory may be specified in square brackets. When using this style, the Source Safe project is dynamically determined by using Source Safe's **locate** command. If the file exists in the base project (**VCS Project** not blank), only projects at or below this project are used. This mode of operation is slower than **ssstree** and **ssonedir** because it requires the **locate** command to execute for each file.

**NOTE** If you are using Source Safe for Windows, use the SCC interface.

## Checkin/Checkout Files Dialog

When the option **Prompt for files** is checked on the Version Control Setup dialog, the Checkin Files or Checkout Files dialog is displayed when checking files in or out, respectively. This allows you to see a list of files in the current workspace or project, or a list of open files (buffers), and select the files to check in or out.

**NOTE** This dialog is not available for CVS and Subversion.



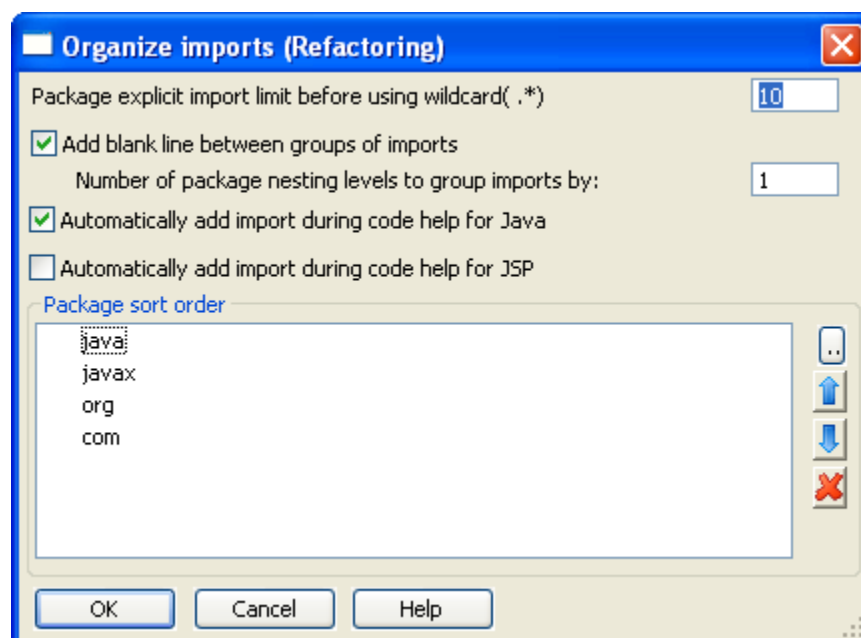
The Checkin Files and Checkout Files dialogs share a similar interface and contain the following elements:

- **Workspace** - When this option is selected, all files in the workspace are added to the list.
- **Project** - When this option is selected, all files in the current project are added to the list.
- **Buffers** - When this option is selected, all open files are added to the list.
- **Available** - When this option is selected, all files available for the check-in or check-out operation are displayed in the list. For example, if you are performing a check-in, all files that have been checked out will be added to the list. This option is only available for SCC systems.
- **Save if modified** - When this option is selected, any unsaved files are saved before check-in.

- **Checkin** or **Checkout** - Click these buttons to perform the check-in or check-out operation on the selected files.
- **Advanced** - This button displays the options dialog specific to the version control system you are using. This option is only available for SCC systems.

## Organize Imports Options Dialog

The behavior of the Organize Imports and Add Import features is controlled by the options on the Organize Imports Options dialog box, pictured below. This dialog can be accessed from the main menu by choosing **Tools > Imports > Options**.



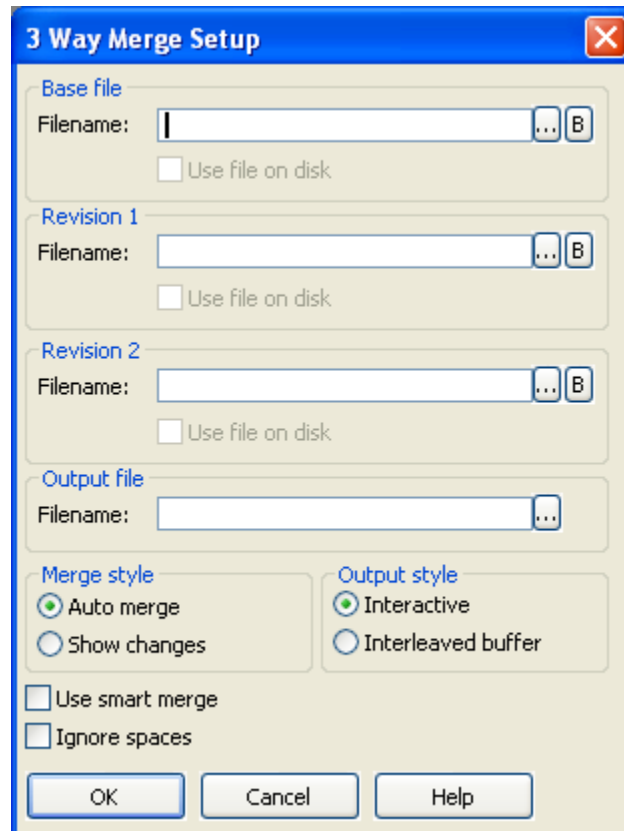
The following settings are available:

- **Package explicit import limit before using wildcard(.\*)** - If more than this number of classes are explicitly imported from the same package in one file, the imports will be replaced with a single wildcard import.
- **Add blank line between groups of imports** - Organize Imports will group imports by package name or top-level package name. Select this option to force Organize Imports to add a blank line between these groups instead of having just one flat list of imports.
- **Number of package nesting levels to group imports by** - If this is set to "1", import statements will be grouped by top level package name only. For example, all your imports from "java." packages would be in a separate group from your imports from "com." packages. If set to "2", import statements will be grouped by second level package names. For example, all your imports from "java.util" would be in a separate group from your imports from "java.awt".
- **Automatically add import during code help for Java** - If selected, SlickEdit® will attempt to automatically add imports as you edit Java code.
- **Automatically add import during code help for JSP** - If selected, SlickEdit will attempt to automatically add imports as you edit Java code embedded in HTML. JSP imports are added using the following notation: `<%@ page import="java.util.Vector"%>`.

- **Package sort order** - This list specifies the order in which package groups are sorted. Use the “...” button to add a new package. Use the up and down arrows to move items. Use the “X” button to delete the currently selected package from the list.

## 3-Way Merge Dialog

The 3-Way Merge dialog (**Tools > File Merge**), shown below, is used for merging file differences.



The **Ellipses** icons to the right of the text boxes are used to select files. The **B** icons to the right of the text boxes are used to select from the open buffers.

The list below describes the remaining fields and settings:

- **Base file** - Specifies the file/buffer name of the original source file before any changes are made.
- **Revision 1 and 2** - Specifies the file/buffer names of the modified versions of the base file.
- **Output file** - Specifies the output file name.
- **Merge style** - The following merge styles are available:
  - **Auto merge** - If selected, if a change does not cause a conflict, the change is automatically applied to the output file and no indication is made that the change was already applied.
  - **Show changes** - If selected, if a change does not cause a conflict, the change is automatically applied to the output file and the change IS indicated, so that using the **Next Conflict** button will show you the change.



- **Output style** - **Output style** has no effect if there are no conflicts. The following output styles are available:
  - **Interactive** - Provides a friendly side-by-side dialog box which lets you pick the change you want in the output file. It also lets you edit.
  - **Interleaved buffer** - Creates an editor buffer which you must edit to resolve conflicts.
- **Use smart merge** - If selected, the number of conflicts found is reduced.
- **Ignore spaces** - If selected, leading and trailing spaces are ignored. The side-by-side output allows you to easily select the change that you want.

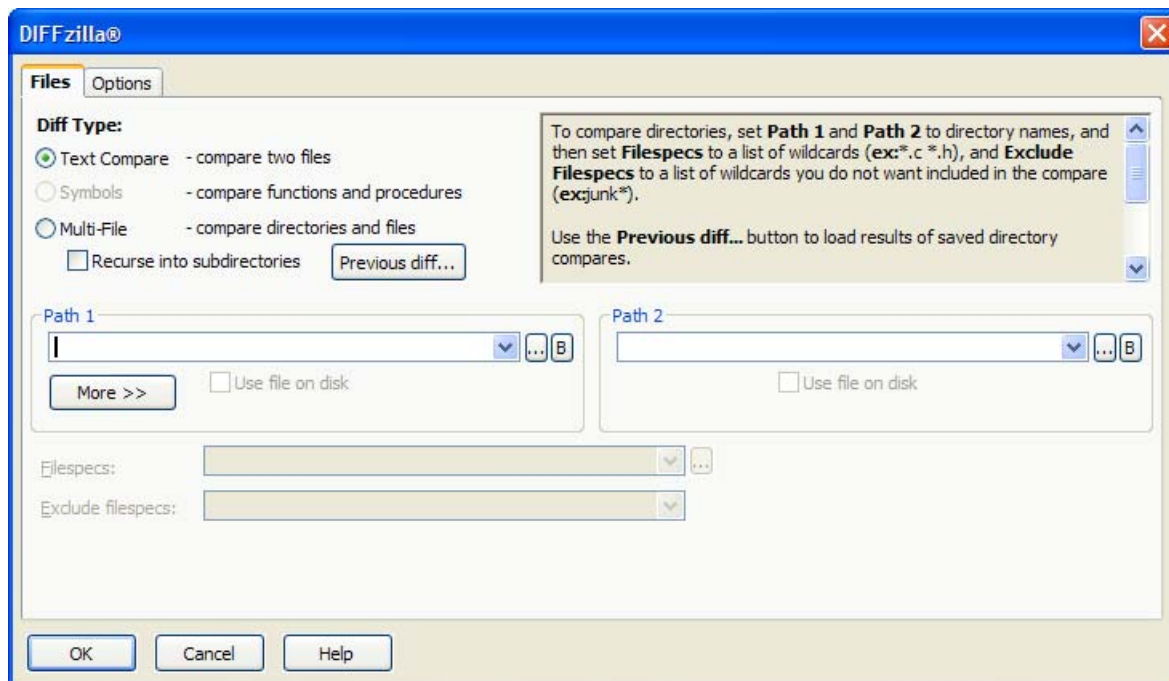
## DIFFzilla® Dialog

The DIFFzilla dialog (**Tools > File Difference**) is used to configure a file differencing operation and begin the diff. The options are categorized into two tabs:

- [DIFFzilla® Files Tab](#)
- [DIFFzilla Options Tab](#)

### DIFFzilla® Files Tab

You can compare two files or two source trees to determine which files have been added or removed and to generate a list of file names. Use this tab to set up the comparison parameters for the files that you wish to compare. After configuring your settings, click **OK** to start the diff.



### Diff Types

The following **Diff types** are available:

- **Text Compare** - Compares two files and shows the differences between them. When this option is selected, after you click **OK** on this dialog to start the comparison, the interactive Diff dialog is displayed, allowing you to preview the differences one-by-one before committing.

If the option on the Options tab, **Instead of an interactive dialog, output one buffer with the differences labeled**, is checked, a buffer with the differences between the two files marked up will be displayed instead.

- **Symbols** - Enables the selection of a symbol in order to set the **Line Range** line numbers. Not all symbol ranges are identified. Ranges for multi-line variable declarations are not identified.
- **Multi-File** - Compares two directories or directory trees, and shows which files do not match. Select **Recurse into subdirectories** to search subdirectories recursively. Click **Previous diff** to load a diff state file (`.diff`), restoring the saved state of a multi-file diff session.

### Path Information and Filespecs

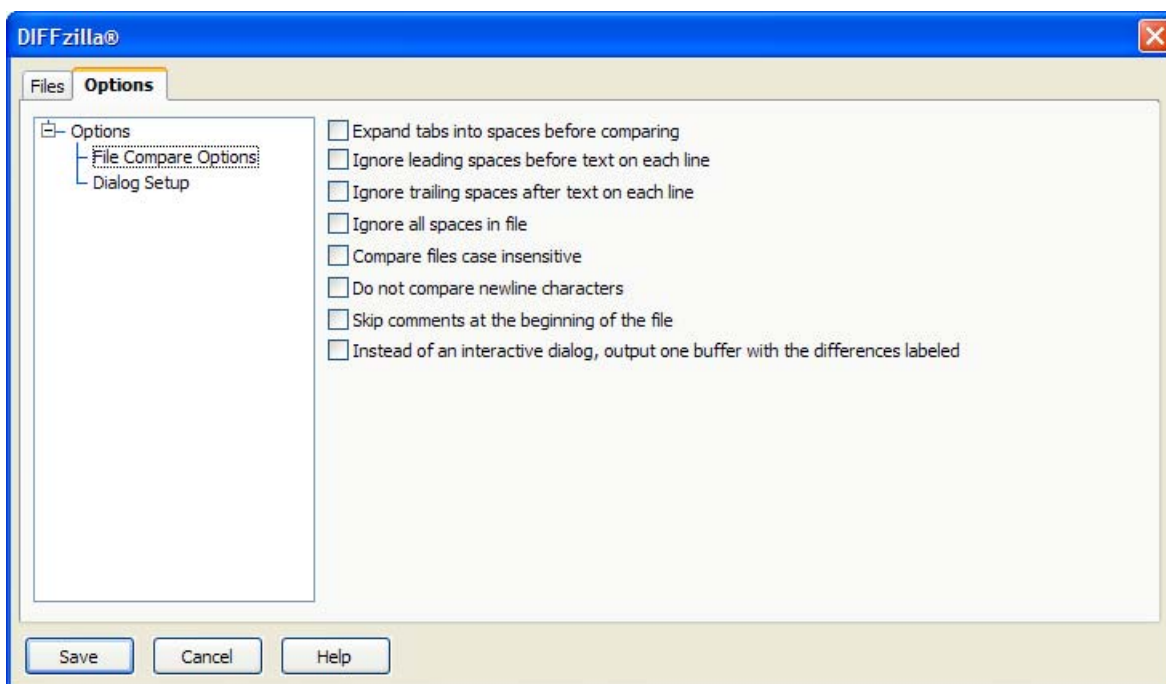
The following list describes the path information and filespec settings that can be entered on the DIFFzilla® dialog:

- **Path 1, Path 2** - To compare directories, set **Path 1** and **Path 2** to directory names. To compare files, set Path 1 and Path 2 to file names. If the file names only differ by path, you only need to specify a directory for Path 2. Click the dotted icon to browse, or the **B** icon to select an open buffer.
- **Use file on disk** - Select this option to diff the file on disk and not the file in the buffer.
- **More** - Click this button to display additional file options.
- **Symbols** - Click this button to select a symbol to diff. The selected symbol will appear next to the word "Symbol" under the **Line range** options. All symbols from Path 1 are diffed against all symbols from Path 2. Performing a multi-file diff always diffs all symbols. When symbols are diffed, there is a multi-file diff on just two files which immediately diffs all symbols. Not all symbol blocks are identified correctly. For an example, see [Comparing Symbols or Parts of Files](#).
- **Line range** - Specifies the start and end line numbers for the range of lines to compare. If you set the start line number and leave the end line number blank, the range extends to the end of the file. When you select a range of lines, you can compare parts of the same file.
- **Record file width** - Specifies the record width to use when reading a file (optional).
- **Filespecs** - Enter a space-delimited list of wildcard file specifications to difference. For example, enter `*.c *.cpp *.h` to difference all files with `.c`, `.cpp`, and `.h` extensions.
- **Exclude filespecs** - Enter a space-delimited list of wildcard file specifications to be excluded from the differencing. For example, enter `junk* test*` to exclude all files with names beginning with the words "junk" or "test".

### DIFFzilla Options Tab

Use this tab to set up file comparison options and options that affect the interactive Diff dialog. Click **Save** to save the options and close this dialog without running DIFFzilla. There are two types of options available: file comparison and dialog setup.

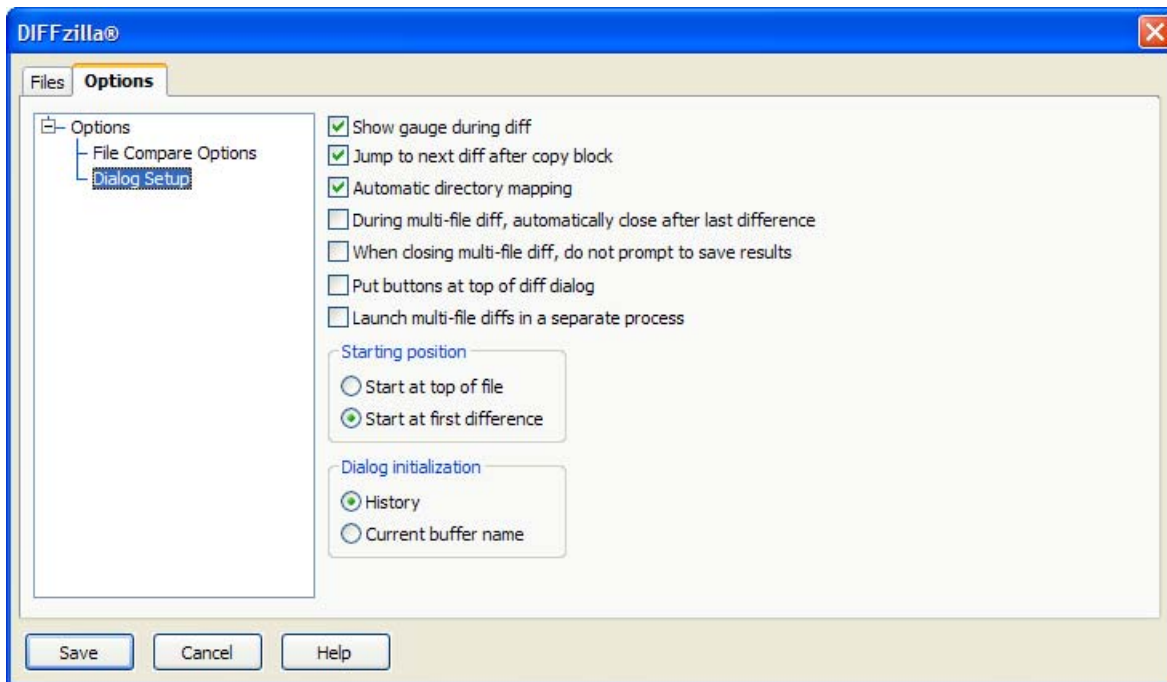
## File Compare Options



The file compare options, shown above, are described as follows:

- **Expand tabs into spaces before comparing** - When selected, tabs are expanded to the appropriate number of spaces before lines from each file that is compared.
- **Ignore leading spaces before text on each line** - When selected, differences in leading spaces of lines are ignored.
- **Ignore trailing spaces after text on each line** - When selected, differences in trailing spaces at the end of lines are ignored.
- **Ignore all spaces in file** - When selected, differences in spacing between characters in lines are ignored.
- **Compare files case insensitive** - When selected, differences in character casing are ignored.
- **Do not compare newline characters** - When selected, differences in end-of-line characters are ignored. This is useful when comparing UNIX-formatted files with DOS-formatted files.
- **Skip comments at the beginning of the file** - When selected, leading comments are ignored. This is useful if you are using a version control system that automatically inserts comment file headers.
- **Instead of an interactive dialog, output one buffer with the differences labeled** - When selected, a new buffer is created that contains color-coded difference output. You can edit the output buffer. When this option is not selected, the Diff dialog box opens displaying the two files side-by-side and the differences are color-coded.

## Dialog Setup Options

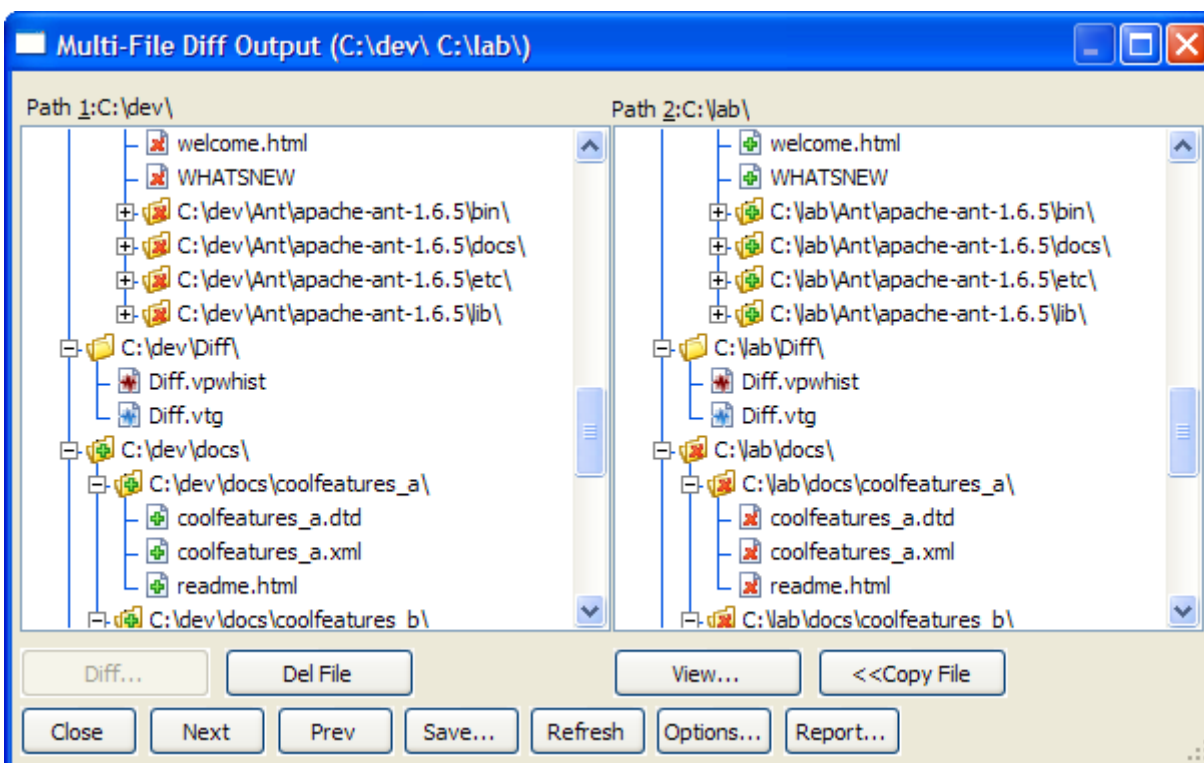


Setup options for the DIFFzilla® dialog are described as follows:

- **Show gauge during diff** - When selected, a gauge control will show various processing statistics while you wait for the differences output to complete.
- **Jump to next diff after copy block** - When selected, the cursor is moved to the next difference when you apply changes from one file to the other. For example, after clicking **Block** on the Diff dialog box, the tab moves to the next difference. This option has no effect on interleaved output.
- **Automatic directory mapping** - When selected, the **Path 2** text box is automatically updated when you type a directory in the **Path 1** text box.
- **During multi-file diff, automatically close after last difference** - When selected, clicking **Next Diff** on the Diff dialog box when there are no more differences, triggers the **Close** button on that dialog box.
- **Put buttons at top of diff dialog** - When selected, the buttons that control operations such as **Next Diff**, **Prev Diff**, and **Block**, are displayed at the top of the Diff dialog box.
- **Launch multi-file diffs in a separate process** - When selected, source trees are diffed in a separate process so you can continue working.
- **Starting position** - Determines whether to place the cursor at the top of the file or at the first difference when the Diff dialog box is displayed. This option has no effect on interleaved output.
- **Dialog initialization** - Determines whether the DIFFzilla dialog box restores previous dialog settings (history) or just places the current buffer name into the Path 1 text box. Press **F7/ F8** to restore the previous next dialog settings, respectively.

## Multi-File Diff Output Dialog

When using DIFFzilla® to perform a directory comparison (**Multi-File** diff type), the results are presented in the Multi-File Diff Output dialog.



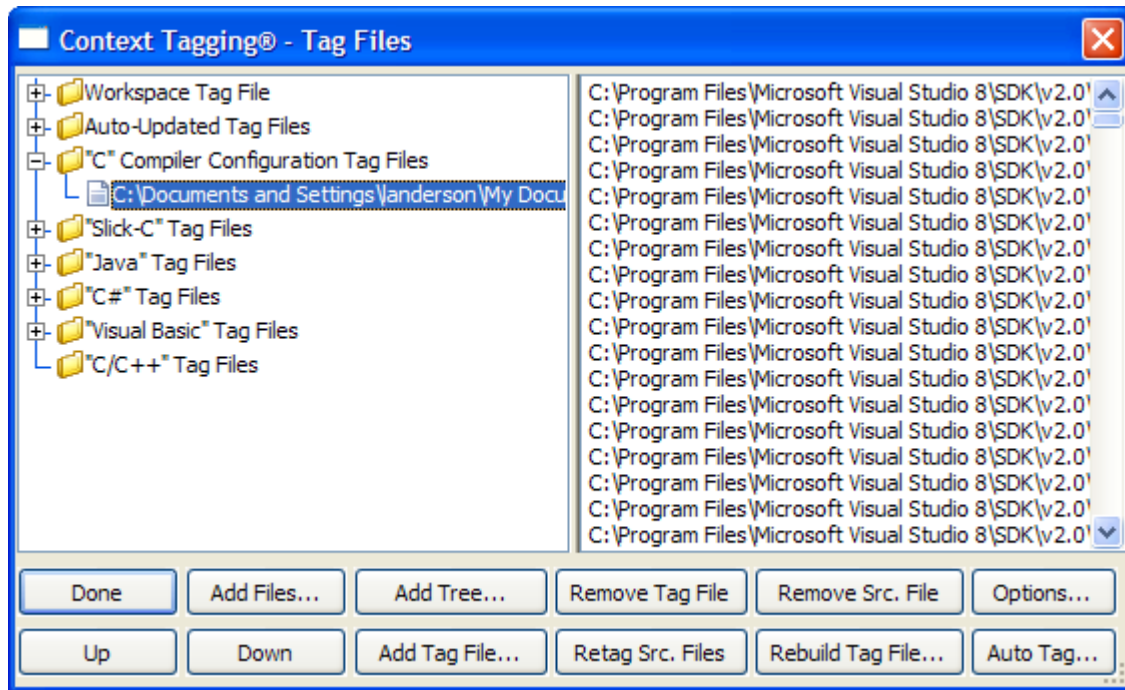
The Multi-File Diff Output dialog box contains the following elements:

- **Diff** - Shows current files in the difference editor when the selected files differ.
- **Del File** - Deletes the selected file(s). Hold Ctrl+click to multi-select in either tree. The **X** bitmap is displayed.
- **View** - Shows current files in the difference editor when the selected files match.
- **Copy File/Copy Tree** - **Copy File** is displayed when the selected files differ or when the selected file only exists in the current source tree. The **+** bitmap is displayed. **Copy Tree** is displayed when the selected item is a directory that only exists in the current source tree. When you click **Copy Tree**, you are prompted as to whether you want to copy the directory source tree recursively.
- **Next** - Moves the cursor to the next set of mismatched files in both source trees.
- **Prev** - Moves the cursor to the previous set of mismatched files in both source trees.
- **Save** - Lets you save a diff state file (.dif) that you can load later with the **Previous diff** button on the DIFFzilla® dialog box. This is especially useful when you have not completed merging files and you want to continue at a later time. Also, you can generate a file list.
- **Refresh** - Rediffs modified files or all files.
- **Options** - Displays the [DIFFzilla Options Tab](#). Options include ignoring spaces, skipping leading comments, and expanding tabs.

- **Report** - Displays a report of the operations you performed in this dialog including file copies, file deletes, and diffs where changes were saved. In addition, you can save the report.

## Context Tagging® - Tag Files Dialog

The Context Tagging® - Tag Files dialog, shown below, is used to manage all your tag files. For more information on tagging, see [Context Tagging® Overview](#). To access the dialog, choose **Tools > Tag Files**.



The left section of the dialog lists all of your tag files, separated into categories. A tag file having a file icon with blue arrows indicates the tag file is built with support for cross-referencing. The right section of the dialog lists all the source files indexed by the currently selected tag file.

For descriptions of the Tag File categories, listed on the left side of the dialog, see [Tag File Categories](#).

The following buttons are available on the Context Tagging® - Tag Files dialog:

- **Done** - Saves tag file settings and closes the dialog box.
- **Add Files** - Displays the Add Source Files dialog box, from which you can add a set of files to the currently selected tag file. This button will be disabled for read-only tag files and auto-updated tag files. If you add files to your workspace tag file, you will be prompted if you want to also add the files to your project.
- **Add Tree** - Displays the Add Tree dialog box, from which you can recursively add a directory of files to the currently selected tag file. This button will be disabled for read-only tag files and auto-updated tag files. If you add files to your workspace tag file, you will be prompted if you want to also add the files to your project.
- **Remove Tag File** - Deletes the currently selected tag file. You will be prompted whether or not to delete the tag file from the list, and then whether or not to permanently delete the tag file from disk. Note that some extension-specific tag files are automatically generated, and thus will be automatically regenerated if you delete them.



- **Remove Src. File** - Removes the selected files from the currently selected tag file. If no files are selected, you will be prompted whether or not to remove all source files from the tag file. If you remove files from your workspace tag file, you will be prompted if you want to also remove the files from your project.
- **Options** - Displays the Context Tagging® Options dialog box for you to configure Context Tagging® options. See [Context Tagging® Options Dialog](#) for more information.
- **Up** - Moves the selected tag file higher in the search order. This primarily applies to extension-specific tag files (see [Creating Extension-Specific Tag Files](#)).
- **Down** - Moves the selected tag file lower in the search order. This primarily applies to extension-specific tag files (see [Creating Extension-Specific Tag Files](#)).
- **Add Tag File** - Displays the Add Tag File dialog box, which allows you to choose from a list of languages the source type for which to insert the tag file. To automatically create tag files for C++, Java, and .NET, you can instead use the Create Tag Files for Run-Time Libraries dialog (see [Creating Tag Files for Run-Time Libraries](#)).
- **Retag Src. Files** - Updates the Context Tagging information for the selected files in the currently selected tag file. If no files are selected, you will be prompted whether or not to retag all source files.
- **Rebuild Tag File** - Displays the Rebuild Tag File dialog box containing options for rebuilding the selected file. See [Rebuilding Tag Files](#).
- **Auto Tag** - Displays the Create Tag Files for Run-Time Libraries dialog box used to automatically create run-time library tag files for C++, Java, and .NET (see [Creating Tag Files for Run-Time Libraries](#)).





# Options

This section describes items on the **Tools > Options** menu and associated dialogs and tool windows.

## Options Menu

The table below describes each item on the **Tools > Options** menu and its corresponding command.

Tools Options Menu Item	Description	Command
General	Displays General Options dialog box. See <a href="#">General Options Dialog</a> .	<b>show -modal _config_form</b>
Emulation	Displays the Emulation dialog, which allows you to view or change the current emulation, which affects key bindings and other configuration options. See <a href="#">Emulations</a> .	<b>show -modal -mdi _emulation_form</b>
File Extension Setup	Displays Extension Options dialog box. See <a href="#">Extension Options Dialog</a> .	<b>setupext</b>
File Options	Displays the File Options dialog, containing options for Load, Save, Backup, File Filters, and AutoSave. See <a href="#">File Options Dialog</a> .	<b>setup_file_options</b>
Key Bindings	Allows you to change your key bindings. See <a href="#">Key Bindings Dialog</a> .	<b>gui_bind_to_key</b>
Redefine Common Keys	Displays the Redefine Common Keys dialog, which allows you to change configuration of common keys like Tab, Enter, Backspace, and more. See <a href="#">Redefine Common Keys Dialog</a> .	<b>show -modal -mdi _rc_keys_form</b>
Tagging Options	Displays Context Tagging® Options dialog box. See <a href="#">Context Tagging® Options Dialog</a> .	<b>show -modal _autotag_options_form</b>
Aliases	Allows you to edit your global or extension-specific aliases. See <a href="#">Aliases</a> .	<b>alias</b>
Color Coding	Displays Color Coding Setup dialog box. See <a href="#">Color Coding Setup Dialog</a> .	<b>show -modal _cc_form</b>
Color	Displays Color Settings dialog box. See <a href="#">Color Settings Dialog</a> .	<b>color</b>
Font	Displays Font Configuration dialog box. See <a href="#">Font Configuration Dialog</a> .	<b>show -mdi -modal _font_config_form</b>
XML/HTML Formatting	Displays the XML/HTML Formatting Scheme Configuration dialog. See <a href="#">XML/HTML Formatting</a> .	<b>xml_html_options</b>
URL Mappings	Displays the URL Mappings dialog, which allows you to redirect URL path or file access to a local or URL path or file. See <a href="#">URL Mappings</a> .	<b>show -modal -xy -mdi _urlmappings_form</b>

Tools Options Menu Item	Description	Command
Proxy Settings	Displays the Proxy Settings dialog, which allows you to configure proxy settings used to transfer remote files via HTTP. See <a href="#">Proxy Settings Dialog</a> .	<b>show -modal -mdi _url_proxy_form</b>
Network Options	Displays the Network Options dialog which allows you to set the Internet Protocol version. See <a href="#">Network Options Dialog</a> .	<b>show -modal -mdi _network_options_form</b>
History	Displays the History dialog, which allows you to delete menu history. See <a href="#">Viewing the History of Opened Items</a> .	<b>show -modal -mdi _history_form</b>
Associate File Types	(Windows only) Displays the Associate File Types dialog, which allows you to associates file types to run SlickEdit® when you open them in Windows Explorer and File Manager. See <a href="#">Setting File Associations</a> .	<b>assocft</b>
Web Browser Setup	Displays the Web Browser Setup dialog, which lets you configure what browser to use when SlickEdit needs to launch one. See <a href="#">Web Browser Setup Dialog</a> .	<b>show -modal -mdi _html_form</b>
Visual C++ Setup	Displays the Visual C++ Setup dialog, which contains options for using Visual C++ with SlickEdit. See <a href="#">Visual C++ Setup Dialog</a> .	<b>show -modal -mdi _vchack_form</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Tools > Options** menu items.

### General Options Dialog

Many common user preferences can be set from the General Options dialog. To work with this dialog, from the main menu, choose **Tools > Options > General**.

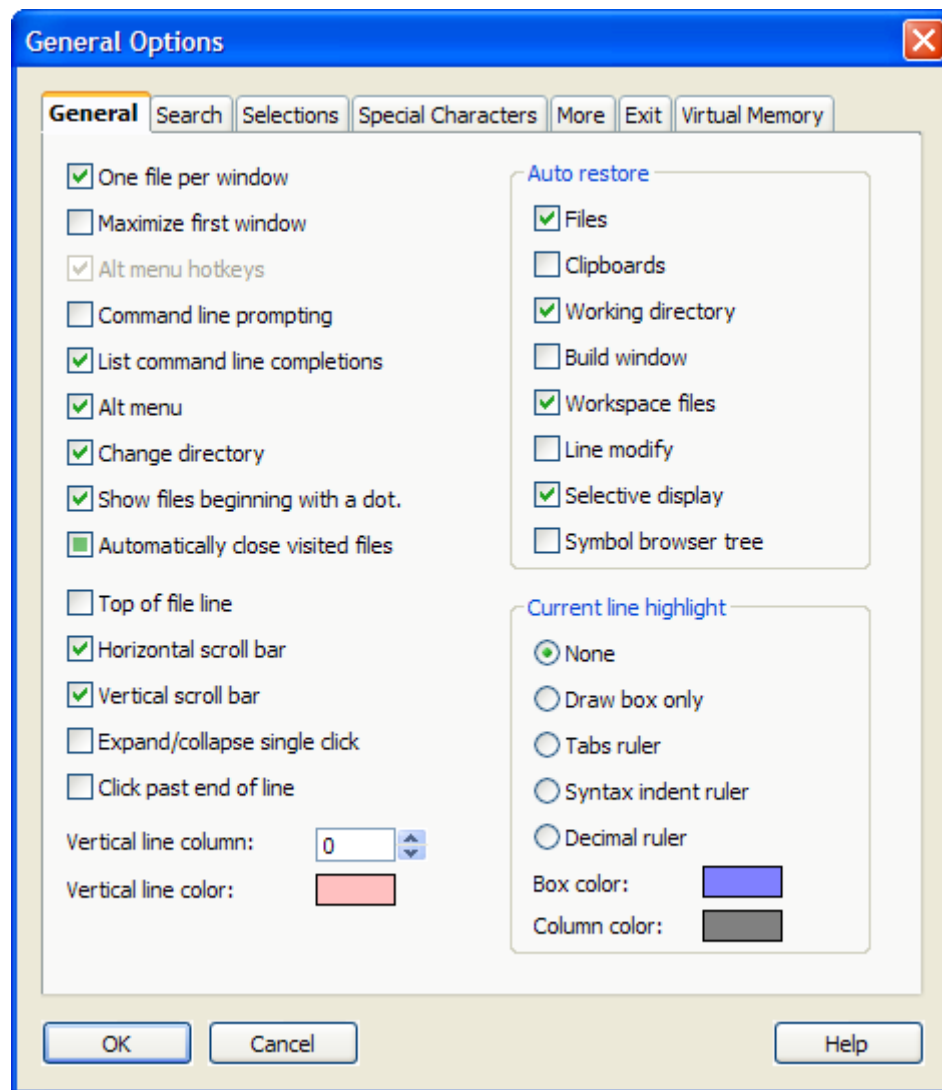
There are seven categories (tabs) available, listed below. Information about working with these general options is located throughout the documentation on a contextual basis.

- [General Tab](#) - The General tab contains general option settings.
- [Search Tab](#) - This tab contains options to control your searching preferences.
- [Selections Tab](#) - The Selections tab allows you to set text selection style preferences.
- [Special Characters Tab](#) - Enabling view of special characters inserts characters into your file to show such items as tabs, spaces, and line endings. When characters are defined in the Special Characters tab, they are displayed instead of the original characters when the view options are enabled. See [Viewing Special Characters](#) for more information.
- [More Tab](#) - The More Tab contains additional option settings.

- [Exit Tab](#)- The Exit tab contains options regarding exiting the program.
- [Virtual Memory Tab](#) - This tab stores information for the virtual memory.

## General Tab

The General tab, pictured below, is where you set general configuration options to use when you are working with SlickEdit®. These options are accessed from the main menu by choosing **Tools > Options > General**, then selecting the **General tab**.



The following options are available:

- **One file per window** - If checked, each file you open will be allocated in its own window. If unchecked, each file will open in the same window.
- **Maximize first window** - If checked, the first editor window opened will be maximized.
- **Alt menu hotkeys** - If checked, "Alt" prefixed keyboard shortcuts will display the corresponding drop-down menu. If unchecked, you can be more selective about key bindings because you are permitted to bind Alt keys you normally could not, such as Alt+F. Do not check this option if you

bind Alt keys that are normally menu keys, because you will lose these key bindings. This option is unavailable using the CUA emulation.

- **Command line prompting** - Many commands that display dialog boxes have equivalent commands that prompt for arguments on the command line. For faster prompting than the dialog boxes allow, check this option. See [Command Line Prompting](#) for more information.
- **List command line completions** - If checked, when typing a command on the command line, a list of possible commands and argument completions will be displayed above or below the command line. See [Command Line Completions](#) for more information.
- **Alt menu** - If checked, when the Alt key is pressed without following it with another key, the cursor will pop to the menu bar.
- **Change directory** - If checked, the current directory is changed in the editor when the directory is changed in the Change Directory dialog (**File > Change Directory**) and the Open and Save As dialogs (**File > Open** and **File > Save As**).
- **Show files beginning with a dot.** - This option controls the default value of the **Show hidden files** option on the UNIX File Open and Save dialogs (and the Open tool window for all platforms). Check this option to have the **Show hidden files** option checked by default each time the dialogs/tool window are displayed. The value of **Show hidden files** is controlled by the global variable **def\_filelist\_show\_dotfiles**. By default, this option is on for Windows, and off for UNIX platforms.
- **Automatically close visited files** - If selected, a visited file will be automatically closed when it is navigated away from. If not selected, the auto-close feature will be disabled. If left in the mixed state, you will be prompted whether or not you want to close files. A file is considered as visited if it is opened as a result of a symbol navigation or search operation, and not modified, and subsequently navigated away from, for example, by using **pop\_bookmark** (Ctrl+),).
- **Top of file line** - If selected, each buffer displays a line which contains the text "Top of File". This indicator for the location of the top of the file is displayed at line "0" which does not affect lines of code.

Rather than using the **Top of file line** option, you can use Ctrl+Shift+Enter (Ctrl+Enter in Visual C++ and Visual Studio emulation) to insert a new line above the line where the cursor is located.

- **Horizontal scroll bar** - If checked, each edit window displays a horizontal scroll bar. This does not affect edit window controls on dialog boxes.
- **Vertical scroll bar** - If checked, each edit window displays a vertical scroll bar. This does not affect edit window controls on dialog boxes.
- **Expand/collapse single click** - If checked, selective display plus and minus sign bitmaps can be expanded or collapsed with a single click. This causes selective display to operate similar to the Windows Explorer. However, you will not be able to select a line by clicking to the left of a text line which contains a selective display bitmap. For more information about Selective Display, see [Selective Display](#).
- **Click past end of line** - If checked, the cursor can be placed past the end of a line.
- **Vertical line column** - Specifies the column in which the editor is to display a vertical line. Specify "0" to display no vertical line.
- **Vertical line color** - Click on the colored box to change the color of the line.
- **Auto restore** - The Auto restore options control which elements of your SlickEdit environment are restored when you switch workspaces or close and re-open SlickEdit. See [Restoring Settings on Startup](#) for more information about these options.
- **Current line highlight** - Select from the following options pertaining to the highlight effect of the current line:

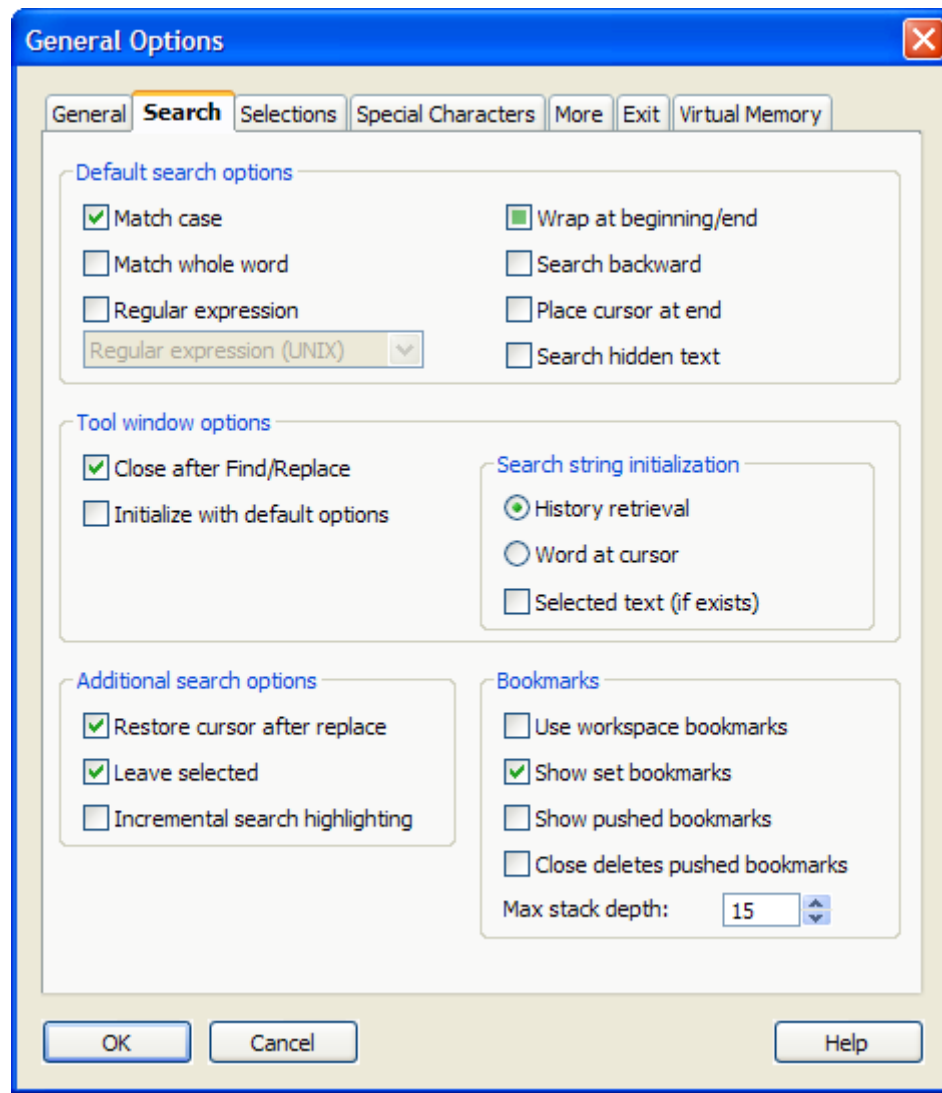
- **None** - If selected, the current line will not be highlighted.
- **Draw box only** - If selected, a dotted box will be drawn around the current line.
- **Tabs ruler** - If selected, a box will be drawn around the current line and tab stops will be marked.
- **Syntax indent ruler** - If selected, a box will be drawn around the current line with the syntax indent levels marked.
- **Decimal ruler** - If selected, a box will be drawn around the current line with marks at multiples of five and 10.
- **Box color** - Click on the colored box to select the dotted box color.
- **Column color** - Click on the colored box to select the column marker color. This is the same as the margin line color.

## Search Tab

The Search tab contains default search options. For more information about working with search and replace operations, see [Find and Replace](#).

There are two ways to access search options:

1. From the Find and Replace tool window (**View > Toolbars > Find and Replace**), right-click on the background and select **Configure Options**. This will display the Search tab of the General Options dialog.
2. From the main menu, choose **Tools > Options > General**, then select the **Search tab**. The Search tab is pictured below.



The following options are available:

- **Default search options** - The following default search options apply to all command line searches, quick searches and incremental searches, and to the Find and Replace tool window when the option **Initialize with default options** is checked.
  - **Match case** - If checked, various search commands default to case-sensitive searches.
  - **Match whole word** - If checked, refines search results to match only the word as a whole. By default, this is unchecked, and search results will match all instances of the word, ignoring characters that are to the left and right of the occurrence.
  - **Regular expression** - If checked, various search commands ( */*, **find**, or **c** ) default to regular expression searching. Specify which syntax to use from the drop-down list.
  - **Wrap at beginning/end** - If checked, various search commands default to wrapping to the beginning or end of a buffer to complete a search.
  - **Search backward** - If checked, searches are performed from the end to the beginning.
  - **Place cursor at end** - If checked, the cursor is placed at the end of the occurrence found.

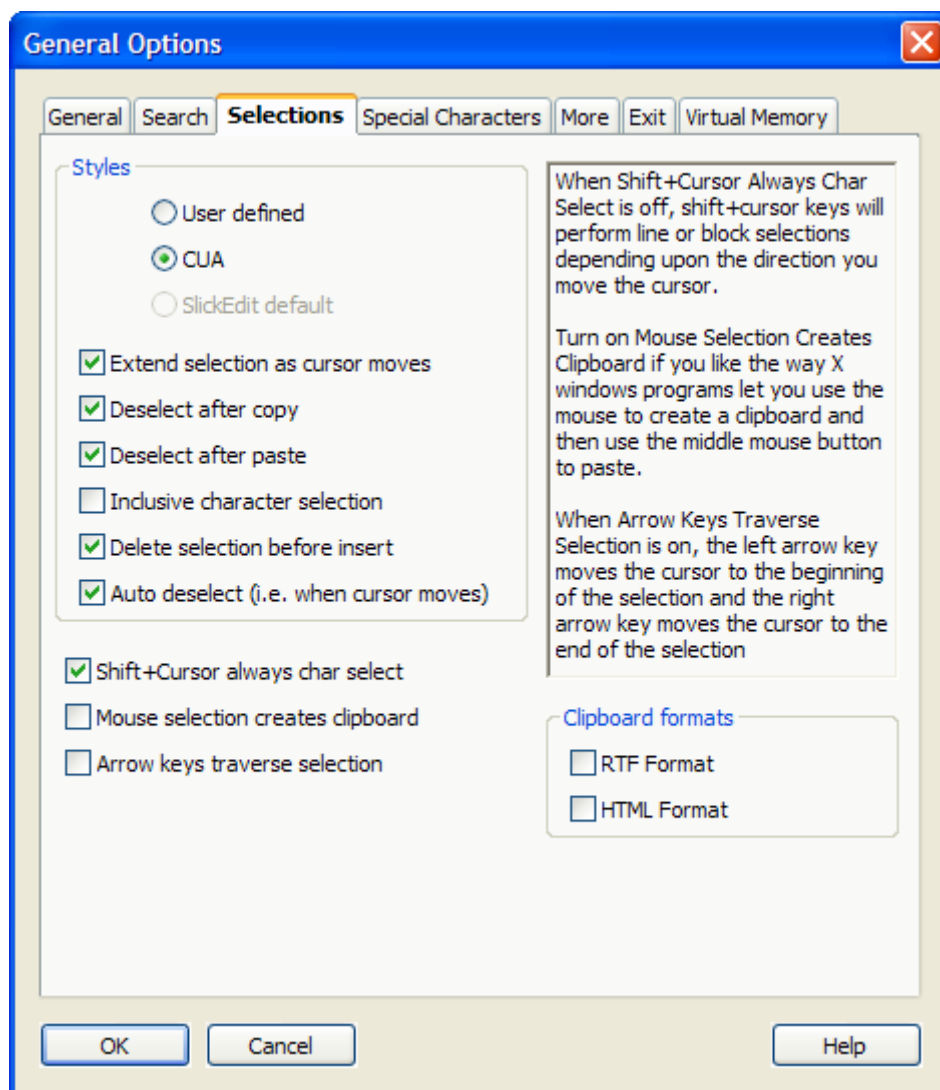
- **Search hidden text** - Check this option to search for text hidden by Selective Display. Matches found that were set to be hidden by Selective Display will be revealed. To enable Selective Display options, from the main menu choose **View > Selective Display**. See [Selective Display](#) for more information.
- **Tool window options** - The following options on the Search tab pertain to the Find and Replace tool window (see [Find and Replace Tool Window](#)):
  - **Close after Find/Replace** - If checked, the Find and Replace tool window is closed after finding text in the buffer.
  - **Initialize with default options** - If checked, the search options in the Find and Replace tool window will be reset to the default options each time it is launched. By default, this option is unchecked, and search options are retained when the Find and Replace tool window is closed and re-opened. The window will retain the current set of options as long as it remains open (this includes auto-hide when docked).
- **Search string initialization options** - The following options on the Search tab provide starting values for when a search and replace operation is activated:
  - **History retrieval** - If selected, the Find and Replace tool window uses the last item that was searched, for the word used when performing a search.
  - **Word at cursor** - If selected, the Find and Replace tool window uses the word that is at the cursor when performing a search.
  - **Selected text (if exists)** - If checked, the Find and Replace tool window uses the text that you select in the build window to perform the search.
- **Additional search options** - The following additional search options are available on the Search tab:
  - **Restore cursor after replace** - If checked, the cursor is restored to its original position after a search and replace operation completes and is not canceled.
  - **Leave selected** - If checked, the last occurrence of a search string that was found is left selected. This also affects whether pressing Esc during a search and replace leaves the search string selected.
  - **Incremental search highlighting** - If checked, when an incremental search is performed, matching occurrences will be highlighted with two colors: one for the current match at the cursor, and one for all possible matches. Highlights are removed when the incremental search command terminates. Highlight colors can be set using the [Color Settings Dialog \(Tools > Options > Color\)](#). Select the items **I-Search Current Match** and **I-Search Highlight** from the element list. See [Incremental Searching](#) for more information.
- **Bookmarks** - These options provide for the control and viewing of bookmarks. See [Bookmarks](#) for more information.
  - **Use workspace bookmarks** - By default, bookmarks are stored globally and are visible in all workspaces. If this option is checked, bookmarks are associated with the workspace used to create them, even if the files they are in are not part of the workspace. When you switch workspaces, the Bookmarks tool window will display only the bookmarks associated with this workspace.
  - **Show set bookmarks** - If checked, a green bookmark icon is displayed in the left margin of the editor control corresponding to the bookmarks you have set.
  - **Show pushed bookmarks** - If checked, a blue bookmark icon is displayed in the left margin of the editor control corresponding with each location on your bookmark stack. This helps you see where "Pop Bookmark" will go.



- **Close deletes pushed bookmarks** - If checked, when a buffer is closed (quit), any pushed bookmarks remaining in that file are removed. This option is helpful for buffer management, because it prevents buffers which were explicitly closed from coming back when you pop up out of your bookmark stack.
- **Max stack depth** - By default, the maximum stack depth for the bookmark stack is set to 15 entries. If you push more than this number of bookmarks, the oldest bookmark will be removed. Enter the number of entries that you want the bookmark stack to hold in this field.

## Selections Tab

The Selections tab, pictured below, is where you can set preferences for selections. These options are accessed from the main menu by choosing **Tools > Options > General**, then selecting the **Selections tab**. For more information about working with selections, see [Selections](#).



The following settings are available:

- **Styles** - Choose the selection style you wish to use from the following options:



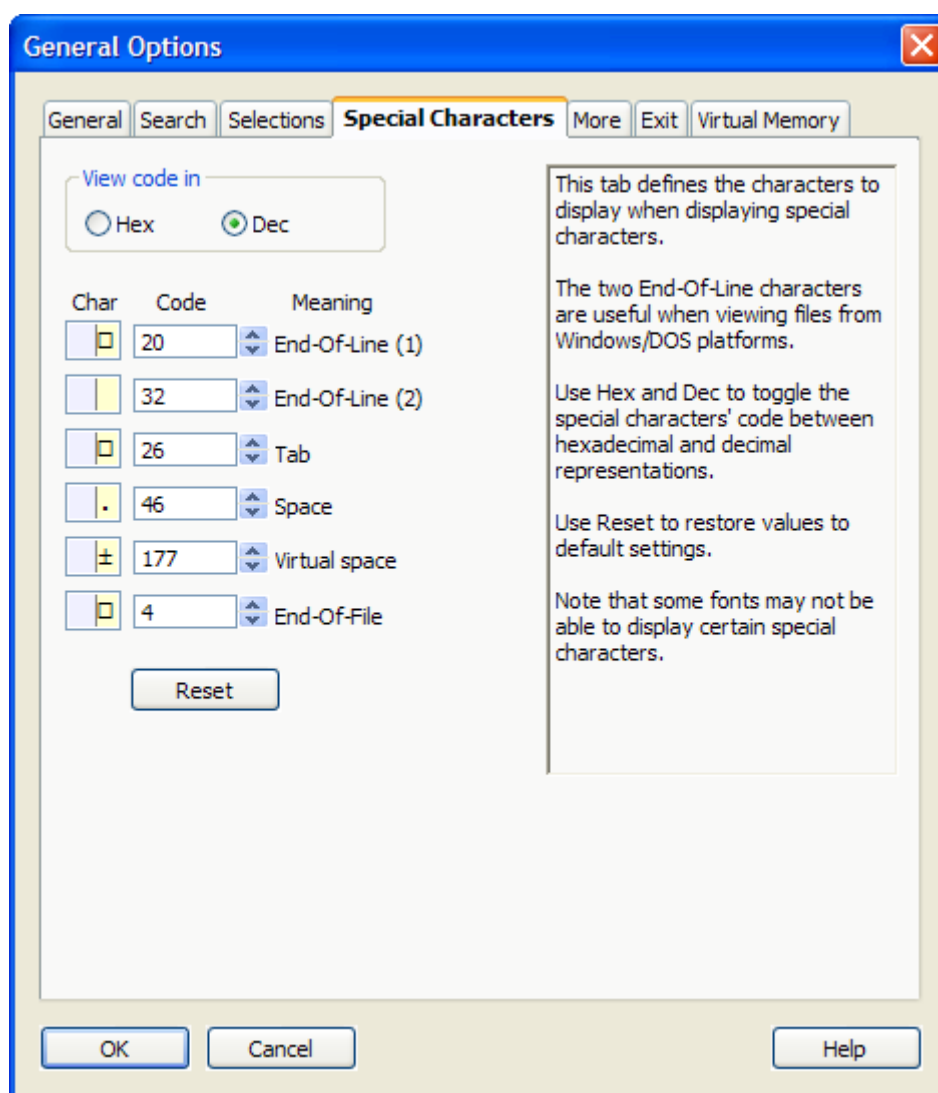
- **User defined** - This option is for setting your own selection preferences. Any changes that are made to the CUA behaviors automatically select **User Defined**. Selecting CUA automatically resets the select behaviors.
- **CUA (default)** - When this style is selected, selected text is deleted before a paste or character is inserted unless the selection is locked. Pressing the Backspace or Delete keys deletes the selection unless the selection is locked. Advanced selections (those selections not started with the mouse or Shift+ <arrow keys>) are extended as the cursor moves. Locking a selection requires one of the emulation commands **select\_line**, **select\_block**, or **select\_char**. To access these commands from Edit pull-down menu, select this option in any emulation.
- **SlickEdit default** - When this option is selected, SlickEdit® uses the default styles that are enabled when the product is installed.
- **Extend selection as cursor moves** - When this check box is selected, the selection is extended to cursor position. This option is not available if using Brief or Emacs emulation.
- **Deselect after copy** - Indicates whether copied text is selected. This is not available if using Brief or Emacs emulation.
- **Deselect after paste** - Indicates whether pasted text is selected. This is not available if using Brief or Emacs emulation.
- **Inclusive character selection** - When this check box is selected, a character selection includes the character following the cursor. This option is not available if using Brief or Emacs emulation.
- **Delete selection before insert** - Indicates whether a selection is deleted before new text is inserted. This option is not available if using a Brief or Emacs emulation.
- **Auto deselect** - Select this option to clear a selection when the cursor moves or one of a few other editor operations occurs. This option is not available if using a Brief or Emacs emulation.
- **Shift+Cursor always char select** - When this check box is cleared, pressing the Shift+<arrow keys> will select line or block selections, depending upon the direction the cursor moves. This is not available if using a Brief emulation.
- **Mouse selection creates clipboard** - Select this option to use the left mouse button to create a clipboard and to use the middle mouse button to paste.
- **Arrow keys traverse selection** - If selected, the left arrow key moves the cursor to the beginning of the selection and the right arrow key moves the cursor to the end of the selection.
- **Clipboard formats** - Select the type of editing format for clipboards, allowing you to paste formatted and color-coded text to other applications (as well as plain text). Choose from **Rich Text Format** or **HTML**.

## Special Characters Tab

Enabling view of special characters inserts characters into your file to show such items as tabs, spaces and line endings. Characters defined in the Special Characters tab are displayed instead of the original characters when the view options are selected.

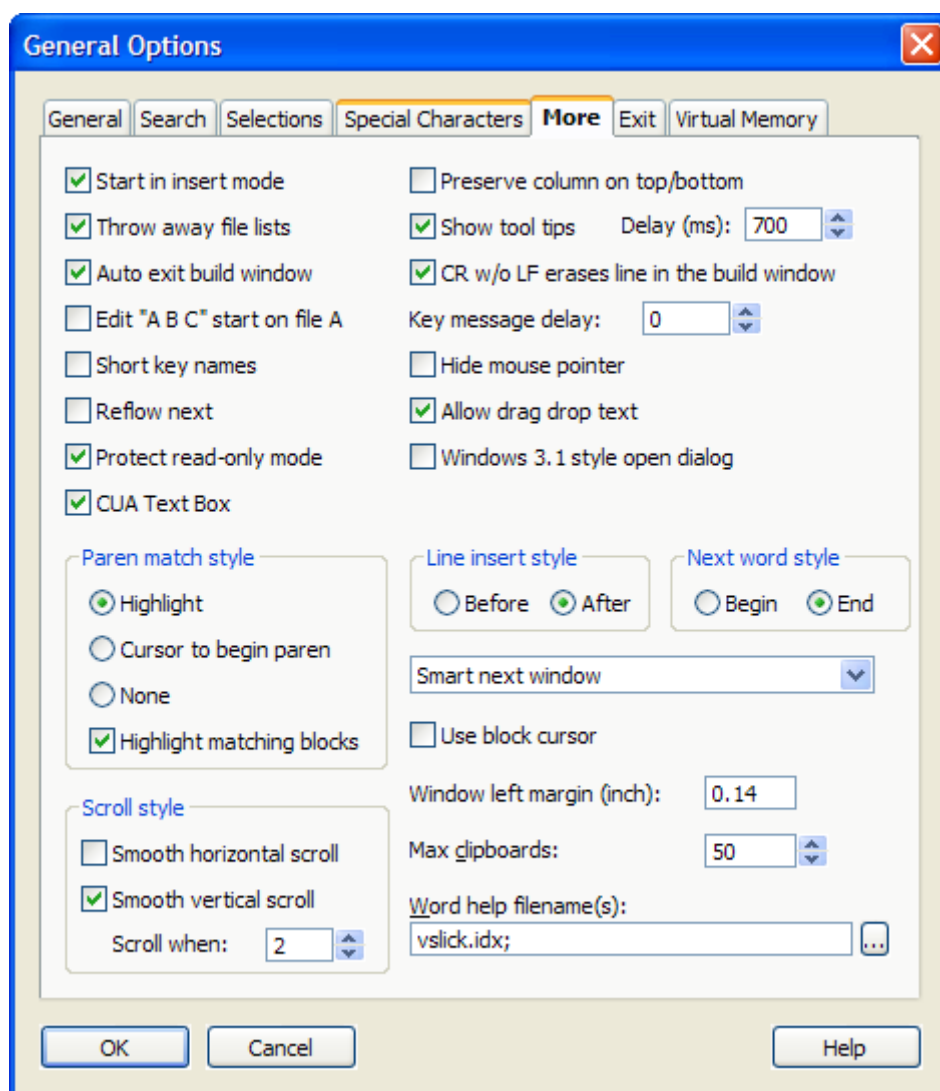
**NOTE** Viewing special characters is only available for ASCII files.

To access Special Characters options, from the main menu choose **Tools > Options > General**, then select the **Special Characters tab**, which is pictured below. See [Viewing Special Characters](#) for more information about these settings.



## More Tab

The More tab, pictured below, contains additional options that can be set for working with SlickEdit®. To access these options, choose **Tools > Options > General**, then select the **More tab**.



The following options are available:

- **Start in insert mode** - If selected, the insert mode is set to insert when the editor is invoked. Otherwise, the insert mode is set to replace.
- **Throw away file lists** - If selected, the modified file manager file lists can be modified and closed without being prompted to save.
- **Auto exit build window** - If selected, the concurrent build window is automatically exited when the buffer is closed or when exiting the editor.
- **Edit "A B C" start on file A** - If selected, the first file opened becomes the active buffer.
- **Short key names** - If selected, key names in the MDI menu bar are condensed (non-CUA). For long CUA key names, clear this check box.
- **Reflow next** - If selected, the **reflow\_paragraph** command places the cursor on the next paragraph after it has reformatted the current paragraph. Otherwise, the cursor is kept at the same location within the current paragraph.

- **Protect read-only mode** - If selected, the editor will not let you modify a file that is in read-only mode. The **save** command always prompts for a different output file name if the file is in read-only mode.
- **CUA text box** - If selected, the keys Ctrl+X, Ctrl+C, and Ctrl+V perform cut, copy, and paste commands respectively for a text box other than the command line. In addition, the dialog manager takes over the keys Alt+A through Alt+Z for selecting controls.

**NOTE** Do not mark this check box if you want all of the keys to operate the same in a text box as they do in the command line and edit windows.

- **Paren match style** - This feature always uses fast brace matching. Select from the following options (the first three are mutually exclusive):
  - **Highlight** - When selected, typing a closing parenthesis temporarily block-selects the text within the parenthesis pair.
  - **Cursor to begin paren** - When selected, typing a closing parenthesis temporarily places the cursor on the matching begin parenthesis.
  - **None** - When selected, typing a closing parenthesis just inserts the close parenthesis.
  - **Highlight matching blocks** - When selected, the corresponding parenthesis, brace, bracket, or begin/end word pairs under the cursor are automatically highlighted.

**TIP** To customize the highlighting color, go to **Tools > Options > Color**, and select the **Block Matching** screen element. To adjust the delay in milliseconds before the highlighting is updated, go to **Macro > Set Macro Variable** and modify the variable `def_match_paren_idle`. See [Setting Colors for Screen Elements](#) and [Setting Macro Variables](#) for more information.

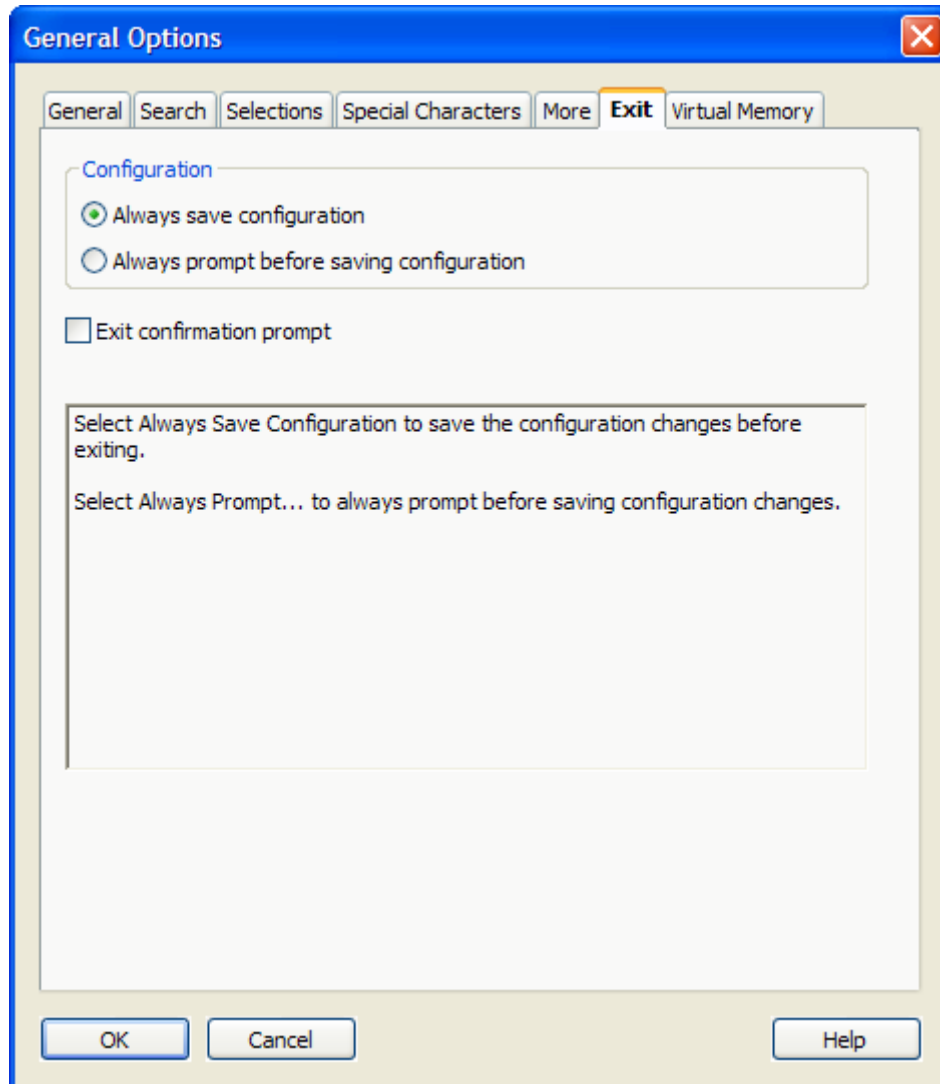
- **Scroll style** - Select from the following options:
  - **Smooth horizontal scroll** - When selected, this option specifies that the window should be scrolled column by column when the cursor moves out of view. When this option is deselected, the cursor will be centered and the text will be scrolled one fourth the width of the window when the cursor moves out of view.
  - **Smooth vertical scroll** - When selected, this option specifies that the window should be scrolled line by line when the cursor moves out of view. When this option is deselected, the cursor will be centered and the text will be scrolled half the height of the window when the cursor moves out of view.
  - **Scroll when** - Specifies how close (in number of lines) the cursor may get to the top or bottom of the window before scrolling occurs. Does not affect horizontal scrolling.
- **Preserve column on top/bottom** - If selected, the `top_of_buffer` (Ctrl+Home) and `bottom_of_buffer` (Ctrl+End) commands do not change the column position unless already at the top or bottom of the buffer.
- **Show tool tips** - If selected, the pop-up tool tip help messages are displayed when the mouse pointer rolls over a button.
- **Delay (ms)** - Selecting this option specifies delay in tenths of a second before tool tip messages are displayed.
- **CR w/o LF erases line in the build window** - It is possible that output sent to the Build tool window may contain carriage return (CR) characters without subsequent line feed (LF) characters.

This causes the line to be erased in the Build window. To prevent SlickEdit from erasing lines in this situation, deselect this option.

- **Key message delay** - Selecting this option specifies the delay before a prefix key is displayed in tenths of a second. The prefix key is not displayed if the next key is pressed before the delay specified in this text box.
- **Hide mouse pointer** - Selecting this option hides the mouse pointer when typing. The mouse pointer is displayed when moving the mouse or when a dialog box is displayed.
- **Allow drag drop text** - If selected, selected text can be copied or moved by dragging and dropping the selected text using the left mouse button.
- **Windows 3.1 style open dialog** - (Windows only) When selected, a Windows 3.1-style Open dialog box is used to open and save files. This dialog does not support all the features of the default Open dialog (for example, encoding options are not supported). For more information on this dialog, see [Open Dialog - Linux, UNIX, and Mac](#).
- **Line insert style** - SlickEdit treats line selections differently than character selections. Line selections are pasted either above or below the current line, saving you from tediously positioning the cursor at the beginning or end of a line prior to pasting. When the style is set to **Before**, lines of text are inserted before the current line. When the style is set to **After** (the default), lines of text are inserted after the current line.
- **Next word style** - When the next word style is set to **Begin**, the **next\_word** command (Ctrl+right arrow) places the cursor on the beginning of the next word. When the next word style is set to **End**, the cursor is placed at the end of the next word.
- **Smart next window** - Use this combo box to select from the following different windowing styles:
  - **Smart next window** - This is the default style. It allows you to press **Ctrl+Tab** (**next\_window** command) to switch the focus between the two most frequently used open editor windows, rather than always going to the next window. Press **Ctrl+Shift+Tab** (**prev\_window** command) to switch between all open editor windows. This style is similar to how Ctrl+Tab and Ctrl+Shift+Tab work in other Windows MDI applications, like Visual Studio.
  - **Reorder windows** - If selected, activating an existing window reinserts the window after the current window. Neither Ctrl+Tab nor Ctrl+Shift+Tab reorders the windows. This option is very good for switching between more than two files, but it is not the Windows standard (which means you're probably not used to it). It's similar to the way SlickEdit reorders buffers.
  - **No window reordering** - If selected, newly opened windows are inserted after the current window, like in all settings. Activating an existing window, pressing Ctrl+Tab, or pressing Ctrl+Shift+Tab does not reorder windows. This option is best if you like to memorize the hot key numbers on the Window menu (for example, Alt+W 1) because it attempts to keep the hot key numbers the same.
- **Use block cursor** - If selected, a text mode-style cursor is used instead of a vertical cursor.
- **Window left margin (inch)** - If selected, this specifies the space between the left edge of the window and the editor text in inches. This value has no effect when there are bitmaps displayed in the left margin since more space is necessary.
- **Max clipboards** - Specifies the maximum number of clipboards saved. By default, a stack of your last 50 clipboards are kept, any one of which can be pasted with Ctr+Shift+V.
- **Word help filename** - Specifies the word help file names used by the **wh** command (**Help > F1 Index Help** or Ctrl+F1), **wh2** command (Ctrl+F2), and **wh3** command. See [Changing the Default Word Help File\(s\)](#) for more information about this setting.

## Exit Tab

The Exit tab, pictured below, contains settings that occur when exiting the editor. To access these settings, choose **Tools > Options > General**, then select the **Exit** tab. For more information about exiting the editor, see [Exiting the Program](#).

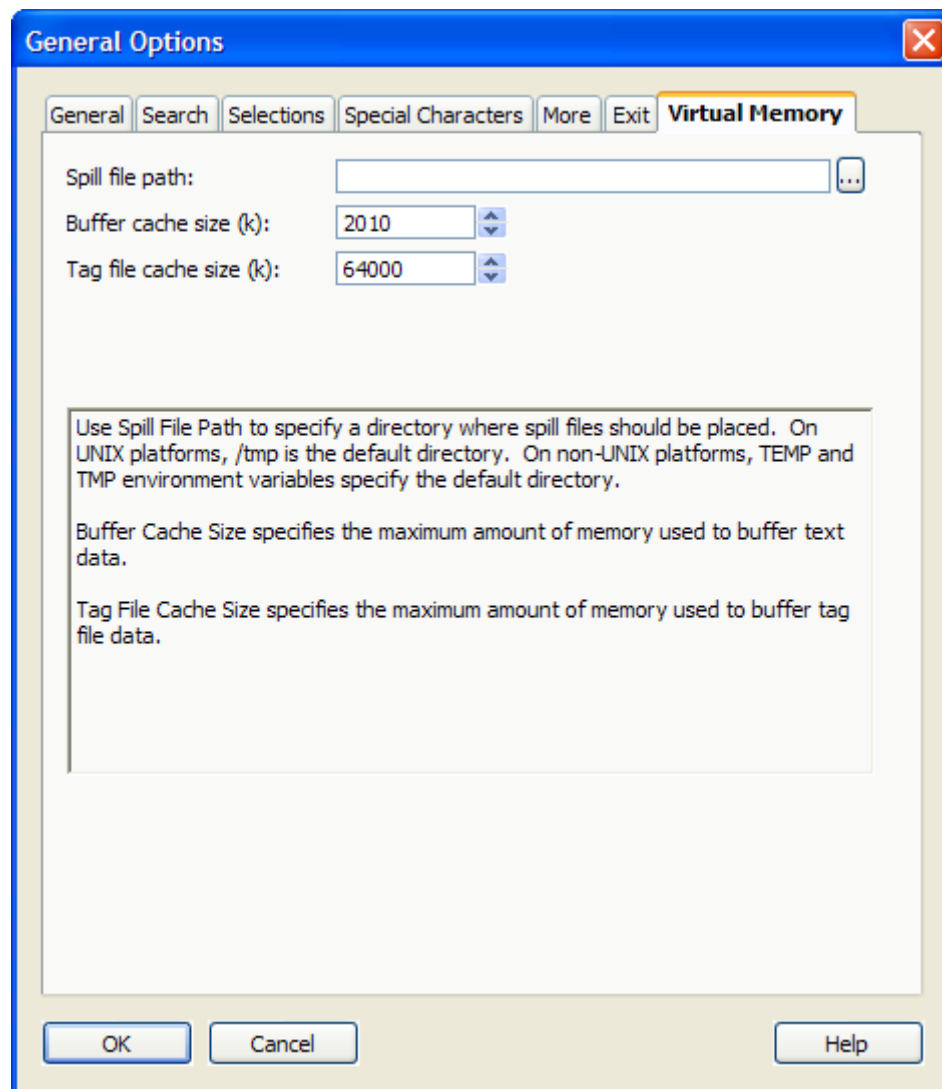


The following options are available on the Exit tab:

- **Always save configuration** - If selected, configuration changes are saved without prompting.
- **Always prompt before saving configuration** - Select this option to always receive a prompt before saving changes that you make to the configuration of the software.
- **Exit confirmation prompt** - If selected, you will always be prompted when exiting the editor.

## Virtual Memory Tab

To access virtual memory options, choose **Tools > Options > General**, then select the **Virtual Memory** tab (pictured below). For more information about working with files, see [Creating, Opening, and Saving Files](#).



The following options are available:

- **Spill file path** - This text box specifies a directory where spill files and temporary files should be placed.

On Windows, this defaults to the directory specified the TEMP environment variable. If it does not exist, the directory specified by the TMP environment variable is used. On UNIX, this defaults to the directory specified by the TMP environment variable.

- **Buffer cache size** - The value in this field, specifies the maximum amount of memory used to store text buffer data in kilobytes. A value that is less than zero specifies all available memory.

**CAUTION** If the operating system starts the swapping process before the cache is full, performance might be degraded. The cache size must be smaller than the amount of actual memory available.

- **Tag file cache size** - You can improve tagging performance by adjusting the tag file cache size to better match the size of your tag files. Generally, a tag file cache size that matches the total size of

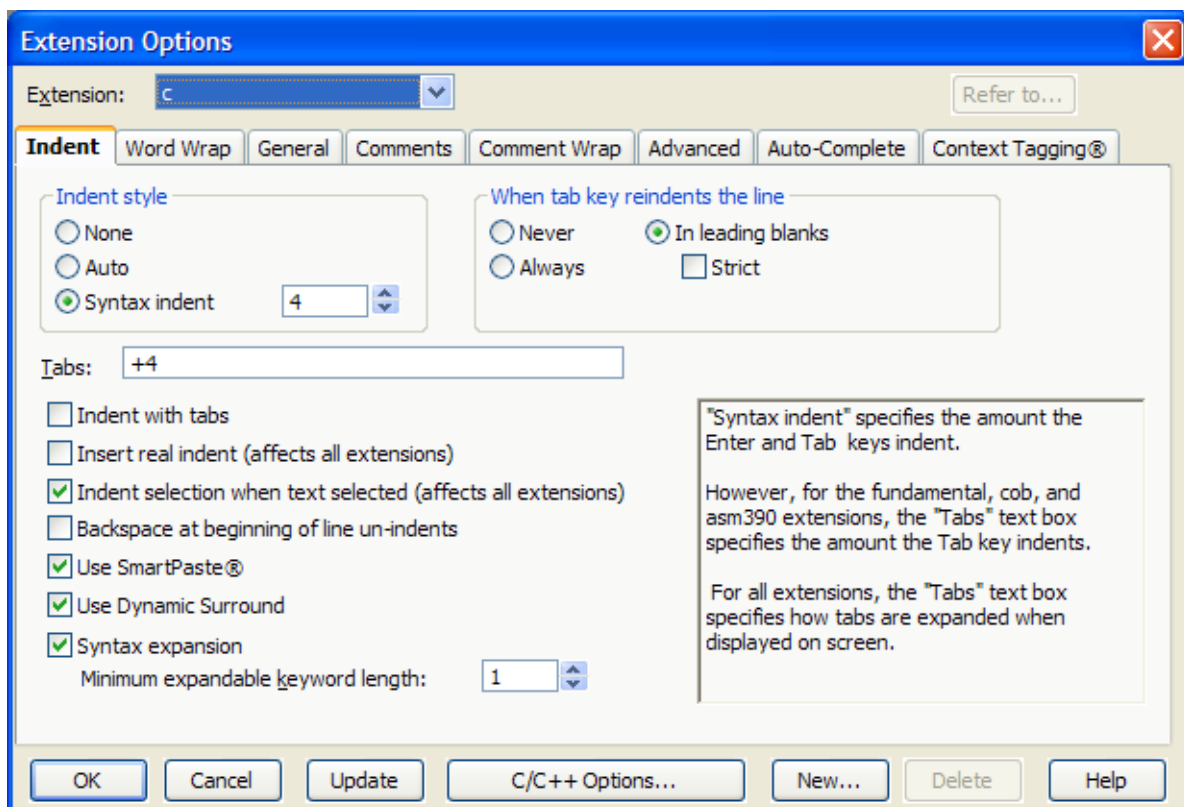
## OPTIONS

the tag files being used will provide the best performance. For example, if the tag files for your source code and libraries adds up to 100 MB, you should set your cache size to 100 MB. You may have to experiment to find the optimum value. Use the recommendations below as a guide. Note that this is the same option as found on the [Context Tagging® Options Dialog](#). For more information about tagging, see [Building and Managing Tag Files](#).

- Minimum – 8 MB
- Default – 64 MB
- Ideal – Sum of tag file sizes
- Maximum – 25% of physical system memory

## Extension Options Dialog

The behavior of the editor can be customized for files based on specific language extensions. The Extension Options dialog box (shown below) contains the settings that can be configured for file extensions. This dialog can be accessed from the main menu by choosing **Tools > Options > File Extension Setup**, or by using the **setupext** command.



General settings on this dialog box are described below (see [Extension Options - General Dialog Settings](#)). Other options are categorized into the following tabs. Click on an item to go to that section in the documentation:

- [Indent Tab](#)
- [Word Wrap Tab](#)
- [General Tab](#)
- [Comments Tab](#)
- [Comment Wrap Tab](#)



- [Advanced Tab](#)
- [Auto-Complete Tab](#)
- [Context Tagging® Tab](#)

## Extension Options - General Dialog Settings

The following fields and buttons are available on the Extension Options dialog:

- **Extension** - The Extension drop-down list (located above the tabs) contains a list of pre-loaded extensions that support Extension Options. To enable Extension Options for an extension that is not in the list, you must first load or create a file with that extension.

**NOTE** Before configuring any of the Extension Options, use the **Extension** drop-down list to select the extension you wish to affect.

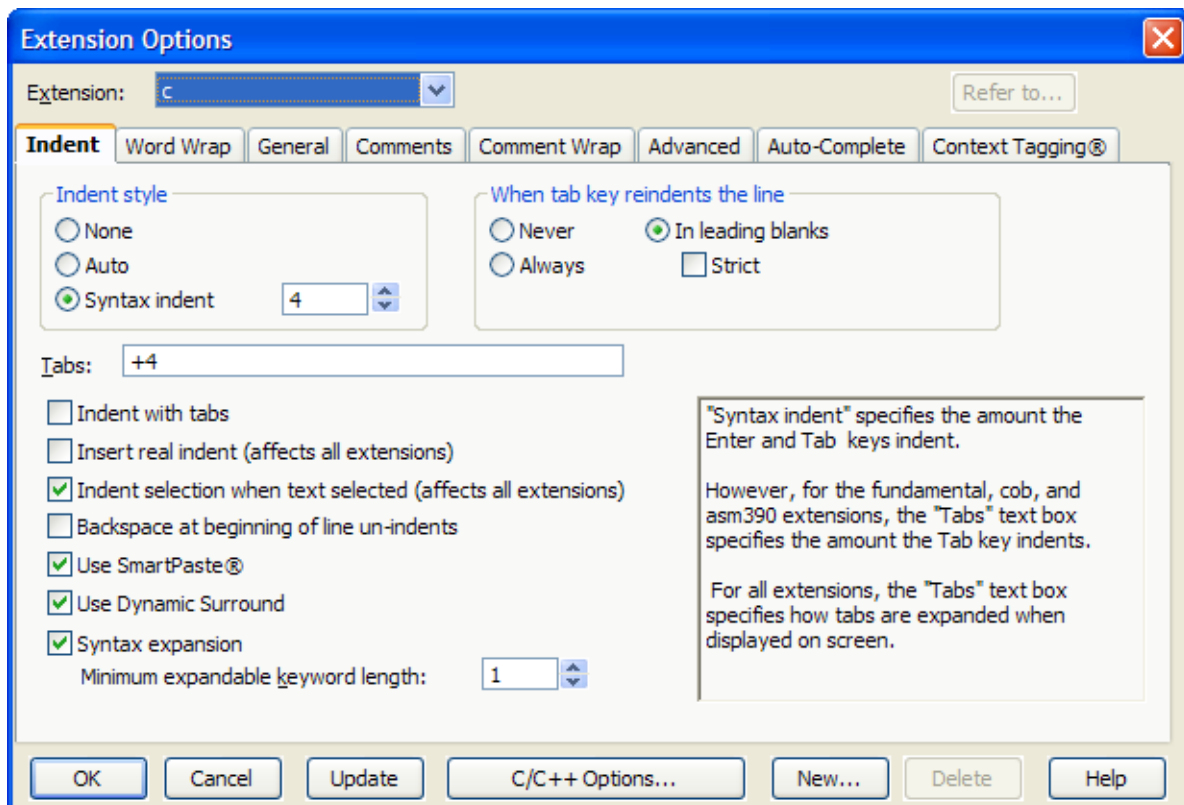
- **Refer to** - When an extension refers to another extension, both extensions operate exactly the same. That is, all Context Tagging®, template editing, word processing options, and all other extension options are the same. In addition, modifying the extension option information for either extension updates both extensions. For example, by default, the `.h` and `.cpp` extensions refer to the `.c` file extension. Modify the `.h` or `.cpp` extension setup to modify the extension setup for all three extensions. In addition, the `.h` and `.cpp` extensions use the same Context Tagging® settings as the `.c` extension.

To have the setup data for one extension refer to another extension, click the **Refer to** button (located at the top right corner of the Extension Options dialog). This button is not enabled if other extensions already refer to this one. **Refer to** is also available on the New Extension dialog box to set when adding a new extension.

- **Update** - To update the extension setup, click this button (located at the bottom of the Extension Options dialog). This is useful when modifying the options for more than one extension, as it keeps the Extension Options dialog box displayed.
- **Options** - To access formatting options such as brace styles, indentation, and other code style settings, click the **Options** button (located at the bottom of the Extension Options dialog). This will display a Formatting Options dialog box that contains options specific to the extension that is selected. Each dialog is described in the appropriate section in the chapter [Language-Specific Editing](#).
- **New** - To create a new extension and assign extension-specific options to it, including **Refer to**, click the **New** button (located at the bottom of the Extension Options dialog). The New Extension dialog box is displayed. See [Creating a New Extension](#) for more information.
- **Delete** - To delete the selected file extension's setup information, click the **Delete** button (located at the bottom of the Extension Options dialog). Note that the Fundamental extension setup information cannot be deleted. An extension such as C, that has other extensions (such as H, CPP, and CXX) that refer to it, cannot be deleted until all of the extensions that refer to it are deleted.

## Indent Tab

You can set the indent configurations for specific file types. To access these settings, from the main menu choose **Tools > Options > File Extension Setup**, then select the **Indent tab** (pictured below). For more information about working with the features controlled by these options, see [Syntax Indent and Smart-Paste®](#).



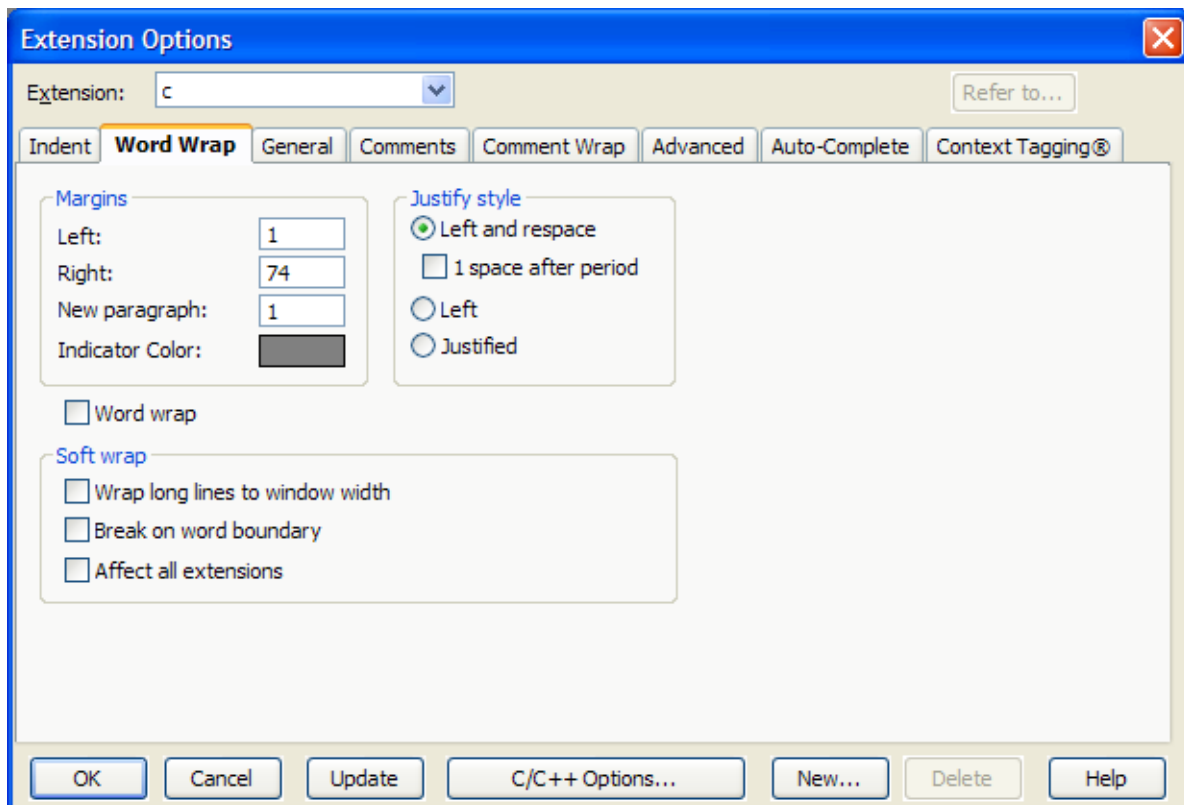
The following options and settings are available:

- **Indent style** - Select from the following indent styles:
  - **None** - When this option is selected, the Enter key will put the cursor at the beginning of the line.
  - **Auto** - When this option is selected, the Enter key indents according to the previous line.
  - **Syntax indent** - When this option is selected, the Enter key indents according to language syntax. The value in the text box specifies the amount to indent for each level. See [Syntax Indent](#) for more information.
- **When tab key reindents the line** - These options specify that the Tab key be used to beautify or reindent the current line. Select from the following settings:
  - **Never** - When this option is selected, pressing Tab will never reindent the line. It will indent to the next tab stop.
  - **Always** - Pressing the Tab key in any column will reindent the current line.
  - **In leading blanks** - Pressing the Tab key will reindent the line if the cursor is positioned within the leading white space of the line. Otherwise it will indent to the next tab stop. This option is further controlled by the **Strict** check box.
  - **Strict** - Strict only applies to the **In leading blanks** option. When this option is selected, it reindents the line only if the cursor position is before the intended indent location; otherwise, it will insert an additional tab stop. When this option is deselected, it reindents the line when the cursor is located on the leading whitespace, regardless of whether the column is before or after the intended indent location.

- **Tabs** - Set tabs in increments of a specific value or at specific column positions. To specify an increment of three, enter "3" in the text box. To specify columns, for example, enter "1 8 27 44", to specify tab stops that are not an increment of a specific value.
- **Indent with tabs** - Determines whether Tab key, Enter key, and paragraph reformat commands indent with spaces or tabs. See [Indenting with Tabs](#) for more information.
- **Insert real indent (affects all extensions)** - When this option is selected, the Enter key inserts real spaces or tabs representing the indent instead of virtual spaces. This check box affects all extensions when it is selected. This option enables the function for the End key on the keyboard to place the cursor after blank text where new text can be typed.
- **Indent selection when text selected (affects all extensions)** - When this option is selected, it affects all of the extensions, and pressing Tab or Shift+Tab indents or un-indents the selected text.
- **Backspace at beginning of line un-indents** - When this option is selected and the cursor is located before the first non-blank character, pressing the Backspace key un-indents the current line by one indent level. See also [Setting the Backspace Unindent Style](#).
- **Use SmartPaste®** - Specifies whether copied or pasted text should be reindented according to what the editor thinks is the correct indent level. See [SmartPaste®](#) for more information.
- **Use Dynamic Surround** - Provides the ability to surround a group of statements with a block statement, indented to the correct levels according to your indent settings on this tab. In order for Dynamic Surround to work, the option **Syntax Expansion** must also be selected (see below). See [Dynamic Surround](#) for more information on how to use this feature.
- **Syntax expansion** - Enables the Syntax Expansion feature. When this option is selected, pressing the spacebar after typing a word such as "if" or "for" will cause that syntax element to be expanded, inserting the rest of the **if** or **for** statement. Alternately, you can bind a space command to a key other than the spacebar. See [Syntax Expansion](#) for more information on using this feature.
- **Minimum expandable keyword length** - Sets the minimum length for a keyword that will trigger syntax expansion. For example, if this is set to 3, then two-letter keywords such as "if" will not be expanded.

## Word Wrap Tab

The Word Wrap tab contains options for controlling how text is wrapped. To access these options, choose **Tools > Options > File Extension Setup**, then select the **Word Wrap** tab (pictured below).



The following settings are available:

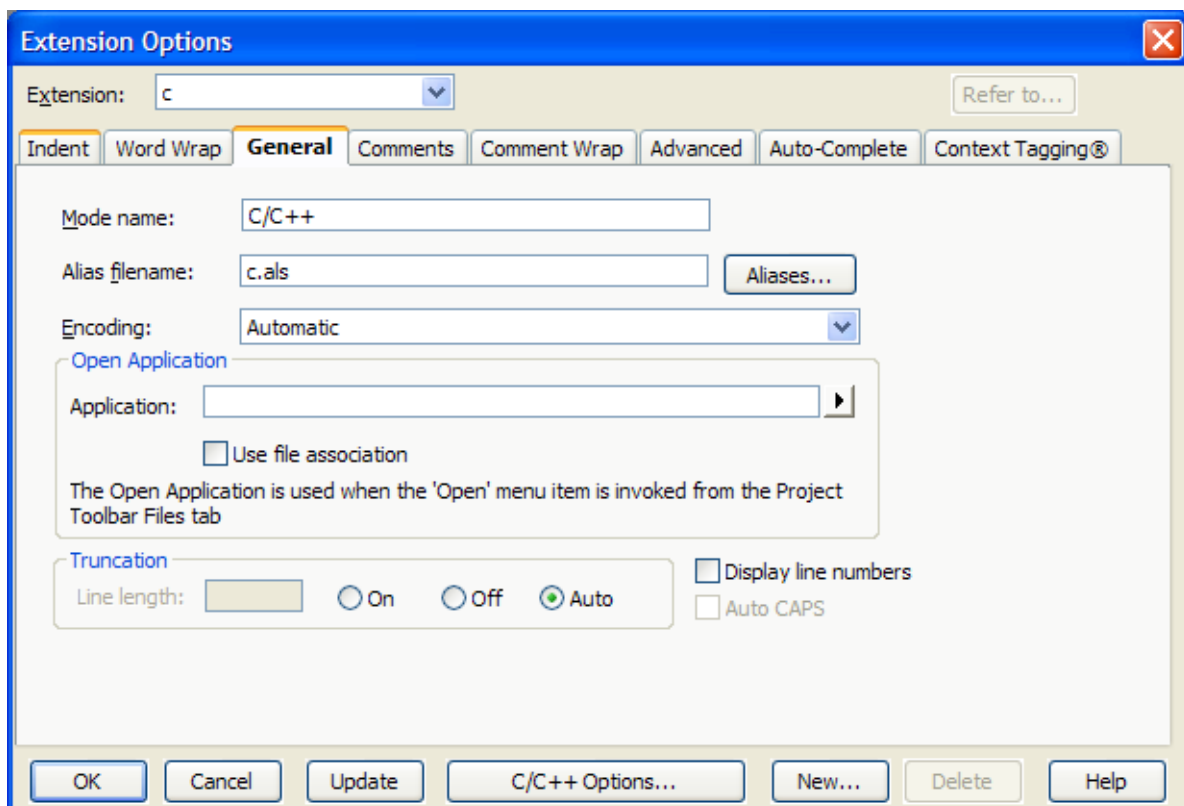
- **Margins** - Sets the left, right, and new paragraph margins. Specify the column number at which each margin should begin. Click the colored box next to the **Indicator Color** label to set the color of the margin indicator. The margin indicator will only appear if the **Word wrap** option is selected, which enables word wrapping.
- **Justify style** - Select from the following justification styles:
  - **Left and respaced** - Left justification with space character reformatting. One space is placed between words except after the punctuation characters (.,?, and !) that get two spaces. To have only one space after the period, question mark, and exclamation point (.,?, and !) punctuation characters, then turn on **1 space after period**.
  - **Left** - Left justification with respect for space characters between words. This setting requires the save options to be set such that trailing spaces are not stripped when a buffer is saved. See [Save Tab](#) for more information on save options.
  - **Justified** - Full justification. Left and right edges of text will align exactly at margins.
- **Word wrap** - This option enables/disables word wrapping. When selected, the editor keeps the cursor within the margins when entering text, moving the cursor, and deleting characters.
- **Soft wrap** - Soft Wrap makes it easy to view long lines of code without scrolling. Each line is wrapped as though a carriage return was inserted, however, the file itself is not modified. The options are as follows:
  - **Wrap long lines to window width** - This option enables Soft Wrap. A curved arrow is displayed at the end of each line, along the right-hand border of the edit pane, indicating that

the text continues on the next line. The horizontal scrollbar disappears as it is no longer needed.

- **Break on word boundary** - Breaks the text at the end of line so that words are kept whole. This makes for easier reading, especially in text files.
- **Affects all extensions** - When selected, this option applies the Soft Wrap configuration that you want on all extensions without having to manually set up each one. For example, if you want Soft Wrap enabled for all extension types, complete the following steps:
  1. Mark **Wrap long lines to window width**.
  2. Mark **Break on word boundary**.
  3. Then mark **Affects all extensions** and click either **Update** or **OK**. Each time that you open a file, Soft Wrap is automatically turned on.
  4. To make this feature unavailable for all extensions, clear the **Wrap long lines to window width** check box, then click either **Update** or **OK**.

## General Tab

The General tab on the file extensions tab provides general and Alias options. To access these options, choose **Tools > Options > File Extension Setup**, then select the **General tab** (pictured below).



The following options are available:

- **Mode name** - Allows you to enter a more meaningful name for this extension setup. Define a mode name here for the **Document > Select Mode** menu item to work well.
- **Alias filename** - An alias defines a snippet of text that is inserted when the alias is expanded. Each extension can have one alias file, allowing aliases to be defined that do not affect other

extensions. An example would be a comment header that is used a lot. See [Extension-Specific Aliases](#) for more information.

- **Aliases** - Click the Aliases button to easily define extension-specific aliases. See [Extension-Specific Aliases](#) for more information.
- **Encoding** - Each extension can have its own encoding specification. Both the extension-specific and global settings are overridden if an encoding is previously specified in the Open dialog box. The encoding used to override default encoding settings is recorded and this setting is used the next time the same file is opened. This provides per-file encoding support. If the extension-specific encoding is set to **Default**, then the global setting defined in the File Options dialog (**Tools > Options > File Options, Load** tab) is used. Note that Unicode support is required to work with encodings. For more information about working with encodings and Unicode, see [Encoding](#).
- **Truncation** - When **On** or **Auto** is selected, all editor operations prevent the data from the right of the truncation line length to be moved or to be modified. For example, search and replace operations do not find data to the right of the truncating line length. In addition, when a replace occurs, the data to the right of the truncation line length will not move.

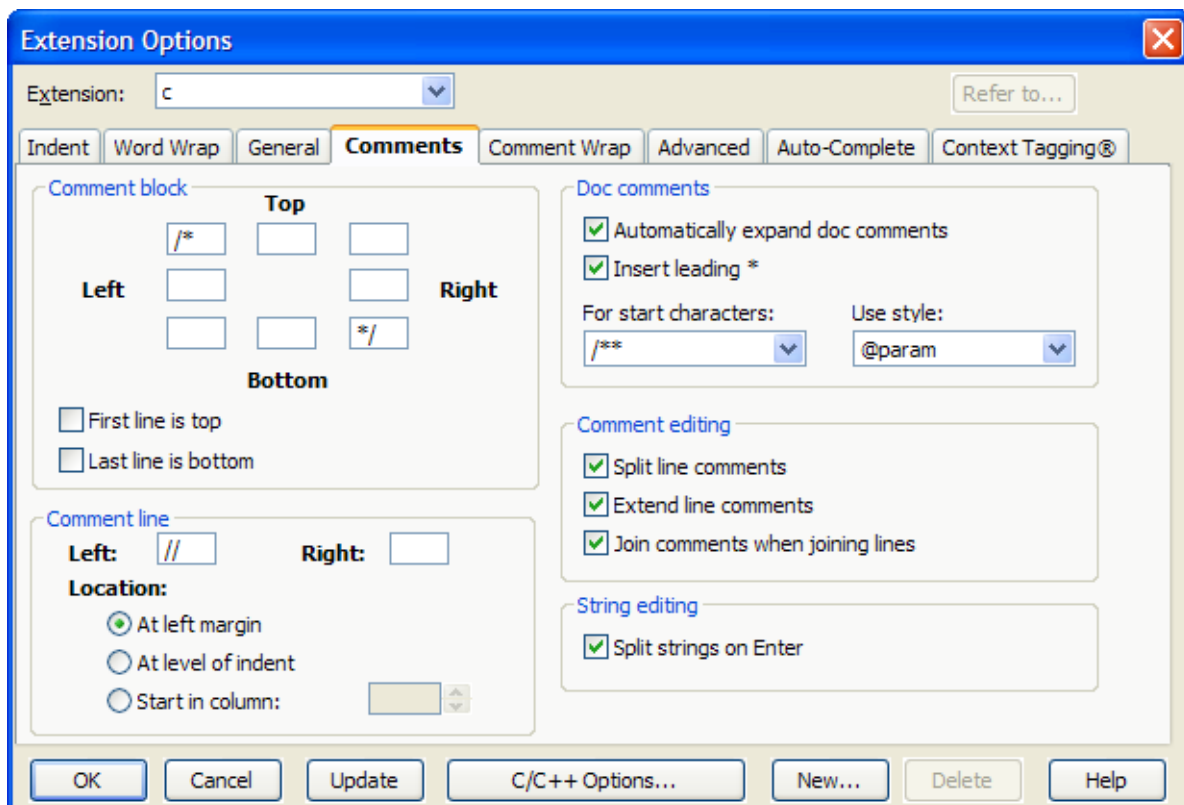
Set this to **Auto** for the editor to determine the truncation line length based on the record format of the file. For files that do not have a record format, the truncation length is turned off. For example, when **Auto** is on and the record width of the file is 80, 72 is used as the truncation line length (the record length minus eight).

- **Display line numbers** - When this option is selected, line numbers are displayed for any file with the selected extension. To toggle the display of line numbers on a per-document basis, from the main menu, click **View > Line Numbers**, or use the **view\_line\_numbers\_toggle** command on the SlickEdit command line.
- **Auto CAPS** - If selected, and a file is opened that does not contain any lowercase characters, caps mode is turned on (not the same as caps lock). When caps mode is on, all text is inserted in uppercase. This feature is intended to emulate ISPF.

## Comments Tab

The Comments tab provides options to control the creation of block and line comments. To comment out selected lines, select text in the editor and then select **Document > Comment Block** or **Document > Comment Lines** (**box** and **comment** commands, respectively). These operations will use the matching comment style to comment out all text on the lines containing the selection. A Comment Block will surround multiple lines with a single block comment. Comment Lines will comment out each line in the selection with a line comment. See [Commenting](#) for more information.

To access comment options, from the main menu click **Document > Comment Setup** (**comment\_setup** command), or **Tools > Options > File Extension Setup**, then select the **Comments** tab.



### Comment block

These settings are used when you comment out a selected block of text (**Document > Comment Block** or **box** command). SlickEdit® provides eight fields to specify the characters used in your commenting style. If you want to apply a comment with no additional decoration, fill in the upper-left and lower-right fields with the characters to begin and end a block comment. To draw a box around the comment, fill in additional characters in the other fields. For example, you might put an asterisk in each of the other fields to draw a box of asterisks around the block comment.

SlickEdit interprets the contents of these fields literally. If you want the asterisks on the left-hand side to line up, you need to put a space before the asterisk in the left, middle field. Likewise you would put a space before the asterisk and slash in the field containing the end of comment characters. Trailing spaces are ignored on the right-hand fields.

To illustrate, the following code sample is a selection:

```
if (!enabled) {
    tabState = TIS_DISABLED;
}
```

From the main menu, click **Document > Comment Block**, and the selection is commented out as follows:

```
/*
    if (!enabled) {
        tabState = TIS_DISABLED;
    }
*/
```

Select from the following comment block options:

## OPTIONS

- **First line is top** - When this option is selected, the first line of the text selection is used as the first line of the comment. The top border is not drawn. Otherwise the open comment characters will appear on their own line.

If this option is selected for the preceding code sample, the comment will instead be formatted as follows:

```
/*          if (!enabled) {
            tabState = TIS_DISABLED;
        }
*/
```

- **Last line is bottom** - When this option is selected, the last line of the text selection is used as the last line of the comment. The bottom border is not drawn. Otherwise the open comment characters appear on their own line.

Using the same example, if this option is selected, the comment will be formatted as follows:

```
/*
        if (!enabled) {
            tabState = TIS_DISABLED;
        }
*/
```

### Comment line

These settings are used when you comment out selected lines (**Document > Comment Lines** or **comment** command).

- **Left and Right** - Characters that you specify in these boxes are literally inserted to the left and right of the text on each line of the selection when you use SlickEdit® to create a line comment. The placement of the **Left** characters can be controlled through the **Location** options below. Characters specified in the **Right** box are placed and aligned vertically at the end of the longest line of text in the selection. For example, if the **Left** and **Right** boxes both contain the characters **//**, selecting **Document > Comment Line** comments out the example code as follows:

```
//          if (!enabled) {
//          tabState = TIS_DISABLED;
//          }
```

- **Location** - Mutually exclusive location options control where characters specified in the **Left** box are placed:
  - **At left margin** - Places characters flush against the left margin of the editor window, as shown in the previous example. The indent levels are not changed. This provides better visibility for your comments and a way to clearly see the indent level relative to lines that are not commented out.
  - **At level of indent** - Places and aligns characters vertically at the current indent level. For example:

```
//if (!enabled) {
//  tabState = TIS_DISABLED;
//}
```

- **Start in column** - Specifies in which column to start the comment for a line selection. This is useful for column-oriented languages such as COBOL. Type or use the spin box to select the desired column number. The left comment characters are placed at the specified column.

### Doc comments

Select from the following options:



- **Automatically expand doc comments** - When this option is selected, SlickEdit® automatically inserts a skeleton doc comment when you type comment start characters and then press **Enter** on a line directly above a function, class, or variable. The type of skeleton that is inserted is based on your start characters and style settings (specified in the **For start characters** and **Use style** boxes).

**NOTE** In C#, you do not need to press **Enter**, as the skeleton comment is inserted after you type the third slash.

- **Insert leading \*** - If selected, when you press **Enter** inside a doc comment, a leading asterisk is automatically inserted on the next line. For example:

```
/**
 * This is my comment.[CURSOR HERE]
 */
```

Pressing **Enter** will result in:

```
/**
 * This is my comment.
 * [CURSOR HERE]
 */
```

- **For start characters** - Use this box to specify the comment start characters that will trigger the style of reference tags that are automatically inserted as part of the doc comment skeleton. The characters selected here use the reference tag style specified in the **Use style** box. For comments formatted in Javadoc, select **/\*\***. For XMLdoc, select **///**. For Doxygen, select **/\*!** or **///**. See [Doc Comment Examples](#) for more information.
- **Use style** - Select the tag style to use for the corresponding start characters. This tag style is used when SlickEdit creates skeleton doc comments beginning with the matching start characters. For comments formatted in Javadoc, select the **@param** style. For XMLdoc, select the **<param>** style. For Doxygen, select the **\param** style. You can click through the start characters, assigning each one with a particular style and the settings will be remembered. See [Doc Comment Examples](#) for more information.

## Comment editing

The following options control comment editing behaviors. These options will be disabled for non-applicable extensions.

- **Split line comments** - If selected, when you press **Enter** in the middle of a line comment, a new line comment will automatically be started on the new line. For example:

```
// The quick brown fox [CURSOR_HERE]jumped over the lazy dog.
```

Pressing **Enter** will result in:

```
// The quick brown fox
// [CURSOR_HERE]jumped over the lazy dog.
```

- **Extend line comments** - If selected, when you press **Enter** at the end of a line containing a line comment, and there is also an aligned line comment on the line before or after the current line, a new line comment will automatically be started on the new line. For example:

```
// The quick brown fox
// jumped over the lazy dog.[CURSOR_HERE]
```

Pressing **Enter** will result in:

```
// The quick brown fox
// jumped over the lazy dog.
// [CURSOR_HERE]
```

- **Join comments when joining lines** - If selected, when you press **Delete** at the end of a line containing a line comment to join the current line with the next line, and the next line is also a line comment, the line comment characters will automatically be deleted. For example:

```
// The quick brown fox [CURSOR_HERE]
// jumped over the lazy dog.
```

Pressing **Delete** will result in:

```
// The quick brown fox [CURSOR_HERE] jumped over the lazy dog.
```

### String editing

If **Split strings on Enter** is selected, when you press **Enter** to split a line when the cursor is inside of a string, the closing and opening quotes and, if necessary, operators, will automatically be inserted, and the string will be aligned with the original string. For example:

```
String x = "The quick brown fox [CURSOR_HERE] jumped over the lazy dog.";
```

Pressing **Enter** will result in:

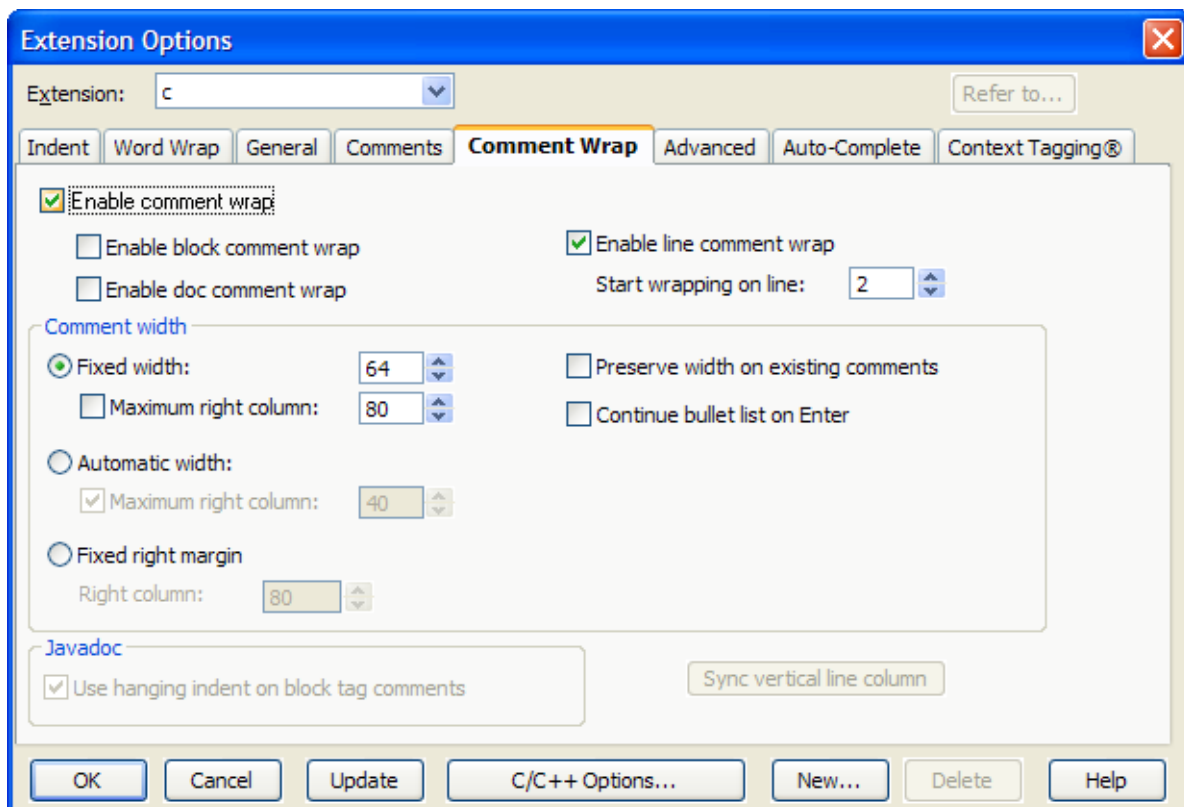
```
String x = "The quick brown fox "+
           "[CURSOR_HERE] jumped over the lazy dog.";
```

### Comment Wrap Tab

Comment wrapping options can be configured for C, C++, C#, Java, and Slick-C® files. These options are currently unavailable for other languages. Use the Comment Wrap tab to enable comment wrapping and configure options for how block, line, and doc comments are wrapped.

**TIP** After configuring comment wrap settings, you can use the Reflow Comment dialog to reflow block comments, paragraphs, or a selection of the current file. See [Reflowing Comments](#).

To access the Comment Wrap tab, from the main menu choose **Tools > Options > File Extension Setup**, make sure the extension you want to work with is selected in the **Extension** box, then select the **Comment Wrap** tab.



The following options are available:

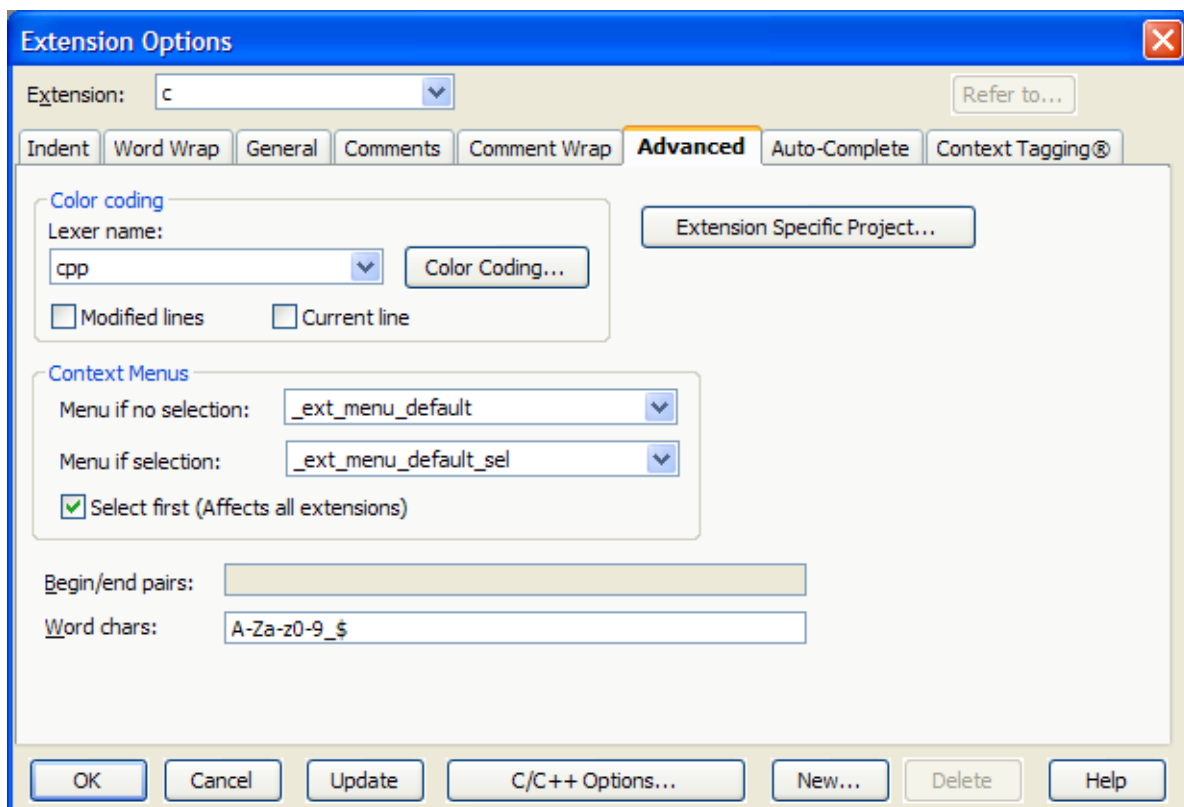
- **Enable comment wrap** - When selected, comments are allowed to be wrapped. You must still enable the type of comments that you want wrapped by selecting one or more of the **Enable** options for block, line, and doc comments.
  - **Start wrapping on line** - This setting pertains to line comments only. Make sure line comment wrapping is enabled, then type or select the number of consecutive line comments that must be present before wrapping is activated. If your code contains many one line descriptive comments, you may want to set this to 2 or more so that comment wrapping will not affect these short line comments.
- **Comment width** - There are three types of width settings for comments:
  - **Fixed width** - If selected, comments are formatted to the specified width. This is useful since comments are typically indented with the corresponding code. This option maintains the original left margin of the comment and adjusts the right margin to meet the target width.  
 If **Maximum right column** is used, comment lines will be wrapped when they reach the specified column, even if they have not reached the specified fixed width. This is useful if coding standards mandate that text should not exceed a specified column.
  - **Automatic width** - If selected, the width of the longest multi-line paragraph in the comment block is used as the width for block comments. This is useful for preserving the formatting of existing comments.  
 If **Maximum right column** is used, comment lines will be wrapped when they reach the specified column, even if they have not reached the specified fixed width. This is useful if coding standards mandate that text should not exceed a specified column.

## OPTIONS

- **Fixed right margin** - If selected, lines will break before the specified number of columns in the **Right column** field has been reached.
- **Preserve width on existing comments** - If selected, when editing an existing comment, SlickEdit® preserves the width of the existing comment. The width is determined by the length of the longest multi-line paragraph. If the width of the existing comment cannot be determined, the formatting option specified under **Comment width** will be used instead.
- **Continue bullet list on Enter** - If selected, when Enter is pressed inside a bulleted paragraph, a new bullet will be inserted and the cursor will be placed at the text starting position.
- **Javadoc** - If **Use hanging indent on block tag comments** is selected, the second line of a block tag comment will be automatically aligned to the first non-whitespace character after the first word after the tag.
- **Sync vertical line column** - This button will make visible and move the vertical line column to match the hard margin column (if using fixed right column margins) or the maximum right column (if using fixed width).

## Advanced Tab

The Advanced tab allows you to set color coding, extension-specific project parameters, and more. To access these settings, choose **Tools > Options > File Extension Setup**, then select the **Advanced tab** (pictured below).



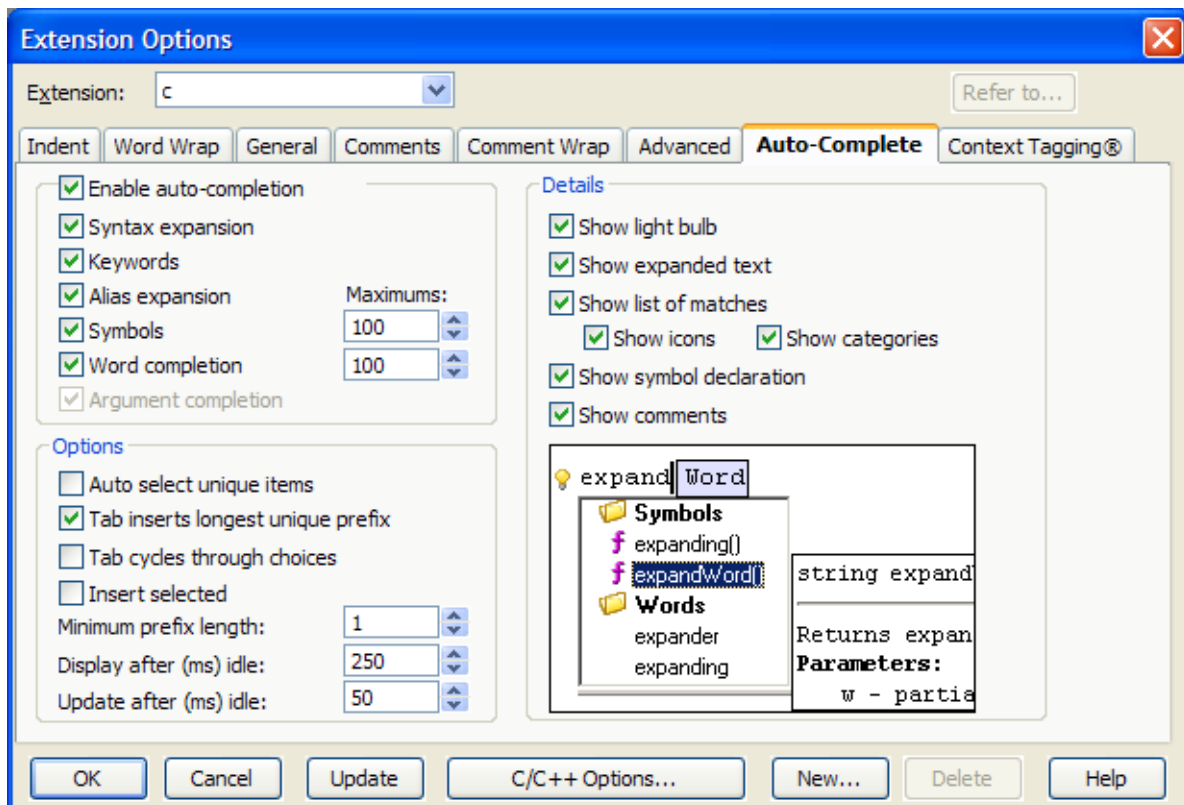
The following options and settings are available:

- **Color coding options** - The options below affect Color Coding. For more information about working with this feature, see [Color Coding](#).

- **Lexer name** - Use the drop-down list to select the lexer to use to recognize elements to be colored.
- **Color Coding** - Click this button to display the Color Coding Setup dialog, allowing modification of language-specific color coding for the current language.
- **Modified lines** - If selected, lines that have been modified are color-coded.
- **Current line** - Check on Current line to color code the current line.
- **Extension specific project** - Click this button to set project properties specific to a file extension. See [Defining Extension-Specific Projects](#) for more information.
- **Context Menus** - These options specify which context menu to display in the editor window base on whether a text selection is made in the editor window.
  - **Menu if no selection** - This specifies the menu that is displayed when right clicking in an edit window that does not have a selection.
  - **Menu if selection** - This specifies the menu that is displayed when right clicking in an edit window that has a selection.
  - **Select first (affects all extensions)** - When checked (default), a selection can be made with the right mouse button instead of displaying the extension-specific menu. When this is not checked, select menu items by clicking and dragging the mouse.
- **Begin/End pairs** - Specify the begin/end pairs to use for the selected extension in a format similar to a regular expression. This text box is disabled for languages that have special begin/end matching built-in. See [Begin/End Structure Matching](#) for more information about begin/end pairs and using this option.
- **Word chars** - The word characters affect the operation of all word-oriented commands, including word searching. You can use a dash (-) character to specify a range, such as "A-Z", which specifies uppercase letters. To specify the dash (-) character as a valid word character, place a dash at the beginning or end of the word character string.

## Auto-Complete Tab

Auto-Complete options in SlickEdit® can be configured for each file extension type. To access these options, choose **Tools > Options > File Extension Setup**, then select the **Auto-Complete** tab (pictured below).



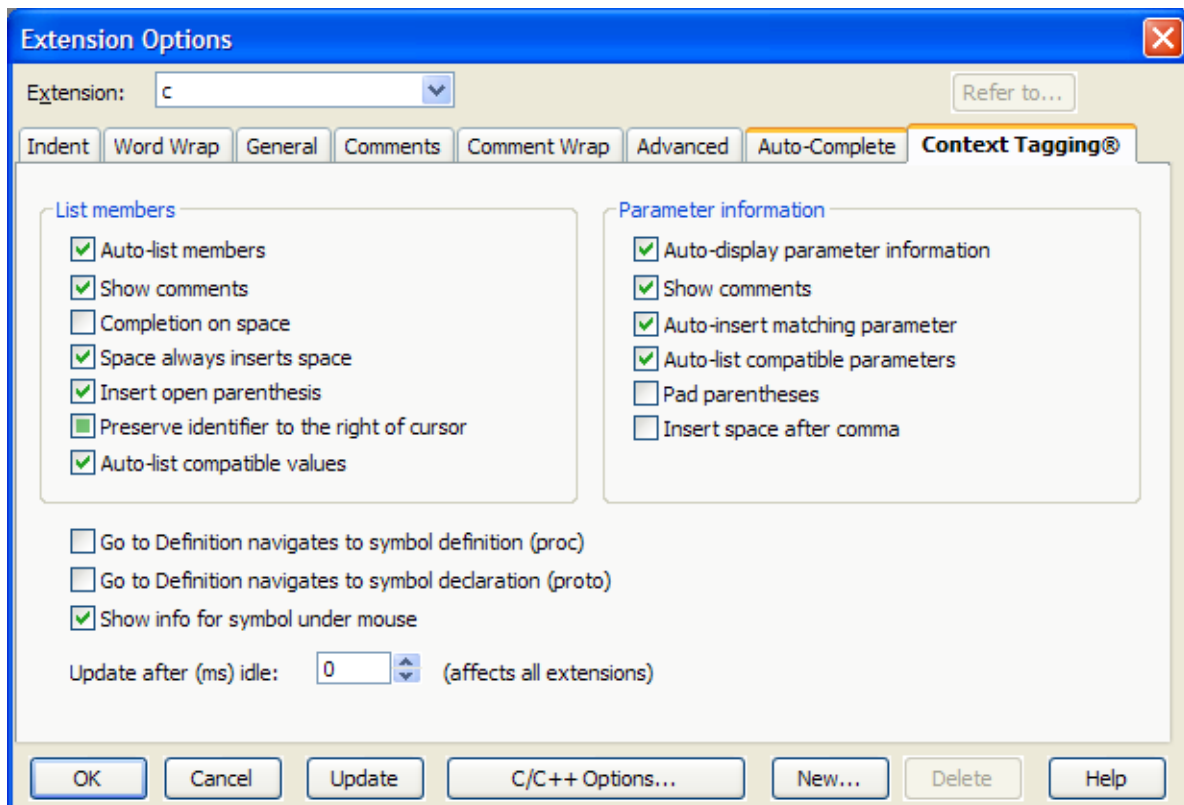
The following options are available:

- **Enable auto-completion** - If selected, activates the auto-completion feature.
- **Syntax expansion** - If selected, auto-completion will show syntax expansion choices for the word prefix under the cursor. Syntax expansion completes syntactic elements of the language, like **if** or **for** statements, putting in the parentheses and braces matching your specified coding style settings.
- **Keywords** - If selected, auto-completion will show keyword choices for the word prefix under the cursor, if it matches one or more keywords in the current language.
- **Alias expansion** - If selected, auto-completion will show the matching alias for the word under the cursor. Aliases names require an exact word match, not just a prefix match. For more information on using aliases, see [Aliases](#).
- **Symbols** - If selected, symbols will be displayed as completion options if the word prefix at the cursor matches one or more symbols using a strict, context-sensitive and language-specific tag search.
  - **Maximums (Symbols)** - For performance tuning, you can limit the maximum number of symbols displayed by auto-completion. This setting affects all file extensions.
- **Word completion** - If selected, word completions will be displayed if the word prefix under the cursor matches one or more words in the current file. The strength of this option is that it ties into SlickEdit's word and line completion features. After you select a word completion, you can press Ctrl+Shift+Space to complete the rest of the line from which the original word came.

- **Maximums (Word completion)** - For performance tuning, you can limit the maximum number of word completions displayed by auto-completion. This setting affects all file extensions. This is especially useful when editing large files.
- **Argument completion** - If selected, turns auto-completion on in the Build tool window for completing filenames and paths.
- **Auto-select unique items** - If this option is selected and auto-completion finds exactly one word match, it will automatically select that match for completion. If this option is turned off, you must select a word to complete using the Tab, up arrow, or down arrow keys.
- **Tab inserts longest unique prefix** - If selected, pressing Tab will cause auto-completion to attempt to insert the longest unique prefix match of all its completions. If the word prefix cannot be extended, Tab will cycle to the next completion choices.
- **Tab cycles through choices** - Select this option if you want to use Tab and Shift+Tab to cycle through completion choices, as is done in some command shells. If deselected, Tab will attempt to insert the longest unique prefix (if selected), or insert the selected completion, or cancel auto-completion and behave normally if there is no completion selected.
- **Insert selected** - If selected, when cycling through completion choices and a choice is selected, the selected choice will replace the current text. This modifies the file as you work.
- **Minimum prefix length** - The minimum number of characters the word prefix must contain before auto-completions will be displayed automatically.
- **Display after (ms) idle** - The number of milliseconds the editor must be idle before auto-completion suggestions will be displayed. This setting affects all extensions.
- **Update after (ms) idle** - The number of milliseconds the editor must be idle before auto-completion suggestions will be refreshed. This setting affects all extensions.
- **Show light bulb** - If selected, displays the light bulb as a reminder when auto-completion suggestions are available for the current word prefix.
- **Show expanded text** - If selected, shows the rest of the word or statement being completed.
- **Show list of matches** - If selected, shows the list of matches underneath the word prefix.
  - **Show icons** - If selected, displays symbol icons and folder icons. Turn this feature off to get a more compact list containing only completions.
  - **Show categories** - If selected, shows completions in a categorized list for each type. If deselected, all completions will be shown in one flat, sorted list.
- **Show symbol declaration** - If selected, for symbol completions, this will show the symbol declaration as a comment to the right of the symbol completion.
- **Show comments** - If selected, for symbol completions, this will display the symbol's comments to the right of the symbol completion.

## Context Tagging® Tab

Context Tagging options can be configured for each file extension type. This allows you to enable and disable particular features on a per-language basis. To change these options, from the main menu choose **Tools > Options > File Extension Setup**, then select the **Context Tagging® tab**.



### List Members Options

The following options apply to list members:

- **Auto-list members** - If selected, typing a member access operator (for example, "." or "->" in C++) will trigger SlickEdit® to display a list of the members for the corresponding type. To access this feature on demand, press Alt+Dot. If you use this feature on demand, and you are not in a member expression, this feature will display a list of locals, global variables, current class members, and so on.
- **Show comments** - If selected, the comments are displayed for the currently selected symbol in the list displayed by list members. When a symbol has multiple definitions or overloads, and multiple sets of comments, the comments will indicate that you are looking at item "< 1 of n >". Click on the arrows or use Ctrl+PageUp and Ctrl+PageDown, to cycle through the comment sets.
- **Completion on space** - If selected, pressing the spacebar when list members is displayed will insert the longest unique matching prefix from the symbols in the list. For example, if the list contains **FLAG\_CHAR** and **FLAG\_LONG**, then typing **FL<Alt+Dot><spacebar>** completes the line of code up to **FLAG\_**. If this option is not selected, use Ctrl+Space when list members is displayed to perform completion.
- **Space always inserts space** - If selected, pressing the spacebar when list members is displayed will insert the current item and a space in the list after the current item. If unselected, pressing the spacebar will only insert the current item with no extra space.
- **Insert open parenthesis** - If selected, selecting an item in the list inserts the current item in the list and any extra characters that are required by the symbol. For example, an open parenthesis is inserted after a function name for languages that require an open parenthesis after a function name. For C++, the less-than symbol (<) is inserted after a template class name.



- **Preserve identifier to right of cursor** - If selected, only the identifier characters before the cursor are replaced with an item selected from a list members dialog. Identifier characters after the cursor are preserved. When this option is not selected, identifier characters following the cursor are replaced with the item selected from a list members dialog. When this option is in the mixed state, trailing identifier characters are preserved for auto list members but not when listing members on demand by pressing Alt+Dot.

For example, if list members is active and the current line is:

```
this->foo<cursor_here>Bar
```

If this option is selected and you choose a symbol named “foodForThought” from the list members list, the line will be changed to:

```
this->foodForThought<cursor here>Bar
```

If this option is not selected, doing the same would result in:

```
this->foodForThoughtBar<cursor here>
```

- **Auto-list compatible values** - If selected, compatible variables are automatically listed after typing a space after assignment operators and return statements. Global (non-module) variables are not listed. This only affects C, C++, and Java. To access this feature on demand, press Alt+Comma.

## Parameter Information

The following options apply to parameters:

- **Auto-display parameter information** - If selected, the prototype and comments for a function are automatically displayed when a function operator such as the open parenthesis is typed, and highlights the current argument within the displayed prototype. To access this feature on demand, press Alt+Comma.
- **Show comments** - If selected, comments are displayed when Parameter Info is displayed. When a symbol has multiple definitions, and multiple sets of comments, the comments will indicate that you are looking at item “< 1 of n >”. Click on the arrows or use Ctrl+PageUp and Ctrl+PageDown, to cycle through the comment sets.
- **Auto-insert matching parameter** - If selected, when auto parameter info is displayed and the name of the current formal parameter matches the name of a symbol in the current scope of the appropriate type or class, the name is automatically inserted. When the name is inserted, it is also selected so that you can type over it, or you can type a comma, space, tab, or close parenthesis to use the automatically inserted parameter.
- **Auto-list compatible parameters** - If selected, compatible variables are automatically listed when parameter info is active and typing the arguments to a function call. Global (non-module) variables are not listed. This only affects C, C++, and Java. To access this feature on demand, press Alt+Comma.
- **Pad parentheses** - If selected, a space is inserted after the open parenthesis when a parameter name is automatically inserted. In addition, if you type a close parenthesis after an automatically inserted parameter, it will insert a space before the close parenthesis.
- **Insert space after comma** - If selected, a space is inserted after the comma when a parameter name is automatically inserted, such as **myfun(a, b, c)**.

## Miscellaneous Options

The following options appear at the bottom of the Context Tagging® tab:

- **Go to Definition navigates to symbol definition (proc) or declaration (proto)** - These options are mutually exclusive. If neither option is selected, the Select a Tag dialog is displayed, prompting you for both definitions and declarations. In any case, if you use Ctrl+Dot to jump to a symbol, you can cycle through the alternate symbols by pressing Ctrl+Dot repeatedly. You can step backwards through the list of matches by pressing Ctrl+Comma. However, once you reach the first match, Ctrl+Comma will then pop you back to your original location before you pressed Ctrl+Dot.

Independent of the settings for these options, in the following circumstances, SlickEdit® will jump directly to the definition or declaration.

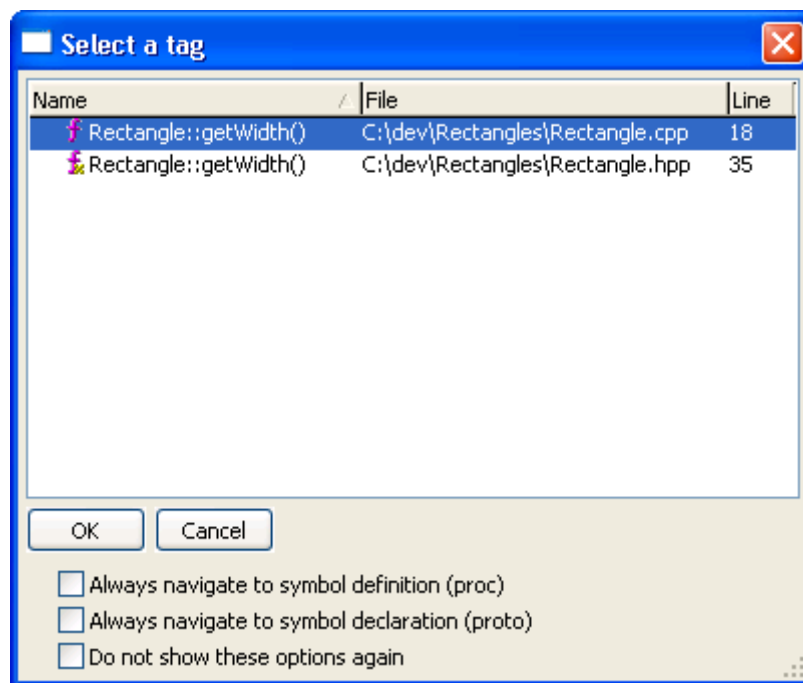
- If the cursor is on the first line of a symbol's declaration, it will jump directly to the definition, provided it is unique.
- If the cursor is on the first line of a symbol's definition, it will jump directly to the declaration, provided it is unique.

This behavior is particularly convenient for C++ programmers to navigate from a function to its prototype and vice-versa.

- **Show info for symbol under mouse** - When selected, as the mouse cursor floats over a symbol, the information and comments for that symbol are displayed.
- **Update after (ms) idle** - The value that you type in this field contains the idle time in milliseconds before the Auto List Members feature displays the list. To prevent the Auto List Members list from being displayed when you are typing fast, set the idle time to 300 milliseconds.

## Select a Tag Dialog

The Select a Tag dialog is used to prompt whether you want to navigate to the symbol's definition or declaration when you use Go to Definition (Ctrl+Dot, **Search > Go to Definition**, or **push\_tag** command).



Select the definition you want to go to. You can also check the options on this dialog to set the behavior going forward. When you set the **Always navigate to symbol** options, the settings are also updated for

the **Go to Definition** options on the [Context Tagging® Tab](#) of the Extension Options dialog (**Tools > Options > File Extension Setup**).

See [Symbol Navigation](#) for more information.

## File Options Dialog

You can set various options that pertain to loading, saving, and other file operations. To access these options, choose **Tools > Options > File Options**. The File Options dialog is displayed, containing the following four tabs—click the links to go to the appropriate descriptions in the documentation:

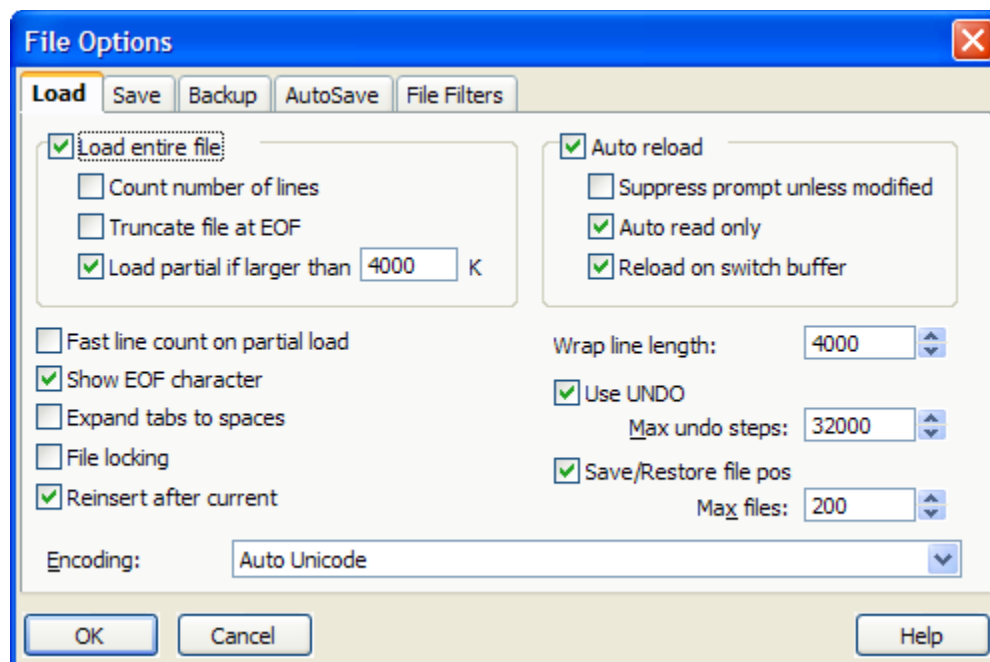
- [Load Tab](#)
- [Save Tab](#)
- [Backup Tab](#)
- [AutoSave Tab](#)
- [File Filters Tab](#)

For more information about working with files, see the chapter [Workspaces, Projects, and Files](#).

### Load Tab

The Load tab, pictured below, offers options to control how files are loaded. For more information about loading and opening files, see [Opening Files](#).

To access the Load tab, choose **Tools > Options > File Options**, then select the **Load tab**.



The following settings are available on the Load tab:

- **Load entire file** - When selected, the entire contents of the opened files are read into memory. The Line indicator (located at the bottom right section of the editor) might become blank if the file does not fit in the editor's cache (defaults to 2 MB). When this option is not selected, Auto Reload does not work until the file is saved. If you are using the **load** command to open files, use the switch **+L** to specify this option.

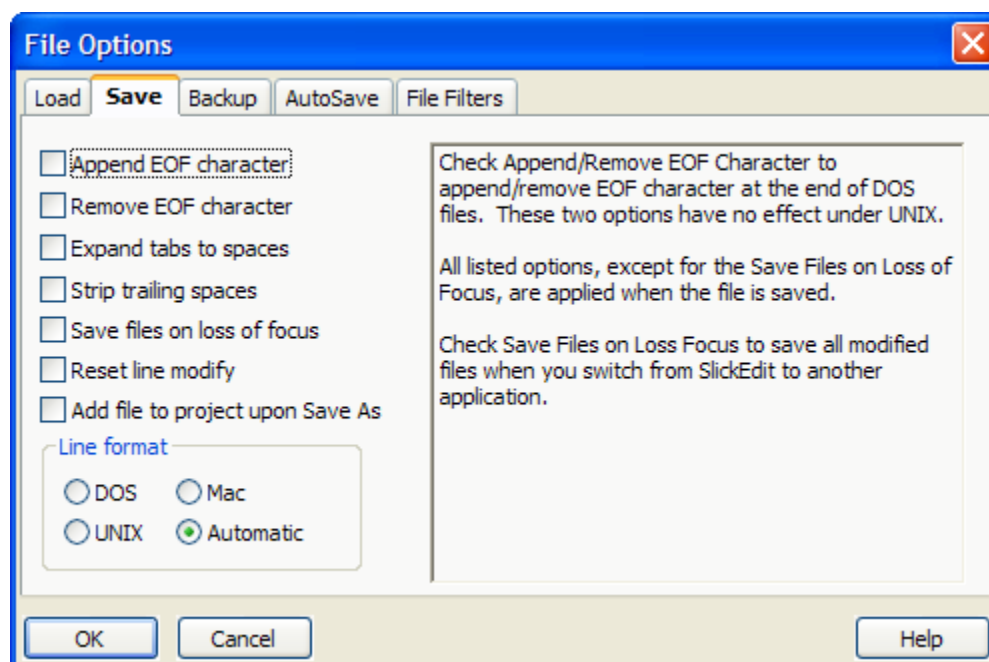
- **Count number of lines** - When selected, the entire contents of the opened files are read into memory and the number of lines in the file are counted. The line number is always displayed in the Line indicator area of the editor. The **Load entire file** check box will have the same affect as this check box when the entire file fits within the cache of the editor (defaults to 2 MB) and does not have to be spilled. If you are using the **load** command to open files, use the switch **+LC** to specify this option.
- **Truncate file at EOF** - When selected, the entire contents of the opened files are read into memory and the number of lines are counted. In addition, DOS format files are truncated when an EOF character is found. The line number is always displayed in the Line indicator area of the editor. This option is useful for REXX `.cmd` files which can have p-code appended to them after the EOF character. If you are using the **load** command to open files, use the switch **+LZ** to specify this option.
- **Load partial if larger than** - When selected, if the file being loaded is greater than the size specified in the **Load partial if larger than** text box, the entire file is not read into memory. Since the file handle remains open to your file, Auto Reload does not work until the file is saved. The Line indicator might be blank unless the **Fast line count on partial load** option is selected.
- **Auto reload** - When selected, the editor detects when files being edited have been modified by other applications and prompt or automatically replace the file with the new copy on disk.
- **Suppress prompt unless modified** - When selected, files being edited that have been modified by other applications will automatically be replaced with the new copy on disk.
- **Auto read only** - When selected, the editor detects when other applications change the read-only attribute of the file and turns the read-only mode on and off.
- **Reload on switch buffer** - When selected, the editor will detect when the file has been modified by other applications when the file is switched to the active editor window. If the option is deselected, the default check is still performed when you switch from another application.
- **Fast line count on partial load** - When selected, this option indicates that the editor is to count the number of lines when files are opened. The line number is always displayed in the Line indicator area of the editor. This option is much faster than the **Count number of lines** option when editing files larger than the cache size (2 MB by default), because very little data is written to the spill file. The Auto Reload feature does not work until the file is saved. If you are using the **load** command to open files, use the switch **+LF** to specify this option.
- **Show EOF character** - When selected, the EOF character is not removed when files are loaded. If you are using the **load** command to open files, use the switch **+LE** to specify this option.
- **Expand tabs to spaces** - When selected, the entire contents of the files are read into memory and tabs are expanded into spaces. If your tab settings for the file being loaded are of the form **+<increment>** (e.g. "+4"), then tabs are expanded in increments of the specified increment. Otherwise, tabs are expanded in increments of eight. To set tabs in a form **+<increment>**, select **Tools > Options > File Extension Setup**. Select your extension from the **Extension** drop-down list, then select the [Indent Tab](#). Enter your values in the **Tabs** text box. For languages such as REXX and Linux containing shell scripts that require the contents of the file be analyzed before the file type is known, the Fundamental mode tab settings are used. If you are using the **load** command to open files, use the switch **+E** to specify this option.
- **File locking** - When selected, this option ensures that a file handle is kept open to the file for locking purposes. This detects when another user is editing the same file. If you are using the **load** command to open files, use the switch **+N** to specify this option.
- **Reinsert after current** - When selected, the editor will switch back to the previous buffer or window with the **prev\_buffer** or **prev\_window** command. If you are using the **load** command to open files, use the switch **+BP** to specify this option.

- **Unmodified block spilling** - When selected, unmodified blocks are spilled when the memory/buffer cache is full. When the spill file destination is faster than the disk the file resides on (such as a floppy drive), select this option for optimal performance. If you are using the **load** command to open files, use the switch **+S** to specify this option.
- **Wrap line length** - When selected, this option improves editing performance on large files with very long lines, by wrapping the long lines at the number of characters specified here. This option is particularly useful for editing very large, single-line XML files.
- **Use undo, Max undo steps** - When **Use undo** is selected, modifications to buffers may be undone. The **Max undo steps** text box specifies the maximum number of steps that are stored. Cursor motion can be undone but is not counted as a step. If you are using the **load** command to open files, use the switch **+U** to specify this option. For example, **+U:32000** turns on undo and specifies a 32000-step max).
- **Save/Restore file pos, max files** - When **Save/Restore file pos** is selected, when you open a file, the cursor position is restored to its previous location when the file was closed. The **Max files** text box specifies the maximum number of cursor positions saved. The most recently closed file positions are stored.
- **Encoding** - Unicode support required. Specifies the global (non-extension specific) file encoding. This setting is overridden if an extension-specific encoding is defined. Both the extension-specific and global setting are overridden if you specify an encoding in the Open dialog. SlickEdit® records the encoding used to override default encoding settings and reuses this setting the next time you open the same file. This provides you with per-file encoding support. Encoding is also supported for Microsoft project files (`vcproj`, `cspj`, `vbproj`) that are XML files but that default to active code page encoding and not UTF-8, like XML. See [Encoding](#) for more information.

## Save Tab

The Save tab, shown below, contains options that affect how files are saved. For more information about saving files, see [Saving Files](#).

To access save options, choose **Tools > Options > File Options**, then select the **Save tab**.



## OPTIONS

The following options are available:

- **Append EOF character** - When this option is set, an EOF character is appended to the end of DOS files when the buffer is saved. This option has no effect on UNIX, Macintosh, or binary files. If you are using the **save** command to save files, use the switch **+Z** to specify this option.
- **Remove EOF character** - When this option is set, the EOF character is removed from the end of DOS files when the buffer is saved. This option has no effect on UNIX, Macintosh, or binary files. If you are using the **save** command to save files, use the switch **+ZR** to specify this option.
- **Expand tabs to spaces** - When this option is set, tabs are expanded to spaces according to the tab stops when the buffer is saved. If you are using the **save** command to save files, use the switch **+E** to specify this option.
- **Strip trailing spaces** - When this option is set, trailing spaces at the end of each line are stripped when the buffer is saved. If you are using the **save** command to save files, use the switch **+S** to specify this option.
- **Save files on loss of focus** - When this option is set, all modified files will be saved when you switch to another application.
- **Reset line modify** - When this option is set, line modify flags are reset when the buffer is saved. If you are using the **save** command to save files, use the switch **+L** to specify this option.
- **Add file to project upon Save As** - This option controls the default value of the **Add to project** option on the Save As dialog. Check this option on the Save tab to have the **Add to project** check box selected by default each time the dialog is invoked. By default, neither option is set. If you are using the **save** command to save files, use the switch **+P** to specify this option.
- **Line format** - By default, the line format is set to **Automatic**, which means files are saved “as is” and there are no changes made to the line end characters. To have line end characters translated when files are saved, set the file format to **DOS**, **Mac**, or **UNIX**.

The Save As dialog also enables the translation of the line end characters for the current file. See [Save As Dialog](#).

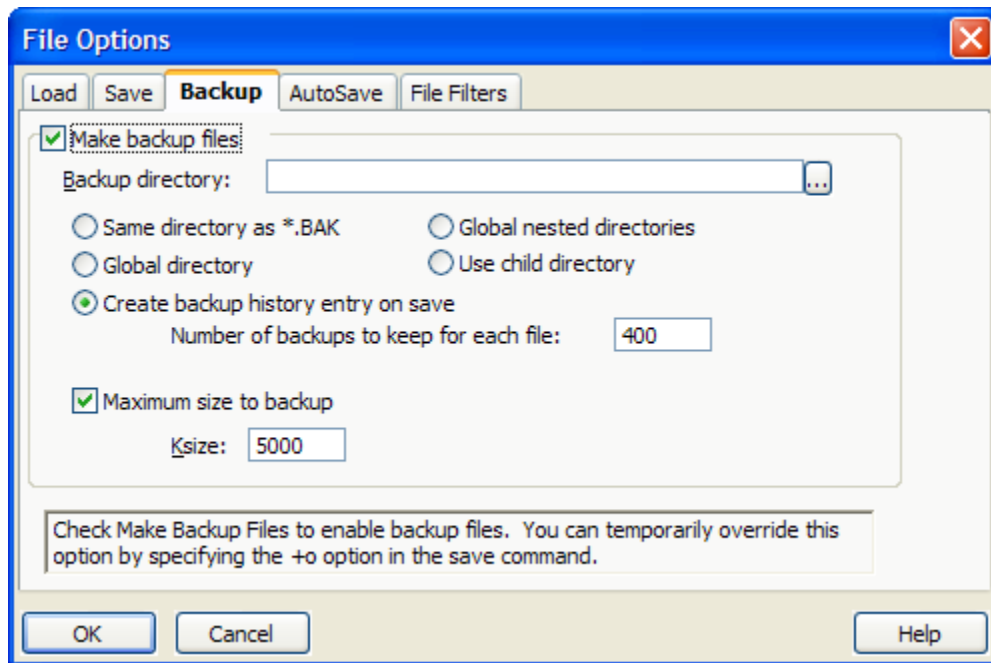
When **Automatic** is set (default), the line breaks are saved automatically in the file format appropriate to the context in which you are working. However, you can designate a file type for the line breaks. For example, if you are working in Windows and using CVS, using UNIX line breaks will make using CVS easier. Therefore, set the file format to **UNIX**.

**NOTE** Classic Mac line endings are a single carriage return (ASCII 13).

## Backup Tab

The Backup tab, shown below, contains options that pertain to file backups. For more information about backing up files, see [File History and Backups](#).

To access backup options, choose **Tools > Options > File Options**, then select the **Backup tab**.



The following settings are available on the Backup tab:

- **Make backup files** - This option turns backup on and off. When selected, a backup is created when you save the file. If you are using the **save** command to save files, use the switch **+O** to specify this option.
- **Backup directory** - Specifies the backup directory used by **Global directory** or **Global nested directories** backup styles. The backup directory may contain a drive specifier. This sets the VSLICKBACKUP environment variable.
- **Same directory as \*.BAK** - When selected, backup files are placed in the same directory as the destination file but the extension is changed to .bak. This option is useful on networks. If you are using the **save** command to save files, use the switch **+DB** to specify this option.
- **Global directory** - When selected, backup files are placed in a single directory. The default backup directory is \vslick\backup\ (UNIX: \$HOME/.vslick/backup). The backup file gets the same name as the destination file. For example, given the destination file c:\project\test.c (UNIX: /project/test.c), the backup file will be c:\vslick\backup\test.c (UNIX: \$HOME/.vslick/backup/test.c). If you are using the **save** command to save files, use the switch **+D** to specify this option.

For a network, you might need to create the backup directory with appropriate access rights manually before saving a file. See [Backing Up Network Files](#).

- **Global nested directories** - When selected, backup file directories are derived from concatenating a backup directory with the path and name of the destination file. The default backup directory is \vslick\backup\ (UNIX: \$HOME/.vslick/backup). For example, given the destination file c:\project\test.c, the backup file will be c:\vslick\backup\project\test.c (UNIX: \$HOME/.vslick/backup/project/test.c). If you are using the **save** command to save files, use the switch **-D** to specify this option.

For a network, you might need to create the backup directory with appropriate access rights manually before saving a file. See [Backing Up Network Files](#).

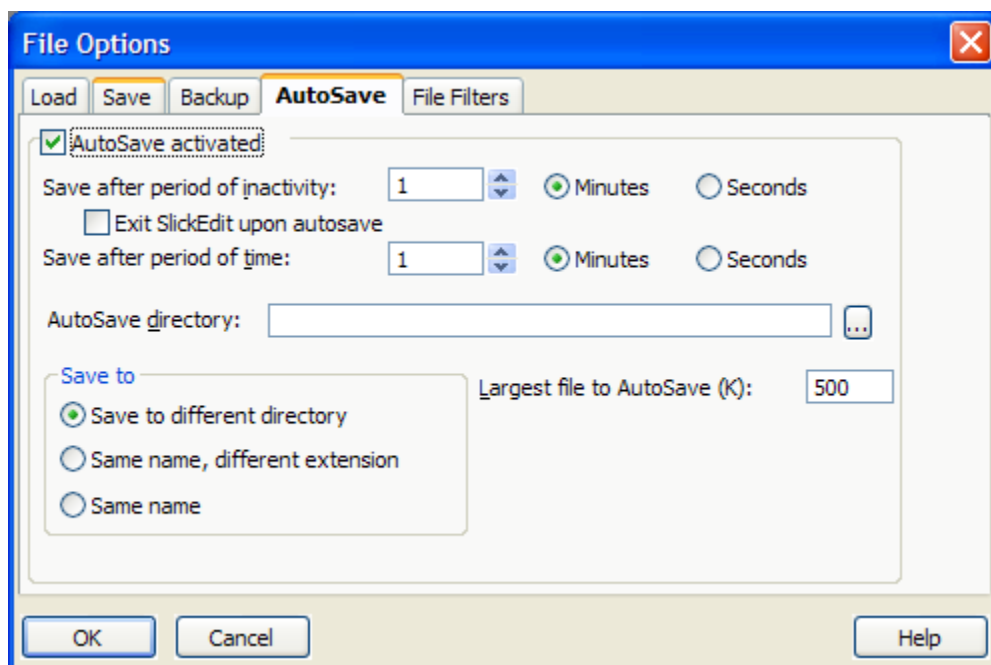


## OPTIONS

- **Use child directory** - When selected, backup files are placed in a directory off the same directory as the destination file. For example, given the destination file `c:\project\test.c` (UNIX: `/project/test.c`), the backup file will be `c:\project\backup\test.c` (UNIX: `/project/backup/test.c`). This option is most convenient on networks because of file permission issues. If you are using the **save** command to save files, use the switch **+DK** to specify this option.
- **Create backup history on save** - Select this option to create a history of the file when it is saved. Specify the number of backups to keep for each file.
- **Maximum size to backup** - Select this option if you wish to specify a file size for backups.

## AutoSave Tab

The AutoSave tab, shown below, contains settings for automatically saving files. For more information about saving files, see [Saving Files](#).



The following options are available:

- **AutoSave activated** - Enables AutoSave, which prevents you from losing data when an abnormal editor exit occurs (possibly from a power loss).

The AutoSave temporary files are placed in a directory named `autosave` in the configuration directory. Usually, the AutoSave temporary files are deleted when you exit SlickEdit®. After a file is saved or closed, the AutoSave temporary file is deleted the next time AutoSave occurs. AutoSave temporary files are only needed for files that are modified.

The current implementation of AutoSave does not save files that have are not named. In addition, AutoRestore does not restore files that do not exist on the disk drive of your system. Save your file at least one time to ensure that the file has a file name and exists on the disk drive.

- **Save after period of inactivity** - The value that you enter in this field specifies the amount of idle time, in minutes or seconds, when modified files should be saved. Set this value to 0 if you do not want this option ignored.



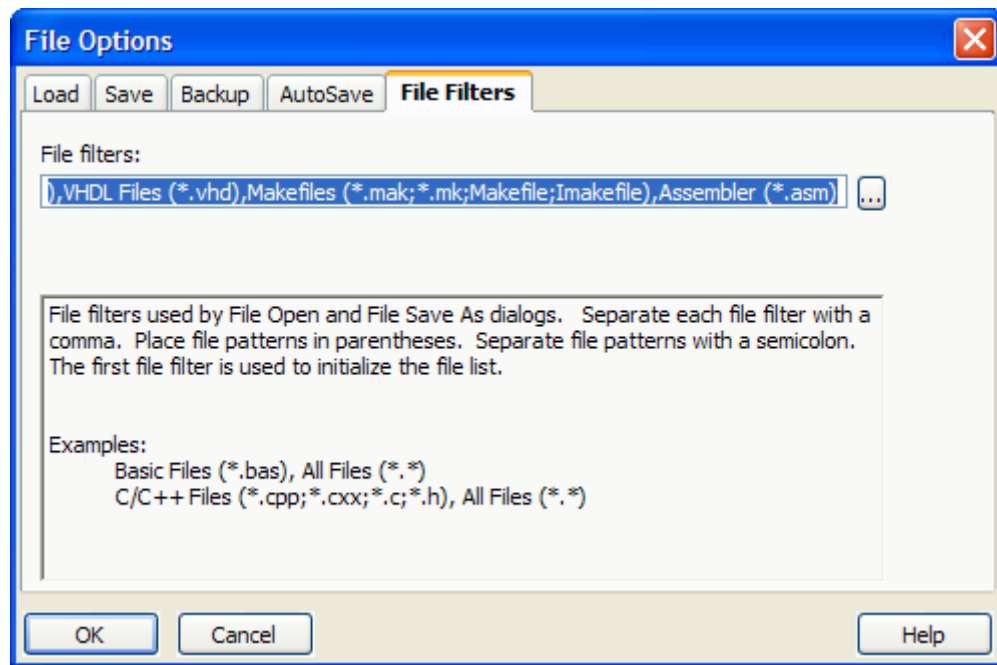
- **Exit SlickEdit on AutoSave** - When this option is selected, the SlickEdit application closes after an AutoSave.
- **Save after period of time** - Specifies amount of time in minutes or seconds when modified files should be saved. Set this value to 0 if you want this option ignored.
- **AutoSave directory** - The directory that you specify in this field is the AutoSave directory if the **Save to different directory** option has been selected. If the **AutoSave directory** is blank, **<configuration\_directory>\autosave** is used. The configuration directory is defined by the VSLICKCONFIG environment variable. If the VSLICKCONFIG environment variable is not set, the directory in the editor executable directory is used as the configuration directory.
- **Save to different directory** - Setting this option specifies that all AutoSave temporary files be placed in the directory specified by the AutoSave directory text box. Use this option to clean up or find all of the AutoSave files if an abnormal editor exit occurs.

**NOTE** When editing two files with the same name but in different directories, one AutoSave temporary file is overwritten by the other.

- **Same name, different extension** - This option, when set, specifies that the AutoSave file be placed in the same directory as the file that is being auto saved but it is given a different extension. The third character of the extension is replaced with a ~ character. The length of the extension is padded with underscores if the length of the extension is less than three characters. For example, the AutoSave file for `test.c` is `test.c_~`. The AutoSave file for `test.prg` is `test.pr~`. If you are editing two files in the same directory which differ only by the third character, one AutoSave temporary file will be overwritten by the other.
- **Same name** - Setting this option specifies that the modified files be automatically saved. No AutoSave temporary files are created.
- **Largest file to AutoSave (K)** - Files greater than the value that you type in this field are not automatically saved. Set this value to 0 if you want all files auto saved.

## File Filters Tab

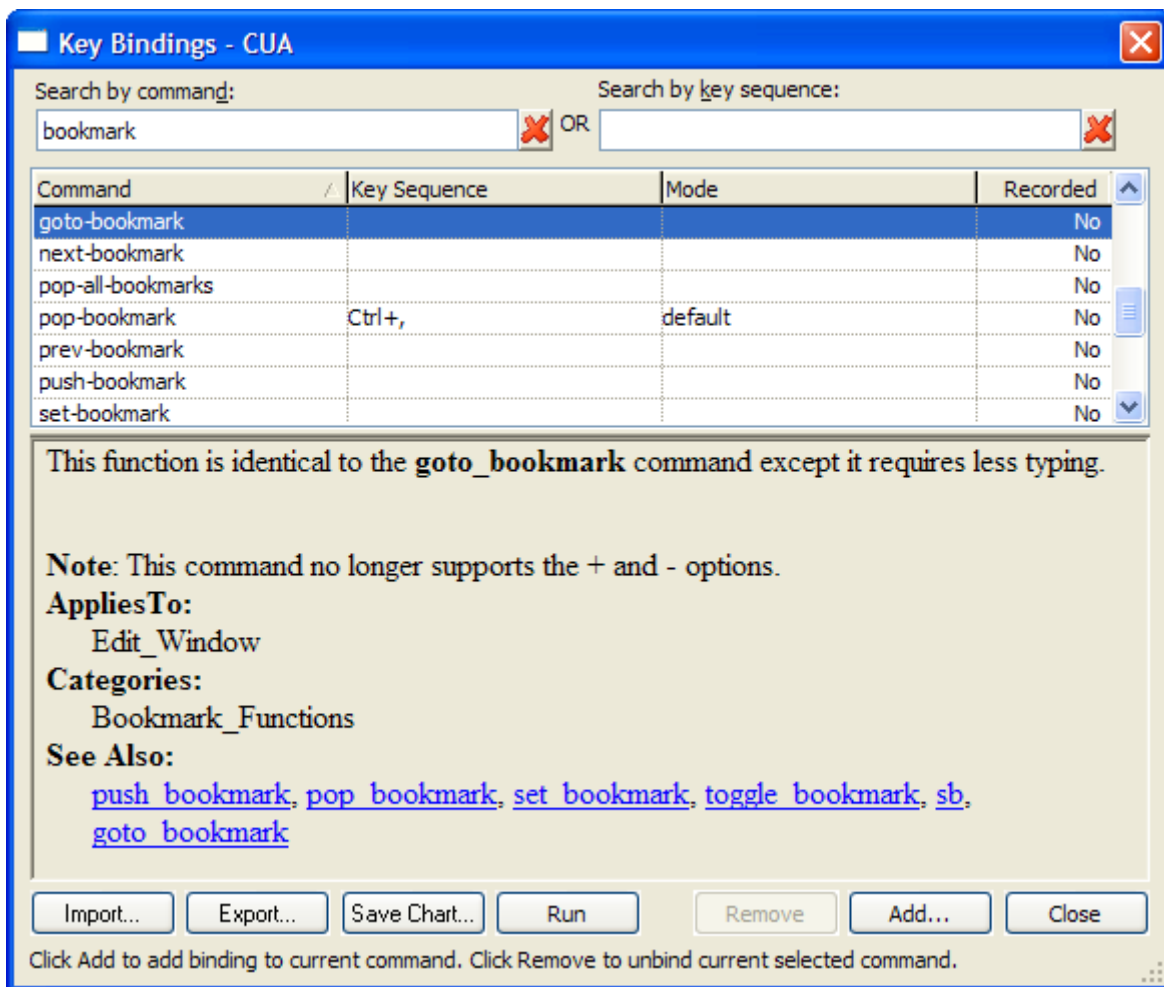
The File Filters tab, shown below, contains filtering options for opening and saving files. To access filters, choose **Tools > Options > File Options**, then select the **File Filters tab**.



In the **File filters** text box, enter the filters you wish to assign. Separate each filter with a comma, and place file patterns in parentheses. Separate file patterns with a semicolon. The first file filter is used to initialize the file list.

## Key Bindings Dialog

The Key Bindings dialog (**Tools > Options > Key Bindings**, or `gui_keybindings` command) is used for creating and viewing mouse and key bindings for commands and user-created macros, as well as for importing and exporting your custom bindings. See [Managing Bindings](#) for detailed information.



## NOTES

- The first time the Key Bindings dialog is invoked, the Building Tag File progress bar may be displayed while Slick-C® macro code is tagged.
- Bindings are based on the editor emulation mode (CUA is the default). The title bar of the Key Bindings dialog shows the current mode. To change the emulation mode, select **Tools > Options > Emulation**. For more information, see [Emulations](#).

The dialog contains the following elements:

- Search by command** - This filter is used for searching commands in the **Command** column. Type a string in the filter box, and the list of commands is filtered as you type to show only those commands that contain the specified string. The red **X** icon is used to clear the text box or you can edit inside the text box manually.
- Search by key sequence** - This filter is used for searching bindings in the **Key Sequence** column. It captures literal keyboard input. For example, when the focus is in this filter, press **Ctrl** and **C** at the same time, and "Ctrl+C" is displayed. Press the **Backspace** key and "Backspace" is displayed. Mouse events inside the filter are literal as well. For example, right-clicking within the filter displays the text "RButtonDn". Because the key sequence filter captures literal keyboard

input, you cannot edit the text or use key functions such as backspacing or tabbing in and out of the field. You must use the red **X** icon to clear the filter.

- **Command** - This column lists, in alphabetical order by default, the SlickEdit® commands and user macros that are or can be bound to keys or mouse events. Click on the column label to sort bindings by this column. An arrow in the column header indicates the sort order (ascending or descending).

If a command/macro has more than one binding, each instance is listed on a separate row. For example, in CUA emulation, the command **gui\_open** is bound to **F7**, **Command+O** (on the Mac), and **Ctrl+O**. Therefore, **gui\_open** appears in the **Command** column three times, once for each binding.

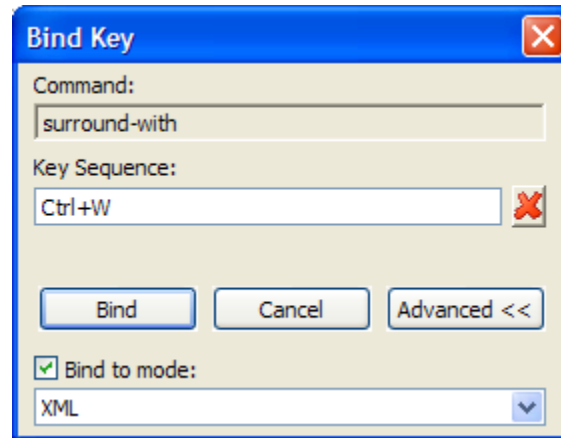
- **Key Sequence** - This column shows the mouse event or key sequence associated with the command or macro. If a **Key Sequence** cell is empty, no binding is associated with that command/macro. Click on the column label to sort bindings by this column. An arrow in the column header indicates the sort order (ascending or descending).
- **Mode** - This column shows the language editing mode to which the key binding applies. The **default** mode causes the binding to work in all language editing modes. However, the default mode will be overridden by any language-specific mode binding to another command/macro. Click on the column label to sort bindings by this column. An arrow in the column header indicates the sort order (ascending or descending).

**NOTE** To change the mode for a command/macro that is already bound, first you should unbind the command/macro, then recreate the binding with the mode you want to use. See [Editing Bindings](#) for more information. For information about editing modes, see [Language Editing Modes](#).

- **Recorded** - This column indicates if the item in the Command column is a SlickEdit command (**No**) or a user-recorded macro (**Yes**).
- **Documentation pane** - The bottom pane of the dialog displays the code documentation for the selected command or macro, if it exists. Click “See Also” hyperlinks (if any exist) to display Help for that item. For See Also links, if a Help entry does not exist, a message box notification is displayed. The documentation pane can be resized by dragging the size bar in the middle of the dialog. The size is remembered the next time the dialog is displayed.
- **Import and Export** - These buttons allow you to import and export bindings. This is useful for creating backups, sharing with other team members, or taking with you should you switch computers. See [Exporting and Importing Bindings](#) for details of these features.
- **Save Chart** - This button allows you to save a reference chart of all current bindings for all language editing modes in the selected emulation. The chart is saved in HTML format with a name and location that you specify. Commands/macros that are not bound are not included.
- **Run** - This button runs the selected command or user-recorded macro.
- **Remove** - This button clears the binding for the selected command/macro. You can also press **Delete** to clear the binding.
- **Add** - This button is used to initiate a new binding, displaying the Bind Key dialog. See [Creating Bindings](#) and [Bind Key Dialog](#) for more information.
- **Close** - Closes the Key Bindings dialog. You can also press **Esc** to close the dialog.
- **Message line** - A message line under the buttons displays contextual instructions for using the dialog.

## Bind Key Dialog

The Bind Key dialog is used to bind a command to a key sequence or mouse event. It is displayed when you click **Add** on the Key Bindings dialog to add a new binding.



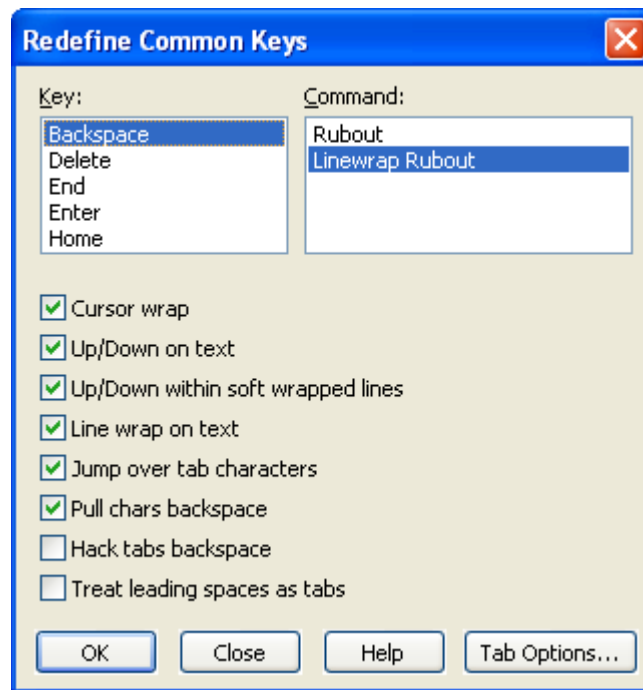
The Bind Key dialog contains the following:

- **Command** - This field shows the command that you have selected to bind.
- **Key Sequence** - This field is used to enter the key sequence or mouse event that you want bound to the command. For example, to enter the key sequence Ctrl+W, literally press the Ctrl and W keys together. It accepts literal keyboard/mouse input, so you cannot edit the text or use key functions such as backspacing or tabbing in and out of the field. You must use the red **X** icon to clear the filter.
- **Bind** - After entering the key sequence or mouse event, click this button to save the binding and close the dialog. Prior to clicking **Bind**, you may want to assign the binding to a specific language editing mode (see below).
- **Cancel** - Click this button to cancel the binding operation and close the dialog.
- **Advanced** - Click this button to expand the language editing mode settings:
  - **Bind to mode** - By default, all new bindings are assigned to the “default” language editing mode, which means that the binding will work in all modes. To assign the binding to a specific language editing mode, select this option and click the language editing mode from the drop-down list. Click **Bind** when finished.

See [Creating Bindings](#) for more information.

## Redefine Common Keys Dialog

The Redefine Common Keys dialog (**Tools > Options > Redefine Common Keys**), shown below, allows you to change the behavior of certain common keys. For more information about redefining keys, see [Redefining Common Keys](#).



## Redefinable Keys

The redefinable keys and their available commands are described below.

- **Backspace** - The following commands are available for binding to the Backspace key:
  - **Rubout** - Deletes the character to the left of the cursor. If Word Wrap is on (**Tools > Options > File Extension Setup, [Word Wrap Tab](#)**), the cursor will wrap to the previous line when the left margin is reached. Otherwise the cursor is not wrapped to the previous line.
  - **Linewrap Rubout** - Deletes the character to the left of the cursor. The cursor always wraps to the previous line when the left margin is reached. If you want line wrapping to occur when column one is reached, select the option **Line wrap on text** on this dialog.
- **Delete** - The following commands are available for binding to the Delete key:
  - **Delete Char** - Deletes the character at the cursor. If Word Wrap is on (**Tools > Options > File Extension Setup, [Word Wrap Tab](#)**) and no more characters exist on the current line, the next line is joined to the current line.
  - **Linewrap Delete Char** - Deletes the character at the cursor. If no more characters exist on the current line, the next line is joined to the current line.
- **End** - The following commands are available for binding to the End key:
  - **End line** - Moves the cursor to the end of the line.
  - **End Line Text Toggle** - Toggles the cursor between the end of the current line and the last non-whitespace character. This is useful for trimming extra spaces from long lines, because it gives you a natural and quick way to get to your vertical line column and the last non-blank column.
- **Enter** - The following commands are available for binding to the Enter key:

- **Nosplit Insert Line** - Inserts a blank line after the current line and aligns the cursor with the first non-blank character of the original line. The current line is not split.
- **Split Insert Line** - Splits the current line at the cursor. Enough blanks are inserted at the beginning of the new line to align it with the first non-blank character of the original line.
- **Maybe Split Insert Line** - If the option **Start in insert mode** is on (**Tools > Options > General**, [More Tab](#)), the current line is split at the cursor. Enough blanks are appended to the beginning of the new line to align it with the first non-blank character of the original line. If **Start in insert mode** is off, the cursor is moved to column one of the next line.

**NOTE** When changing the key binding for the Enter key, the binding for Ctrl+Enter will automatically switch to the opposite setting, depending on whether it is bound to “Split Insert Line” or “Nosplit Insert Line”.

- **Home** - The following commands are available for binding to the Home key:
  - **Begin Line** - Moves the cursor to column one.
  - **Begin Line Text Toggle** - Toggles the cursor between the first non-blank character of the current line and column 1.

## More Options

The settings listed below appear on the Redefine Common Keys dialog box.

- **Cursor wrap** - Determines whether the **cursor\_left** and **cursor\_right** commands wrap to the previous or next line respectively.
- **Up/Down on text** - Determines whether the **cursor\_up** and **cursor\_down** commands place the cursor in virtual space. By default, **cursor\_up** and **cursor\_down** go to the same column of the next or previous line, regardless of the length of the line.
- **Up/Down within soft wrapped lines** - If selected, when Soft Wrap is on (**Tools > Options > File Extension Setup**, [Word Wrap Tab](#)), the **cursor\_down** and **cursor\_up** commands move the cursor up to the next or previous visible line, including line continuations. To force **cursor\_down** and **cursor\_up** to move the cursor to the next or previous physical line (the same position to which the cursor would move if Soft Wrap was off), deselect this option.
- **Line wrap on text** - If selected, line wrapping will occur when column one is reached. If deselected, line wrapping occurs when the left margin is reached. When Word Wrap is on (**Tools > Options > File Extension Setup**, [Word Wrap Tab](#)), wrapping occurs when the left margin is reached regardless of the Left or Backspace key configurations.
- **Jump over tab characters** - If selected, moving the cursor over a tab character with the Left or Right arrow key causes the cursor to jump across the virtual space. To allow the Left and Right arrow keys to cursor into virtual space of tab characters, deselect this option.

This setting also controls whether clicking in the buffer with the mouse to either position the cursor or to make a selection will align the cursor to the nearest tab character, or allow the cursor to be placed in virtual space between tab characters.

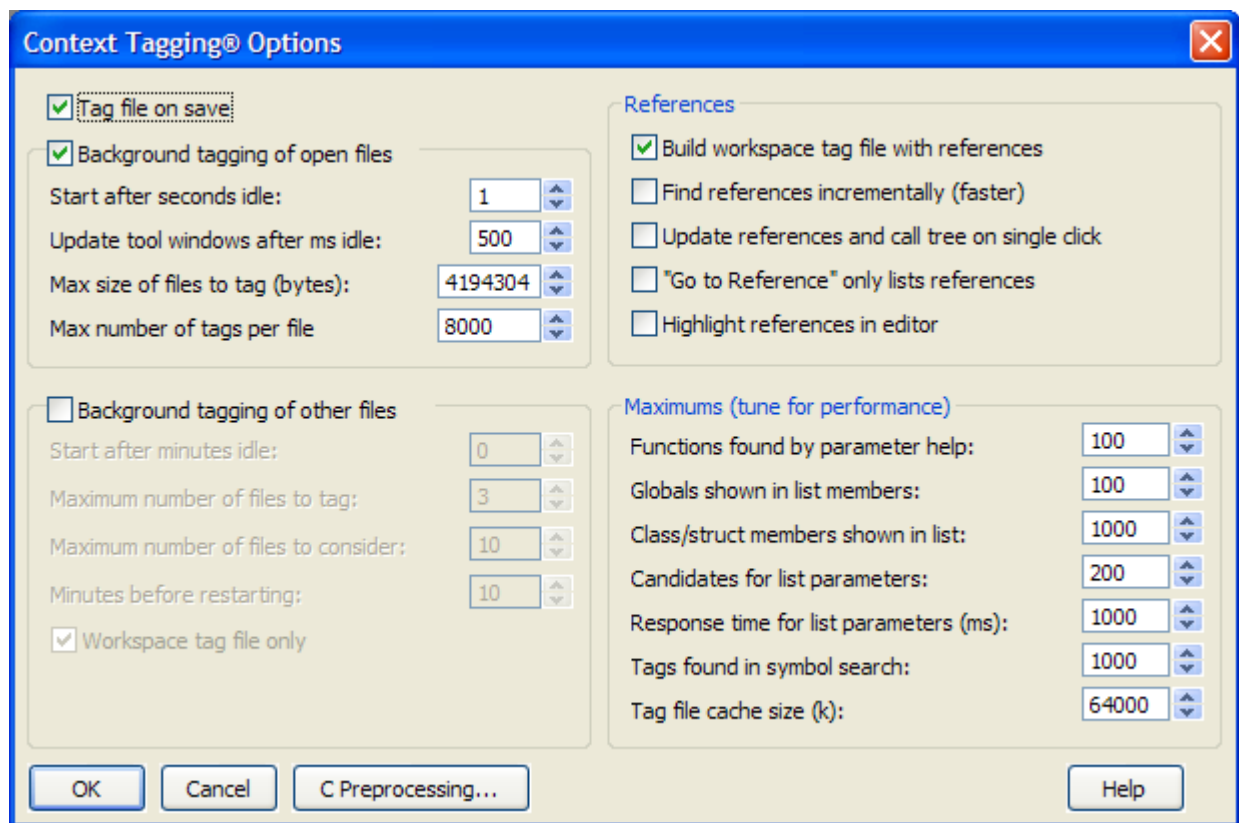
- **Pull chars backspace** - If selected, pressing the Backspace key when in replace mode (when **Start in insert mode** is off [**Tools > Options > General**, [More Tab](#)]) removes the previous character and moves the cursor left. If you want the previous character to be replaced with a space character, deselect this option.
- **Hack tabs backspace** - If selected, pressing the Backspace key when the previous character is a tab causes the rest of the line to be moved to the previous tab stop. If you are using a mode that has a syntax indent for each level that is different from the Tab settings (see [Indenting with Tabs](#)),

deselect this option. If you want your Backspace key to delete through tab characters one column at a time, select this option.

- **Treat leading spaces as tabs** - If selected, the commands **cursor\_left** and **cursor\_right** will move the cursor by tab stops when within leading space. If deselected, the **cursor\_left** and **cursor\_right** commands will move the cursor over by one physical character. The purpose of this option is to emulate the “feel” of real tab characters even if you only use spaces for indentation.

## Context Tagging® Options Dialog

The Context Tagging® Options dialog allows you to set general parameters for the Context Tagging® features. Here, you designate how the Context Tagging is done, how the references function within the application, and you can also tune the application to maximize performance. To set options, from the main menu, select **Tools > Options > Tagging Options**. The dialog box is displayed, as shown below. See [Building and Managing Tag Files](#) for more information on this topic.



The following settings are available:

- **Tag file on save** - When this option is selected, files are retagged when you save a modified file.
- **Background tagging of open files** - When this option is selected, all open files are retagged in the background if they have been modified.
- **Start after seconds idle** - When **Background tagging of buffers** is selected, re-tagging of buffers (opened files) starts after the user has not touched the keyboard or the mouse for this number of seconds.
- **Update tool windows after ms idle** - Number of milliseconds to wait before updating the Defs, Preview, and Current Context tool windows.



- **Max size of files to tag (bytes)** - Limits tagging to the files that have less than this number of bytes.
- **Max number of tags per file** - Limits the Defs and Current Context tool windows to files that have less than this number of tags, including statements if statement tagging is on. (See [Statement Level Tagging](#) for more information.)
- **Background tagging of other files** - Select this option if you want your tag files updated when another application modifies a file. This option is not on by default because it requires SlickEdit® to constantly perform disk I/O to check dates of files on disk.
- **Start after minutes idle** - When **Background tagging of files** is selected, re-tagging of files on disk starts after the user has not touched the keyboard or the mouse for this number of idle minutes.
- **Maximum number of files to tag** - When **Background tagging of files** is selected, this sets a limit to the number of files SlickEdit will re-tag in one pass.
- **Maximum number of files to consider** - When background re-tagging of files starts, you cannot use the editor until it is done with the amount of processing specified by this option or the option **Maximum number of files to tag**. The **Maximum number of files to consider** field specifies the number of file dates to compare. Specifying smaller maximum values means you will be able to regain access to the editor quicker and re-tagging will be slower. Specifying larger maximum values means it will take longer to regain access to the editor, but re-tagging will be quicker.
- **Minutes before restarting** - Specifies the number of minutes to wait, after background tagging has fully tagged all files, until background tagging can restart again. The default is 10 minutes.
- **Workspace tag file only** - When this option is selected, background tagging will cycle through only the workspace tag file. When not selected, background tagging will cycle through all of your extension-specific tag files (listed under **Tools > Tag Files**) in addition to the workspace tag file. .

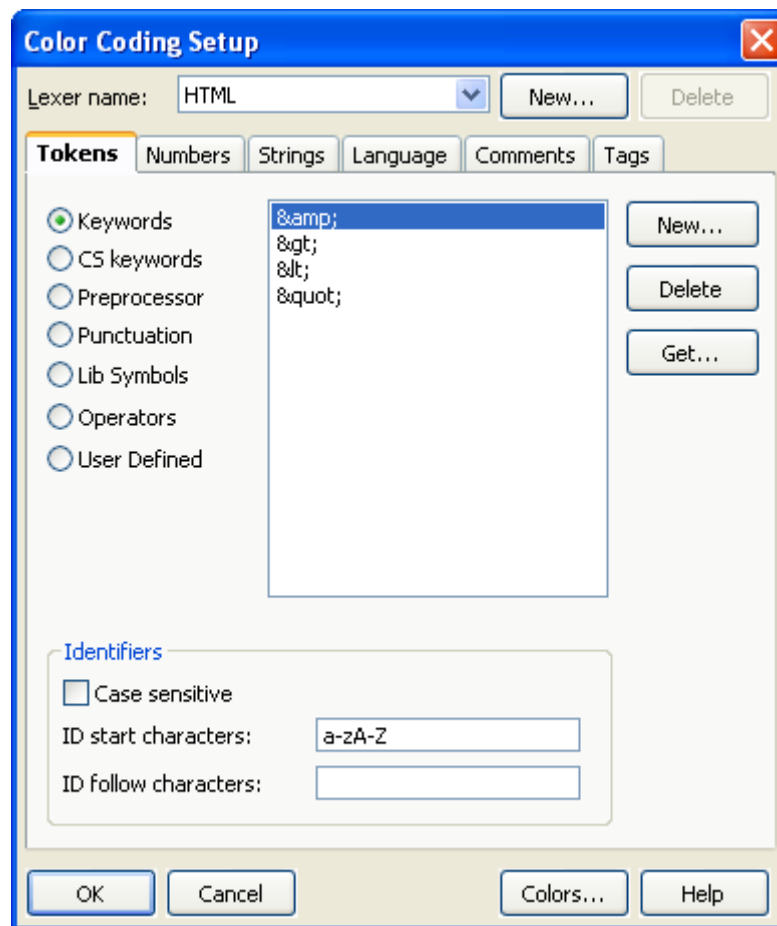
**CAUTION** We do not recommend you run a second copy of the editor to perform tag file updating because it will cause tag file access problems. Under UNIX the editor will crash if multiple editors are updating the same tag files.

- **References** - The following settings apply to References:
  - **Build workspace tag file with references** - When selected, newly-created tag files are built with support for symbol cross-references.
  - **Find references incrementally (faster)** - When unselected, all files with potential references are searched and analyzed so that the files which do not contain any references are removed. When this option is selected, querying references will appear to be faster, since analysis stops when a file is found containing a valid occurrence. However, you may see files which do not have any references to the symbol you are looking for listed in the References tool window.
  - **Update references and call tree on single click** - When this option is selected and you perform a single click on a new symbol in the Symbols tool window, the references in the References tool window are updated. This option is unselected by default because it can cause problems with double-click.
  - **“Go To Reference” only lists references** - When selected, “Go To Reference” will search for references, but it will not jump immediately to the first reference. To find the next reference, invoke the **find\_next** command (**Search > Find Next** or Ctrl+G).
  - **Highlight references in editor** - Select this option to have each reference highlighted within the file.

- **Maximums (tune for performance)** - You can tune Context Tagging® performance and accuracy by adjusting these values. Higher values will find more tags but increase search time. Lower values improve performance but may cause tags to be omitted.
  - **Functions found by parameter help** - When you invoke function parameter help, this setting limits the number of overloaded functions that will be displayed.
  - **Globals shown in list members** - When you invoke list members, this setting limits the number of global symbols that will be inserted into the list.
  - **Class/struct members shown in list** - When you invoke class/struct members, this setting limits the number of members that will be displayed in the list.
  - **Candidates for list parameters** - When you invoke auto list parameters, auto list compatible values are enabled. This setting limits the number of local variables and class members that will be evaluated to determine assignment compatibility.
  - **Response time for list parameters (ms)** - This setting is an upper limit on the amount of time SlickEdit will spend finding compatible parameters. Note that this is not a hard limit; in some cases, evaluating the assignment compatibility of a single variable can be time-consuming, especially when templates are involved.
  - **Tags found in search** - When you invoke the Find Tag dialog box (right-click in the [Symbols Tool Window](#) and select **Find Tag**) the number of tags found is limited by this setting. This setting also controls how many duplicate tags are tried when SlickEdit is attempting to evaluate the type of a symbol.
  - **Tag file cache size (k)** - You can improve tagging performance by adjusting the tag file cache size to better match the size of your tag files. Generally, a tag file cache size that matches the total size of the tag files being used will provide the best performance. For example, if the tag files for your source code and libraries adds up to 100 MB, you should set your cache size to 100 MB. You may have to experiment to find the optimum value. Use the recommendations below as a guide. Note that this is the same option that is found on the [Virtual Memory Tab](#) of the General Options dialog.
    - Minimum - 8 MB
    - Default - 64 MB
    - Ideal - Sum of tag file sizes
    - Maximum - 25% of physical system memory
- **C Preprocessing button** - Displays the C/C++ Preprocessing dialog box. Use this dialog to modify preprocessing so that Context Tagging can better analyze your code. See [C/C++ Preprocessing](#) for more information.

## Color Coding Setup Dialog

The Color Coding Setup dialog provides the capability to specify colors for identifying your code. To configure color coding, from the main menu, select **Tools > Options > Color Coding**. The Color Coding Setup box is displayed, as shown below.



General settings on this dialog box are described below (see [Color Coding Setup Options - General Dialog Settings](#)). Other options are categorized into the following tabs:

- [Tokens Tab](#)
- [Numbers Tab](#)
- [Strings Tab](#)
- [Language Tab](#)
- [Comments Tab](#)
- [Tags Tab](#)

## Color Coding Setup Options - General Dialog Settings

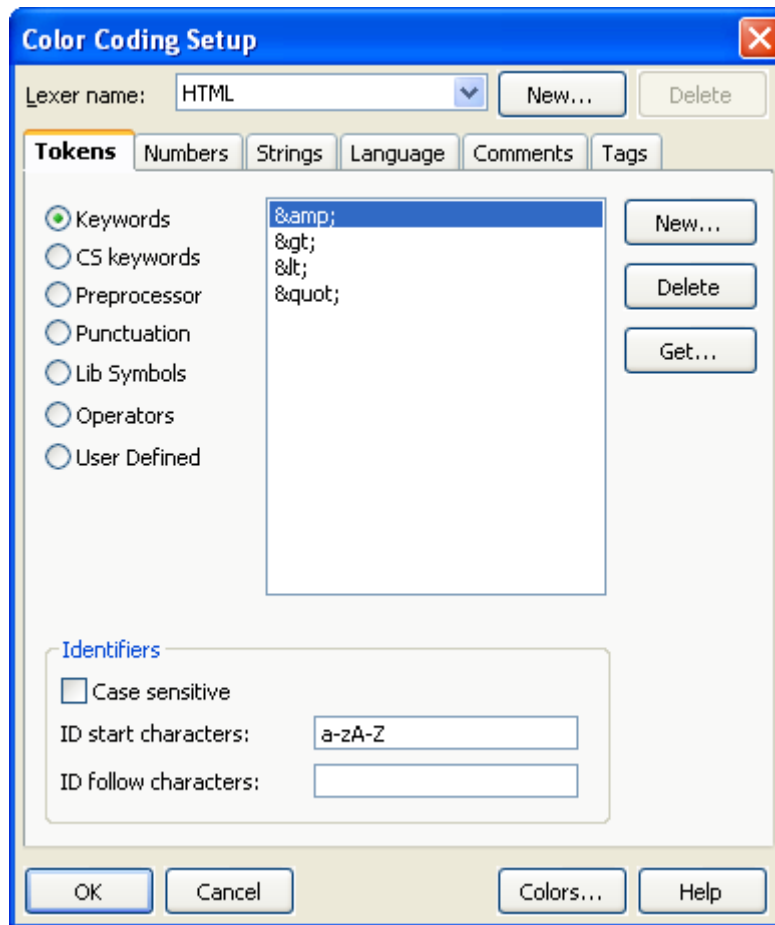
The following fields and buttons are available on the Color Coding Setup dialog:

- **Lexer name** - Select the language that you wish to work with from the **Lexer name** drop-down list. Be sure to select the lexer you wish to affect before using the tabs to make settings.
- **New** - Click **New** to prompt for a lexer name to start a new language-specific color coding definition (see [Creating Color Coding for a New Language](#)).
- **Delete** - Click **Delete** to remove a lexer name from the list. You can only delete user-created lexers.

- **Colors** - Click **Colors** at the bottom of the dialog to display the Color Settings dialog, which allows you to specify the color for color coding elements and other editor elements (see [Setting Colors for Screen Elements](#)).

## Tokens Tab

The Tokens tab, shown below, provides the capability to specify unique tokens to help you when working with your code. To access these settings, choose **Tools > Options > Color Coding**, then select the **Tokens** tab.



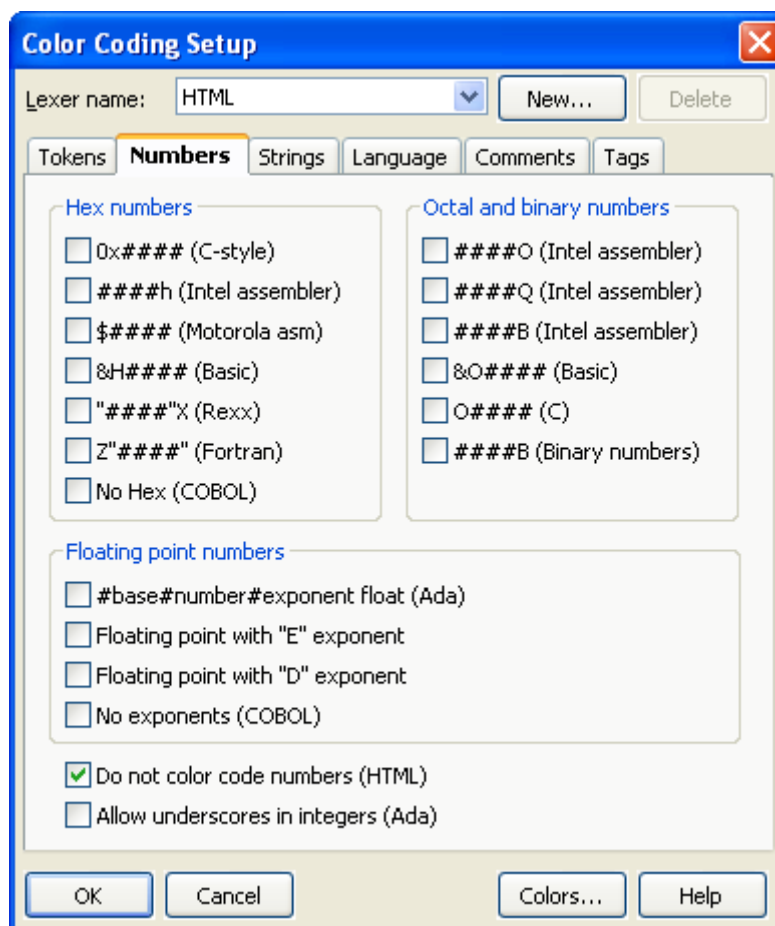
The following options are available:

- **Token type** - Select from the following token types:
  - **Keywords** - When this option is selected, the list box displays the words that have keyword color.
  - **CS keywords** - When this option is selected, the list box to the right displays case sensitive words that have keyword color. These words are always case sensitive even if the **Case Sensitive** check box is not selected.
  - **Preprocessor** - When this option is selected, the list box to the right displays preprocessor keywords in preprocessor color. All preprocessor keywords must start with the same character.

- **Punctuation, Lib Symbols, Operators, User Defined** - When one of these options is selected, the list boxes to the right display the words associated with each.
- **Identifiers** - Select from the following options:
  - **Case sensitive** - Indicates whether identifiers are case sensitive.
  - **ID start characters** - Specifies characters which are valid for the start of an identifier or any part of an identifier.
  - **ID follow characters** - Specifies additional characters which are valid after the first character of an identifier.
- **New** - Click the **New** button on the Tokens tab to add one or more words. Separates each word with a space.
- **Delete** - Deletes selected items in a list box.
- **Get** - Click this button to add words by selecting the file that contains the keywords that you want to add.

## Numbers Tab

The Numbers tab, shown below, provides options for color coding numerical values when working with SlickEdit®. To access these options, choose **Tools > Options > Color Coding**, then select the **Numbers** tab.



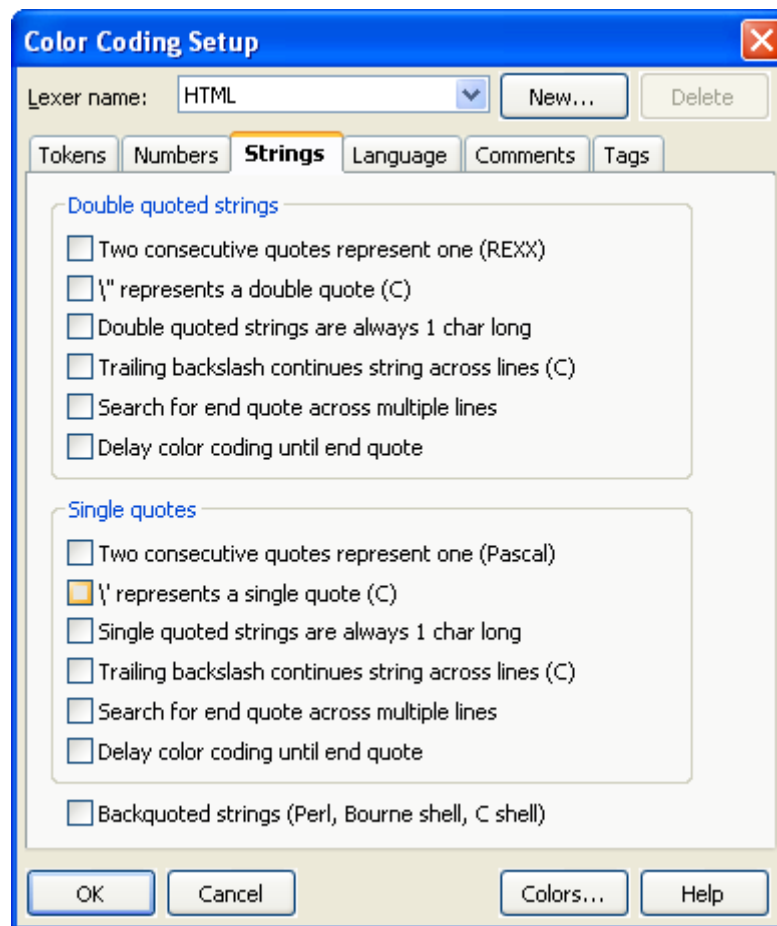
## OPTIONS

The following options are available:

- Hex numbers
  - **0x#### (C-Style)** - When this option is selected, text such as 0x123ABC is color coded in number color.
  - **####h (Intel assembler)** - When this option is selected, text such as 123ABCh is color coded in number color.
  - **\$#### (Motorola)** - When this option is selected, text such as \$123ABC is color coded in number color.
  - **&H#### (Basic)** - When this option is selected, text such as &H123ABC is color coded in number color.
  - **####X (Rexx)** - When this option is selected, strings such as "123ABC"X are color coded in number color.
  - **Z#### (Fortran)** - When this option is selected, strings such as Z"123ABC" are color coded in number color.
  - **No Hex (COBOL)** - When this option is selected, text such as 123ABC is not color coded in number color. By default (for most languages set in Language tab) 123ABC is color coded in number color.
- Floating point numbers
  - **#base#number#exponent float (Ada)** - When this option is selected, text such as #23#56#67 is color coded in number color.
  - **Floating point with E exponent** - When this option is selected, text such as 123.4E24 is color coded in number color.
- **Do not color code numbers (HTML)** - When this option is selected, text such as 123.4E24 and 123ABC is not color coded in number color. By default (for most languages set in Language tab), 123.4E24 and 123ABC is color coded in number color.
- **Allow underscores in integers (Ada)** - When this option is selected, text such as 12\_34 is color coded in number color.

## Strings Tab

The Strings tab, shown below, contains options for color coding strings. To access these settings, choose **Tools > Options > Color Coding**, then select the **Strings tab**.



The following options are available:

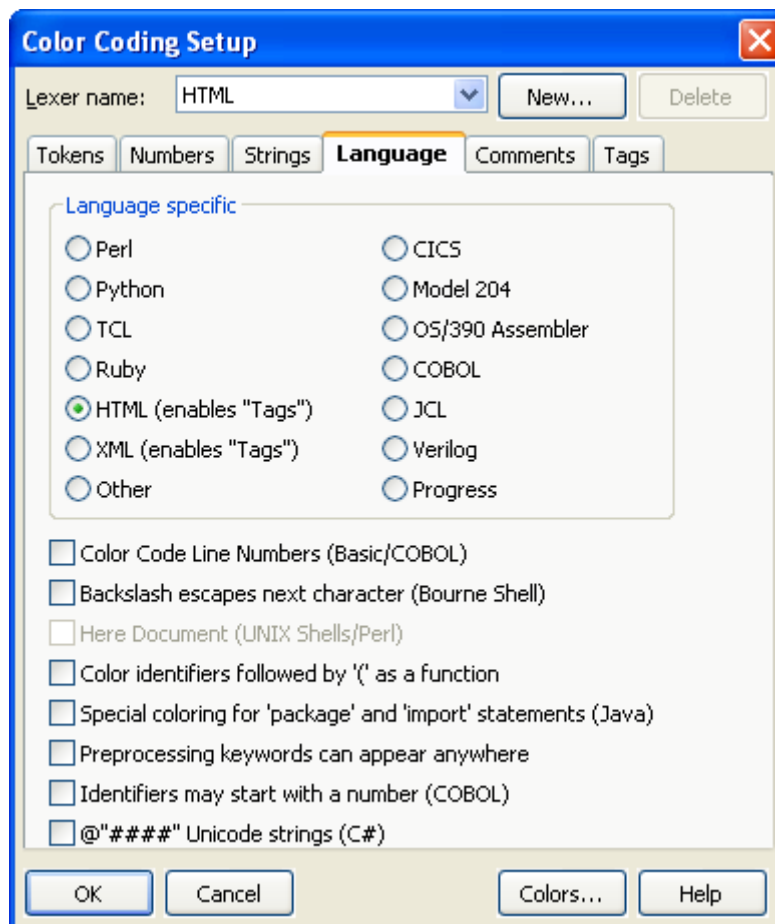
- Double quoted strings
  - **Two consecutive quotes represent one** - "" for REXX represents a string of length one which is a double quote character.
  - **Backslash double quote represents a double quote** - \" for C represents a string of length one which is a double quote character.
  - **Double quoted strings are always 1 char long** - When this option is selected, this means that a double quote character is followed by an additional character and then the terminating double quote character. There is never more than one character between the start and end double quote.
  - **Trailing backslash continues string across lines** - When this option is selected, it indicates that searching for the terminating quote continues to the next line if the lines end with a backslash character.
  - **Search for end quote across multiple lines** - When this option is selected, it indicates that the string does not have to be terminated on the same line as the start quote character.
  - **Delay color coding until end quote** - When this option is selected, a string is not color coded unless an end quote is seen on the same line. This does not support multi-line strings.
- Single quotes

## OPTIONS

- **Two consecutive quotes represent one** - "" (4 consecutive single quote characters) for Pascal represents a string of length one which is a single quote character.
- **Backslash single quote represents a single quote** - \" represents a string of length one which is a single quote character.
- **Single quoted strings are always 1 char long** - When this option is selected, a single quote character is followed by an additional character and then the terminating single quote character. There is never more than one character between the start and end single quote.
- **Trailing backslashes continues string across lines** - When this option is selected, it indicates that searching for the terminating quote continues to the next line if the lines end with a backslash character.
- **Search for end quote across multiple lines** - When this option is selected, it indicates that the string does not have to be terminated on the same line as the start quote character.
- **Delay color coding until end quote** - When this option is selected, a string is not color coded unless an end quote is seen on the same line. This does not support multi-line strings.

## Language Tab

The Language tab, shown below, is where you can set more language-specific color coding options. To access these settings, choose **Tools > Options > Color Coding**, then select the **Language tab**.



The following options are available:



- **Language specific** - To avoid requiring complicated BNF for defining color coding, some hardware language-specific adjustments have been added. You may be able to use one of these language-specific settings for another language, but there's no guarantee it will work.
- **Color code line numbers (Basic)** - When this option is selected, indicates that leading line numbers should be color coded in line number color.
- **Backslash escapes next character (Bourne Shell)** - Backslash escapes the character that follows. This is useful for UNIX shell scripts which use \" to indicate that the double quote is not the start a string.
- **Here Document (UNIX Shells/Perl)** - Enables support for **HERE** documents. Note that if you prefix your terminator with one of our lexer names you will get embedded language color coding. Example of a **HERE** document in Perl, where HTMLEOF is used as the terminator to get HTML embedded language color coding:

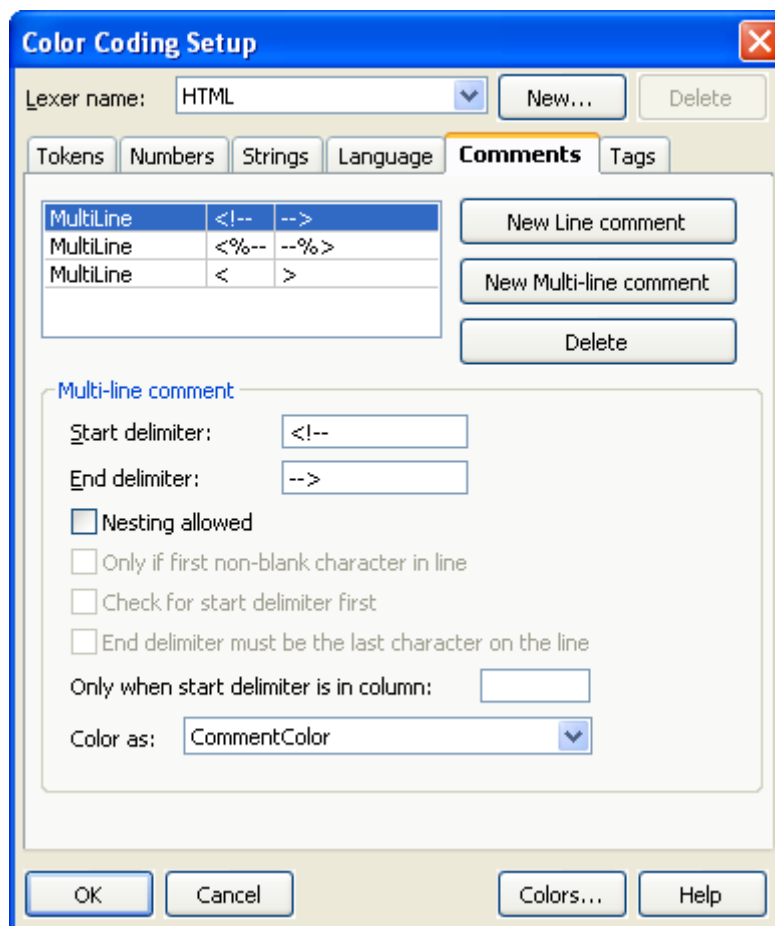
```
print <<HTMLEOF;
<HTML><HEAD><TITLE>...</TITLE></
HEAD>
<BODY>
...
</BODY>
</HTML>
HTMLEOF
```

Unknown languages are color coded in string color. Embedded language colors are user-definable.

- **Color identifiers followed by ( as a function** - For language such as C++, Java, and Slick-C®, an identifier followed by a parenthesis always indicates a function.
- **Special coloring for package and import statements (Java)** - When this option is selected, the Java syntax package and import statements are supported. This option is forced on for the lexer name Java. You must add the **package** and/or **import** keywords to your keyword list in order for this option to have any effect.
- **Preprocessing keywords can appear anywhere** - When this option is selected, preprocessing keywords are color coded even if they are not only preceded by white space.
- **Identifiers may start with a number (COBOL)** - When this option is selected, identifiers may start with one or more decimal digits. By default, leading decimal digits indicate a number.
- **@"####" Unicode strings (C#)** - When this option is selected, text in the form of @"any text" is coded as a string.

## Comments Tab

The Comments tab, shown below, is where you can set comment options for color coding. To access these settings, choose **Tools > Options > Color Coding**, then select the **Comments tab**.



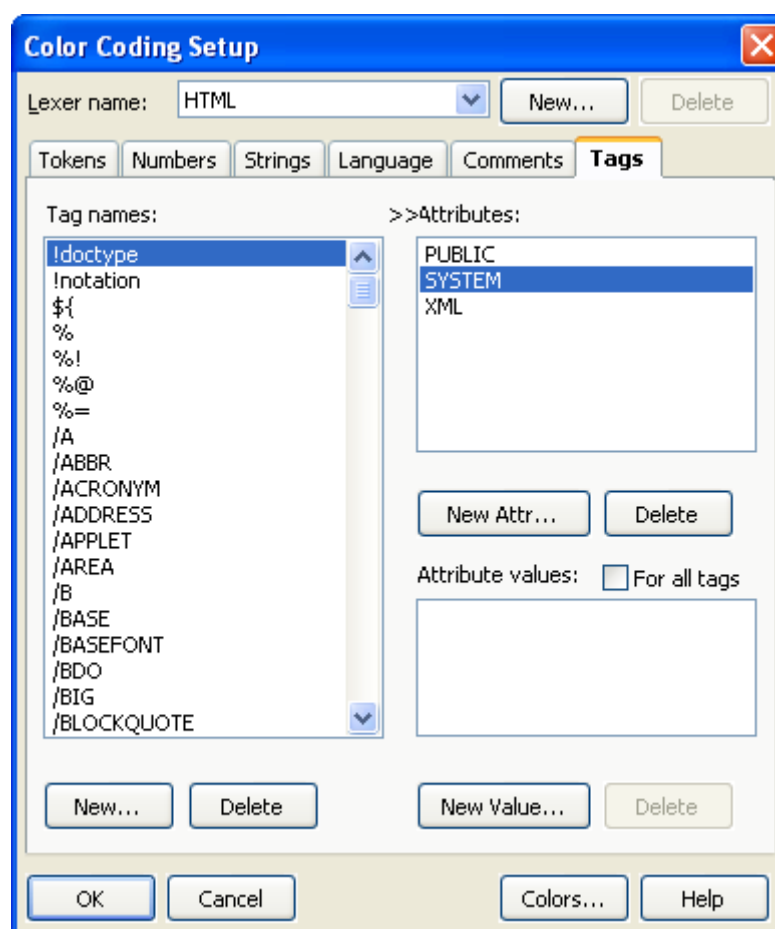
The following options are available:

- **New Line comment** - Click this button to define new single line comments.
- **New Multi-line comment** - Click this button to define new multi-line comments.
- **Line comment options** - The following line comment options are for working with multi-line comments:
  - **Start delimiter** - Delimiter which starts the multi-line comment. Currently, the first character of this string cannot be a valid identifier character.
  - **End delimiter** - Delimiter which ends the multi-line comment. Currently, the first character of this string cannot be a valid identifier character.
  - **Nesting allowed** - When this option is selected, this multi-line comment may have this multi-line comment inside it.
  - **Only if first non-blank character in line** - Indicates the start delimiter must be the first non-blank character in the line in order to be considered the start of a comment. This check box is enabled only when the **Only when start delimiter is in column** text box is completed.
  - **Check for start delimiter first** - When this option is selected, the lexer checks for the start delimiter before looking for other items. When this option is specified, the start delimiter is limited to one character in length.

- **End delimiter must be the last character on the line** - When this option is selected, the end delimiter text must occur at the end of a line to terminate the comment.
- **Only when start delimiter is in column** - Indicates that the start delimiter text starts a comment only when found in the column specified.
- **Color as** - Specifies color used for this comment. This color is not used when the start delimiter is immediately followed by one of the Comment Keywords. When the start delimiter is immediately followed by one of the Comment Keywords, keyword color is used.

## Tags Tab

The Tags tab of the Color Coding Setup dialog, shown below, is used to set attributes when working with languages that can be used with tagging. For example, HTML and XML are tagged-based languages. To access these options, choose **Tools > Options > Color Coding**, then select the **Tags** tab.



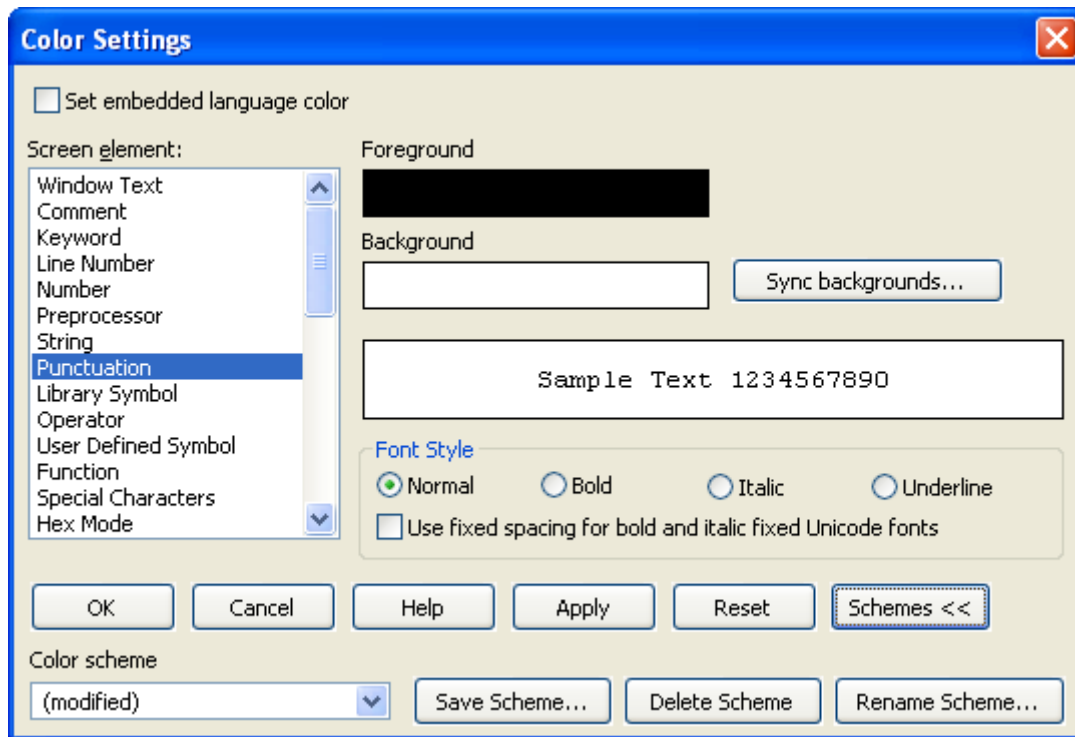
The following fields and settings are available:

- **Tag names** - List box containing tags for HTML or XML. To add or delete tags, use the **New** and **Delete** buttons below this list box.
- **Attributes** - List box containing attributes that belong to the tag selected in the Tag names list box. To add or delete attributes use the **New Attr** and **Delete** buttons below this list box.
- **Attribute values** - List box contains the values for the specified tag and attribute. To add or delete a value use the **New Value** and **Delete** buttons below this list box.

- **For all tags** - When this option is selected, the values in the **Attribute value** list box are applied to all tags that have the specified attribute.

## Color Settings Dialog

The Color Settings dialog contains options for changing embedded language colors and the colors of screen elements. See [Colors](#) for more information about changing these colors. To display the Color Settings dialog, choose **Tools > Options > Color**.



The following options are available:

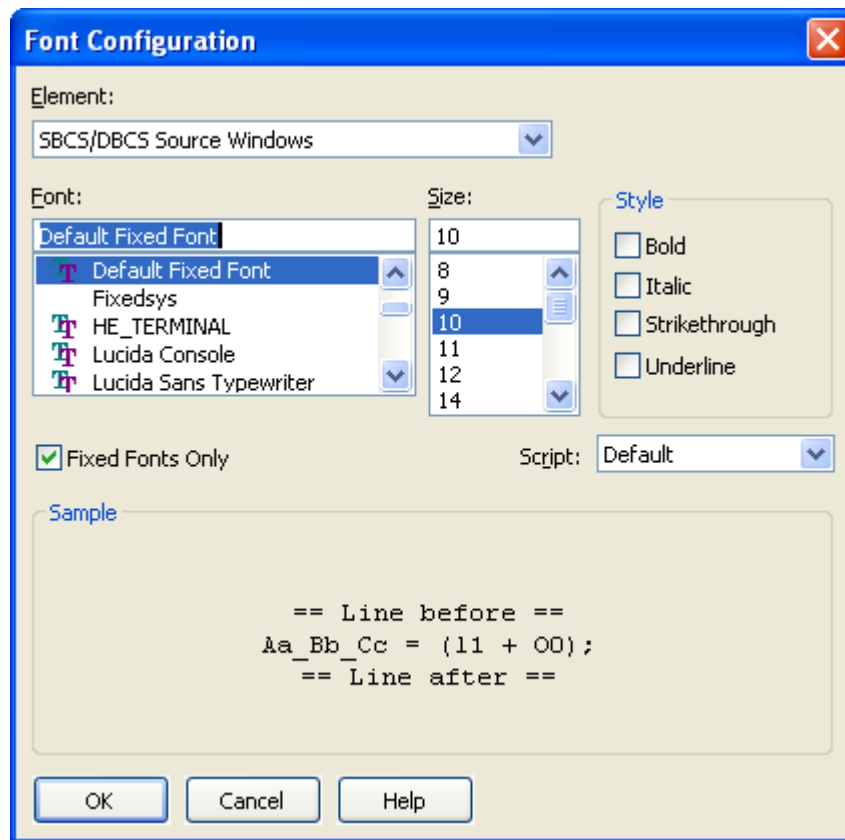
- **Set embedded language color** - When this option is selected, you can define the colors for source code (for example, JavaScript embedded in an HTML file). For HTML, the syntax color coding recognizes the `<script language="???">` tag and uses embedded language colors for the new language. In addition, for Perl and UNIX shell scripts, you can prefix your **HERE** document terminator with one of the color coding lexer names to get embedded language color coding. For an example, see [Setting Colors for Screen Elements](#)
- **Screen element** - Select the screen element before changing the foreground and background colors. Most of the screen element items are obvious except for those in the following list:
  - **Window Text** - This is the color of other text which is not a specific syntax element.
  - **Attribute (HTML only)** - This is the color used for a recognized attribute of an HTML tag. For example, the **src** attribute of the **img** tag gets this color.
  - **Cursor** - The Cursor screen element is displayed in the active edit window when the cursor is placed on the command line. It is not the color of the blinking cursor.
  - **Current Line, Current Selected Line, Selection** - SlickEdit will attempt to render these elements using your normal color settings for the Foreground color. The selected foreground color will only be used if there is not enough contrast between the font colors to be readable. It

is best to specify a Background color for these elements that is as close as possible to your normal background color, ensuring that the color-coded fonts are still easy to read.

- **Foreground/Background** - Click the color squares to change colors for the selected element. The Color Picker dialog is displayed, allowing you to pick a color from the palette or set your own custom color using RGB values.
- **Use system default** - When this option is selected, the operating system default colors are used. Currently, this check box is only enabled for the **Status** and **Message** fields. For UNIX, the System default colors are selected by the editor and not the operating system.
- **Sync backgrounds** - Click to apply the current background color as the background color for other elements. The Select Colors to Update dialog appears, from which you can select specific elements to affect.
- **Font Style** - For color-coded elements, you may choose whether the element is normal, bolded, italicized, or underlined. For example, keywords are bold by default.
- **Use fixed spacing for bold and italic fixed Unicode fonts** - (Unicode support required) When this option is selected, and a fixed font is selected for a Unicode source window, bold and italic color coding is supported. Since this requires the Unicode text to be converted to the active code page, some characters may be displayed incorrectly. The current editor display engine ignores bold and italic settings for proportional fonts or fixed Unicode fonts (which are treated like proportional fonts).
- **OK** - Applies color changes and closes the Color Settings dialog.
- **Cancel** - Restores all colors to the values they were when the Color Settings dialog box was first displayed in the current editor session.
- **Apply** - Updates all modified screen elements, which is useful for previewing what the colors look like. The dialog box is not closed so that you can make further changes if you wish.
- **Reset** - Restores all colors to the values they were when the editor was invoked.
- **Schemes** - Expands the Color Settings dialog box so you can try different color schemes, or define your own. See [Using Color Schemes](#) for more information.

## Font Configuration Dialog

The Font Configuration dialog contains options for changing the fonts and font styles of screen elements. See [Fonts](#) for more information about changing fonts and a list of some recommended fonts. To display the Font Configuration dialog, pictured below, choose **Tools > Options > Font**.



The following settings are available:

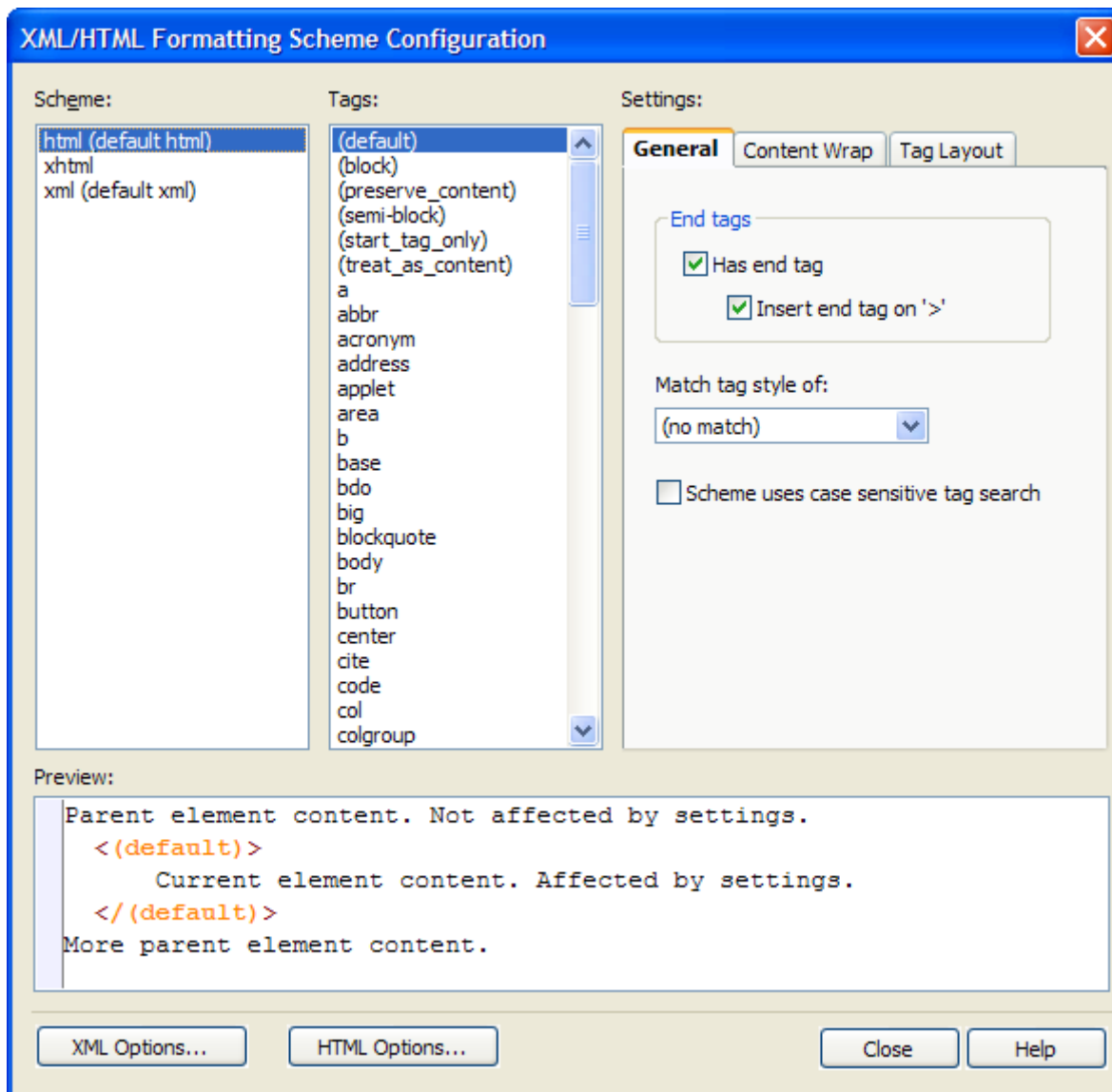
- **Screen Elements** - The **Element** drop-down list of the Font Configuration dialog contains the screen elements for which fonts can be changed. When an element is selected, the font type and size will automatically adjust to the current settings for that element, and a preview of the font will be displayed in the **Sample** area. Select from the following elements:
  - **Command Line** - The SlickEdit® command line displayed at the bottom of the application window.
  - **Status Line** - For status messages displayed at the bottom of the application window.
  - **SBCS/DBCS Source Windows** - Editor windows that are displaying non-Unicode content (for example, plain text).
  - **Hex Source Windows** - Editor windows that are being viewed in Hex Mode (**View > Hex**).
  - **Unicode Source Windows** - Editor windows that are displaying Unicode content (for example, XML).
  - **File Manager Windows** - Controls the display of the SlickEdit File Manager (**File > File Manager**).
  - **Diff Editor Source Windows** - The editor windows used by the DIFFzilla® tool.
  - **Parameter Info** - Controls the fonts used to display pop-ups with information about symbols and parameters.
  - **Parameter Info Fixed** - Used when we need to display a fixed-width font for parameter info, as when displaying example code.

- **Selection List** - The font used for selection lists, like the buffer list (**Document > List Buffers**).
- **Dialog** - Controls the font used in SlickEdit dialogs and tool windows.
- **HTML Proportional** - The default font used by HTML controls for proportional fonts. In particular, this affects the Version Control History dialog, the About SlickEdit dialog, and the Cool Features dialog.
- **HTML Fixed** - The default font used by HTML controls for fixed space fonts.
- **Font Settings** - The following settings and options are available on the Font Configuration dialog:
  - **Font and Size** - The Font and Size fields allow you to make typeface and point size changes to the selected screen element. The fonts that are listed are the fonts that are installed on your computer.
  - **Style** - Styles, such as bold and italic, can be enabled to affect the selected font.
- **Sample area** - This area provides a preview of the selected font, size, and style.
- **Fixed Fonts Only** - Select this option to display only fixed fonts in the Font field. By default, this option is not enabled.
- **Script (Windows only)** - Choose Default unless you are editing files that have characters not in the active code pages. Choose Western to use the typical English characters.

## XML/HTML Formatting Dialog

This dialog is used to configure the way XML and HTML code is automatically formatted as you edit. Note that XML/HTML Formatting must be enabled in order for these settings to work.

To display the XML/HTML Formatting dialog, select **Tools > Options > XML/HTML Formatting**, or use the `xml_html_options` command.



See [XML/HTML Formatting](#) for information about enabling formatting and working with this feature. See [Formatting Settings](#) for information about the General, Content Wrap, and Tag Layout tabs.

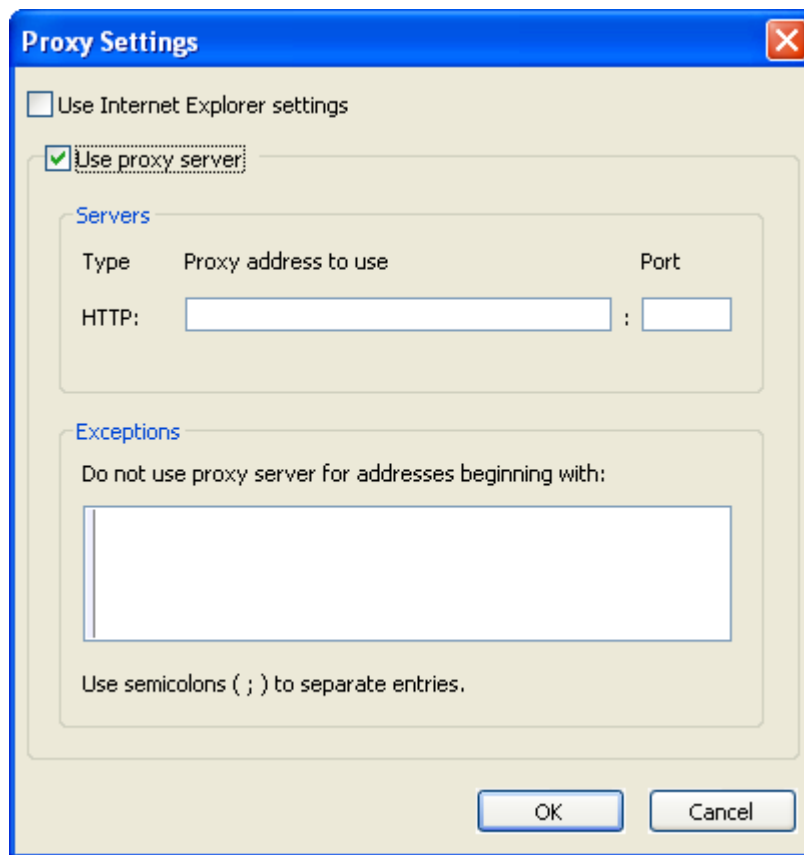
## URL Mappings Dialog

This dialog allows you to map URLs to a different location. See [URL Mappings](#) for information.

## Proxy Settings Dialog

If you need to configure proxy settings for when SlickEdit® needs to use an Internet connection, use the Proxy Settings dialog, pictured below (**Tools > Options > Proxy Settings**).



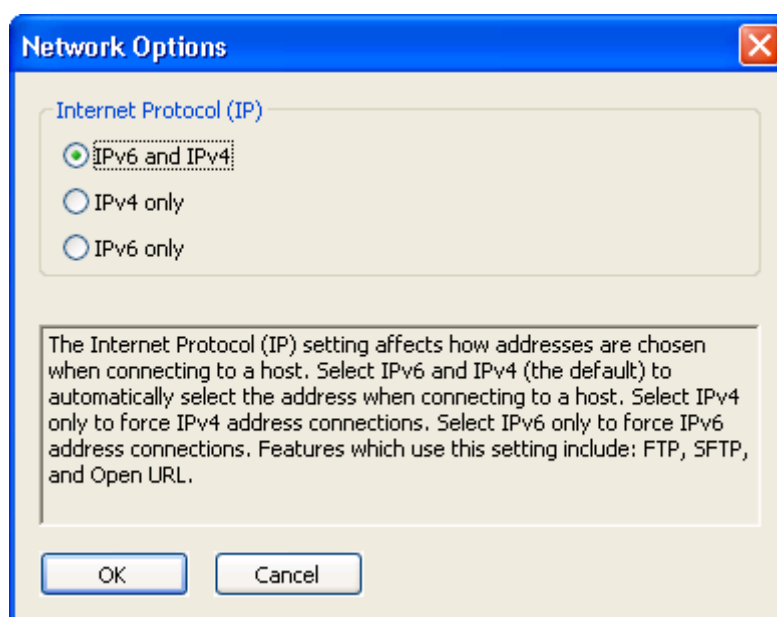


The following options are available:

- **Use Internet Explorer settings** - If selected, Internet Explorer settings will be used, and the remaining options and fields on the dialog will be unavailable.
- **Use proxy server** - If selected, the remaining options and fields will become available.
- **Servers** - Indicates the proxy address and port to use.
- **Exceptions** - Indicates the Web site addresses that the proxy server should disregard. Separate entries with semicolons (;).

## Network Options Dialog

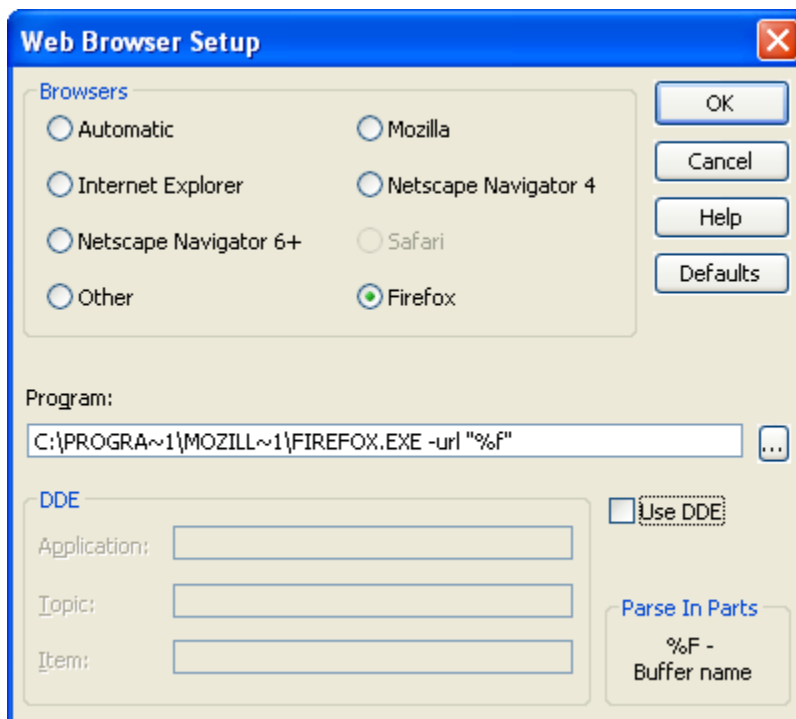
This dialog allows you to set the Internet Protocol (IP) version. The Internet Protocol (IP) setting affects how addresses are chosen when connecting to a host. Select **IPv6** and **IPv4** (the default) for SlickEdit® to automatically select the address when connecting to a host. Select **IPv4** only to force IPv4 address connections. Select **IPv6** only to force IPv6 address connections. Features that use this setting include [FTP](#), [SFTP](#), and [Opening a URL](#).



## Web Browser Setup Dialog

The Web Browser Setup dialog, shown below, contains options for specifying the browser to use when SlickEdit® needs to launch one. To access this dialog, choose **Tools > Options > Web Browser Setup**.

**NOTE** This configuration does not apply to the Help system. See [Supported Web Browsers](#) for more information.

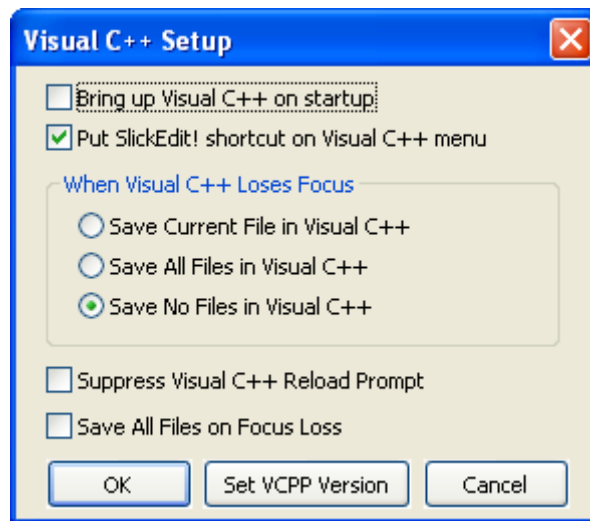


The following settings are available:

- **Browser** - Select which Web browser you want to use. Selecting a preferred browser automatically sets the defaults for the other items in the Web Browser Setup dialog box. Note the following:
  - **Windows** - Your Web browser is automatically detected.
  - **UNIX and Mac OS X** - You need to specify which Web browser you are using. In addition, you need to give the full path to the program executable.
- **Program** - Indicates the program to run. You may specify a **%F** in this text box or any of the other text boxes on this dialog box to have the HTML file name inserted into the command that is executed.
- **DDE** - The **Application**, **Topic**, and **Item** text boxes specify DDE XTYP\_REQUEST parameters and are used only if the **Use DDE** option is selected.

## Visual C++ Setup Dialog

The Visual C++ Setup dialog, shown below, is available for users of Visual C++ for configuring general options. To access this dialog, choose **Tools > Options > Visual C++ Setup**.



The following options are available:

- **Bring up Visual C++ on startup** - If selected, SlickEdit® will start Visual C++ when you invoke the editor.
- **Put SlickEdit! shortcut on Visual C++ menu** - If selected, SlickEdit places a **SlickEdit!** menu item on the Visual C++ menu bar. When you click on the **SlickEdit!** button, the current file is loaded into the editor. The cursor location is transferred as well.
- **When Visual C++ Loses Focus** - Select from the following options:
  - **Save Current File in Visual C++** - If selected, the current file in Visual C++ is saved when Visual C++ loses focus.
  - **Save All Files in Visual C++** - If selected, all files in Visual C++ are saved when Visual C++ loses focus.
  - **Save No Files in Visual C++** - If selected, no files in Visual C++ are saved when Visual C++ loses focus.
- **Suppress Visual C++ Reload Prompt** - If selected, the Visual C++ reload file prompt is not displayed when Visual C++ gets focus.
- **Save All Files on Focus Loss** - If selected, all files in SlickEdit are saved when SlickEdit loses focus.
- **Set VCPP Version** - Click this button to specify the version of Visual C++ that you are using from a selection list.

## Window

---

This section describes items on the **Window** menu and associated dialogs and tool windows. For more information about working with editor windows, see [Buffers and Editor Windows](#).

### Window Menu

The table below describes each item on the **Window** menu and its corresponding command.

Window Menu Item	Description	Command
Cascade	Cascades editor windows.	<b>cascade_windows</b>
Tile	Tiles editor windows.	<b>tile_windows</b>
Tile Horizontal	Tiles editor windows horizontally when there are three or less windows.	<b>tile-windows h</b>
Arrange Icons	Rearranges iconized windows.	<b>arrange_icons</b>
Next	Switches to next window.	<b>next_window</b>
Previous	Switches to previous window.	<b>prev_window</b>
Close	Closes the current window.	<b>close-window</b>
Font	Displays the Window Font dialog, which allows you to set/view fonts for the current editor window or all windows. See <a href="#">Window Font Dialog</a> .	<b>wfont</b>
New Window Size	Sets/views new window size settings.	<b>new_window_size</b>
Split Horizontally	Splits the current window horizontally in half.	<b>hsplit_window</b>
Split Vertically	Splits the current window vertically in half.	<b>vsplit_window</b>
Zoom Toggle	Zooms or unzooms the current window.	<b>zoom_window</b>
One Window	Zooms the current window and deletes all other windows.	<b>one_window</b>
Duplicate	Creates another window linked to the current buffer.	<b>duplicate_window</b>
Link Window	Displays the Link Window dialog, which allows you to select a buffer to display in the current editor window. See <a href="#">Link Window Dialog</a> .	<b>link-window</b>

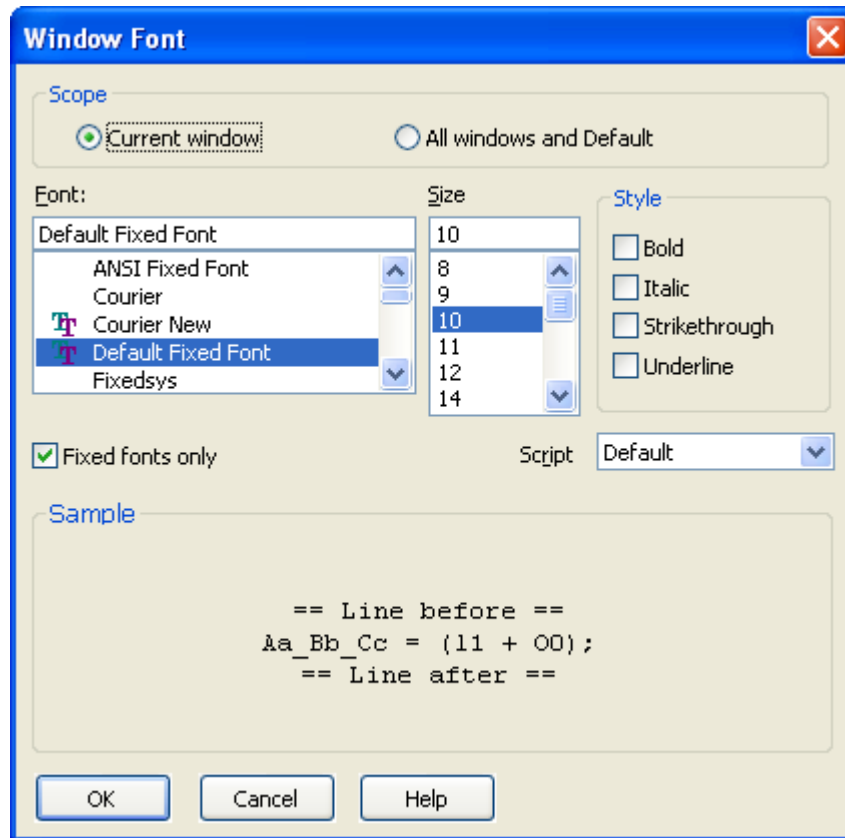
## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Window** menu items.

### Window Font Dialog

The Window Font dialog is used to set the font and font style of editor windows. For more information about setting fonts, see [Setting Fonts and Colors](#).

To access the Window Font dialog, choose **Window > Font**, or use the **wfont** command.



The following options and settings are available:

- **Scope** - Specifies the editor windows to affect.
  - **Current window** - Affects the current editor window only.
  - **All windows and Default** - Affects all open editor windows and all newly-created editor windows.
- **Font** - Displays a selection list of the fonts installed on your computer.
- **Size** - Displays a selection list of the sizes that are available for the selected font.
- **Style** - Displays a selection list of common font style options such as bold, italic, etc.
- **Fixed Fonts Only** - If selected, only fixed fonts that are installed on your computer are displayed in the Font list box.
- **Script** - (Windows only) Displays a selection list of character language settings. Choose **Default** unless you are editing files that have characters not in the active code pages. Choose **Western** to use the typical English characters.
- **Sample** - Displays a preview of the selected font and settings.

## Link Window Dialog

The Link Window dialog is used to link files to editor windows, so that you can view more than one file in one editor window. For more information about working with editor windows, see [Buffers and Editor Windows](#).

To access the Link Window dialog, choose **Window > Link Window**, or use the **link\_window** command. It provides the following options:

- **Link to Window** - Changes the file that is displayed in the current window to be the selected file/buffer in the list box. Modifications made to the buffer that was previously displayed will not be lost.
- **Open File** - Opens a file and displays it in the current window. No additional window is created.
- **Start Process** - Starts a process buffer and displays it in the current window. If a process buffer has already been started, it is linked to the current window.





## Help

---

This section describes items on the **Help** menu and associated dialogs and tool windows. For more information about how to use the Help system and how to obtain product support, see [Help and Product Support](#). For advanced Help configuration topics, such as accessing multiple Help files and configuring F1 MSDN Help, see [Advanced Help Configuration](#).

## Help Menu

The table below describes each item on the **Help** menu and its corresponding command.

Help Menu Item	Description	Command
Contents	Displays the Help system open to the Table of Contents.	<b>help -contents</b>
Index	Displays the Help system open to the Index, where you can search for index items.	<b>help -index</b>
Search	Displays the Help system open to the Search tab, where you can search for any item.	<b>help -search</b>
Cool Features	Displays the Cool Features dialog, which shows SlickEdit® feature tips. See <a href="#">Cool Features Dialog</a> .	<b>cool_features</b>
Keys Help	Displays the Help system open to the emulation key binding reference tables for the current emulation.	<b>help key bindings</b>
What Is Key	Used to discover the command associated with a key binding. Opens the command line, prompting with the text "What is key:".	<b>what-is</b>
Where Is Command	Used to discover the key binding associated with a command. Opens the command line, prompting with the text "Where is command:".	<b>where-is</b>
Macro Functions by Category	Displays the Help system open to this topic, which shows a categorized list of macro functions.	<b>help macro functions by category</b>
Frequently Asked Questions	Invokes a Web browser which opens to the FAQs section of the SlickEdit Web site, which contains answers to common user questions.	<b>goto_faq</b>
F1 Index Help	Displays the Help Index dialog, where you can search the F1 index file for help on a word. See <a href="#">Help Index Dialog</a> .	<b>help_index</b>

Help Menu Item	Description	Command
Configure F1 Index File	Displays the Configure Help Index File dialog, where you can view or modify the index file used by F1 word help. See <a href="#">Configure Help Index File Dialog</a> .	<b>configure_index_file</b>
Configure F1 MSDN Help	Displays a dialog which lets you enable and configure MSDN Library F1 word help. See <a href="#">Configuring F1 MSDN Help</a> .	<b>msdn_configure_collection</b>
Product Updates	Displays the Product Updates menu, from which you can install updates and hot fixes. See <a href="#">Product Updates Menu</a> .	N/A
Register Product	Displays the Register dialog, from which you can begin the SlickEdit on-line registration process.	<b>online_registration</b>
SlickEdit Support Web Site	Invokes a Web browser which opens to the SlickEdit Support Web site.	<b>goto-slickedit</b>
Contact Product Support	Used to invoke a Web browser which opens to a form on the SlickEdit Web site that you can use to contact Product Support.	<b>do-webmail-support</b>
Check Maintenance	Invokes a Web browser which opens to a SlickEdit Web page that shows the status of your Maintenance and Support Agreement.	<b>check-maintenance</b>
About SlickEdit	Displays a property sheet containing information about your product, such as serial and version numbers, as well as release notes and the license agreement.	<b>version</b>

## Product Updates Menu

The table below describes each item on the **Help > Product Updates** menu and its corresponding command.

Product Updates Menu Item	Description	Command
New Updates	Checks for new updates to the product.	<b>upcheck_display</b>
Options	Displays the Update Manager Options dialog, used to set the frequency of automatic checking of new updates. See <a href="#">Update Manager Options Dialog</a> .	<b>upcheck_options</b>
Load Hot Fix	Displays an Open-style dialog, to begin the process of installing a hot fix. See <a href="#">Hot Fixes</a> .	<b>load_hotfix</b>

Product Updates Menu Item	Description	Command
List Installed Fixes	Displays a summary sheet of hot fixes that are installed on your computer. See <a href="#">Hot Fixes</a> .	<b>list_hotfixes</b>

## Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Help** menu items.

### Cool Features Dialog

The Cool Features dialog appears after the product installation has completed. To access this dialog, choose **Help > Cool Features**. The following options and buttons are available:

- **Options for Feature** - Displays the dialog from which you can make settings for the selected feature.
- **Help on Feature** - Displays the Help system open to the documentation for the selected feature.
- **View Demonstration** - Invokes a Web browser which navigates to a SlickEdit Web page containing an audio/visual demonstration of the feature in action.
- **Topics** - Displays a table of contents from which you can select a Cool Feature to learn more about.
- **Prev** - Scrolls to the previous Cool Feature.
- **Next** - Scrolls to the next Cool Feature.
- **Show on startup** - If selected, disables the Cool Features dialog from appearing each time the editor is started.

### Help Index Dialog

The Help Index dialog scans a help .idx file for help files that contain a prefix match or exact match of the keyword under the cursor. To display this dialog, press **F1** when the cursor is under a keyword, or choose **Help > F1 Index Help**. The following options and buttons are available:

- **Help Keyword** - Specifies the keyword to search for. Type directly into this field, or, if you are seeking help for a keyword under the cursor, the keyword will be automatically filled in.
- **Help Files list box** - Displays a list of the help files in which the keyword was found. Select the help file to use.
- **Close on choose** - If selected, the dialog is closed when you click OK.
- **Exact match** - If selected, only help files which contain an exact match of the keyword specified are displayed.
- **Use default** - If selected, no search is performed for a valid help item. A DDE or shell command is performed to get help on the Help Keyword specified. The exact action here depends on the Default Help dialog (see [Default Help Dialog](#)). Visual C++ help usually works without selecting this option.
- **Configure Index** - Displays the Configure Help Index File dialog, which lets you specify the help files to be used when searching for help. See [Configure Help Index File Dialog](#) below.

## Configure Help Index File Dialog

The Configure Help Index File dialog is used to build and manage the help files that are to be used when searching for help. To access this dialog, choose **Help > Configure F1 Index File**. The following options and buttons are available:

- **Help files list box** - Displays a list of the help files that you have added. Use the Up and Down buttons to move the selected files up and down in the list.
- **Add** - Allows you to browse for another help file on your system to add.
- **Remove** - Removes the selected help file from the list.
- **Edit** - Allows you to change the description of the selected help file.
- **Export** - Allows you to save the list of keywords for the selected help file in Help List Text File format (\*.hlt).
- **Help file path** - Type a list of directories, separated with semicolons, in this text field. These directories are used when searching for help files. These directories are required because .idx files contain the help file names without path information.
- **Load** - Allows you to browse for a Help Index File in .idx format to load. If the file doesn't exist, it will be created.
- **Scan** - Scans for help files which contain a prefix match or exact match of the keyword selected.
- **More** - Expands the dialog to show a list of keywords in the selected help file if the option **Show keywords** is selected.
- **Default** - Displays the Default Help dialog. See [Default Help Dialog](#) below.

## Default Help Dialog

To access this dialog, choose **Help > Configure F1 Index File**, then click the **Default** button.

The Default Help dialog is typically used to interface with Microsoft Visual C++ MVB files. However, you could disable the **Use DDE** option and configure the **Program** text box to run a program like Microsoft Quick Help (**qh**) and pass the word at the cursor (**%k**). However, you may prefer to bind the **qh** command to a key to separate Quick Help word help access.

- **Program** - Indicates the program and arguments to run. The help program is only executed if no other help file contains information on the word at the cursor or if using DDE requires a program to be started before the DDE request can be made.
- **DDE** - The **Application**, **Topic**, and **Command** text boxes specify DDE XTYP\_REQUEST parameters and are used only if the **Use DDE** option is selected.

## Update Manager Options Dialog

The Update Manager checks for new product updates. To set the frequency of automatic updates, use the Update Manager Options dialog (**Help > Product Updates > Options**). Click Proxy Settings to display the Proxy Settings dialog (see [Proxy Settings Dialog](#)).

## Menu Editing

---

For information about accessing menus in SlickEdit and associated options, see [Accessing Menus](#).

## Creating and Editing Menus

Menus in SlickEdit® are controlled by Slick-C® macro files. You can customize menus by editing these files.

If you plan to customize your menu items, be sure to back up your configuration directory before installing any updates or new versions of SlickEdit, as they will overwrite your changes.

Menus can be managed using the Open Menu dialog. From this dialog, you can pick a menu to edit, create a new menu, run a menu as a popup. To access this dialog, from the main menu, choose **Macro > Menus** (or use the **open\_menu** command). The following buttons are available:

- **Open** - Opens the menu specified in the combo box for editing with the Menu Editor. If the menu specified does not already exist, it is created.
- **New** - Creates a new menu with a unique name for editing with the Menu Editor. The Menu Editor allows you to change the name of the menu.
- **Delete** - Deletes the specified menu from the combo box.
- **Show** - Runs the menu by displaying it as a pop-up. Use this button during macro recording to create a command which runs a menu by displaying it as a pop-up. If you bind the command to a left or right button mouse event, the menu will be displayed at the cursor position.

You can use the Menu Editor to create a new menu, or modify the SlickEdit menu bar or an existing menu resource, which can be displayed as a popup or menu bar. To access the Menu Editor, from the main menu, choose **Macro > Menus**. The Open Menu dialog is displayed.

## Creating a New Menu Resource

To create a new menu resource, from the main menu choose **Macro > Menus** (or use the **open\_menu** command). The Open Menu dialog box is displayed. Click **New** and the Menu Editor is displayed.

To create a command which runs a menu by displaying it as a pop-up, after creating a menu, while macro recording, click the **Show** button on the Open Menu dialog box. If you bind the recorded command to a left or right mouse button event, the menu will be displayed at the cursor position. You DO NOT need to specify key bindings for menu items because the Menu Editor automatically determines the key bindings for you. To choose between short and long key names, from the main menu choose **Tools > Options > General**, select the [More Tab](#), and change the option **Short key names**.

See the *Slick-C® Macro Programming Guide* for information on creating forms with menu bars or advanced information.

## Editing Menus

To select a menu for editing, from the main menu choose **Macro > Menus** (or use the **open\_menu** command). Select the menu to edit from the list, then click **Open**. The Menu Editor will be displayed. See [Menu Editor Dialog](#) for a list of the available options.

## Defining Menu Item Aliases

The Menu Item Alias dialog box allows you to define aliases (which are similar commands) for the command that is being executed. Enter each alias command on a separate line. If one of the alias commands are bound to a key, that key name will be displayed to the right of the menu item. For example, the **e** and **edit** commands are absolutely identically in function except that the **e** command requires fewer characters to type. The **gui\_open** command is identical to the **edit** command except that it prompts the user with a dialog box, whereas the **edit** command prompts for files on the command line. These two examples illustrate the best reasons for using aliases.

## Enabling/Disabling Menu Items

SlickEdit® has some enable or disable attributes for predefines which you can specify for any command. When these predefined auto-enabling attributes are not enough, then you need to implement a callback which determines the enable or disable state of the command. See the *Slick-C® Macro Programming Guide* for information on enabling and disabling menu items with your own callback.

The Auto Enable Properties dialog box is used for these settings, and can be accessed from the main menu by choosing **Macro > Menus**. When the Open Menu dialog box is displayed, click **New** to display the Menu Editor. Click the **Auto Enable** button, and the Auto Enable Properties dialog is displayed.

For descriptions of the options on this dialog, see [Auto Enable Properties Dialog](#).

# Appendix

This chapter contains the following topics:

- [Tutorials](#)
- [Encoding](#)
- [Invocation Options](#)
- [Environment Variables](#)
- [Configuration Variables](#)
- [Configuration Directories and Files](#)
- [File Search Order](#)
- [Advanced Help Configuration](#)
- [VLX File and Color Coding](#)
- [Editing the Key Binding Source](#)
- [Using the ISPF and XEDIT Emulations](#)
- [Regular Expression Syntax](#)
- [Emulation Tables](#)





## Tutorials

---

### Hello World Tutorial (C/C++)

This tutorial outlines the steps to create, build, and run a sample Hello World program using the auto-build system for GNU C/C++ projects.

The sample C++ program prints the text **hello world** to the standard output on the Console view. Follow these steps to create a Hello World program using the GNU C/C++ wizard and without using the wizard.

#### Create a Project Using GNU C/C++ Wizard

1. From the main menu, select **Project > New**.
2. From the Project tab, select **GNU C/C++ Wizard**.
3. Specify a project name, "HelloWorld". Change this location if you want.
4. Select the Executable Project Type and Source Type of C++.
5. Click **Next**.
6. For the Application type, select **A Hello World application**.
7. Click **Next**.
8. Select **Build without a makefile**.
9. Click **Finish**. Click **OK**. The wizard constructs a workspace by the name of HelloWorld.vpw, a project called HelloWorld.vpj, and a program file called HelloWorld.cpp.

#### Create a Project Without Using the GNU C/C++ Wizard

1. Create a workspace.
2. From the Project menu, select **New**.
3. Select the Workspace tab and enter the name of the workspace, **HelloWorld**. By default this creates the workspace in the directory `C:\HelloWorld`. You can change this location if you want. This creates a workspace named HelloWorld.vpw.
4. Now, create the project.
5. From the main menu, select **Project > New**.
6. Select the **Project** tab and type the Project Name, **HelloWorld**.
7. Select **Add to current workspace**. By default, this adds a new subdirectory named HelloWorld to the workspace directory `c:\HelloWorld\HelloWorld`. To produce the same results as above, revise the **Location** field to remove the second HelloWorld subdirectory.
8. Select the Executable Project Type and Source Type of C++. For the Application type, select **A HelloWorld application**. Select **Build without a makefile**.
9. Click **Finish**.

#### Build the Project

To build this project, from the main menu, select **Build**.

## Run the Program

To run the program, from the main menu, select **Build > Execute**. The application displays “Hello World” in the output window.

## Comments

When creating a new project in a new workspace, a new workspace does not have to be explicitly created.

The workspace is created automatically when the project is created. The workspace will be given the same name as the project.

For large projects, multiple projects most likely will be created and the workspace name should be distinct from the project names for easier organization.

## Hello World Tutorial (Java)

This tutorial outlines the steps to create, build, and run a sample Hello World program for Java projects. The sample Java program prints the text **Hello World** to the standard output .

### Create the Project

1. From the main menu, select **Project > New**.
2. On the Project tab select **Java > Empty Project**.
3. Type the Project Name, “HelloWorld”. If you already completed the C/C++ tutorial you will need to enter a different name, like “HelloWorldJava”.
4. This creates a workspace and project by the name “HelloWorld” at `c:\HelloWorld`.

### Create the File

1. From the **Project** menu, select **New**.
2. Select the **File** tab.
3. Select **Java** from the list and enter a file name, “HelloWorld.java”. Be sure to type the file extension.
4. Check the Add to Project check box or the file will be created but will not be able to be built.
5. By default the file is created in the directory created in the previous step.

### Edit the File

Edit the file to enter a Hello World program:

#### Example

```
public class HelloWorld {  
    public static void main (String args[]) {  
        System.out.println("hello world");  
    }  
}
```

## Build the Project

From the main menu, select **Build**.

## Run the Program

From the main menu, select **Run > Execute**. The words “Hello World” are displayed in the window.



# Encoding

---

Encodings are used to convert a file to either SBCS/DBCS for the active code page or Unicode (more specifically UTF-8) data. By default, XML and Unicode files with signatures (UTF-8, UTF-16 and UTF-32) files are automatically loaded as Unicode UTF-8 data, while other more common program source files like .c, .java, and .cs source files are loaded as SBCS/DBCS active code page data.

All file data can be configured to Unicode UTF-8 data, but this would cause some problems. Loading files containing SBCS/DBCS data would take significantly longer, slowing down parsing by Context Tagging® and any other multi-file operations. In addition, Unicode editors cannot support all the features supported by SBCS/DBCS editors due to font limitations. For more information, see [Unicode Limitations](#).

To provide better support for editing Unicode and non-Unicode files, two modes of editing exist: Unicode and SBCS/DBCS mode. Files that contain Unicode, XML, or code page data not compatible with the active code page should be opened as Unicode files.

The following are non-Unicode encodings and put the editor in SBCS/DBCS editing mode: **Default, Text, SBCS/DBCS mode, Binary, SBCS/DBCS mode**, and **EBCDIC, SBCS/DBCS mode**. In addition, the **Auto Unicode, Auto Unicode2, Auto EBCDIC and Unicode**, and **Auto EBCDIC and Unicode2** encodings put the editor into SBCS/DBCS editing mode when the file is determined not to be Unicode. All other encodings put the editor in Unicode mode and require that the file data be converted to UTF-8.

There are many encodings available, including:

- **Auto XML** - This encoding specifies that the file encoding be determined based on XML standards and that the file be loaded as Unicode data. The encoding is determined based on the encoding specified by the `?xml` tag. If the encoding is not specified by the `?xml`, the file data is assumed to be UTF-8 data which is consistent with XML standards. We applied some modifications to the standard XML encoding determination to allow for some user error. If the file has a standard Unicode signature, the Unicode signature is assumed to be correct and the encoding defined by the `?xml` tag is ignored.
- **Auto Unicode** - When this encoding is chosen and the file has a standard Unicode signature, the file is loaded as Unicode data. Otherwise the file is loaded as SBCS/DBCS data.
- **Auto Unicode2** - When this encoding is chosen and the file has a standard Unicode signature or looks like a Unicode File, the file is loaded as Unicode data. Otherwise the file is loaded as SBCS/DBCS data. This option is NOT fool-proof and may give incorrect results.
- **Auto EBCDIC** - When this encoding is chosen and the file looks like an EBCDIC file, the file is loaded as Unicode data. Otherwise, the file is loaded as SBCS/DBCS data. This option is NOT fool-proof and may give incorrect results. The option does attempt to support binary EBCDIC files.
- **Auto EBCDIC and Unicode2** - This encoding is a combination of the Auto EBCDIC and Auto Unicode2 encodings described above.

## Using Unicode

To use encodings in SlickEdit®, Unicode support is required (OEMs typically turn this feature off). Unicode is supported for the following list of features:

- All Context Tagging® features.
- Color Coding.
- Level 1 regular expressions as defined by the Unicode consortium.
- Multi-file search and replace.

- Support for many encodings including UTF-8, UTF-16, UTF-32, and many code pages. Automatic encoding recognition for XML files. Configure encoding recognition per extension or globally. Optionally store signatures and specify little endian or big endian. Use the Save As or Write Selection dialog to convert data to a particular file encoding.
- Support for converting Unicode to UNC data and visa versa. Supported UCN formats include `\xHHHH`, `\x{HHHH}`, `\uHHHH`, `&xHHHH;`, and `&xDDDD;`. This is useful for specifying Unicode character strings in SBCS/DBCS active code page source files. See [Converting Unicode to UCN](#).
- Multiple clipboards.
- Sorting.
- 3-Way Merge.
- Support for composite and surrogate characters.
- Support for storing up to 31-bit Unicode characters.
- SmartPaste®.
- Syntax Expansion and Syntax Indenting.
- Code beautifiers.
- Support for almost all of SlickEdit's SBCS/DBCS active code page features.

## Unicode File Recognition

By default, XML and Unicode files with signatures (UTF-8, UTF-16 and UTF-32) files are automatically loaded as Unicode. If you have Unicode files that are not XML and do not have signatures, configure default options to get the best recognition possible. This is important because some features such as drag/drop files and DIFFzilla® do not prompt you for the file encoding.

Each extension may have its own encoding specification. If the extension-specific encoding is set to Default, then the global setting defined in the File Options dialog (**Tools > Options > File Options**, select the [Load Tab](#)) is used. Both the extension-specific and global setting are overridden if you previously specified an encoding in the Open dialog. The encoding used to override default encoding settings is recorded. The setting is then reused the next time you open the same file. This provides you with per-file encoding support.

If you have non-XML UTF-16 files that have signatures, then try selecting **Auto Unicode2** as an extension-specific or global encoding. Since there is no option for recognizing UTF-8 or UTF-32 files (other than Auto XML) by looking at the file contents, you will either need to set an extension-specific encoding, or specify the encoding in the Open dialog the first time you open the file.

Some compilers (such as Visual C++) let you specify the code page in the source file (in fact, more than one code page can be used in the file). This is not supported, so the assumption is that the file is SBCS/DBCS active code page data.

## Opening Unicode Files

To open a Unicode file, complete the following steps:

1. Use the Open dialog (**File > Open**).
2. Specify the encoding if necessary.
3. Press **Enter**.

## Surrogate Support

Unicode data is stored as UTF-8 and not UTF-16. Since the Windows Win32 calls are used to implement some Unicode features there are some issues. By default, Windows does not support surrogates. You must use the **regedit** program to turn on surrogate support.

To turn on surrogate support, run the **regedit** program and go to the following key location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LanguagePack
```

Set the value for **SURROGATE** to **0x00000002**.

Casing features (uppercase, lowercase, ignore case) do not support surrogates. Windows is used for casing support and Windows casing features do not support surrogates.

## Converting Unicode to UCN

You can convert a selection from Unicode to UCN or vice versa. SlickEdit® conversion features are located on the **Edit > Other** menu. The **Unicode to UCN** conversion feature is most useful for specifying Unicode character strings in SBCS/DBCS active code page source files. For example, here are the steps to store some UCN in a Java source file:

1. Open the Unicode file containing the Unicode characters or create a new Unicode file and enter the characters you want to convert.
2. Select the Unicode characters you want to convert.
3. Execute the **Java/C# (UTF-16 \uHHHH)** menu item (**Edit > Other > Copy Unicode As**).
4. Open the Java source file and paste (**Edit > Paste**) the UCN data into the file.

## Unicode Limitations

The following is a list of Unicode limitations:

- Bold and italics color coding is not supported. Support for this will be added in a future version.
- Tab character operations are not fully supported. Tab display, the **Expand tabs to spaces** save option (**Tools > Options > File Options**, select the [Save Tab](#)), and save with tabs (**save +t**) only work correctly if all the characters are below 128. The **Expand tabs to spaces** load option (**Tools > Options > File Options**, select the [Load Tab](#)) is ignored.
- Column selections do not fully support Unicode. If all the characters are below 128 and the font is fixed then it works. Support for this will be added in a future version.
- Word wrap does not fully support Unicode. If all the characters are below 128 and the font is fixed then it works. Support for this will be added in a future version.
- The Unicode line end character 0x2048 is not supported.
- Hex editing is not supported. The current character (Composite character) is displayed on the status line. Also, use the Open dialog with the Binary, SBCS/DBCS mode encoding to view a Unicode file in hexadecimal.
- Casing features (upper case, lower case, ignore case) do not support surrogates. Windows is relied upon for casing support, and Windows casing features do not support surrogates. See [Surrogate Support](#).
- Vertical line column (**Tools > Options > General**, select the [General Tab](#)) is not supported.
- Truncation line length is not supported.

- Record width on the File Open dialog is not supported.
- DDE is not supported. Unicode DDE does not work with Internet Explorer or Netscape®. You can view files with Unicode data in Internet Explorer; however, this feature will fail if the file name contains characters not in the active code page.
- Version control supports files containing Unicode data but does not support file names that contain characters not in the active code page.
- Special character display is not supported for Unicode buffers.
- The grew program does not support Unicode and can only be used on SBCS/DBCS active code page text.
- If you load the same source file in Unicode and SBCS/DBCS mode, the Context Tagging® database will have incorrect seek positions. It is important to use the default load options and to always load source files in the same encoding so that the Context Tagging seek positions match the editor seek positions.
- The install (`setup.exe`), unionist (`uninstall.exe`), and update (`update.exe`) programs are not Unicode applications so the installation directory must contain characters in the active code page.

## Unicode Implementation

Native Unicode and SBCS/DBCS editing modes are supported. When you edit a SBCS/DBCS (active code page) file such as a `.c`, `.h`, or `.java` file, the data is loaded as SBCS/DBCS data and is not converted to Unicode. When you edit a Unicode file, such as an XML file, the data is converted to UTF-8 that is one of the standard formats for supporting Unicode files. There are several advantages to this implementation:

- Since almost all source files for programming are stored as SBCS/DBCS, loading these files is significantly faster. This is very important to our customers who expect superior performance from SlickEdit®.
- Unicode editing modes cannot support all the features you were used to when editing SBCS/DBCS files (see [Unicode Limitations](#)).
- Macros can be written once to support both editing modes. This was very important to us because we wanted to reduce development time.
- Since Unicode is stored as UTF-8, only one set of binaries is required. Most products that support SBCS/DBCS and Unicode (UTF-16), use preprocessing. This requires two sets of binaries.



## Invocation Options

SlickEdit® can be invoked with different options to control certain system settings. The command line syntax for invoking SlickEdit is as follows:

```
vs {options} file1 {options} file2
```

The **vs** executable is stored in the `win` subdirectory on Windows and the `bin` directory on Linux/UNIX/Mac.

(Windows) To invoke SlickEdit, double-click on the SlickEdit icon created by the install program. However, the command line syntax for invoking SlickEdit is often useful. For example, you might want to set up multiple icons that open different SlickEdit projects, or start the editor from a shell prompt.

The table below shows a list of available invocation options.

Invocation Option	Description
<b>+ or -new</b>	Indicates whether a new instance of the editor should be created or if the existing instance should process the command line parameters. <b>+new</b> creates a new instance. Default is <b>-new</b> .
<b>-sc config_path</b>	Specifies the configuration directory. This directory will be used to find and save configuration files. Sets the VSLICK-CONFIG environment variable to <b>config_path</b> .
<b>-supf kbdfile</b>	(UNIX only) Specifies a keyboard file for mapping the keyboard at the X (modmap) level. Some UNIX systems have keyboard mappings that need to be modified so that keys like Backspace and Delete (and sometimes others) function properly. Use the <b>xmapkeys</b> program to generate the keyboard file.
<b>-sr restore_path</b>	Specifies the directory containing auto-restore files. Sets the VSLICKRESTORE environment variable to <b>restore_path</b> .
<b>-si kmax_var_space max_Nofvars</b>	<b>kmax_var_space</b> specifies the amount of memory in kilobytes allocated for storing the contents of interpreter variables (default is 1000 kilobits). <b>max_Nofvars</b> specifies the maximum number of local, static, and global variables there may be (default is 10000).
<b>-st state_ksize</b>	Specifies the maximum amount of swappable state file data in <code>vslick.sta</code> (UNIX: <code>vslick.stu</code> ) to be kept in memory, in kilobytes. "-1" specifies no limit. "0" specifies that the editor preload all state file data. The default is 200 K.
<b>-sm max_file_size</b>	Specifies the maximum amount of buffer text (in megabytes) that may be edited at one time. Additional memory will automatically be allocated to allow for the editing of up to 2 GB of files. Choosing this option will provide you with better performance.
<b>-suxft</b>	(Linux only) Disables Xft font support. Use this option only if you have problems with fonts under Linux and would like to use non-Xft fonts, like previous versions of SlickEdit. See note below.

Invocation Option	Description
<b>-summ</b> "[x_1 y_1] width_1 height_1, [x_2 y_2] width_2 height_2"	<p>(UNIX only) Specifies multiple monitor configuration. You must specify at least two monitors. By default, SlickEdit tries to automatically detect if you have two monitors. However, this only works if your monitors have the same width, height, and y values. In a left-to-right monitor configuration, <b>x</b> and <b>y</b> are not necessary. The following two examples are equivalent because the monitors are in a left-to-right configuration:</p> <p><b>-summ "1024 768,1024 768"</b>  <b>-summ "0 0 1024 768,1024 0 1024 768"</b></p> <p>Specifying the <b>-summ</b> option in the previous example would not be necessary because it would be automatically detected correctly. However, the following monitor configurations would not be detected correctly:</p> <p><b>-summ "1600 1200,1024 768"</b>  <b>-summ "0 0 1024 768,0 768 1024 768"</b>  <b>-summ "1024 768,1024 768,1024 768"</b></p> <p>The above examples represent the following configurations: left-to-right, 2 monitors; top-to-bottom, 2 monitors; left-to-right, 3 monitors.</p> <p>Note that you can specify this option in the VSLICK environment variable and set it in your <code>vslick.ini</code> file so you don't need to specify this for every invocation. For more information, see <a href="#">Setting Environment Variables in vslick.ini</a>.</p>
<b>-x pcode_name</b>	Alternate state file (.sta) or pcode file (.ex).
<b>-m menu_file</b>	Name of menu resource to use for the SlickEdit menu bar.
<b>-p cmdline</b>	Execute command with arguments given and exit. No other options or file names can be specified after this option since the rest of the command line is assumed to be the program name and space-delimited arguments for this option.
<b>-r cmdline</b>	Execute command with arguments given and remain resident. No other options or file names can be specified after this option since the rest of the command line is assumed to be the program name and space-delimited arguments for this option.
<b>file1 file2</b>	Files to edit. File names may contain wildcard characters ("*" and "?").
<b>-#command</b>	Execute command on active buffer. For example, <b>vs test.c -#bottom-of-buffer</b> places the cursor at the end of <code>test.c</code> . Use double quotes if the command has spaces ( <b>vs test.c "-#goto-col 50"</b> ).
<b>+ or -L[C]</b>	Turn on/off load entire file switch. The optional "C" suffix specifies counting the number of lines in the file.
<b>+nnn</b>	Load binary file(s) that follow with a record width <b>nnn</b> .

Invocation Option	Description
<b>+T [buf_name]</b>	Start a default operating system format temporary buffer with name <b>buf_name</b> .
<b>+TU [buf_name]</b>	Start a UNIX format temporary buffer with name <b>buf_name</b> .
<b>+TM [buf_name]</b>	Start a Macintosh format temporary buffer with name <b>buf_name</b> . Classic Mac line endings are a single carriage return (ASCII 13).
<b>+TD [buf_name]</b>	Start a DOS format temporary buffer with name <b>buf_name</b> .
<b>+ or -E</b>	Turn on/off expand tabs to spaces when loading file. Default is off.



# Environment Variables

Below is a list of environment variables that can be used within SlickEdit®. Configuration environment variables are set in `vslick.ini`.

You can also use the **set** command to temporarily change one of the configuration environment variables or any other environment variable. See [Using the set Command](#) for more information.

**CAUTION** Do not set the VSLICKCONFIG environment variable in `vslick.ini`. VSLICKCONFIG determines where the editor looks for `vslick.ini`. When the editor starts up, it sets the value of environment variables specified in `vslick.ini`. For more information, see [Setting Environment Variables in vslick.ini](#).

Environment Variable	Description
VSLICKRESTORE	Directory to store Auto Restore files.
VSLICKCONFIG	Directory where user's local configuration files are stored. Used in multi-user environments. Defaults to: <ul style="list-style-type: none"> <li>(Windows) <code>.../My Documents/My SlickEdit Config/Editor_Version/</code></li> <li>(Linux, UNIX, Mac) <code>\$HOME/.slickedit/editor_version/</code></li> </ul>
VSLICK	Specifies additional command line arguments to editor as if you were typing them in when invoking the editor.
VSLICKPATH	One or more directories separated with “;” (UNIX: “:”) where batch macros or executable files are searched.
VSLICKMACROS	One or more directories separated with “;” (UNIX: “:”) that contain macro files (*.e). VSLICKPATH must also contain the directories listed here.
VSLICKBIN	One or more directories separated with “;” (UNIX: “:”) that contain binary files. VSLICKPATH must also contain the directories listed here.
VSLICKBITMAPS	One or more directories separated with “;” (UNIX: “:”) that contain bitmap files (*.bmp). VSLICKPATH must also contain the directories listed here.
VSLICKMISC	One or more directories separated with “;” (UNIX: “:”) that contain miscellaneous files including *.vlx, *.slk, *.api, *.idx, vslick.sta (UNIX: vslick.stu), *.hlp, scommon.lst, main.dct, *.pif, *.ini (except for vslick.ini), and *.lst. VSLICKPATH must also contain the directories listed here.
VSLICKALIAS	One or more file names separated with “;” (UNIX: “:”) that contain ALIAS definitions.
VSLICKTAGS	Specifies global tag files. One or more file names separated with “;” (UNIX: “:”) that contain tags. Do not put this environment variable in <code>vslick.ini</code> .
VSLICKBACKUP	Directory to place backup files. Affects <b>+D</b> (default) and <b>-D</b> backup configurations only.

Environment Variable	Description
VSLICKSAVE	Allows save options to be specified per drive.
VSLICKLOAD	Allows load options to be specified per drive.
VSLICKXTERM	(UNIX only) Allows you to specify the default xterm program and arguments used by the <b>dos</b> command and <b>shell</b> function. The complete path to the xterm program must be specified. You may not specify the <b>-e</b> option in the command string. For example, setting VSLICKXTERM to: <b>/usr/X11/bin/xterm -geometry 80x40</b> will create xterm windows with a width of 80 characters and a height of 40 characters.
VSUSER	The license manager handles system crashes better if each user sets the VSUSER environment variable to a unique name.
VST	Specifies additional command line arguments to the macro compiler as if you typed them in when invoking the compiler.
VSLICKXNOBLINK	Suppresses the blinking cursor.
VSLICKXNOPLUSNEWMSG	Suppresses a message when starting a second instance of SlickEdit.

## Setting Environment Variables in vslick.ini

Place configuration environment settings in the file `vslick.ini`. This file is located in the following default directory based on your platform (if it does not exist, it can be created manually):

- Windows: `.../My Documents/My SlickEdit Config/Editor_Version/`
- Linux, UNIX, Mac: `$HOME/.slickedit/editor_version/`

Below is a sample `vslick.ini` file with an environment section.

```
[Environment]
VSLICKPATH=c:\vslick\win;c:\vslick\macros;c:\vslick\bitmaps;c:\vmacro
s
VSLICKALIAS=c:\vmacros\alias.slk
VSLICKINCLUDE=c:\vslick\macros;c:\vmacros
VSLICKLOAD=a: +l b: +l
VSLICKSAVE=a: +o b: +o
MYPROJECTVERSION=c:\myprog4.2
```

When the editor starts, the following environment variables are created by the editor:

- **VSDRIVE** - Drive letter followed by colon where editor executable resides.
- **VSDIR** - Directory of editor executable with trailing backslash (UNIX: slash).

Environment variables can be embedded in any line within a section by placing “%” characters around the environment variable.

## Using the set Command

Change or view the environment while running through the **set** command. The operation of the built-in **set** command is almost identical to the DOS SET command. Use the **set** command to temporarily change one

of the configuration environment variables or any other environment variable. For a complete listing of configuration environment variables, see [Environment Variables](#). The syntax of the **set** command is:

```
set [envvar_name [=value]]
```

When you invoke the **set** command with no parameters, a new buffer is created and the current environment variable settings are inserted. The current value of an individual environment variable may be retrieved by executing the **set** command followed by the name of the environment variable. Specify the name of the environment variable followed by an equal sign and the new value will replace the value of an existing environment variable or assign a value to a new environment variable.

To remove an environment variable, specify the name of the environment variable followed by an equal sign, but omit the *value* parameter (ex. **set classpath=**). The DOS command shell removes environment variables in this way also.

The following steps are a convenient way to change the PATH environment variable:

1. Press **Esc** to toggle the cursor to the command line.
2. Type **set path** and press **Enter**. This will place the current value of the PATH variable on the command line.
3. Edit the current value and press **Enter**.

You can use the above steps to change the value of any other environment variable by specifying a different environment variable name in the second step. The **set** command supports completion on the environment variable name. Typing **set ?** on the command line will give you a selection list of all of the environment variable names.





## Configuration Variables

---

SlickEdit® has many behaviors that are controlled through properties not exposed in the options dialogs. They are set through global configuration macro variables in Slick-C®, using the **set\_var** command. The most commonly used of these variables are listed in the table below.

### Viewing Configuration Variables

To view the complete list of configuration variables, bring up the SlickEdit® command line and type **set\_var def-** (note the hyphen at the end). The completion list will provide the full list of available variables. Use the Help system to look up information on a variable by typing the name of the variable into the Index search field. You can also see a list of variables in the Help under **Macro Functions by Category > Configuration Variables**.

Alternatively, you can use the [Symbols Tool Window](#) to find where the variable is defined in the Slick-C® code. Expand the **Slick-C** folder and then expand the **Global Variables** folder. If Slick-C hasn't already been tagged, type **fp** into the SlickEdit command line. This is an abbreviation of the **find\_proc** command, which will trigger tagging if it hasn't already been done.

### Setting/Changing Configuration Variables

There are two ways to set/change these macro variables:

- From the SlickEdit® menu, choose **Macro > Set Macro Variable** and enter the macro variable in the **Variable** field. The current value of the variable will be shown in the **Value** text box. Click **Edit** to edit this variable, then click **OK** to accept the change.
- From the SlickEdit command line, invoke the **set\_var** command with the macro variable name (for example, **set\_var def\_auto\_linecomment**), then press **Enter** to view the current value. You can edit this value, then press **Enter** to accept the change.

See [Programmable Macros](#) for more information on loading macros and setting variables.

### Table of Configuration Variables

The table below provides a list of the most commonly used configuration variables.

Configuration Variable	Description
<b>def_auto_linecomment</b>	Change to 0 to turn off automatic line comment insertion.
<b>def_binary_ext</b>	Default is <b>.ex .obj .exe .lib</b> . This variable is used by the <b>editflst.e</b> macro for Brief emulation. When the <b>editflst.e</b> macro is loaded, the space-delimited extensions listed by this variable are filtered out by the <b>edit</b> command's completion.

Configuration Variable	Description
<b>def_buflist</b>	<p>Change this variable to find the initial file in Buffer List. The default is 3. This macro variable determines how the <b>list_buffers</b> commands displays the buffer list. By default, the buffer list is sorted and path information is in a separate column to the right of the name. This macro variable is composed with the following flags:</p> <ul style="list-style-type: none"> <li>• SORT_BUFLIST_FLAG - 1</li> <li>• SEPARATE_PATH_FLAG - 2</li> </ul> <p>Add the flags together to select a configuration. Leaving out a flag removes the features. If the buffer list is not sorted, the list will be in the order of the buffer ring.</p> <p>If you set this variable to 1, it will show the full path, which you can order according to path. The default (3) will show an alphabetical list of the files in the left column and the directories in the right column.</p>
<b>def_ctags_flags</b>	<p>This variable is a safeguard against parsing past the end of a proc when the braces mismatch. To have SlickEdit® recognize the second <b>dd</b>, go to <b>Macro &gt; Set Macro Variable</b>, enter <b>def_ctags_flags</b>, and set the value to 10.</p>
<b>def_debug_logging</b>	<p>If you change this value to 1, then run the integrated debugger and let it time out, a <code>vs.log</code> file will be created in your config directory.</p>
<b>def_deselect_copy</b>	<p>Set to 1 in Brief emulation to deselect after a copy.</p>
<b>def_do_block_mode_key</b>	<p>Set this variable's value equal to 0 to stop SlickEdit from inserting characters on every line of a block selection.</p>
<b>def_error_re2</b>	<p>Edit this variable to change from the SlickEdit regular expression used for compile/build errors.</p>
<b>def_filelist_show_dotfiles</b>	<p>Controls the global <b>Show hidden files</b> option on the <a href="#">General Tab</a> of the General Options dialog. On Windows, the default value of this variable is 1; change to 0 to view dot files. On UNIX platforms, the default value is 0; change to 1 to hide dot files.</p>
<b>def_from_cursor</b>	<p>Default is 0. If non-zero, the commands <b>upcase_word</b>, <b>lowcase_word</b>, and <b>cap_word</b> will start case change from the cursor position instead of the beginning of the current word.</p>
<b>def_linewrap</b>	<p>Default is set to 1. If you are at the end of a line that has whitespace only on the line below it (spaces or tabs) and you press Delete, this will bring the whitespace below it up to the end of the line that you are on. When the value is set to 0, if you press Delete while at the end of a line that has whitespace only on the line below it (spaces or tabs), the whitespace is removed entirely—acting as a line delete.</p>
<b>def_linux1_shell</b>	<p>To use an alternate shell, set this variable to the shell that you want to run (for example, <b>/bin/bash -i</b>). This will cause the editor to use your process shell.</p>

Configuration Variable	Description
<b>def_max_filehist</b>	Increases the number of files displayed in the file history of the File menu. Enter the number of files you want to see in the history.
<b>def_max_mffind_output</b>	This variable is set for performance reasons. You can increase the amount of information displayed in the Output tool window during a multi-file search by changing this to your desired setting.
<b>def_max_workspace_hist</b>	Increases the length of the Workspace history list in the Project menu. Enter the number of files you want to see in the history.
<b>def_modal_paste</b>	Default is 0. If non-zero, commands that insert a BLOCK type clipboard will overwrite the destination text if the cursor is in replace mode.
<b>def_plusminus_blocks</b>	When the value is set to true (1), the <b>plusminus</b> command will try to find code blocks to expand or collapse if the cursor is on a line that does not have a +/- bitmap on it. The default is true (1).
<b>def_preplace</b>	Default is 1. If zero, the <b>save</b> command will NOT prompt you if you are inadvertently overwriting a file. For example, if you invoke the command <b>save xyz</b> , and an <b>xyz</b> file already exists, and <b>xyz</b> is not the name of the current buffer, you are prompted by default whether you wish to overwrite the file.
<b>def_rwprompt</b>	Default is 1. Change this to 0 to suppress the pop-up that states: "Do you want to update the read-only attribute of the file on disk."
<b>def_save_macro</b>	Default is 1. Set this variable to 0 if you do not want to be prompted with the Save Macro dialog box after ending macro recording.
<b>def_shift_updown_line_select</b>	Set this value to 1 for Shift+Up or Shift+Down to select the current line.
<b>def_show_makefile_target_menu</b>	This variable can be set to decrease the time that it takes for menus to open. Set to 0 to disable all makefile submenus (such as the Build menu and the Projects tool window). Set to 1 to enable all makefile submenus (this is the default). Set to 2 to enable makefile submenus only in the Projects tool window (the Build menu makefile targets are disabled).
<b>def_switchbuf_cd</b>	Set this variable equal to 1 to change the current working directory to the file that currently has focus in the editor.
<b>def_top_bottom_push_bookmark</b>	Set this variable to 1 to push a bookmark whenever you jump to the top or bottom of the buffer. Note that even when this variable is set, no bookmarks are pushed when using the current buffer as a build window ( <code>.process</code> buffer). The default value is 0.
<b>def_undo_with_cursor</b>	Set this value to 1 to enable the undo of each cursor movement.

Configuration Variable	Description
<b>def_update_context_max_file_size</b>	This variable increases the array size in bytes of a file that is too large. The default size of files that can be processed by Context Tagging® is 4 MB. The size can be lowered by changing this variable and setting it to equal the size that you want (in bytes).
<b>def_vc_advanced_options</b>	Set to this variable to 0 to remove advanced options that decrease performance when using ClearCase version control.
<b>def_vtg_tornado</b>	Set this variable value to 0 to prevent Context Tagging of Tornado files.
<b>def_xml_no_schema_list</b>	To prevent SlickEdit from accessing the Internet to validate and get color coding information from DTD's, add your XML extension to this variable. Set the value to a list of space-delimited extensions that you want excluded for actual schema validation. For example: <b>.xml .xsl .xsd</b> . This will prevent SlickEdit from attempting to connect to the Internet for these extensions.

# Configuration Directories and Files

---

## User Configuration Directory

Your SlickEdit® configuration directory contains configuration files representing the changes you have made through setting editor options, and it preserves the state of SlickEdit by using the state file, [vslick.sta](#).

## Configuration Directory Location

By default, the configuration directory is located in `My Documents/My SlickEdit Config` on Windows, and `$HOME/.slickedit` on UNIX, Linux, and Mac. If you want to use a different directory for your config files, set the [VSLICKCONFIG](#) environment variable or specify the location by using the **-sc** invocation option on the command line (see [Environment Variables](#) or [Invocation Options](#)). You can view the path to the config directory by selecting **Help > About SlickEdit**.

## Backing Up the Configuration Directory

You should make periodic backups of your SlickEdit® configuration directory. If you experience a problem in the editor, you can often solve it by using a saved config directory. SlickEdit Product Support may also ask you to use a default configuration to help debug problems. This is accomplished by backing up your config directory and then deleting its contents.

## Table of User Configuration Files

The table below provides a list of the user configuration files.

User Config File	Description
*.als	A text file that contains user-defined extension-specific aliases.
alias.slk	A text file that contains user-defined global aliases (directory aliases).
ftp.ini (UNIX: uftp.ini)	A text file that contains user-defined FTP configurations.
project.vpe (UNIX: uproject.vpe)	A text file that contains user-defined extension-specific projects.
ubox.ini	A text file that contains user-defined box and line comment styles.
uformat.ini	A text file that contains user-defined beautifier schemes.
uprint.ini	A text file that contains user-defined printing schemes.
uscheme.ini	A text file that contains user-defined color schemes.
user.vlx	A text file that contains color coding changes (keywords, etc.). This file is updated when you close the Color Coding dialog box.
usercpp.h	A text file that contains defines (default preprocessing) for Context Tagging® of C++ and C code.
uservc.slk	A text file that contains user-defined version control systems.
usrprjtemplates.vpt	A text file that contains user-defined project packages.

User Config File	Description
vrestore.slk	A text file that contains auto-restore information. The workspace files also contain auto-restore information, but only for the files/windows previously open.
vslick.ini	A text file that contains a few miscellaneous options. The user-configured backup directory is stored here. In addition, some customizable environment variables for path searching for macros, bitmaps, and binary files are stored here as well.
vslick.sta (UNIX: vslick.stu)	A binary file that contains dialog boxes, menus, macro pcode, key bindings, and all other configuration data not stored in one of the other configuration files. Both user and system configuration information is stored here.
vusrdefs.e (UNIX: vunxdefs.e)	A Slick-C® text file that contains the emulation setting, key bindings, color settings, file extension setup information, and some other miscellaneous options.
vusrobjs.e (UNIX: vunxobjs.e)	A text file that contains user-defined dialog boxes and menus in Slick-C syntax.
vusr*.e (UNIX: vunxs*.e)	A text file that contains system modified dialog boxes and menus. These changes are NOT automatically transferred unless the version encoding matches. For example, vusrsl0e.e.

## System Configuration Files

System configuration files are located in the SlickEdit® installation directory.

Typically, these files are only modified by SlickEdit Inc. or OEM customers. OEM customers might want to modify one of these files to ship a customized version of SlickEdit.

### Table of System Configuration Files

The table below provides a list of the system configuration files.

System Config File	Description
alias.slk	A text file that contains default global aliases (for example, directory aliases).
box.ini	A text file that contains default box and line comment styles. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
format.ini	A text file that contains default beautifier schemes.
print.ini	A text file that contains default printing schemes. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
prjtemplates.vpt	A text file that contains default project packages. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
syscpp.h (UNIX: usyscpp.h)	A text file that contains system-defined default preprocessing for Context Tagging® of C++ and C code.

System Config File	Description
vcsystem.slk (UNIX: uvcsys.slk)	A text file that contains default version control systems. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
vslick.ini	A text file that contains a few miscellaneous options. Some customizable environment variables for path searching for macros, bitmaps, and binary files are stored here as well. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
vslick.sta (UNIX: vslick.stu)	A binary file that contains default dialog boxes, menus, macro pcode, key bindings, and all other configuration data not stored in one of the other configuration files.
vslick.vlx	A text file that contains default color coding lexer definitions.
vsscheme.ini	A text file that contains default color schemes. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.





## File Search Order

---

### Search Order for Configuration Files

Several files are automatically searched for, either immediately when the editor is invoked or during the course of operation. The search order for configuration files such as `vslick.ini`, `vslick.sta`, and `vre-store.slk` is:

- Configuration directory. The configuration directory is defined by the `VSLICKCONFIG` environment variable.
- If `VSLICKCONFIG` is not defined, then `My Documents\My SlickEdit Config` is used.
- Current directory.
- Paths specified in `VSLICKPATH` environment variable.
- Paths specified in `PATH` environment variable.

### Search Order for Executable Files

The search order for executable files, batch macro programs, and miscellaneous files is:

- Current directory.
- Configuration directory. The configuration directory is defined by the `VSLICKCONFIG` environment variable.
- If `VSLICKCONFIG` is not defined, then `My Documents\My SlickEdit Config` is used.
- Paths specified in `VSLICKPATH` environment variable.
- Paths specified in `PATH` environment variable.



## Advanced Help Configuration

---

This section outlines the ways that multiple and alternative help file access can be configured. For a complete listing of the options and settings on the dialogs mentioned in this section, see [Dialogs and Tool Windows](#).

### Accessing Multiple Help Files

The Help system uses help files to determine the help information that is displayed. To add or remove help files from SlickEdit®, from the main menu choose **Help > Configure F1 Index File**. This will invoke the Configure Help Index File dialog box. This dialog box creates an IDX file which is typically used for providing word help on multiple help files that may have duplicate help index entries. If you want your IDX file to be used when you press Ctrl+F1 (**wh** command), make sure the **Word help filename(s)** text box on the General Options dialog (**Tools > Options > General**, [More Tab](#)) specifies this file name.

### Building the Help Index File vslick.idx

1. From the main menu, select **Help > Configure F1 Index File**.
2. Click “Scan” to automatically add help files (.hlp) on the PATH. This should also find Microsoft Visual C++, C++ Builder, and Delphi help files.
3. If scanning did not pick up some help files you want, add the other help directories to the **Help File Path** text box. Click **Scan** again.
4. Click **OK** to build the vslick.idx file.

Usually, you only need one help index file for all the compiler packages. However, you may prefer to separate the help files into groups that you can access on different keys.

### Changing the Default Word Help File(s)

To change the default word help file name(s) for Ctrl+F1 and Ctrl+F2 (**wh** and **wh2** commands), complete the following steps:

1. To set the Word Help Filename, from the main menu, select **Tools > Options > General**. Select the [More Tab](#).
2. In the **Word help filename(s)** text box, specify the help files for the **wh** command (Ctrl+F1), each separated with a plus sign. You can only specify one IDX or MVB file per command.
3. Use a semicolon to indicate the start of the help files for the **wh2** command (Ctrl+F2) and specify its help files separated with a plus sign.
4. You can use another semicolon to indicate the start of the help files for the **wh3** command; however, this command is not bound to a key.

Help files have the extension .mvp, .hlp, or .idx. Do not use a plus sign to specify more than one IDX or MVB file. For example:

- **CtrlF1a.hlp + CtrlF1b.hlp; CtrlF2.hlp**
- **vslick.idx; win31wh.hlp+mscxx.hlp+errors.hlp+vslick.hlp**
- **CtrlF1.idx; CtrlF2.idx**
- **vslick.idx; c:\msdev\help\vcbooks40.mvp**

Environment variables may be embedded in the word help file name(s) string by the notation: **%(envvar-name)**.

## Configuring F1 MSDN Help

You can configure F1 help to display information from Microsoft's MSDN Help system when searching for help in a Microsoft solution or project. Any MSDN library that is installed with Visual Studio 2003 and later installs this help viewer. To enable this configuration, from the main menu choose **Help > Configure F1 MSDN Help**.

**Installed Collections** is a list of the help collections that are installed on your computer. By default, Help searches the collections with the word "combined" in the name, first. It searches in the index list as well as the keywords.

## VLX File and Color Coding

---

For more basic information about using Color Coding, see [Color Coding](#).

To modify the color coding for VLX files, use one of the following methods:

- From the main menu, select **Tools > Options > Color Coding**. Use the Color Coding set-up dialog box.
- Or, modify the `vslick.vlx` file.
- Or, create a new VLX file.

The `vslick.vlx` file defines language-specific coloring support for the following languages:

- Ada
- Assembler
- AWK
- C
- C++
- CFScript
- CICS
- COBOL
- dBASE
- Delphi
- Fortran
- HTML
- Java
- Modula-2
- Pascal
- Perl
- Python
- REXX
- Slick-C®
- VHDL
- Visual Basic .NET

### Modifying the VLX File to Change a Color Definition

To modify an existing language-specific coloring definition, complete the following steps:

1. Edit `vslick.vlx`.
2. Search for one of the section names: CPP, Java, Delphi, Pascal, AWK, REXX, Perl, HTML, Modula-2, AWK, COBOL, Python, CICS, Fortran, Visual Basic .NET, Ada, or Slick-C®.
3. Modify the definition. See below for information on syntax of definitions.
4. Invoke the **load** command from the command line. If the current buffer has a `.vlx` extension, it will be loaded. Otherwise you will be prompted to specify a file name. Specify `vslick.vlx` including path as the file name.

### Creating a Lexer Name and a New VLX File

To create a new lexer name (and thus a new section in the VLX file), complete the steps below. Perform all of the above steps and continue with the following steps.

1. From the main menu, select **Tools > Options > File Extension Setup**. The Extension Options dialog box appears.
2. Select the [Advanced Tab](#).
3. If this lexer definition is for a new extension, create the extension with the **New** button. Otherwise, choose the appropriate extension.
4. Set the **Lexer Name** for the new lexer definition you created.
5. Turn on the **Language Specific** check box.
6. Click **Update** to commit the changes.

Files with a `.vlx` extension are text files that have a syntax similar to a `.ini` file. If the first non-blank character in a line is a semicolon, the line is considered a comment. Each definition of a language starts with a section name (the lexer name) enclosed in square brackets. Within each section are statements that look like `name=value`.

The table below shows the statements that can be used.

Statement	Description
<b>case-sensitive= [Y   N]</b>	Defines the case sensitivity for the language. This statement must be the first or second statement within the section.
<b>idchars=start_id_chars after_id_chars</b>	Defines the characters that are the start of a valid identifier and additional valid characters that may follow. This statement must be the first or second statement within the section. You may use a hyphen ("-") character to specify a range, for example, <b>A-Z</b> specifies uppercase letters. To specify a hyphen or backslash ("\") character as a valid word character, place a backslash before the character.
<b>styles= style</b>	Defines zero or more styles. See <a href="#">Table of style Values</a> below for a list of available styles.
<b>mlcomment= start_symbol end_symbol [nesting] [followedby idchars] [colorname]</b>	Defines a multi-line comment. <i>start_symbol</i> and <i>end_symbol</i> define strings which start and end the comment. Specify <b>nesting</b> if the lexer should look for another occurrence of <i>start_symbol</i> when looking for the end comment symbol. The <b>followedby idchars</b> is used to require certain characters to follow <i>start_symbol</i> . You can use a '-' character to specify a range such as <b>A-Z</b> which specifies uppercase letters. To specify a hyphen ("-") or backslash ("\") character as a valid word character, place a backslash before the character. <b>followedby</b> is ignored when the <b>html</b> style is specified. Currently, <i>start_symbol</i> and <i>end_symbol</i> may not be valid identifiers. No more than four multi-line comments may be defined. <i>colorname</i> can be used to indicate that a different color such as keyword color be used instead of comment color when a match is found. <i>colorname</i> may be <b>keywordcolor</b> , <b>numbercolor</b> , <b>stringcolor</b> , <b>commentcolor</b> , <b>ppkeywordcolor</b> , <b>linenumcolor</b> , <b>symbol1color</b> , <b>symbol2color</b> , <b>symbol3color</b> , or <b>symbol4color</b> .

Statement	Description
<b>mlcomment=</b> <i>start_symbol</i> <i>start_col</i> [ <b>checkfirst</b>   <b>leading</b> ] <i>end_symbol</i> [ <b>lastchar</b> ]	Defines a multi-line comment. This construct was designed to handle comments for the ATLAS language. <i>start_symbol</i> and <i>end_symbol</i> define strings which start and end the comment. <i>start_symbol</i> is only considered the start of a comment if it appears in column <i>start_col</i> . <b>checkfirst</b> specifies that the lexer should check if the line is a comment before determining the color coding of symbols in the line. When the <b>checkfirst</b> option is specified, <i>start_symbol</i> is limited to one character in length. <b>leading</b> specifies that <b>symbol</b> is considered a line comment only if it appears as the first non-blank character. Space or tab characters are considered blanks. Currently, <i>end_symbol</i> may not be a valid identifier. <b>lastchar</b> specifies that <i>end_symbol</i> must appear as the last character on a line to terminate the comment. No more than two multi-line comments may be defined.
<b>mlkeywords=</b> [ <i>keyword</i> ] [ <i>keyword</i> ] ...	Defines keywords for the last mlcomment statement. When one of these keywords follows the <i>start_symbol</i> defined for the last mlcomment statement the keyword color is used to color the comment instead of comment color. Keywords do not have to be valid identifiers. This statement is useful for tag languages like HTML. See the HTML definition in the file <code>vslick.vlx</code> for an example.
<b>keywordattrs=</b> [ <i>mlkeyword</i> ] [ <i>attribute</i> ] [ <i>attribute</i> ] ...	Defines attributes for the <i>mlkeyword</i> specified which belongs to the last mlcomment statement. Currently this statement only supports HTML syntax attributes and requires that the <b>HTML</b> style be specified. For example, <b>keywordattrs=SCRIPT LANGUAGE SRC</b> .
<b>linecomment=</b> [ <i>symbol</i> ] [ <i>col</i>   <i>col+</i>   <i>start_col</i> - <i>end_col</i> ] [ <b>checkfirst</b>   <b>leading</b> ]	Defines a line comment. <i>symbol</i> defines the character(s) which start this line comment. If no column limits are specified, the remainder of the line is considered a comment regardless of where <i>symbol</i> appears. A plus sign (“+”) after a column specifies an unlimited <i>end_col</i> . <b>checkfirst</b> specifies that the lexer should check if the line is a comment before determining the color coding of symbols in the line. When the <b>checkfirst</b> option is specified, <i>symbol</i> is limited to one character in length. If <i>symbol</i> is not specified, all characters will be ignored at or after the column specified (ex. <b>linecomment=73+</b> ). This is useful for Fortran which requires that all characters at or after column 73 be ignored. <b>leading</b> specifies that <b>symbol</b> is considered a line comment only if it appears as the first non-blank character. Space or tab characters are considered blanks.
<b>keywords=</b> [ <i>keyword</i> ] [ <i>keyword</i> ] ...	Defines words that should be displayed in keyword color. Keywords do not have to be valid identifiers.

Statement	Description
<b>cskeywords=</b> [keyword] [keyword] ...	(Case-sensitive keywords) Defines words that should be displayed in keyword color only if found in the case specified. This statement should only be used for languages such as HTML which are case insensitive except for a few words. For other languages, use the <b>case-sensitive</b> and <b>keywords</b> statements. Keywords do not have to be valid identifiers.
<b>ppkeywords=</b> [keyword] [keyword] ...	Defines words that should be displayed in preprocessor color. The first character of a preprocessor keyword must not be a valid identifier. Preprocessing keywords must appear as the first non-blank symbol in the line.
<b>symbol1=</b> [keyword] [keyword] ...	Defines words that should be displayed in <b>symbol1</b> color. Keywords do not have to be valid identifiers.
<b>symbol2=</b> [keyword] [keyword] ...	Defines words that should be displayed in <b>symbol2</b> color. Keywords do not have to be valid identifiers.
<b>symbol3=</b> [keyword] [keyword] ...	Defines words that should be displayed in <b>symbol3</b> color. Keywords do not have to be valid identifiers.
<b>symbol4=</b> [keyword] [keyword] ...	Defines words that should be displayed in <b>symbol4</b> color. Keywords do not have to be valid identifiers.

### Table of *style* Values

The table below describes the **style** values that can be used:

Value of <i>style</i>	Description
<b>linenum</b>	Line numbers may be found as the first non-blank symbol of a line like BASIC.
<b>dqbackslash</b>	Color double-quoted strings. Characters following a backslash in a double quoted string are included in the string (like C).
<b>dqbackslashml</b>	Color double-quoted strings. If a double-quoted string ends in a backslash, it continues the string to the next line (like C).
<b>dqmultiline</b>	Color double-quoted strings. String may span multiple lines.
<b>dqdoubles</b>	Color double-quoted strings. Two double quotes represent one double quote.
<b>dqterminate</b>	Do not color code a double-quoted string until the string is terminated. This style does not support <b>dqmultiline</b> or <b>dqbackslashml</b> .
<b>dqlen1</b>	Color double-quoted strings. Double-quoted strings contain exactly one character.
<b>sqbackslash</b>	Color single-quoted strings. Characters following a backslash in a single-quoted string are included in the string (like C).



Value of <i>style</i>	Description
<b>sqbackslashml</b>	Color single-quoted strings. If a double-quoted string ends in a backslash, it continues the string to the next line.
<b>sqmultiline</b>	Color single-quoted strings. String may span multiple lines.
<b>sqdoubles</b>	Color single-quoted strings. Two consecutive single quotes represent one single quote (like Pascal).
<b>sqterminate</b>	Do not color code a single-quoted string until the string is terminated. This style does not support <b>sqmultiline</b> or <b>sqbackslashml</b> .
<b>sqlen1</b>	Single-quoted strings contain exactly one character (like Ada).
<b>amphhex</b>	Hexadecimal numbers are of the form <b>&amp;Hdddd</b> (like BASIC).
<b>ampoct</b>	Octal numbers are of the form <b>&amp;Odddd</b> (like BASIC).
<b>hexh</b>	Hexadecimal numbers are of the form <b>ddddH</b> (like Intel Assembler).
<b>octo</b>	Octal numbers are of the form <b>ddddO</b> (like Intel Assembler).
<b>octq</b>	Octal numbers are of the form <b>ddddQ</b> (like Intel Assembler).
<b>poundbase</b>	Based numbers are of the form <b>#base#number#exponent</b> (like Ada).
<b>underlineint</b>	Numbers may have underlines between the numbers (like Ada).
<b>xhex</b>	Hexadecimal numbers are of the form <b>0xhhhh</b> (like C).
<b>nonumbers</b>	Do not color code numbers. This style is useful for tag languages like HTML. Using this style with other number color coding styles will produce unpredictable results.
<b>rexxhex</b>	Hexadecimal strings are followed by an upper or lowercase letter X. For example, <b>'414141'X</b> or <b>414141X</b> are REXX-style hexadecimal strings that are both equivalent to the string <b>AAA</b> .
<b>packageimport</b>	Language has Java syntax package and import statement where non-quoted file name follows <b>package</b> and <b>import</b> keyword.
<b>idparenfunction</b>	An identifier followed by an open parenthesis indicates a function (like C++ and Java).
<b>html</b>	Enables HTML syntax embedded languages and attribute coloring.
<b>backslashescapechars</b>	Backslash escapes the character that follows.

Value of <i>style</i>	Description
<b>heredocument</b>	Enables support for <code>Here</code> documents. Note that if you prefix the terminator with one of the lexer names you will get embedded language color coding.
<b>perl</b>	Adds support for Perl <b>format</b> statement and some other Perl specific changes.
<b>tcl</b>	Special support for TCL language color coding.
<b>bquote</b>	Perl and Linux Shell style backquote (subshell).
<b>model204</b>	Special support for Model 204 language.
<b>cics</b>	Special support for CICS embedded in COBOL.
<b>python</b>	Special support for Python.

## Editing the Key Binding Source

---

If you are creating a new emulation or if you change many key bindings, you might want to edit your key binding source instead of using the [Key Bindings Dialog](#). To create a Slick-C® batch macro containing your current key bindings, enter the command **list\_source** on the command line. One of the files generated by this command is `vusrdefs.e`. It is placed in your configuration directory if the `VSLICKCONFIG` environment variable is set (defaults to `$HOME/.vslick` under UNIX). Otherwise, it is placed in your `macros` directory. If you open this file (Ctrl+O), the first part of the source code is your key binding, which look like the following:

- `defeventtab default_keys`
- `def 'A-a'-'A-z' =`
- `def 'A-F6' =`
- `def 'F10' =`
- `def 'C-A' = select_all`
- `def 'C-B' = select_block`
- `def 'C-C' = copy_to_clipboard`
- `def 'C-D' = gui_cd`

The **default\_keys** are the key bindings that are active in fundamental mode. The other event tables defined by the **defeventtab** primitive are mode event tables containing key bindings which override the fundamental mode key bindings. Make changes to this buffer by adding or modifying the `def keyname =` command lines and then save the buffer by pressing Ctrl+S. The valid key names are listed in the Help system under “Event Names.” You can also list the key names of the keys through the Help by invoking the command **help Event Names**. To run this batch program, type the name **vusrdefs** without the extension on the command line. The path is not necessary if it is included in your `VSLICKPATH` or `PATH` environment variable.



## Using the ISPF and XEDIT Emulations

This information describes the features of the ISPF editor emulation, as well as outlining some XEDIT line commands.

### ISPF Options Dialog

The ISPF Options dialog is used to tune various ISPF emulation options. When you are in ISPF emulation, you can access this dialog from the main menu by choosing **Tools > Options > ISPF Options**. The following settings are available:

- **Prefix area width** - The number of characters to display in the prefix area (default is 6). Note that some line commands require four characters (e.g. BNDS, TABS, COLS, MASK). To completely remove the prefix area, set the prefix area width to 0.

Only the following line commands are allowed in read-only mode:

<b>ISPF Line Labels</b>	Define a label
<b>ISPF Line Command BNDS</b>	Insert a column boundary ruler line
<b>ISPF Line Command COLS</b>	Insert a column ruler line
<b>ISPF Line Command First</b>	Expose one or more lines at the beginning of a block of excluded lines
<b>ISPF Line Command Last</b>	Expose one or more lines at the beginning of a block of excluded lines
<b>ISPF Line Command Show</b>	Expose one or more lines having the left-most indentation level in a block of excluded lines
<b>ISPF Line Command TABS</b>	Displays the tab definition line
<b>ISPF Line Command Exclude</b>	Specifies one or more lines to be hidden (excluded)
<b>ISPF Line Command Select</b>	Select a block of lines

- **Enter places cursor in prefix area** - When on, the Enter key places the cursor in the prefix area of the next line. When off, the Enter key places the cursor in column 1 of the next line.
- **Right CTRL = Enter/Send** - When this option is enabled, the Enter key places the cursor at the beginning of the next line, and the Right Ctrl key is used to execute line commands. When off, the Right Ctrl key acts like a normal control key and the Enter key is used to execute line commands.
- **Cursor page up/down** - When on, the display is scrolled up/down until the line the cursor is on becomes the last/first line displayed, respectively. If the cursor is already on the top/bottom display line, the display is scrolled one page. When disabled, page up/down always scrolls one page.
- **END command saves the file** - When this option is enabled, changes to the buffer are saved automatically when the **ispf\_end** (F3) command is performed. Otherwise, you will be prompted if you want to save changes before closing the file.
- **XEDIT line commands** - When on, the prefix area will support XEDIT-style line commands.
- **Home places cursor on command line** - When on, the home key places the cursor on the command line. By default, this option is off, and the home key simply moves the cursor to the beginning of the line.

Further ISPF related options are available on the [General Tab](#) of the Extension Options dialog box (**Tools > Options > File Extension Setup**). These options include auto CAPS mode, edit boundaries, and the truncation column.

## ISPF Primary Commands

The following standard ISPF primary commands are supported in the ISPF emulation mode. Primary commands are entered by placing the cursor on the command line, (see [ISPF Command Line and Text Box Editing](#) for details about command line editing features).

To place the cursor on the command line, either hit the Esc key, click on the message bar, or use **ispf\_retrieve** (F12). If configured to do so, the Home key will also place the cursor on the command line. Once on the command line, you may use the cursor up/down keys to retrieve the previous/last command entered, respectively.

Though primary commands may be typed at the command line explicitly, for convenience you can simply type the last part of the command name in the command line and it will automatically be mapped to the ISPF specific command. For example to execute the ISPF reset command, simply type **reset** at the command line instead of **ispf\_reset**.

Note that some standard built-in commands conflict with ISPF emulation commands. These conflicts include: copy, cut, delete, find, hex, move, and paste. To access the built-in command, you may be able to use a menu option or consult the help for that command for specific instructions.

<b>ispf_autosave</b>	Turn on or off prompting to save changes.
<b>ispf_bounds</b>	Set or reset the left and right edit boundaries.
<b>ispf_bnds</b>	Set or reset the left and right edit boundaries
<b>ispf_browse</b>	Browse a data set or member.
<b>ispf_cancel</b>	Closes the current file or PDS member without saving changes.
<b>ispf_caps</b>	Turn on or off automatic capitalization mode.
<b>ispf_change</b>	Replace one string with another within the current buffer.
<b>ispf_chg</b>	Replace one string with another within the current buffer.
<b>ispf_compare</b>	Compares the file you are editing with another file.
<b>ispf_copy</b>	Inserts the contents of a file or PDS member into the buffer.
<b>ispf_create</b>	Create a new file or PDS member containing the contents of the buffer.
<b>ispf_cut</b>	Cut lines out of the current buffer and place them in the clipboard.
<b>ispf_delete</b>	Deletes lines in the given line range, or the entire buffer.

<b>ispf_edit</b>	This command is identical to the built-in <b>edit</b> command.
<b>ispf_end</b>	Closes the current file.
<b>ispf_exclude</b>	Hides (excludes) lines that match the given search string.
<b>ispf_find</b>	Find occurrences of the given search string in the current buffer.
<b>ispf_flip</b>	Reverses the exclude status of lines.
<b>ispf_hex</b>	Toggles display of the document in hexadecimal mode.
<b>ispf_hilite</b>	Specify the use of color coding in the editor.
<b>ispf_locate</b>	Find lines with a specific line prefix.
<b>ispf_move</b>	Moves the contents of a file or PDS member into the buffer.
<b>ispf_nonumber</b>	Turns off numbering mode.
<b>ispf_number</b>	Controls line numbering mode. Unlike ISPF, this command does effect how lines are inserted.
<b>ispf_paste</b>	Copies lines from the clipboard to the buffer.
<b>ispf_preserve</b>	Controls saving of trailing blanks.
<b>ispf_rchange</b>	Repeats the change requested by the most recent change command.
<b>ispf_renumber</b>	Immediately updates the line numbers in a file.
<b>ispf_replace</b>	Save to contents of the current buffer to an existing file.
<b>ispf_reset</b>	Resets the contents of the line prefix area.
<b>ispf_return</b>	Closes the current file.
<b>ispf_rfind</b>	Repeat the last find operation requested.
<b>ispf_save</b>	This command is identical to the built-in <b>save</b> command.
<b>ispf_sort</b>	Sorts lines of data in a specified order.
<b>ispf_submit</b>	Submit the contents of the current buffer for batch processing.
<b>ispf_swap</b>	Switches to the next buffer.
<b>ispf_tabs</b>	Define logical tab positions.
<b>ispf_unnumber</b>	Blanks out the line numbers in a file.
<b>ispf_undo</b>	This command is identical to the undo command.

## ISPF Line Commands

The following ISPF edit line commands are supported in the ISPF emulation mode:

- Enter line commands by typing over the prefix area (on the left hand side of the editor control) which contains either ===== or the line number. To place the cursor in the prefix area, click there, cursor left, or back tab until the cursor is in the prefix area. In addition, enter will place the cursor in the prefix area of the next line, unless an insert or text entry command is executed.
- Edit line commands operate on either a single line or a block of lines. The commands that operate on blocks require you to place the command on both the first and last lines of the block.
- Line commands are processed using the **ispf\_do\_lc** command when you press Enter, Ctrl+Enter or the right control key, depending on your preferences. Several commands or line labels can be entered and then processed at one time. The **ispf\_reset** command is used to clear the prefix area.

<b>ISPF Line Labels</b>	Define a label
<b>ISPF Line Command Shift</b>	Shift data left or right
<b>ISPF Line Command A</b>	Identifies a line after which lines are to be inserted
<b>ISPF Line Command B</b>	Identifies a line before which lines are to be inserted
<b>ISPF Line Command BNDS</b>	Insert a column boundary ruler line
<b>ISPF Line Command Copy S</b>	Specify lines to be copied to another location
<b>ISPF Line Command COL</b>	Insert a column ruler line
<b>ISPF Line Command Delete</b>	Deletes one or more lines
<b>ISPF Line Command First</b>	Expose one or more lines at the beginning of a block of excluded lines
<b>ISPF Line Command I</b>	Insert one or more blank data entry lines
<b>ISPF Line Command Lower-case</b>	Converts all uppercase letter alphabetic characters in one or more lines to lowercase
<b>ISPF Line Command Last</b>	Expose one or more lines at the beginning of a block of excluded lines
<b>ISPF Line Command Move</b>	Specify lines to be moved to another location
<b>ISPF Line Command MASK</b>	Displays the contents of the mask used with the insert (I) and text entry (TE) line commands
<b>ISPF Line Command Make Data</b>	Converts one or more no-save lines to data so that they may be saved when the buffer is saved
<b>ISPF Line Command Overlay</b>	Identifies one or more lines over which the copy or move block is to be overlaid
<b>ISPF Line Command Repeat</b>	Specify lines to be repeated immediately following this line or block



<b>ISPF Line Command Show</b>	Expose one or more lines having the left-most indentation level in a block of excluded lines
<b>ISPF Line Command TABS</b>	Displays the tab definition line
<b>ISPF Line Command TE</b>	Inserts one or more blank lines to allow power typing for text entry
<b>ISPF Line Command TF</b>	Reflows paragraphs according to the current column boundary settings
<b>ISPF Line Command TJ</b>	Join this line with the next line
<b>ISPF Line Command TS</b>	Divides a line so that data can be added
<b>ISPF Line Command Upper-case</b>	Converts all lowercase letter alphabetic characters in one or more lines to upper-case
<b>ISPF Line Command Exclude</b>	Specifies one or more lines to be hidden (excluded)
<b>ISPF Line Command Select</b>	Select a block of lines

## ISPF Line Labels *.label*

**Usage** *.label*, where *label* does not start with 'z'

**Remarks** Define a label to be used as a marker to identify the given line. Labels are used to specify a particular line, such as in the **ispf\_locate** command, or to specify a range of lines for an primary command to operate on. The following labels are built in to the ISPF emulation.

<b>.zfirst</b>	The first line in the buffer (abbreviated <b>.zf</b> ).
<b>.zlast</b>	The last line in the buffer (abbreviated <b>.zl</b> ).
<b>.zcsr</b>	The current line the cursor is on (abbreviated <b>.zc</b> ).

See Also

**ispf\_change**, **ispf\_copy**, **ispf\_delete**, **ispf\_exclude**, **ispf\_find**, **ispf\_flip**, **ispf\_locate**, **ispf\_paste**, **ispf\_reset**, **ispf\_sort**

## ISPF Shift Lines Left or Right

**Usage** ( *[n]* ) Shift the current line *n* columns left, default 2

<b>(( <i>[n]</i> )</b>	Shift the block of lines <i>n</i> columns left, default 2
<b>) <i>[n]</i> )</b>	Shift the current line <i>n</i> columns right, default 2
<b>)) <i>[n]</i> )</b>	Shift the block of lines <i>n</i> columns right, default 2

< [n]	Data shift the current line <i>n</i> columns left, default 2
<< [n]	Data shift the block of lines <i>n</i> columns left, default 2
> [n]	Data shift the current line <i>n</i> columns right, default 2
>> [n]	Data shift the block of lines <i>n</i> columns right, default 2

**Remarks** This set of commands are used for shifting data left or right. The versions using parenthesis shift text literally, while the other versions attempts to intelligently shift text without disturbing line numbers or comments. In all cases, the default number of columns that the text is shifted is two.

There are two forms to these commands. The single character forms ( , ), <, or > specifies that the line and the subsequent *n-1* lines are to be shifted. The two character block forms are placed on the first and last lines of the block to be shifted.

Data is shifted only within the columns defined by the current bounds, or if bounds is turned off, but there is a truncation column, between column 1 and the truncation column. If the shift operation results in data moving beyond the right or left margins, it is truncated and there is no warning message.

See Also

**ispf\_bounds**

## ISPF Insert After A

*Usage* A [n]

**Remarks** Identifies a line after which copied or moved lines are to be inserted *n* times. You are allowed to specify multiple **A**, **B**, or **O** line commands to have the same copy or move block inserted in multiple places.

See Also

**ispf\_copy**, **ispf\_paste**, **ISPF Line Command B**, **ISPF Line Command Copy**, **ISPF Line Command Move**, **ISPF Line Command Overlay**

## ISPF Insert Before B

*Usage* B [n]

**Remarks** Identifies a line before which copied or moved lines are to be inserted *n* times. You are allowed to specify multiple **A**, **B**, or **O** line commands to have the same copy or move block inserted in multiple places.

See Also

**ispf\_copy**, **ispf\_paste**, **ISPF Line Command B**, **ISPF Line Command Copy**, **ISPF Line Command Move**, **ISPF Line Command Overlay**

## ISPF Insert Bounds Ruler BNDS

*Usage* BNDS

**Remarks** Insert a column boundary ruler line. After this line is inserted, the < and > marks may be moved in order to adjust the column boundaries. Note that if you have multiple bounds lines, and you change one, the subsequent bounds lines will also be changed.

A column boundary line with one < sign indicates a left boundary and no right boundary (unbounded). A column boundary with one > sign indicates a single column boundary (left and right bounds are same).

See Also

**ispf\_bounds, ISPF Line Command Shift, ISPF Line Command Overlay**

## ISPF Copy Lines C and CC for blocks

**Usage** **C [n]** Copy *n* lines starting with the line with the command.

**CC** Copy a block of lines, must match another **CC**.

**Remarks** Specify lines to be copied to another location. There are two forms to this command, the first form (**C [n]**) specifies that the line and the subsequent *n-1* lines are to be copied. The second (block) form (**CC**) is placed on the first and last lines of the block to be copied. There can be only one copy block specified. Furthermore, you can not have both a move block and a copy block specified at the same time.

See Also

**ISPF Line Command A, ISPF Line Command B, ISPF Line Command Move, ISPF Line Command Overlay**

## ISPF Insert Columns Ruler COLS or SCALE

**Usage** COLS

SCALE

**Remarks** Insert a column ruler line. The column ruler line is read-only.

See Also

**ispf\_bounds, ispf\_tabs, ISPF Line Command BNDS, ISPF Line Command TABS**

## ISPF Delete Lines D and DD for blocks

**Usage** **D [n]** Delete *n* lines starting with the line with the command.

**DD** Delete a block of lines, must match another **DD**.

**Remarks** Deletes one or more lines. There are two forms to this command. The first form (**D [n]**) specifies that the line and the subsequent *n-1* lines are to be deleted. The second (block) form (**DD**) is placed on the first and last lines of the block to be deleted.

See Also

**ispf\_delete**

## ISPF Expose First Lines F and FF

**Usage** **F [n]** Unexclude (expose) the first *n* lines of an excluded block.

**FF** Unexclude (expose) an entire excluded block.

**Remarks** Expose one or more lines at the beginning of a block of excluded lines. The **FF** line command exposes the entire block of lines and is to **F[m]** where *m* is the number of lines in the block of excluded lines.

See Also

**ispf\_exclude, ispf\_reset, ISPF Line Command Last, ISPF Line Command Show, ISPF Line Command Exclude**

## ISPF Insert Lines

*Usage* I [n]

**Remarks** Insert one or more blank data entry lines.

See Also

**ispf\_enter, ISPF Line Command TE**

## ISPF Lowercase Lines LC, LCC and LCLC for blocks

*Usage* LC [n] Lowercase n lines starting with the line with the command.

**LCC** Lowercase a block of lines, must match another **LCC** or **LCLC**.

**LCLC** Lowercase a block of lines, must match another **LCC** or **LCLC**.

**Remarks** Converts all uppercase letter alphabetic characters in one or more lines to lowercase. This command only operates on text within the edit boundary columns. There are two forms to this command. The first form (**LC [n]**) specifies that the line and the subsequent n-1 lines are to be converted. The second (block) form (**LCLC** or **LCC**) is placed on the first and last lines of the block to be converted.

See Also

**ispf\_caps, ISPF Line Command Uppercase, lowercase, upcase**

## ISPF Expose Last Lines L and LL

*Usage* L [n] Unexclude (expose) the last n lines of an excluded block.

**LL** Unexclude (expose) an entire excluded block (identical to **FF**).

**Remarks** Expose one or more lines at the end of a block of excluded lines. The **LL** line command exposes the entire block of lines and is to **L[m]** where m is the number of lines in the block of excluded lines.

See Also

**ispf\_exclude, ispf\_reset, ISPF Line Command First, ISPF Line Command Show, ISPF Line Command Exclude**

## ISPF Move Lines M and MM for blocks

*Usage* M [n] Move n lines starting with the line with the command.

**MM** Move a block of lines, must match another **MM**.

**Remarks** Specify lines to be moved to another location. There are two forms to this command, the first form (**M [n]**) specifies that the line and the subsequent n-1 lines are to be moved. The second (block) form (**MM**) is placed on the first and last lines of the block to be moved. There can be only one move block specified. Furthermore, you can not have both a move block and a copy block specified at the same time.

See Also

**ISPF Line Command A, ISPF Line Command B, ISPF Line Command Copy, ISPF Line Command Overlay**

## ISPF Insert Mask Line MASK

*Usage* MASK

**Remarks** Displays the contents of the mask used with the insert (**I**) and text entry (**TE**) line commands. Normally, when a line is inserted, the line is initially blank. By specifying an insert mask, you can insert a block of lines with a particular template. The **MASK** line is editable. Note that if you specify multiple masks in one file, only the first mask is used.

See Also

ISPF Line Command I, ISPF Line Command TE, ISPF Line Command TS

## ISPF Make Data Lines MD, MDD and MDMD for blocks

**Usage** MD [*n*] Make *n* data lines starting with the line with the command.

**MDD** Make a block of lines data, must match another **MDD** or **MDMD**.

**MDMD** Make a block of lines data, must match another **MDD** or **MDMD**.

**Remarks** Converts one or more no-save lines to data so that they may be saved when the buffer is saved. There are two forms to this command. The first form (**MD [*n*]**) specifies that the line and the subsequent *n*-1 lines are to be converted. The second (block) form (**MDMD** or **MDD**) is placed on the first and last lines of the block to be converted.

See Also

ISPF Line Commands, ISPF Line Command COLS, ISPF Line Command BNDS, ISPF Line Command MASK, ISPF Line Command TABS

## ISPF Overlay Lines O and OO for blocks

**Usage** O [*n*] Overlay *n* lines starting with the line with the command.

**OO** Overlay a block of lines, must match another **OO**.

**Remarks** Identifies one or more lines over which the copy or move block is to be overlaid. Text is only overlaid within the column boundaries. If the copy or move block has less lines than the overlay, it is repeated until it fills the entire overlay block.

There are two forms to this command. The first form (**O [*n*]**) specifies that the line and the subsequent *n*-1 lines are to be overlaid. The second (block) form (**OO**) is placed on the first and last lines of the block to be overlaid.

You are allowed to specify multiple **A**, **B**, or **O** line commands to have the same copy or move block inserted or overlaid in multiple places.

See Also

ispf\_copy, ispf\_paste, ISPF Line Command A, ISPF Line Command B, ISPF Line Command Copy, ISPF Line Command Move, ISPF Line Command Overlay

## ISPF Repeat Lines

**Usage** R [*n*] Repeat the line with the command *n* times.

**RR [*n*]** Repeat the block *n* times, must match another **RR**.

**Remarks** Specify lines to be repeated immediately following this line or block. There are two forms to this command, the first form (**R[n]**) specifies that the line is to be repeated *n* times. The second (block) form (**RR[n]**) is placed on the first and last lines of the block to be repeated *n* times.

See Also

**ISPF Line Command A, ISPF Line Command B, ISPF Line Command Copy**

## ISPF Expose Next Level of Code S and SS

**Usage** **S [n]** Unexclude (expose) the first *n* lines of an excluded block.

**SS** Unexclude (expose) an entire excluded block.

**Remarks** Expose one or more lines having the leftmost indentation level in a block of excluded lines. The **SS** line command exposes the entire block of lines and is to **S[m]** where *m* is the number of lines in the block of excluded lines.

See Also

**ispf\_exclude, ispf\_reset, ISPF Line Command First, ISPF Line Command Last, ISPF Line Command Exclude**

## ISPF Insert Tabs Ruler TABS or TABL

**Usage** TABS

TABL

**Remarks** Displays the tab definition line. After this line is inserted, the \* marks may be moved in order to adjust the tab positions. Note that if you have multiple tabs lines, and you change one, the subsequent tabs lines will also be changed.

See Also

**ispf\_tabs, tabs**

## ISPF Insert Text TE

**Usage** TE [n]

**Remarks** Inserts one or more blank lines to allow power typing for text entry. This command is identical to the insert (I) command, except that it switches the mode to wrap lines.

See Also

**ispf\_enter, ISPF Line Command I, ISPF Line Command MASK**

## ISPF Insert Lines TF

**Usage** TF

**Remarks** Reflows paragraphs according to the current column boundary settings.

See Also

**reflow\_paragraph**

## ISPF Join Lines TJ

**Usage** TJ

**Remarks** Join this line with the next line.

See Also

ISPF Line Command TS, [join\\_line](#)

## ISPF Split Line TS

*Usage* TS

**Remarks** Divides a line so that data can be added. The line is split at the column in which the cursor is in when you hit Enter. This command does not support multiple lines.

See Also

ISPF Line Command TJ, [split\\_insert\\_line](#)

## ISPF Uppercase Lines UC, UCC and UCUC for blocks

**Usage** UC [*n*] Uppercase *n* lines starting with the line with the command.

**UCC** Uppercase a block of lines, must match another **UCC** or **UCC**.

**UCUC** Uppercase a block of lines, must match another **UCC** or **UCUC**.

**Remarks** Converts all lowercase letter alphabetic characters in one or more lines to uppercase. This command only operates on text within the edit boundary columns. There are two forms to this command. The first form (**UC [*n*]**) specifies that the line and the subsequent *n*-1 lines are to be converted. The second (block) form (**UCUC** or **UCC**) is placed on the first and last lines of the block to be converted.

See Also

[ispf\\_caps](#), ISPF Line Command Lowercase, lowercase, upcase

## ISPF Exclude Lines X and XX for blocks

**Usage** X [*n*] Exclude *n* lines starting with the line with the command.

**XX** Exclude a block of lines, must match another **XX**.

**Remarks** Specifies one or more lines to be hidden (excluded). There are two forms to this command. The first form (**X [*n*]**) specifies that the line and the subsequent *n*-1 lines are to be excluded. The second (block) form (**XX**) is placed on the first and last lines of the block to be excluded.

See Also

[ispf\\_exclude](#), [ispf\\_reset](#), ISPF Line Command First, ISPF Line Command Last, ISPF Line Command Show

## ISPF Select Lines Z and ZZ for blocks

**Usage** Z [*n*] Select *n* lines starting with the line with the command.

**ZZ** Select a block of lines, must match another **ZZ**.

**Remarks** Select a block of lines. There are two forms to this command. The first form (**Z [*n*]**) specifies that the line and the subsequent *n*-1 lines are to be selected. The second (block) form (**ZZ**) is placed on the first and last lines of the block to be selected.

See Also

[ispf\\_cut](#), [ispf\\_paste](#)

## XEDIT Line Commands

The following XEDIT line commands are supported and override the like-named ISPF commands when there is a conflict. XEDIT commands can be enabled using the **ISPF Options dialog box**.

XEDIT	ISPF	Description
/	R	Repeat the marked line
	R	Repeat the marked line
F	A	Paste text following line
A	I	Add (insert) line(s)
P	B	Paste text before line
L	LC	Make line lowercase
LL	LCC	Make block lowercase
U	UC	Make line uppercase
UU	UCC	Make block uppercase

Note the following conflicts with standard ISPF edit line commands:

- **F** conflicts with unexclude first (F).
- **A** conflicts with paste after (A).
- **L** conflicts with unexclude last (L).
- **LL** conflicts with unexclude block (LL).

## ISPF Unsupported Primary Commands

The following ISPF primary commands are not supported by the ISPF Emulation mode. The unsupported commands fall into three categories. First some ISPF commands are made obsolete by more powerful features, such as recovery, profile, and setundo. Second, some commands reflect features that we chose not to implement for the emulation, such as ISPF macros, PDF statistics, model, and pack.



## ISPF Emulation Commands

<b>autolist</b>	Controls the automatic printing of data to the ISPF list data set.
<b>builtin</b>	Process a built-in command, even if overloaded by a macro.
<b>define</b>	Define a name as an alias or macro.
<b>imacro</b>	Saves the name of an initial macro in the edit profile.
<b>level</b>	Sets the modification level number in PDF library statistics.
<b>model</b>	Copies a model into the buffer or defines a model class.
<b>notes</b>	Controls whether the MODEL command display notes or not.
<b>nulls</b>	Controls null spaces.
<b>pack</b>	Controls whether data is to be stored compressed or not.
<b>profile</b>	Display edit profile.
<b>recovery</b>	Specifies edit recovery options.
<b>rmacro</b>	Saves a recovery macro in the edit profile.
<b>setundo</b>	Controls the UNDO mode.
<b>stats</b>	Generate library statistics.
<b>version</b>	Sets the version number in the PDF library statistics.
<b>view</b>	Save as browse command but prompts on save.

The following commands are supported in ISPF emulation mode.

<b>ispf_bottom</b>	Moves cursor to the end of the buffer
<b>ispf_down</b>	Moves cursor to next page of text
<b>ispf_enter</b>	Handle the Enter key or right-control key in ISPF emulation
<b>ispf_home</b>	Places the focus on the command line in ISPF emulation
<b>ispf_retrieve</b>	Does command line retrieval, getting the next command line from the list
<b>ispf_retrieve_back</b>	Identical to the ispf_retrieve back command
<b>ispf_top</b>	Moves cursor up to the top of the buffer
<b>ispf_up</b>	Moves cursor up to the previous page of text
<b>ispf_do_lc</b>	Immediately process all commands found in the line prefix area



# Regular Expression Syntax

This section provides lists of the UNIX, SlickEdit®, and Brief regular expression syntaxes, samples, and Unicode category specifications.

## UNIX Regular Expressions

UNIX regular expressions are defined in the following table.

UNIX Regular Expression	Definition
<code>^</code>	Matches beginning of line.
<code>\$</code>	Matches end of line.
<code>.</code>	Matches any character except newline.
<code>X+</code>	Maximal match of one or more occurrences of X. See <a href="#">Minimal versus Maximal Matching</a> .
<code>X*</code>	Maximal match of zero or more occurrences of X.
<code>X?</code>	Maximal match of zero or one occurrences of X.
<code>X{n1}</code>	Match exactly <i>n1</i> occurrences of X.
<code>X{n1,}</code>	Maximal match of at least <i>n1</i> occurrences of X.
<code>X{,n2}</code>	Maximal match of at least 0 occurrences but not more than <i>n2</i> occurrences of X.
<code>X{n1,n2}</code>	Maximal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
<code>X+?</code>	Minimal match of one or more occurrences of X.
<code>X*?</code>	Minimal match of zero or more occurrences of X.
<code>X??</code>	Minimal match of zero or one occurrences of X.
<code>X{n1}? </code>	Matches exactly <i>n1</i> occurrences of X.
<code>X{n1,}? </code>	Minimal match of at least <i>n1</i> occurrences of X.
<code>X{,n2}? </code>	Minimal match of at least 0 occurrences but not more than <i>n2</i> occurrences of X.
<code>X{n1,n2}? </code>	Minimal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
<code>(?!X)</code>	Search fails if expression X is matched. The expression <code>^(?!if)</code> matches the beginning of all lines that do not start with "if".
<code>(X)</code>	Matches sub-expression X and specifies a new tagged expression (see <a href="#">Using Tagged Search Expressions</a> ). No more tagged expressions are defined once an explicit tagged expression number is specified as shown below.
<code>(?dX)</code>	Matches sub-expression X and specifies to use tagged expression number <i>d</i> where $0 \leq d \leq 9$ . No more tagged expressions are defined by the sub-expression syntax (X) once this sub-expression syntax is used. This is the best way to make sure you have enough tagged expressions.
<code>(?:X)</code>	Matches sub-expression X but does not define a tagged expression.

UNIX Regular Expression	Definition
X Y	Matches X or Y.
[ <i>char-set</i> ]	Matches any one of the characters specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. '\' may be used inside the square brackets to define literal characters or define ASCII characters. For example, \- specifies a literal dash character. The expression [\d0-\d27] matches ASCII character codes 0..27. The expression [] matches a right bracket. In SlickEdit® regular expressions, [] matches no characters. In both syntaxes, the expression [)] matches a right bracket. The expression [^] matches a '^' character but this does not work for SlickEdit regular expressions. In both syntaxes, [^] matches a '^' character.
[^ <i>char-set</i> ]	Matches any character not specified by <i>char-set</i> . A '-' character may be used to specify ranges.
[ <i>char-set1</i> - [ <i>char-set2</i> ]]	Character set subtraction. Matches all characters in <i>char-set1</i> except the characters in <i>char-set2</i> . The expression [^A-Z] matches all characters except uppercase letters. For example, "[a-z-[qw]]" matches all English lowercase letters except 'q' and 'w'. "[\p{L}-[qw]]" matches all Unicode lowercase letters except 'q' and 'w'.
[ <i>char-set1</i> & [ <i>char-set2</i> ]]	Character set intersection. Matches all characters in <i>char-set1</i> that are also in <i>char-set2</i> . For example, "[\x{0}-\x{7f}&[\p{L}]]" matches all letters between 0 and 127.
\x{ <i>hhhh</i> }	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhhh</i> .
\p{ <i>UnicodeCategorySpec</i> }	(Only valid in character set) Matches characters in <i>UnicodeCategorySpec</i> . Where <i>UnicodeCategorySpec</i> uses the standard general categories specified by the Unicode consortium. For example, "[\p{L}]" matches all letters. "[\p{Lu}]" matches all uppercase letters. See "Unicode Category Specifications for Regular Expressions" on page 706.
\P{ <i>UnicodeCategorySpec</i> }	(Only valid in character set) Matches characters not in <i>UnicodeCategorySpec</i> . For example, "[\P{L}]" matches all characters that are not letters. This is equivalent to "[^\p{L}]". "[\P{Lu}]" matches all characters that are not uppercase letters. See <a href="#">Unicode Category Specifications for Regular Expressions</a> .
\p{ <i>UnicodeIsBlockSpec</i> }	(Only valid in character set) Matches characters in <i>UnicodeIsBlockSpec</i> . Where <i>UnicodeIsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, "[\p{isGreek}]" matches Unicode characters in the Greek block. See <a href="#">Unicode Character Blocks for Regular Expressions</a> .
\P{ <i>UnicodeIsBlockSpec</i> }	(Only valid in character set) Matches characters not in <i>UnicodeIsBlockSpec</i> . For example, "[\P{isGreek}]" matches all characters that are not in the Unicode Greek block. This is equivalent to "[^\p{isGreek}]". See <a href="#">Unicode Character Blocks for Regular Expressions</a> .
\xhh	Matches hexadecimal character <i>hh</i> where 0<= <i>hh</i> <=0xff.

UNIX Regular Expression	Definition
<code>\dddd</code>	Matches decimal character <i>ddd</i> where $0 \leq ddd \leq 255$ .
<code>\d</code>	Defines a back reference to tagged expression number <i>d</i> . For example, <b>{abc}def\d</b> matches the string <b>abcdefabc</b> . If the tagged expression has not been set, the search fails.
<code>\c</code>	Specifies cursor position if match is found. If the expression <code>xyz\c</code> is found the cursor is placed after the <code>z</code> .
<code>\n</code>	Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file. Use <code>\d10</code> if you want to match an ASCII 10 character.
<code>\r</code>	Matches carriage return (ASCII 13). What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file.
<code>\t</code>	Matches tab character.
<code>\f</code>	Matches form feed character.
<code>\od</code>	Matches any 2 byte DBCS character. This escape is only valid in a match set ( <code>[...\od...]</code> ). <code>[\od]</code> matches any single byte character excluding end of line characters. When used to search Unicode text, this escape does nothing.
<code>\om</code>	Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end of line characters. For example, <b>\om.+</b> matches the rest of the buffer.
<code>\ol</code>	Turns off multi-line matching (default). You can still use <code>\n</code> to create regular expressions which match one or more lines. However, expressions like <code>.+</code> will not match multiple lines. This is much safer and usually faster than using the <code>\om</code> option.
<code>\char</code>	Declares character after slash to be literal. For example, <code>\*</code> represents the star character.

UNIX Regular Expression	Definition
<code>\:char</code>	<p>Matches predefined expression corresponding to <i>char</i>. The predefined expressions are:</p> <ul style="list-style-type: none"> <li><code>\:a [A-Za-z0-9]</code> Matches an alphanumeric character.</li> <li><code>\:c [A-Za-z]</code> Matches an alphabetic character.</li> <li><code>\:b (?[ \t]+)</code> Matches blanks.</li> <li><code>\:d [0-9]</code> Matches a digit.</li> <li><code>\:f (?[^\:\\&lt;&gt; =+;, \t"]+)</code> Windows: Matches a filename part.</li> <li><code>\:f (?[^\:\/\t"]+)</code> UNIX: Matches a filename part.</li> <li><code>\:h (?[0-9A-Fa-f]+)</code> Matches a hex number. Matches a filename part.</li> <li><code>\:i (?[0-9]+)</code> Matches an integer.</li> <li><code>\:n (?:[0-9]+(?:\.[0-9]+) \.[0-9]+)(?:[Ee](?:\+ -)[0-9]+))</code> Matches a floating number.</li> <li><code>\:p (?:([A-Za-z:])(?:\\ /)(?:\f(?:\\ /))*\f)</code> Windows: Matches a path.</li> <li><code>\:p (?:([:/])?:([f()])*\f)</code> UNIX: Matches a path.</li> <li><code>\:q (?\"[^\"]*\" '[^']*')</code> Matches a quoted string.</li> <li><code>\:v ([A-Za-z_\$][A-Za-z0-9_\$]*)</code> Matches a C variable.</li> <li><code>\:w ([A-Za-z]+)</code> Matches a word.</li> </ul>

The precedence of operators, from highest to lowest, is as follows:

- `+, *, ?, {}, +?, *?, ??, {}?` (These operators have the same precedence.)
- concatenation
- `|`

## UNIX Regular Expression Examples

Sample UNIX Regular Expression	Description
<b>^defproc</b>	Matches lines that begin with the word defproc.
<b>^definit\$</b>	Matches lines that only contain the word definit.
<b>^\<i>*name</i></b>	Matches lines that begin with the string <i>*name</i> . Notice that the backslash must prefix the special character <i>'*</i> .
<b>[<i>t</i> ]</b>	Matches tab and space characters.
<b>[<i>d9\</i>d32]</b>	Matches tab and space characters.
<b>[<i>x9\</i>x20]</b>	Matches tab and space characters.
<b>p.t</b>	Matches any three letter string starting with the letter 'p' and ending with the letter 't'. Two possible matches are pot and pat.
<b>s.*?t</b>	Matches the letter 's' followed by any number of characters followed by the nearest letter 't'. Two possible matches are seat and st.
<b>for while</b>	Matches the strings for or while.
<b>^\<i>:</i>p</b>	Matches lines beginning with a file name.
<b>xy+z</b>	Matches x followed by one or more occurrences of y followed by z.
<b>[<i>a-z-</i>qw]</b>	Character set subtraction. Matches all English lowercase letters except 'q' and 'w'.
<b>[<i>p{isGreek}&amp;</i>[p{L}]]</b>	Character set intersection. Matches all Unicode letters in the Greek block.
<b>\x{6587}</b>	Matches Unicode character with hexadecimal value 6587. Character set intersection. Matches all Unicode letters in the Greek block.
<b>[<i>p{L-}</i>qw]</b>	Matches all Unicode letters except 'q' and 'w'.
<b>[<i>p{L}</i>]</b>	Matches all Unicode letters.
<b>[<i>p{Lul}</i>]</b>	Matches all Unicode uppercase and lowercase letters.
<b>[<i>P{L}</i>]</b>	Matches all Unicode characters that are not letters.
<b>[<i>p{isGreek}</i>]</b>	Matches all Unicode characters in the Greek block.

## SlickEdit® Regular Expressions

SlickEdit regular expressions are defined in the following table.

SlickEdit Regular Expression	Definition
<b>^</b>	Matches beginning of line.
<b>\$</b>	Matches end of line.
<b>?</b>	Matches any character except newline.

SlickEdit Regular Expression	Definition
X+	Minimal match of one or more occurrences of X. See <a href="#">Minimal versus Maximal Matching</a> for more information.
X#	Maximal match of one or more occurrences of X.
X*	Minimal match of zero or more occurrences of X.
X@	Maximal match of zero or more occurrences of X.
X:n1	Matches exactly n1 occurrences of X. Use () to avoid ambiguous expressions. For example a:9()1 searches for 9 a's followed by a 1.
X:n1,	Maximal match of at least n1 occurrences of X.
X:n1,n2	Maximal match of at least n1 occurrences but not more than n2 occurrences of X.
X:*n1,	Minimal match of at least n1 occurrences of X.
X:*n1,n2	Minimal match of at least n1 occurrences but not more than n2 occurrences of X.
~X	Search fails if expression X is matched. The expression ^~(if) matches the beginning of all lines that do not start with if.
(X)	Matches sub-expression X.
{X}	Matches sub-expression X and specifies a new tagged expression. See <a href="#">Using Tagged Search Expressions</a> for more information.
{#dX}	Matches sub-expression X and specifies to use tagged expression number d where 0<=d<=9.
X Y	Matches X or Y.
[char-set]	Matches any one of the characters specified by char-set. A '-' character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. '\' may be used inside the square brackets to define literal characters or define ASCII characters. For example \- specifies a literal dash character. The expression [0-127] matches ASCII character codes 0..27. The expression [] matches no characters. In UNIX regular expressions [] matches a right bracket. In both syntaxes, the expression [^] matches a right bracket. The expression [^] matches a '^' character in both syntaxes.
[~char-set]	Matches any character not specified by char-set. A '-' character may be used to specify ranges. The expression [~A-Z] matches all characters except uppercase letters.
[^char-set]	Same as [~char-set] above.
[char-set1 - [char-set2]]	Character set subtraction. Matches all characters in char-set1 except the characters in char-set2. For example, "[a-z-[qw]]" matches all English lowercase letters except 'q' and 'w'. "[p{L}-[qw]]" matches all Unicode lower case letters except 'q' and 'w'.
[char-set1 & [char-set2]]	Character set intersection. Matches all characters in char-set1 that are also in char-set2. For example, "[x{0}-x{7f}&[p{L}]]" matches all letters between 0 and 127.



SlickEdit Regular Expression	Definition
<code>\x{hhhh}</code>	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhhh</i> .
<code>\p{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeCategorySpec</i> . Where <i>UnicodeCategorySpec</i> uses the standard general categories specified by the Unicode consortium. For example, “[ <b>p{L}</b> ]” matches all letters. “[ <b>p{Lu}</b> ]” matches all uppercase letters. See <a href="#">Unicode Category Specifications for Regular Expressions</a> .
<code>\P{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeCategorySpec</i> . For example, “[ <b>P{L}</b> ]” matches all characters that are not letters. This is equivalent to “[ <b>^p{L}</b> ]”. “[ <b>P{Lu}</b> ]” matches all characters that are not uppercase letters. See <a href="#">Unicode Category Specifications for Regular Expressions</a> .
<code>\p{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeIsBlockSpec</i> . Where <i>UnicodeIsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, “[ <b>p{isGreek}</b> ]” matches Unicode characters in the Greek block. See <a href="#">Unicode Character Blocks for Regular Expressions</a> .
<code>\P{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeIsBlockSpec</i> . For example, “[ <b>P{isGreek}</b> ]” matches all characters that are not in the Unicode Greek block. This is equivalent to “[ <b>^p{isGreek}</b> ]”. See <a href="#">Unicode Character Blocks for Regular Expressions</a> .
<code>\xhh</code>	Matches hexadecimal character <i>hh</i> where $0 \leq hh \leq 0xff$ .
<code>\ddd</code>	Matches decimal character <i>ddd</i> where $0 \leq ddd \leq 255$ .
<code>\gd</code>	Defines a back reference to tagged expression number <i>d</i> . For example, {abc}def\g0 matches the string abcdefabc. If the tagged expression has not been set, the search fails.
<code>\c</code>	Specifies cursor position if match is found. If the expression xyz\c is found the cursor is placed after the z.
<code>\n</code>	Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file. Use \d10 if you want to match an ASCII 10 character.
<code>\r</code>	Matches carriage return.
<code>\t</code>	Matches tab character.
<code>\b</code>	Matches backspace character.
<code>\f</code>	Matches form feed character.
<code>\od</code>	Matches any 2 byte DBCS character. This escape is only valid in a match set ([...od...]). [ <b>^od</b> ] matches any single byte character excluding end of line characters. When used to search Unicode text, this escape does nothing.

SlickEdit Regular Expression	Definition
<code>\om</code>	Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end of line characters. For example, <code>\om?#</code> matches the rest of the buffer. Note: Test the regular expression on a very small file before using the regular expression on a large file. This option may cause the editor to use a lot of memory.
<code>\ol</code>	Turns off multi-line matching (default). You can still use <code>\n</code> to create regular expressions which match one or more lines. However, expressions like <code>?#</code> will not match multiple lines. This is much safer and usually faster than using the <code>\om</code> option.
<code>\char</code>	Declares character after slash to be literal. For example, <code>\:</code> represents the colon character.

SlickEdit Regular Expression	Definition
: <i>char</i>	<p>Matches predefined expression corresponding to <i>char</i>. The predefined expressions are:</p> <ul style="list-style-type: none"> <li>• :a [A-Za-z0-9] Matches an alphanumeric character</li> <li>• :b ([\t]#) Matches blanks - note that :b is not like the Perl/.NET \s</li> <li>• :c [A-Za-z] Matches an alphabetic character</li> <li>• :d [0-9] Matches a digit</li> <li>• :f ([~\[\]\: \V&lt;&gt; =+;, \t"]#) Windows: Matches a filename part</li> <li>• :f ([~/\t"]#) UNIX: Matches a filename part.</li> <li>• :h ([0-9A-Fa-f]#) Matches a hex number</li> <li>• :i ([0-9]#) Matches an integer</li> <li>• :n (([0-9]#([0-9]#) [0-9]#)([Ee](\+ -)[0-9]# )) Matches a floating number</li> <li>• :p (([A-Za-z]\: )(\\ / )(:f(\\ / ))@:f) Windows: Matches a path</li> <li>• :p ((/ )(:f(/ ))@:f) UNIX: Matches a path</li> <li>• :q ("[\~"]@\" '[\~']@') Matches a quoted string</li> <li>• :v ([A-Za-z_\$][A-Za-z0-9_\$]@) Matches a C variable</li> <li>• :w ([A-Za-z]#) Matches a word</li> </ul>

The precedence of operators, from highest to lowest, is as follows:

- +, #, \*, @, :, .\* (These operators have the same precedence.)
- concatenation
- |

## SlickEdit® Regular Expression Examples

Sample SlickEdit Regular Expression	Description
<b>^defproc</b>	Matches lines that begin with the word defproc.
<b>^definit\$</b>	Matches lines that only contain the word definit.
<b>^\:name</b>	Matches lines that begin with the string :name. Notice that the backslash must prefix the special character ':'.
<b>[t ]</b>	Matches tab and space characters.
<b>[\9\32]</b>	Matches tab and space characters.
<b>[\x9\x20]</b>	Matches tab and space characters.
<b>p?t</b>	Matches any three letter string starting with the letter 'p' and ending with the letter 't'. Two possible matches are pot and pat.
<b>s?*t</b>	Matches the letter 's' followed by any number of characters followed by the nearest letter 't'. Two possible matches are seat and st.
<b>for while</b>	Matches the strings "for" or "while".
<b>^:p</b>	Matches lines beginning with a file name.
<b>xy+z</b>	Matches x followed by one or more occurrences of y followed by z.

## Brief Regular Expressions

Brief regular expressions are defined in the following table.

Brief Regular Expression	Definition
<b>%</b>	Matches beginning of line.
<b>&lt;</b>	Matches beginning of line.
<b>\$</b>	Matches end of line.
<b>&gt;</b>	Matches end of line.
<b>?</b>	Matches any character except newline.
<b>*</b>	Minimal match of zero or more of any character except newline. This is the same as ?@.
<b>X+</b>	Minimal match of one or more occurrences of X. See <a href="#">Minimal versus Maximal Matching</a> for more information.
<b>X\.*</b>	Maximal match of zero or more of any character except newline. This is the same as ?\:@.
<b>X\:@</b>	Maximal match of zero or more occurrences of X.
<b>X\:+</b>	Maximal match of one or more occurrences of X.
<b>X\:n1</b>	Matches exactly n1 occurrences of X. Use {} to avoid ambiguous expressions. For example a:9{}1 searches for 9 a's followed by a 1.

Brief Regular Expression	Definition
$X\backslash:n1,$	Maximal match of at least $n1$ occurrences of $X$ .
$X\backslash:,n2$	Maximal match of at least 0 occurrences but not more than $n2$ occurrences of $X$ .
$X\backslash:n1,n2$	Maximal match of at least $n1$ occurrences but not more than $n2$ occurrences of $X$ .
$X\backslash:n1?$	Match exactly $n1$ occurrences of $X$ .
$X\backslash:n1,?$	Minimal match of at least $n1$ occurrences of $X$ .
$X\backslash:,n2?$	Minimal match of at least 0 occurrences but not more than $n2$ occurrences of $X$ .
$X\backslash:n1,n2?$	Minimal match of at least $n1$ occurrences but not more than $n2$ occurrences of $X$ .
$\backslash(X)$	Matches sub-expression $X$ but does not define a tagged expression.
$\{X\}$	Matches sub-expression $X$ and specifies a new tagged expression. See <a href="#">Using Tagged Search Expressions</a> for more information.
$\{@dX\}$	Matches sub-expression $X$ and specifies to use tagged expression number $d$ where $0 \leq d \leq 9$ . No more tagged expressions are defined by the sub-expression syntax $\{X\}$ once this sub-expression syntax is used. This is the best way to make sure you have enough tagged expressions.
$X Y$	Matches $X$ or $Y$ .
$[char-set]$	Matches any one of the characters specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression $[A-Z]$ matches any uppercase letter. \ can be used inside the square brackets to define literal characters or define ASCII characters. For example \- specifies a literal dash character. The expression $[0-27]$ matches ASCII character codes 0..27. The expression $[]$ matches a right bracket. In SlickEdit® regular expressions $[]$ matches no characters. In both syntaxes, the expression $[\backslash]$ matches a right bracket.
$[ \sim char-set ]$	Matches any character not specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression $[ \sim A-Z ]$ matches all characters except uppercase letters. The expression $[ \sim ]$ matches any character except newline.
$[char-set1 - [char-set2]]$	Character set subtraction. Matches all characters in <i>char-set1</i> except the characters in <i>char-set2</i> . For example, $[a-z-[qw]]$ matches all English lower case letters except 'q' and 'w'. $[\backslash p\{L\}-[qw]]$ matches all Unicode lower case letters except 'q' and 'w'.
$[char-set1 \& [char-set2]]$	Character set intersection. Matches all characters in <i>char-set1</i> that are also in <i>char-set2</i> . For example, $[\backslash x\{0\}-\backslash x\{7f\} \& [\backslash p\{L\}]]$ matches all letters between 0 and 127.
$\backslash x\{hhhh\}$	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhhh</i> .

Brief Regular Expression	Definition
<code>\p{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeCategorySpec</i> . Where <i>UnicodeCategorySpec</i> uses the standard general categories specified by the Unicode consortium. For example, “[\p{L}]” matches all letters. “[\p{Lu}]” matches all uppercase letters. See <a href="#">Unicode Category Specifications for Regular Expressions</a> .
<code>\P{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeCategorySpec</i> . For example, “[\P{L}]” matches all characters that are not letters. This is equivalent to “[^\p{L}]”. “[\P{Lu}]” matches all characters that are not uppercase letters. See <a href="#">Unicode Category Specifications for Regular Expressions</a> .
<code>\p{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeIsBlockSpec</i> . Where <i>UnicodeIsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, “[\p{isGreek}]” matches Unicode characters in the Greek block. See <a href="#">Unicode Character Blocks for Regular Expressions</a> .
<code>\P{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeIsBlockSpec</i> . For example, “[\P{isGreek}]” matches all characters that are not in the Unicode Greek block. This is equivalent to “[^\p{isGreek}]”. See <a href="#">Unicode Character Blocks for Regular Expressions</a> .
<code>\xhh</code>	Matches hexadecimal character <i>hh</i> where $0 \leq hh \leq 0xff$ .
<code>\dddd</code>	Matches decimal character <i>ddd</i> where $0 \leq ddd \leq 255$ .
<code>\d</code>	Defines a back reference to tagged expression number <i>d</i> . For example, <b>{abc}def\d</b> matches the string <b>abcdefabc</b> . If the tagged expression has not been set, the search fails.
<code>\c</code>	Specifies cursor position if match is found. If the expression <i>xyz\c</i> is found the cursor is placed after the <i>z</i> .
<code>\n</code>	Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file. Use <code>\d10</code> if you want to match a 10 character.
<code>\r</code>	Matches carriage return.
<code>\t</code>	Matches tab character.
<code>\b</code>	Matches backspace character.
<code>\f</code>	Matches form feed character.
<code>\od</code>	Matches any 2 byte DBCS character. This escape is only valid in a match set ([...\od...]). <code>[\~\od]</code> matches any single byte character excluding end of line characters. When used to search Unicode text, this escape does nothing.
<code>\om</code>	Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end of line characters. For example, <code>\om? \@</code> matches the rest of the buffer.

Brief Regular Expression	Definition
<code>\ol</code>	Turns off multi-line matching (default). You can still use <code>\n</code> to create regular expressions which match one or more lines. However, expressions like <code>?\@</code> will not match multiple lines. This is much safer and usually faster than using the <code>\om</code> option.
<code>\char</code>	Declares character after slash to be literal. For example, <code>\*</code> represents the star character.
<code>\:char</code>	Matches predefined expression corresponding to <i>char</i> . The predefined expressions are: <ul style="list-style-type: none"> <li><code>\:a [A-Za-z0-9]</code> Matches an alphanumeric character</li> <li><code>\:b \([ \t]#\)</code> Matches blanks</li> <li><code>\:c [A-Za-z]</code> Matches an alphabetic character</li> <li><code>\:d [0-9]</code> Matches a digit</li> <li><code>\:f \([~\[\]\: \V&lt;&gt; =+;,\t"]#\)</code> Matches a filename part</li> <li><code>\:f \([~/\t"]#\)</code> UNIX: Matches a filename part.</li> <li><code>\:h \([0-9A-Fa-f]#\)</code> Matches a hex number</li> <li><code>\:i \([0-9]#\)</code> Matches an integer</li> <li><code>\:n \([0-9]#(\.[0-9]# ) \.[0-9]#\)\([Ee]\([+ -]\)[0-9]#\)\)</code> Matches a floating number</li> <li><code>\:p \([A-Za-z]\: \\ \/ \\ :\f(\ /)\)\@:f)</code> Windows: Matches a path</li> <li><code>\:p \([/ \\ :\f(\ /)\)\@:f)</code> UNIX: Matches a path</li> <li><code>\:q \(["'~\"]@\"' ~']@\\)</code> Matches a quoted string</li> <li><code>\:v \([A-Za-z_\$][A-Za-z0-9_\$]@\)</code> Matches a C variable</li> <li><code>\:w \([A-Za-z]#\)</code> Matches a word</li> </ul>

## Brief Regular Expression Examples

Sample Brief Regular Expression	Description
<defproc	Matches lines that begin with the word <b>defproc</b> .
<definit>	Matches lines that only contain the word <b>definit</b> .
<\*name	Matches lines that begin with the string *name. Notice that the backslash must prefix the special character *.
[ \t ]	Matches tab and space characters.
[\d9\d32]	Matches tab and space characters.
[\x9\x20]	Matches tab and space characters.
p?t	Matches any three letter string starting with the letter p and ending with the letter t. Two possible matches are pot and pat.
s*t	Matches the letter s followed by any number of characters followed by the nearest letter t. Two possible matches are seat and st.
{for} {while}	Matches the strings for or while.
^\:p	Matches lines beginning with a file name.
xy+z	Matches x followed by one or more occurrences of y followed by <b>z</b> .

## Unicode Category Specifications for Regular Expressions

The Unicode consortium standard regular expression categories are supported. The syntax for specifying categories is:

```
\p{MainCategoryLetter Subcategories}
```

which matches the categories specified, and:

```
\P{MainCategoryLetter Subcategories}
```

which matches all characters not in the categories specified.

The \p and \P notations can only be used inside a character set specification. *MainCategoryLetter* can be 'L', 'M', 'N', 'P', 'S', 'Z', or 'C'. The valid *Subcategories* depend on the *MainCategoryLetter* specified. If no *Subcategories* are specified, all are assumed. For example:

- “[\p{L}]” matches all Unicode letters



- “[p{Lu}]” matches all uppercase and lowercase letters
- “[P{L}]” matches all characters that are not letters

The following table lists the valid subcategories for a specific main category. These character tables were generated using the “UnicodeData-3.1.0.txt” found on the Unicode consortium Web site.

Subcategory	Description
Lu	Letter, Uppercase
Ll	Letter, Lowercase
Lt	Letter, Titlecase
Lo	Letter, Other
Mn	Mark, Non-Spacing
Mc	Mark, Spacing Combining
Me	Mark, Enclosing
Nd	Number, Decimal Digit
Nl	Number, Letter
No	Number, Other
Pc	Punctuation, Connector
Pd	Punctuation, Dash
Ps	Punctuation, Open
Pe	Punctuation, Close
Pi	Punctuation, Initial quote (may behave like Ps or Pe depending on usage)
Pf	Punctuation, Final quote (may behave like Ps or Pe depending on usage)
Po	Punctuation, Other
Sm	Symbol, Math
Sc	Symbol, Currency
Sk	Symbol, Modifier
So	Symbol, Other
Zs	Separator, Space
Zl	Separator, Line
Zp	Separator, Paragraph
Cc	Other, Control
Cf	Other, Format
Cs	Other, Surrogate
Co	Other, Private Use
Cn	Other, Not Assigned (no characters in the file have this property)

## Unicode Character Blocks for Regular Expressions

The Unicode consortium standard regular expression block categories are supported. The syntax for specifying a character block is:

```
\\ p{IsBlockName}
```

which matches the characters in the block specified, and:

```
\\P{IsBlockName}
```

which matches all characters not in the block specified.

The `\p` and `\P` notations may only be used inside a character set specification. For example, “`[\\p{isBasicLatin}]`” matches all characters in the Greek block. “`[\\P{isBasicLatin}]`” matches all characters that are not in the Greek block.

The following table lists the non-standard valid character block names. These character tables were generated from XML standards found at the World Wide Web Consortium Web site.

Block Name	Description
XMLNameStartChar	All characters that are valid for the start of an XML tag name.
XMLNameChar	All characters that are valid in an XML tag name.

The following table lists the valid character block names. These character tables were generated using the “**blocks.txt**” found on the Unicode consortium’s web site.

Range	Block Name
0000..007F	BasicLatin
0080..00FF	Latin-1Supplement
0100..017F	LatinExtended-A
0180..024F	LatinExtended-B
0250..02AF	IPAExtensions
02B0..02FF	SpacingModifierLetters
0300..036F	CombiningDiacriticalMarks
0370..03FF	Greek
0400..04FF	Cyrillic
0530..058F	Armenian
0590..05FF	Hebrew

Range	Block Name
0600..06FF	Arabic
0700..074F	Syriac
0780..07BF	Thaana
0900..097F	Devanagari
0980..09FF	Bengali
0A00..0A7F	Gurmukhi
0A80..0AFF	Gujarati
0B00..0B7F	Oriya
0B80..0BFF	Tamil
0C00..0C7F	Telugu
0C80..0CFF	Kannada
0D00..0D7F	Malayalam
0D80..0DFF	Sinhala
0E00..0E7F	Thai
0E80..0EFF	Lao
0F00..0FFF	Tibetan
1000..109F	Myanmar
10A0..10FF	Georgian
1100..11FF	HangulJamo
1200..137F	Ethiopic
13A0..13FF	Cherokee
1400..167F	UnifiedCanadianAboriginalSyllabics
1680..169F	Ogham
16A0..16FF	Runic
1780..17FF	Khmer
1800..18AF	Mongolian
1E00..1EFF	LatinExtendedAdditional
1F00..1FFF	GreekExtended
2000..206F	GeneralPunctuation
2070..209F	SuperscriptsandSubscripts
20A0..20CF	CurrencySymbols
20D0..20FF	CombiningMarksforSymbols
2100..214F	LetterlikeSymbols
2150..218F	NumberForms
2190..21FF	Arrows
2200..22FF	MathematicalOperators
2300..23FF	MiscellaneousTechnical
2400..243F	ControlPictures
2440..245F	OpticalCharacterRecognition

Range	Block Name
2460..24FF	EnclosedAlphanumerics
2500..257F	BoxDrawing
2580..259F	BlockElements
25A0..25FF	GeometricShapes
2600..26FF	MiscellaneousSymbols
2700..27BF	Dingbats
2800..28FF	BraillePatterns
2E80..2EFF	CJKRadicalsSupplement
2F00..2FDF	KangxiRadicals
2FF0..2FFF	IdeographicDescriptionCharacters
3000..303F	CJKSymbolsandPunctuation
3040..309F	Hiragana
30A0..30FF	Katakana
3100..312F	Bopomofo
3130..318F	HangulCompatibilityJamo
3190..319F	Kanbun
31A0..31BF	BopomofoExtended
3200..32FF	EnclosedCJKLettersandMonths
3300..33FF	CJKCompatibility
3400..4DB5	CJKUnifiedIdeographsExtensionA
4E00..9FFF	CJKUnifiedIdeographs
A000..A48F	YiSyllables
A490..A4CF	YiRadicals
AC00..D7A3	HangulSyllables
D800..DB7F	HighSurrogates
DB80..DBFF	HighPrivateUseSurrogates
DC00..DFFF	LowSurrogates
E000..F8FF	PrivateUse
F900..FAFF	CJKCompatibilityIdeographs
FB00..FB4F	AlphabeticPresentationForms
FB50..FDFF	ArabicPresentationForms-A
FE20..FE2F	CombiningHalfMarks
FE30..FE4F	CJKCompatibilityForms
FE50..FE6F	SmallFormVariants
FE70..FEFE	ArabicPresentationForms-B
FEFF..FEFF	Specials
FF00..FFEF	HalfwidthandFullwidthForms
FFF0..FFFD	Specials
10300..1032F	OldItalic

Range	Block Name
10330..1034F	Gothic
10400..1044F	Deseret
1D000..1D0FF	ByzantineMusicalSymbols
1D100..1D1FF	MusicalSymbols
1D400..1D7FF	MathematicalAlphanumericSymbols
20000..2A6D6	CJKUnifiedIdeographsExtensionB
2F800..2FA1F	CJKCompatibilityIdeographsSupplement
E0000..E007F	Tags



# Glossary

## 3-Way Merge

Typically used after two people make a local copy of the same source file and make some modifications to their local copy. The 3-way merge takes both sets of changes and creates a new source file. A wizard lets you select the change desired in the output file. The output can be viewed side-by-side or interleaved.

## API

Application Programming Interface. A functional interface that allows an application program written in a high-level language to use specific data or functions of the operating system or another program. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program.

## Auto List Members

Automatically lists members when you type a member access operator. Also access this feature by pressing Alt+Dot.

## Auto List Parameters

When you type a function operator, a list of compatible variables and expressions for the current argument is displayed. For performance reasons, not all possible variables and expressions are listed. Press Alt+Dot if the symbol you want is not listed. To access Auto List Parameters on demand, press Alt+Comma.

## Auto Parameter Info

Automatically displays the prototype for a function when you type a function operator, and highlights the current argument within the displayed prototype.

## Binding

The attachment of a command to a key.

## Bookmark stack

An internal list of pushed bookmarks.

## Breakpoint

A point designated in the code to break or stop during a debug. View a list of all breakpoints in the Task view.

## Buffer

A file that has been loaded into the application. When a file is loaded, you can safely perform modifications to the buffer without modifying the file on disk until you save the buffer.

## Class

A compiled Java source file.

## Clipboard

A temporary storage area used to transfer text or dialog box controls from one place to another. Multiple text clipboards are available to store multiple instances of copied material.

### **Context Tagging®**

A feature set that performs expression type, scope, and inheritance analysis as well as symbol look-up within the current context to help you navigate and write code. Context Tagging uses an engine that parses your code and builds a database of symbol definitions and declarations—commonly referred to as *tags*. Context Tagging features work with *your* source code, not just standard APIs (application program interfaces), and the features are dynamic, in the sense that symbols are updated immediately or in the background as you edit your source code.

### **CVS**

An open-source, network-transparent version control system.

### **DIFFzilla®**

Allows you to view and merge changes from one version of a file to another. Difference two files, two directories or two source trees. Provides the ability to view and merge differences for specific symbols such as functions or classes, or a specified range of lines, from two files or the same file.

### **Edit window**

A rectangular viewing area used to display and edit buffers.

### **Emulation**

The ability of a program or device to imitate another program or device. Change the keyboard bindings or shortcuts to emulate favorite shortcuts. Nine emulations available.

### **Enscript**

Enscript is an external, command-line program that prints a text file to a printer using PostScript, which allows for print formatting such as font, page layout, margins, colors, etc. Enscript is included in most Linux distributions. However, it is also shipped with SlickEdit® to ensure availability of the program.

### **File Extension Setup**

Provides an easy way to set default indent, word wrap, and other options for specific file extensions.

### **FLEXnet™ Publisher**

Licensing option for multiple users on a server.

### **Hotkey**

A keyboard shortcut that is bound to a menu item.

### **IDE**

Integrated development environment. A set of software development tools such as editors, compilers, and debuggers, that are accessible from a single user interface.

### **Incremental search**

Allows searching as letters are typed.

### **Key binding**

A key or combination of keys that a user can press to perform an action that is available from a menu. Also known as a shortcut key.



**pcode**

The binary result of a translation of Slick-C® source code. The translation is done to speed up the interpretation of source code.

**Project**

A group of folders, files, classes or packages.

**Refactoring**

A comprehensive code editing feature to help improve, stabilize, and maintain code. It allows a system-wide coding change without affecting the semantic behavior of the system.

**Run-time**

The time period that a computer program is executing. A run-time environment is an execution environment.

**Schema**

In database programming, the representation of a database that will be mapped.

**Selection**

A highlighted region of text typically operated by a command which affects only the region. In the dialog editor, the selection is indicated by eight square handles which surround the control.

**Selective Display**

A SlickEdit feature that allows you to select which lines are visible or hidden based on the content of the lines. Also known as *code folding*.

**Slick-C®**

The SlickEdit® macro programming language.

**SmartPaste®**

Pasted or dropped source code is automatically re-indented to the correct indentation level.

**Source folder**

A folder that contains packages, classes, and files.

**State file**

A file that stores configuration information and allows quick state restoration in subsequent edit sessions.

**Window**

A rectangular viewing area. We also use this term in the more advanced sections of this manual to refer to the operating system resource known as a window.

**Workspace**

A workspace is for managing resources.



# Index



**A**

- About SlickEdit 52
- ActionScript Formatting Options dialog 353
- active project 127
- Ada Beautifier dialog box 397
- Ada Formatting Options dialog 397
- Add Dialog Box 422
- Add File dialog box (Code Templates) 160
- add file to project 600
- Add FTP Profile Advanced Tab 476
- Add FTP Profile General Tab 475
- Add import (Refactoring) 356
- Add New Item dialog box (Code Templates) 161
- Add Parameter dialog box (Code Templates) 160
- Add Tag File dialog 184
- Add Tree dialog box 519
- Add (Code Templates Add New Item dialog box) 162
- adding files to a project 132
- advanced tab (file ext setup) 590
- Advanced tab (FTP Options dialog box) 477
- after header 90
- Alias Editor dialog box 259
- alias file 259
- aliases 257
  - creating from selection 262
  - directory aliases 257
  - escape sequences 260
  - expansion option 592
  - extension-specific aliases 258
  - filename option 583
  - parameter prompting 261
  - using the Alias Editor 259
- align on equal 346
- align on parens 346
- always save configuration 576
- Annotation Editor dialog 281
- Annotation File Manager dialog 286
- Annotation Types dialog 283
- anonymous e-mail address 477
- anonymous login 475
- ANSI C Formatting Options dialog 353
- ant targets 360
- apache ant 360
  - adding xml files to a project 360
  - opening 360
- append eof character 600
- argument completion 255
  - options for 593
- arrays 450
- Associate File Types dialog 115
- associating workspaces 115
- At left margin (Comments tab) 586
- At level of indent (Comments tab) 586
- attach debugger 200
  - core file 201
  - GDB 201
  - Java 201
  - running process 201
- Attach Debugger Menu 526
- attach debugger (GDB) 201
- attach debugger (Java) 201
- auto 580
- auto caps mode 139
- auto deselect 571
- Auto Enable Properties dialog box 542
- auto hide toolbar 63
- auto increment search results window 494
- Auto List Compatible Parameters 181
- auto list members 179
- auto merge 554
- auto parameter information 180
- auto read only 598
- auto refresh 476
- auto reload 598
- auto save activated 602
- auto save directory 603
- auto save tab 142
- auto validate on open 364
- auto-change directory 139
- auto-close visited files 139
- Auto-Complete 253
- Auto-complete tab 591
- Auto-display parameter (tagging tab) 595
- Auto-insert matching param (tagging tab) 595
- Auto-list compatible params (tagging tab) 595
- Auto-list members (tagging tab) 594
- Auto-list values (tagging tab) 595
- automatic directory mapping 417
- automatic width (comment wrapping) 589
- Automatically expand doc comments (Comments tab) 587
- auto-restore 113
- Autos tool window 69
- autosave tab 142
- Auto-select unique items (Auto-Complete tab) 593
- auto-updated tag files 186
- AWK Formatting Options dialog 353

**B**

- backing up network files 149
- backup 149
  - configuration directory 663
  - history 149
  - network files 149
  - options 150
- Backup dialog box 146
- backup directory 601
- backup files 601
- Backup History 149
  - setting options for 150
- Backup History tool window 64
- backup tab 600
- backward 493
- Base Classes dialog box 230
- base file 418
- beautifier 307
- beautify 343

- Beautifying code 307
- before footer 90
- begin end structure matching 212
  - setting match style 213
  - viewing/defining 212
- Begin-End Style tab (C/C++ Formatting) 339
- binary character searching 303
- binary files (editing) 140
- Bind Key dialog 607
- bind macro to key 446
- block cursor 575
- Block insert mode 236
- block selections 235
- block spilling (unmodified) 599
- bookmarks 275
  - deleting named 276
  - named 275
  - navigating named 276
  - options 278
  - popping a bookmark 277
  - pushed 277
  - pushing a bookmark 277
  - tool window 276
  - viewing pushed 277
- Bookmarks dialog box 275
- Bookmarks tool window 276
- breakpoints 201
  - conditional 201
  - exceptions 202
- breakpoints tab 201
- Breakpoints tool window 64
- Brief regular expressions 702
  - examples 706
- buffer cache 577
- Buffer tabs (visibility) 65
- buffers 71
  - closing 76
  - listing open 74
  - switching between 73
- build and compile 191
  - auto-build on save 192
  - build errors 193
  - build methods 192
  - build on save 192
  - commands 192
  - GNU C/C++ 192
  - makefiles 192
  - operations 191
  - parsing errors 194
  - projects 191
  - setting shortcuts for Ant targets 360
  - Xcode 193
- build errors 193
  - listing 194
  - parsing with reg expressions 194
  - viewing and navigating 193
- Build Menu 523
- build methods 192
  - GNU C/C++ 192

- Xcode 193
- build options 131
- build settings (projects) 130
- Build tab (Project Properties) 515
- Build tag file (C++ Refactoring) 351
- Build tool window 64
- build window (auto exit) 573
- build window (auto-restore) 113
- build window (erased lines) 574
- building tag files 184
- byte offset navigation 215

## C

- cache size (for buffers) 577
- cache size (for tagging) 577
- calculating (see mathematics)
- calculations (see mathematics)
- calculator dialog box 437
- Call Stack tool window 69
- call tree 229
- case for hex values 376
- case sensitive 244
- Categories (Code Templates Add New Item dialog box) 161
- Categories (Template Manager dialog box) 159
- CFML Formatting Options dialog box 375
- CFScript Formatting Options dialog 353
- Ch Formatting Options dialog 353
- Change Directory dialog box 566
- change directory (option) 566
- changing emulations 98
- changing fonts and colors 109
- character count indicator 60
- character selection 235
  - inclusive option 571
- characters (inserting literal) 245
- characters (searching for with expressions) 303
- Check In Dialog Box 422
- Check Out Dialog Box 422
- Checkin/Checkout Files dialog 552
- CICS Formatting Options dialog 403
- Class Exclusion Manager dialog 219
- class properties 69
- Class tool window 217
  - exclusion manager 219
  - filtering hierarchy pane 219
  - filtering/sorting members pane 219
- Classes (debug) tool window 69
- clear modified lines 247
- clear\_bookmarks 276
- click past end of line 243
- clipboards 242
  - setting max number of 243
- clipboards (auto-restore) 113
- close workspace 122
- closing files 138
  - automatically 139
- closing SlickEdit (see exiting SlickEdit)
- clr debugger 202

- Cobol Formatting Options dialog 403
- code annotations 279
  - code marker 280
  - copying 282
  - creating 281
  - deleting 283
  - editing 283
  - for code reviews 287
  - for recording tasks 287
  - go to annotation 282
  - handling type conflicts 284
  - locations 279
  - managing files 285
  - moving 282
  - private and shared 280
  - types 279
  - user-defined 286
  - view and filter 282
- Code Annotations tool window 280
- code folding 328
- code navigation 211
- code reviews with annotations 287
- Code Templates 153
  - Add File dialog box 160
  - Add New Item dialog box 161
  - Add Parameter dialog box 160
  - Creating 155
  - Creating a multi-file template 163
  - Global substitution parameters 160
  - Locating Templates 162
  - Manually creating a template 162
  - Metadata file reference 164
  - Organizing templates 158
  - Pre-defined substitution parameters 156
  - Substitution parameters 156
  - Template Manager dialog box 158
  - Template Options dialog box 160
- collapse code block 329
- color
  - see colors
- color coding 247
  - adding keywords 247
  - advanced configuration 249
  - clear modified line color 247
  - configuration 248
  - creating support for a new language 247
  - default options 249
- Color Coding Search Options dialog box 492
- Color Coding Setup dialog box 612
- Color Picker dialog 112
- Color Settings dialog box 622
- colors 111
  - customizing 111
  - for embedded languages 112
  - of special characters 328
  - schemes 112
  - sync backgrounds 623
- column indicators 60
- Columns per Line 474
- command line (see SlickEdit command line) 79
- commands 82
  - binding to keys 103
  - viewing associated key bindings 80
  - see also SlickEdit command line
- Comment width (Comment Wrap tab) 535
- comment wrap tab 588
- comment wrapping 292
  - Comment Wrap tab 588
  - enable 589
  - reflowing comments 292
  - sync vertical line 590
  - width settings 589
- comments 289
  - block comments 289
  - commenting lines 289
  - comments tab options 584
  - configuration 289
  - configure start column 586
  - configuring block comments 585
  - configuring doc comments 586
  - configuring line comments 586
  - creating doc comments 290
  - doc comment examples 290
  - Doxygen 291
  - editing options 587
  - Javadoc 290
  - line comments 289
  - reflowing 292
  - removing 289
  - string editing 292
  - string editing options 588
  - wrapping 292
  - XMLdoc 291
- Comments tab 584
- Comments tab (Beautifier) 346
- Comments tab (Color Coding Setup) 619
- common keys (redefining) 86
- Compare Options dialog box 420
- comparing directories 415
- comparing files 413
- comparing parts of files 414
- comparing symbols in one file 414
- comparing symbols of two files 415
- compile 191
  - projects 191
  - Visual C++ 192
  - vsbuild 192
  - see also build and compile
- Compiler Properties 349
- Compiler properties (C++ Refactoring) 349
- Compile/Link tab (Project Properties) 517
- completion 253
  - word/variable 254
- Completion on space (tagging tab) 594
- Completion (Context Tagging) 182
- Completions in dialogs 255
- concurrent process buffer (Build window) 64
- conditional breakpoints 201

- ul style="list-style-type: none; padding-left: 0;">
- config (see configuration)
- configuration 663
  - backing up config directory 663
  - changes not on menu 656
  - configuration variables 659
  - F1 index file 669
  - F1 MSDN help 670
  - help files 669
  - of project directories 128
  - of project tools 128
  - of projects 127
  - options for saving 576
  - set command 656
  - system config files 664
  - table of sys config files 664
  - table of user config files 663
  - user config directory 663
  - vslick.ini 656
  - see also environment variables
- Configurations tab (Debugger Options) 207
- configuraton
  - of build settings 130
- Configure F1 MSDN help 670
- Configure Index File dialog box 638
- contact support 52
- context menus 77, 591
- Context Tagging 179
  - Auto List Compatible Parameters 181
  - building tag files 183
  - Completions 182
  - configuring for COBOL 185
  - for a new extension 184
  - List Members 179
  - managing tag files 186
  - options for 188
  - Parameter Information 180
  - Statement Level Tagging 182
  - Symbol Browsing 182
  - Tag-Driven Navigation 179
- Context Tagging - Tag Files dialog box 560
- Context Tagging Options dialog box 610
- Context Tagging tab (file ext setup) 593
- Context Tagging toolbar 68
- continuation indent 345
- Continue bullet list on Enter (comment wrapping) 590
- control characters (inserting) 91
- conventions 35
  - code syntax 37
  - menus and dialogs 36
- Convert global to static field (C++ Refactoring) 318
- Convert local to field (C++ Refactoring) 319
- Convert Static to Instance Method 318
- convert Unicode to UCN 649
- Cool Features 29
- Copy dialog box 146
- Copy source to template directory (Code Templates Add File dialog box) 160
- Copy Unicode As ucn Menu 485
- copying code annotations 282
- core file 201
- count number of lines 598
- count selected characters 241
- count selected lines 241
- Create New Configuration dialog 127
- Create standard methods (C++ Refactoring) 320
- Create Tag Files for Run-Time Libraries dialog 183
- create workspace 123
- creating a file from a selection 135
- Creating a multi-file template 163
- Creating a new category (Template Manager dialog box) 158
- Creating a new template (Template Manager dialog box) 158
- creating code annotations 281
- creating custom project types 126
- creating directory aliases 257
- creating extension-specific aliases 258
- creating files 135
- Creating new configurations (C++ Refactoring) 350
- creating project files 134
- creating projects 126
- Creating templates 155
- Ctrl+Comma (pop bookmark) 211
- Ctrl+Dot (go to definition) 211
- Ctrl+/ (go to reference) 211
- Ctrl+Shift+V 243, 575
- cua text box 574
- CUA (default emulation) 35
- current character 60
- Current Context tool window 220
- Current Document Options dialog 535
- current line highlight 243
- Current paragraph (Comment Wrap tab) 535
- cursor navigation 213
- cursor style 107
- Custom Parameters tab (Template Manager dialog box) 159
- custom project types 126
- CVS 425
- C# Formatting Options dialog 353
- c++ refactoring 310
- C++ Refactoring Configuration Test dialog box 320
- C++ Refactoring Menu 547
- C++ Refactoring Setup dialog box 349
- c/c++ beautifier 343
- C/C++ Formatting Options dialog 339
- C/C++ Preprocessing dialog box 351
- ## D
- dBASE Formatting Options dialog 407
  - debug key bindings 199
  - Debug Menu 525
  - Debug Sessions tool window 69
  - debug tab (FTP Options) 480
  - Debug toolbar 68
  - Debug Windows Menu 526
  - debug (see also debugging) 199
  - debugger options 202



- Debugger Options dialog box 202
  - Debugger Options Directories Tab 206
  - debugger options info tab 202
  - Debugger Options Runtime Filters Tab 205
  - debugger options (configurations tab) 207
  - debugger options (filters tab) 205
  - debugger (see also debugging) 199
  - debugging 199
    - attach to core file 201
    - attach to remote process 201
    - attach to remote VM 201
    - attach to running process 201
    - GDB 201
    - generate debug 202
    - hot-swap debugger 204
    - mixed mode view 199
    - multiple sessions 200
    - named sessions 200
    - setting breakpoints 201
    - setting options for 202
    - tool windows 208
  - Default Help dialog box 638
  - default key binding mode 35
  - default local directory 477
  - DefaultName element (Code Templates metadata file reference) 165
  - defining project dependencies 127
  - Defs tool window 221
    - options 221
  - Delete Code Block dialog 274
  - Delete Menu 484
  - delete selection 571
  - Deleting a template (Template Manager dialog box) 158
  - deleting an extension 337
  - deleting code annotations 283
  - deleting code blocks 274
  - Dependencies tab (Project Properties) 518
  - dependencies (defining for projects) 127
  - Derived Classes dialog 230
  - Description element (Code Templates metadata file reference) 166
  - Details tab (Template Manager dialog box) 159
  - Diff dialog box 413
  - diff options tab 558
  - Diff Setup dialog box 558
  - different extension 603
  - diffing (see DIFFzilla) 413
  - DIFFzilla 413
    - backup history 417
    - comparing directories 415
    - comparing symbols in two files 415
    - comparing symbols/parts of files 414
    - comparing two files 413
    - directory mapping 417
    - dynamic difference editing 413
    - generating file lists 416
    - options 556
    - path info and filespecs 556
    - types 555
  - DIFFzilla dialog box 555
  - Directories tab (Debugger Options) 206
  - Directories tab (Project Properties) 512
  - directories (changing) 566
  - directories (comparing) 415
  - directories (for projects) 128
  - directory aliases 257
    - creating 257
    - embedding env variables 258
    - using 258
  - directory mapping 417
  - directory (auto-change) 139
  - directory (auto-restore) 113
  - Display after idle (Auto-Complete tab) 593
  - display line numbers 584
  - display symbol info under mouse 596
  - display tool tips 107
  - do not upload 477
  - doc comments 290
    - configuration 586
  - docking toolbars 63
  - document math 439
  - Document Menu 529
  - documentation 35
    - feedback 35
  - documentation conventions 35
  - Doxygen comments 291
  - drag and drop text 242
  - drag drop 575
  - draw box around line 243
  - DTD caching 372
  - dual monitors 83
  - dynamic difference editing 413
  - Dynamic Surround 267
  - dynamic tagging 179
  - dynamic-language projects 125
- ## E
- eat spaces after 348
  - Edit Menu 483
  - edit other copy unicode as ucn menu 649
  - Edit Other Menu 485
  - Edit Select Menu 484
  - Edit toolbar 68
  - Editing an existing template (Template Manager dialog box) 158
  - editing code annotations 283
  - editing recorded macros 448
  - editor windows 71
    - cascading and tiling 72
    - closing 76
    - duplicating 72
    - left margin width 71
    - linking 75
    - maximize/minimize 72
    - splitting 71
  - Elements (Code Templates metadata file reference) 165
  - else on same line as } 344
  - embedded asp dialect 376

- embedded language color 112
- embedded languages 40
  - HTML 41
  - Perl 41
  - UNIX 41
- embedding env variables in aliases 258
- emulations 97
  - changing 98
  - default (CUA) 35
  - determining keys/functions 99
  - supported 97
- Emulations (list) 97
- Enable auto-completion (Auto-Complete tab) 592
- enable comment wrap 589
- Encapsulate field (C++ Refactoring) 315
- encoding 647
  - SBCS/DBCS 647
  - Unicode 647
  - UTF-8 647
- end of line 328
- Enter Alias Parameter dialog 261
- Enter New Alias Name dialog 259
- Enter New Lexer Name dialog 247
- Enter New Tag Name dialog 392
- Entire block comment (Comment Wrap tab) 535
- Enumerate dialog box 487
- Enumeration 241
- environment variables 655
  - set command 656
  - setting in vslick.ini 656
- Error File dialog box 194
- error parsing 194
  - configuration 194
  - exclusions 196
  - sample expression 197
  - testing expressions 198
- Error Regular Expressions dialog box 194
- escape sequences 260
- evaluating regular expressions 435
- examples of Brief reg expressions 706
- examples of SlickEdit reg expressions 702
- examples of UNIX reg expressions 697
- exception breakpoints 202
- Exceptions tool window 65
- exclusions (error parsing) 196
- exit options 48
- exit process 64, 573
- exit tab 576
- exiting SlickEdit 48
  - default options 48
  - with modified buffers 48
- expand code block 329
- Expand tabs 463
- expand tabs to spaces (Load tab) 598
- expand tabs to spaces (Save tab) 600
- Explorer Open dialog box 136
- export key bindings 105
- export to HTML 374
- extend file history 661

- Extend line comments (Comments tab) 587
- extend workspace history 661
- extension 579
- extension options 335
- extension options dialog box 578
- extension-specific aliases 258
  - creating 258
  - creating from selection 262
  - escape sequences 260
  - parameter prompting 261
- extension-specific projects 132
- Extract class (C++ Refactoring) 315
- Extract method (C++ Refactoring) 311
- Extract method (Quick Refactoring) 309
- Extract super class (C++ Refactoring) 316

## F

- f command 212
- failed saves 138
- faq 49
- file associations 115
- File element (Code Templates metadata file reference) 167
- file extension 335
- file extension setup dialog 578
- file filters 143
- File Filters tab 143
- file format 465
- file history (File menu) 661
- file history (Project menu) 661
- file lists 145, 573
- file lists (generating) 416
- file locking 598
- File Manager Files Menu 458
- File Manager Menu 456
- File Manager Select Menu 457
- File Manager (Using) 145
- File Menu 455
- File Open dialog box 461
- file options 139
- File Options dialog box 140
- file search order 667
- File Sort dialog box 146
- File Tabs tool window 65
- File tabs (visibility) 65
- file types 115
- fileman select menu 146
- files 135
  - auto-close 139
  - auto-restore settings 113
  - autosave options 142
  - closing 138
  - comparing parts of 414
  - comparing symbols in two files 415
  - comparing two files 413
  - configuration files and directories 663
  - creating from selections 135
  - diffing file history 417
  - encoding 647

- failed saves 138
  - filters 143
  - finding files to open 137
  - first opened is active 139
  - FTP 433
    - generating lists 416
    - history and backups 149
    - inserting into buffers 137
    - listing open files/buffers 74
    - load and save options 140
    - merging 417
    - navigating between 213
    - opening 135
    - quick create/open 135
    - save as 137
    - saving 137
    - setting associations 115
    - setting options 139
    - virtual memory 143
  - Files element (Code Templates metadata file reference) 168
  - files menu 455
  - Files tab (DIFFzilla) 555
  - Files tab (Project Properties) 511
  - Files tab (Template Manager dialog box) 159
  - Files tool window 65
  - filters 476
  - find
    - see also searching
  - Find and Replace tool window 490
  - Find File dialog box 137
  - Find in Files tab (Find and Replace) 493
  - Find Symbol tool window 495
  - Find tab (Find and Replace) 492
  - finding files 137
  - firewall proxy tab 478
  - First line is top (Comments tab) 586
  - first time running SlickEdit 47
  - fixed right margin (comment wrapping) 590
  - fixed width (comment wrapping) 589
  - float toolbar 63
  - Font Configuration dialog box 109
  - Font dialog box 110
  - fonts 109
    - changing 109
    - in editor windows 110
    - recommended 110
    - UNIX 110
  - footer (printing) 90
  - For start characters (Comments tab) 587
  - force parens on return 348
  - foreground search 494
  - form feed after last page 473
  - format file 600
  - Fortran Formatting Options dialog 403
  - fp command 212
  - frequently asked questions 49
  - FTP 431
    - configuration options 433
    - connecting 432
    - create connection profile 431
    - disconnecting 433
    - opening files 433
    - tool windows 431
  - ftp connection profile 431
  - FTP Menu 456
  - FTP Options Advanced Tab 477
  - ftp options firewall/proxy tab 478
  - ftp options general tab 475
  - ftp options ssh/sftp tab 479
  - FTP port 478
  - FTP tool windows 431
  - full screen mode 83
  - function braces 344
  - F1 Index Help 49
- ## G
- GDB projects 125
  - GDB (attaching to remote process) 201
  - GDB (multiple session debugging) 200
  - GDB (newer version) 208
  - GDB (setting configurations) 207
  - general options dialog box 564
  - general tab (file ext setup) 583
  - general tab (general options) 565
  - Generate debug 202
  - generate file list 416
  - generate references 226
  - Generic Project type 124
  - Get Dialog Box 422
  - global directory 601
  - Global Find dialog 147
  - global nested directories 601
  - Global Replace dialog box 146
  - Global Substitution Parameters (Code Templates) 160
  - glossary 713
  - GNU C/C++ build methods 192
  - GNU C/C++ projects 124
  - go to annotation 282
  - Go to Bookmark dialog box 276
  - go to definition 211
    - navigation option 596
  - Go To Definition dialog box 497
  - go to reference 211
  - go to reference only lists references 611
  - Grid Settings dialog 541
- ## H
- hash tables editing 450
  - header (printing) 90
  - help 49
    - F1 49
    - help system 49
    - key shortcuts 50
    - supported browsers 50
    - UNIX, Linux, Mac OS X 49
  - Help Index dialog box 49
  - Help Menu 635

- ul style="list-style-type: none; padding-left: 0;">
- hexadecimal display 327
- hide mouse pointer 107
- hide toolbar 63
- highlight line 243
- history 149
  - backup 149
  - network files 149
  - of opened items 150
- History dialog box 150
- history (increase File menu) 661
- history (increase Project menu) 661
- HOME Key Configurations 86
- horizontal scroll bar 107
- host name of firewall 479
- host name of FTP server 475
- host type 475
- hot fixes 51
  - installing 52
  - list of installed 52
  - unloading 52
- hot key 78, 565
- hot-swap debugger 204
- HTML 373
  - beautifying 377
  - browser configuration 374
  - exporting current file 374
- HTML Beautifier Advanced tab 383
- HTML Beautifier Attributes Values tab 381
- HTML Beautifier Comments tab 382
- HTML Beautifier dialog box 377
- HTML Beautifier Indent tab 378
- HTML Beautifier Schemes tab 384
- HTML Beautifier Tags tab 379
- HTML Comments dialog 383
- HTML Formatting Options dialog 375
- HTML toolbar 374
- I**
- iconized windows 538
  - ignore spaces 555
  - Import File List dialog 134
  - import key bindings 105
  - Import options (Refactoring) 356
  - importing files to projects 134
  - Imports 356
  - Imports Menu 547
  - increase file history (File menu) 661
  - increase workspace history (Project menu) 661
  - incremental searching 293
  - Indent access specifier 345
  - indent case 345
  - indent first level of code 345
  - indent for each level 345
  - indent inside block 348
  - indent preprocessing 348
  - indent stand alone comments 346
  - indent style 580
  - indent tab 579
  - indent with tabs 345
  - Indentation tab (C/C++ Formatting) 340
  - indenting tab 344
  - indicator (line/char count) 60
  - indicator (macro recording) 60
  - Info tab (Debugger Options) 202
  - initial directory 475
  - Insert File dialog box 137
  - Insert Literal dialog box 245
  - insert mode 573
  - Insert open paren (tagging tab) 594
  - insert real indent 581
  - Insert selected (Auto-Complete tab) 593
  - Insert space after comma (tagging tab) 595
  - insert toggle 60
  - Insert \* for Javadoc (Comments tab) 587
  - inserted attribute (casing) 376
  - inserted case (attributes) 364
  - inserted case (single word values) 364
  - inserted case (tag) 364
  - inserted tag (casing) 376
  - Inserting characters 245
  - Inserting control characters 91
  - inserting files into buffers 137
  - Installed templates (Locating) 162
  - installing SlickEdit 43
  - installing X11 for Mac OS X 43
  - InstallScript Formatting Options dialog 353
  - invocation options 651
  - invoking ant targets 360
  - ISPF Copy Lines 685
  - ISPF Delete Lines 685
  - ISPF Emulation 679
  - ISPF Emulation Commands 691
  - ISPF Exclude Lines 689
  - ISPF Expose First Lines 685
  - ISPF Expose Last Line 686
  - ISPF Expose Next Level of Code 688
  - ISPF Insert After 684
  - ISPF Insert Before 684
  - ISPF Insert Bounds Ruler 684
  - ISPF Insert Columns Ruler 685
  - ISPF Insert Lines 686, 688
  - ISPF Insert Mask Line 687
  - ISPF Insert Tabs Ruler 688
  - ISPF Insert Text 688
  - ISPF Join Lines 688
  - ISPF Line Command A 682
  - ISPF Line Command B 682
  - ISPF Line Command Delete 682
  - ISPF Line Command Exclude 683
  - ISPF Line Command First 682
  - ISPF Line Command Uppercase 683
  - ISPF Line Commands 683
  - ISPF Line Labels 683
  - ISPF Lowercase Lines 686
  - ISPF Make Data Lines 687
  - ISPF Move Lines 686
  - ISPF Options Dialog Box 679
  - ISPF Overlay Lines 687

- ISPF Repeat Lines 687
- ISPF Select Lines 689
- ISPF Shift Lines Left or Right 683
- ISPF Split Line 689
- ISPF Unsupported Primary Commands 690
- ISPF Uppercase Lines 689

## J

- Java Beautifier dialog 355
- Java compiler 358
- Java Formatting Options dialog 353
- Java Live Errors 358
- Java Options Appletviewer tab 358
- Java Options Classpath tab 358
- Java Options Compiler tab 358
- Java Options Debugger tab 358
- Java Options dialog 358
- Java Options Jar tab 358
- Java Options Javadoc tab 358
- Java Options JRE tab 358
- Java Options J2ME tab 358
- Java Options Live Errors tab 358
- Java Organize imports options 356
- Java projects 125
- Javadoc Beautifier dialog box 355
- Javadoc comments 290
- Javadoc Editor dialog box 356
- JavaScript Formatting Options dialog 353
- JAWS screen reader 83
  - configuration 83
- JCL Formatting Options dialog 403
- Join comments (Comments tab) 588
- JSP TagLib Formatting Options dialog 375
- jumping to a tag 211
- justification dialog box 582
- justified 582
- justify style 582
- J# Formatting Options dialog 353

## K

- keep alive default FTP 478
- keep alive FTP connection 476
- key bindings 101
  - binding macros 446
  - creating 103
  - definitions of terms 101
  - emulations 97
  - export/import 105
  - key message delay 106
  - unbinding 105
  - used in debugging 199
  - viewing associated commands 80
- Key Bindings dialog 604
- key definition 80
- key message 575
- key message delay 106
- key names 78, 573
- key shortcuts in text boxes 85
- Keys Help 99

- Keywords (Auto-Complete tab) 592

## L

- language support 39
- Language tab (Color Coding Setup) 618
- largest file to autosave 603
- Last line is bottom (Comments tab) 586
- leave selected 569
- left and respace 582
- Left and Right (Comments tab) 586
- lexer name 591
- lexer name (creating in VLX file) 671
- libraries 122
- license agreement tab (about slickedit) 47
- licensing options 43
- Line Comment Options 620
- line count indicator 60
- line indicators 60
- line insert 575
- line insert style 243
- line modification flags (auto-restore) 113
- line navigation 215
- line numbers 331
- line numbers (global option) 584
- line selections 236
- Line separator char 463
- lines 243
  - clearing modified line coloring 247
  - click past end of 243
  - current line highlight 243
  - line numbers 331
  - preserving columns 243
  - setting insert style 243
- Lines per Page 474
- Link Window dialog box 75
- Linking to a window 75
- Linux (printing on) 91
- list clipboards 242
- list clipboards dialog box 242
- List errors 194
- List Files dialog box 145
- List files of type 462
- list hot fixes 52
- List Members 179
- List members (tagging tab) 594
- list open files/buffers 74
- list project files 531
- list workspace files 531
- literal characters (inserting) 245
- live errors 358
- load entire file 597
- Load Files dialog box 134
- Load Module dialog box 450
- load partial if larger than 141
- load partial (option) 598
- load tab 140
- load (for different drives) 141
- loading project files 134
- Locals tool window 69

- Locating templates 162
  - Installed templates 162
  - User templates 162
- Location (Code Templates Add New Item dialog box) 162
- Location (Comments tab) 586
- locations for code annotations 279
- locked files 141
- look in 492
- lower case file names 377

## M

- Mac OS X 35
  - format buffer 653
  - printing 91
  - special features 40
- Macro Functions by Category 450
- Macro Menu 537
- macros
  - programmable 450
  - recorded 445
- maintenance and support 51
- make backup files 601
- makefile (build with auto) 193
- makefile (building without) 192
- makefile (custom build command) 193
- makefiles 127
- managing code annotation files 285
- managing project source files 132
- managing projects 124
- managing projects in workspace 123
- managing workspaces 122
- Manually creating a template 162
- margins 582
- margins tab 90
- Match block comment setting (Comment Wrap tab) 535
- match case 492, 568
- match whole word 492
- math (see mathematics)
- mathematics 437
  - document math 439
  - expressions with mixed bases 437
  - math commands 438
  - math commands (examples) 438
  - overflow/underflow 439
  - prime numbers 439
  - using the Calculator 437
- max clipboards 243, 575
- max undo 599
- maximize window 565
- Maximum Symbols (Auto-Complete tab) 592
- Maximum Word completion (Auto-Complete tab) 593
- Members tool window 69
- Memory tool window 70
- Menu Editor dialog box 639
- menu if no selection 77, 591
- menu if selection 77, 591
- Merge dialog box 417
- merging files 417
- Message line 60

- Metadata file reference (Code Templates) 164
- Minimum prefix (Auto-Complete tab) 593
- mixed mode view (debugging) 199
- mode name 583
- Modified Buffers dialog box 48
- modified line flags (auto-restore) 113
- Modify parameter list (C++ Refactoring) 312
- Modify parameter list (Quick Refactoring) 310
- modify (reset line) 600
- modifying selected text 237
- modifying Surround With templates 271
- more tab 572
- mouse pointer 575
- mouse selection 571
- mouseover symbol info 596
- Move dialog box 146
- Move method (C++ Refactoring) 317
- Move static field (C++ Refactoring) 318
- moving code annotations 282
- multi-file diff output dialog box 416
- multi-file undo/redo 305
- Multi-line Comment dialog box 620
- multiple file search and replace 493
- multiple help file access 669
- multiple instances 47
- multiple monitors 83
- multiple monitors (configuring) 652

## N

- Name element (Code Templates metadata file reference) 169
- Name (Code Templates Add New Item dialog box) 161
- Name (Code Templates Add Parameter dialog box) 161
- named sessions (debugging) 200
- namespaces 351
- navigation 211
  - between buffers/windows 73
  - between symbols 211
  - between words 214
  - cursor movements 213
  - go to definition 211
  - go to offset 215
  - go to reference 211
  - in pages and files 213
  - in statements and tags 214
  - pop bookmark 211
  - symbol browsing 217
  - to a specific line 215
- nested directories (global) 601
- network backup 149
- Network Options dialog 627
- never 580
- New Annotation dialog 281
- new configurations 350
- new extension 579
- New Field dialog 283
- New File Tab 135
- New Folder Name dialog 122
- New Formatting Scheme dialog 390



- New Project Tab 459
- New Workspace Tab 460
- New (File) dialog box 135
- next word 214, 575
- no space before paren 344
- no window reordering 575
- none 580
- non-mdi editor control 538
- number lines every 471
- number of copies 471
- Numbers tab (Color Coding Setup) 615
- Numbers tab (Debugger Options) 204
- numeric overflow or underflow 439
- numeric sort 244
- numerical values 377

**O**

- Objective-C Formatting Options dialog 353
- offset navigation 215
- OMG IDL Formatting Options dialog 353
- one file per window 565
- Open dialog box 136
- Open Menu dialog box 639
- Open Other Workspace Menu 508
- open site 479
- Open tab (Project Properties) 519
- Open tool window 66
- Open URL dialog box 137
- open workspace 122
- opening files 135
  - quick open 135
  - UNIX, Linux, Mac OS X 136
  - Windows 136
- opening Unicode files 648
- opening URLs 137
- options 579
  - auto-restore 113
  - search 568
- options for common keyboard keys 86
- Options menu 545
- Options tab (DIFFzilla) 556
- options (for files) 139
- options (using macros to discover/control) 449
- order 244
- Organize All Workspaces dialog 122
- Organize Imports Options 356
- Organize imports (Refactoring) 356
- organize\_imports\_options 356
- organizing projects 120
- Organizing templates 158
- original absolute column 347
- orientation 470
- original relative column 347
- original tab size 345
- OS prompt 81
- OS/390 Assembler Formatting Options dialog 403
- Other menu 485
- other tab 347
- Other tab (C/C++ Formatting) 342

- output file 554
- output style 555
- Output tool window 66

## P

- pad condition 348
- Pad parens (tagging tab) 595
- Parameter element (Code Templates metadata file reference) 170
- Parameter info (tagging tab) 595
- Parameter Information 180
- parameter prompting 261
- Parameters element (Code Templates metadata file reference) 171
- paren style 213, 574
- Parsing configuration (C++ Refactoring) 320
- partial load (fast line count) 598
- Pascal Formatting Options dialog 405
- passive transfers 479
- password 475, 479
- paste 242
  - deselect 571
- Perl Formatting Options dialog 353
- PHP Formatting Options dialog 375
- platform support 35
- PL/I Formatting Options dialog 407
- PL/SQL Formatting Options dialog 407
- port 476, 478, 479
- Pre-defined substitution parameters (Code Templates) 156
- preprocessing for C/C++ 351
- preserve column 243, 574
- Preserve trailing (tagging tab) 595
- preserve width on existing (comment wrapping) 590
- Preview tool window 223
  - what is displayed 224
- Preview (Code Templates Add File dialog box) 160
- prime numbers 439
- print color 470
- print color coding 470
- Print Command 474
- Print Device 474
- Print dialog box 468
- print dialog (text mode) 472
- Print Filter 474
- print hex 470
- print schemes 90
- print (see also printing) 89
- printing 89
  - insert formfeed 91
  - on UNIX, Linux, Mac OS X 91
  - on Windows 89
  - schemes 90
- process buffer 64, 574
- procs and prototypes 211
- Product Updates Menu 636
- profile name 475
- program info tab (about slickedit) 47
- programmable macros 450

- loading 450
- setting variables 450
- Progress 4GL Formatting Options dialog 353
- Project Build Commands tab 515
- Project Configuration Settings dialog 127
- project directories tab 512
- project files 132
  - adding and removing 132
  - creating 134
  - importing 134
  - listing 531
  - loading 134
  - makefiles 127
- Project Files Tab 511
- Project Menu 507
- Project Properties dialog 509
- Project properties (Tools) 513
- Project Tools Tab 513
- Project Tools toolbar 68
- project types 124
  - dynamic languages 125
  - GNU C/C++ 124
  - Java 125
  - other compilers 125
  - Visual Studio 125
- projects 119
  - build settings 130
  - build system options 131
  - command line execution 130
  - configurations 127
  - configuring directories 128
  - configuring tools 128
  - creating 126
  - creating custom types 126
  - defining dependencies 127
  - defining extension-specific 132
  - importing files 134
  - libraries 122
  - loading files 134
  - makefiles 127
  - managing 124
  - managing source files 132
  - managing within a workspace 123
  - organizing 120
  - project types 124
  - setting active 127
  - sharing between workspaces 123
  - specifying command directory 130
  - version control 121
  - wildcards 122
  - with one file 132
- Projects tool window 67
- Prompt for value (Code Templates Add Parameter dialog box) 161
- prompt replace dialog box 494
- Prompt string (Code Templates Add Parameter dialog box) 161
- protect read-only 574
- Proxy Settings dialog 626

- Pull up to super class (C++ Refactoring) 314
- Push down to derived class (C++ Refactoring) 312
- PVCS 424
- Python Formatting Options dialog 353

## Q

- quick create 135
- quick extract method 309
- quick modify parameter list 310
- quick open 135
- quick refactoring 309
  - undo/redo 309
- Quick Refactoring Menu 547
- quick rename 309
- quick replace 293
- quick replace literal with constant 310
- quick search 293
- Quick Start 53
- quotes 377
- quotes (numerical values) 377
- quotes (single values) 377
- quote\_key command 91

## R

- Read List dialog box 146
- read only indicator 60
- read only mode (macros) 538
- read only mode (protect) 574
- read only (open as) 464
- rebuild 360
- recognizing Unicode files 648
- Record width 463
- recorded macros 445
  - binding to keys 446
  - deleting 449
  - execute\_last\_macro\_key 447
  - operations 445
  - recording a new macro 445
  - running 448
  - saving and editing 448
  - using to discover/control options 449
- recording tasks with annotations 287
- Redefine common keys dialog box 607
- redo replacement 305
- redo (multi-file) 305
- refactoring 309
  - convert global to static field 318
  - convert local to field 319
  - convert static to instance method 318
  - create standard methods 320
  - encapsulate field 315
  - extract class 315
  - extract method 311
  - extract super class 316
  - modify parameter list 312
  - move method 317
  - move static field 318
  - pull up to super class 314
  - push down to derived class 312



- quick refactorings 309
  - rename 311
  - replace literal with constant 319
  - test parsing config 320
  - refactoring menu 310
  - Refactoring results 323
  - refer to 336
  - Refer to dialog box 579
  - References tool window 225
    - options 226
  - References view 225
  - references (options) 611
  - Referencing new extensions 336
  - reflow comment dialog box 534
  - reflow next 573
  - reflowing comments 292
  - Regex Evaluator 435
  - Regex Evaluator tool window 435
    - entering expressions 435
    - entering test cases 435
    - options 436
  - Registers tool window 70
  - regular expressions 693
    - Brief 702
    - Brief examples 706
    - evaluating 435
    - minimal versus maximal matching 304
    - search and replace with 301
    - SlickEdit 697
    - SlickEdit examples 702
    - tagged search expressions 303
    - Unicode category specifications 706
    - Unicode character blocks 708
    - UNIX 693
    - UNIX examples 697
  - reinsert after current 598
  - release notes tab (about slickedit) 47
  - Reload on switch buffer 598
  - reload prompt 598
  - remote process (debugging) 201
  - remote to local directory mapping 476
  - remote VM 201
  - Remove Dialog Box 422
  - remove duplicate lines 244
  - remove eof character 600
  - removing files from a project 132
  - removing SlickEdit 44
  - Rename (C++ Refactoring) 311
  - Rename (Quick Refactoring) 309
  - reorder windows 575
  - repeat command on selected dialog box 457
  - replace
    - see also replacing
  - Replace in Files tab (Find and Replace) 495
  - Replace literal with constant (C++ Refactoring) 319
  - Replace literal with constant (Quick Refactoring) 310
  - Replace params in target file (Code Templates Add File dialog box) 160
  - Replace tab (Find and Replace) 495
  - replace toggle 60
  - replacing 293
    - Find and Replace tool window 299
    - minimal vs maximal matching 304
    - quick replace 293
    - replace and c commands 297
    - undo/redo 305
    - using regular expressions 301
    - see also searching
  - reset 343
  - resolve links 476
  - revision one 554
  - revision two 554
  - REXX Formatting Options dialog 407
  - router 479
  - Ruby Formatting Options dialog 353
  - running a program 199
  - running multiple instances 47
  - running recorded macros 448
  - Runtime Filters tab (Debugger Options) 205
  - runtime functions 205
  - run-time libraries 185
- ## S
- safe exit 48
  - same name 603
  - save 599
  - save after period of inactivity 602
  - save after period of time 603
  - Save As dialog box 137
  - save as (using) 137
  - Save Failed dialog box 138
  - save files on loss of focus 600
  - save macro dialog box 538
  - Save Multi-File Diff Output dialog box 417
  - save password 475
  - save settings 343, 472
  - save tab 141
  - save to different directory 603
  - saving files 137
  - saving recorded macros 448
  - SBCS/DBCS 647
  - SCC 423
    - configuration 424
    - opening a project 424
  - schemes tab 348
  - Schemes (Beautify) 348
  - schemes (colors) 112
  - schemes (printing) 90
  - screen layout 59
  - screen management 83
  - scroll bars 107
  - scroll style 574
  - scroll when 574
  - search
    - see also searching 293
  - search for 492
  - Search Menu 489
  - search options 568

- Search Options dialog box 492
- search order 667
  - configuration files 667
  - executable files 667
- Search Results tool window 67
- search tab 567
- search (go to offset) 215
- searching 293
  - classes 65
  - Find and Replace tool window 299
  - find and slash commands 294
  - Find Symbol tool window 301
  - for binary characters 303
  - history retrieval 569
  - incrementally 293
  - initialization options 569
  - members 65
  - minimal vs maximal matching 304
  - multiple files 493
  - quick search 293
  - syntax-driven 300
  - tagged search expressions 303
  - undo/redo 305
  - using regular expression syntax 568
  - using regular expressions 301
  - see also replacing
- seek dialog box 215
- seek offset position 215
- Select a Buffer dialog box 75
- Select a Tag dialog 596
- Select Alias File dialog 259
- Select Colors to Update dialog 112
- select first 77, 591
- Select Mode dialog box 335
- Select Text to Paste dialog 486
- selecting a code block 271
- Selecting a mode 335
- selecting text 235
  - block insert mode 236
  - blocks (columns) 235
  - characters 235
  - commands 237
  - counting lines/characters 241
  - drag-and-drop 242
  - enumeration 241
  - keyboard shortcuts 237
  - lines 236
  - modifying a selection 237
  - selection options 242
- selection only 470
- selection restriction 343
- selection styles 570
- Selection (Comment Wrap tab) 535
- selection (creating alias from) 262
- Selections tab 570
- Selective Display 328
- Selective Display dialog box 500
- Selective Display Function definitions 501
- Selective Display Multi-level 502
- Selective Display Preprocessor directives 502
- Selective Display Search Text 501
- Selective Display toolbar 68
- selective display (auto-restore) 113
- server type 475
- service name 480
- Set Attributes dialog 146
- set command 656
- set scroll style 107
- Set Variable dialog box 450
- SETemplate element (Code Templates metadata file reference) 172
- setting active project 127
- setting fonts and colors 109
- Settings tab (Debugger Options) 203, 527
- SFTP 431
- sftp tab 479
- sharing code annotations 280
- sharing projects 123
- shell prompt 81
- shell scripts 40
- shelling 81
- shortcuts for build and rebuild 360
- shortcuts in text boxes 85
- Show categories (Auto-Complete tab) 593
- show changes 554
- Show comments (Auto-Complete tab) 593
- Show comments (List members-tagging tab) 594
- Show comments (Parameters-tagging tab) 595
- show eof character 598
- Show expanded text (Auto-Complete tab) 593
- show hidden files (Open dialog) 465
- show hidden files (Save As dialog) 467
- Show icons (Auto-Complete tab) 593
- Show info for symbol under mouse 596
- Show light bulb (Auto-Complete tab) 593
- Show list of matches (Auto-Complete tab) 593
- Show symbol declaration (Auto-Complete tab) 593
- Show symbol info (tagging tab) 596
- single word values 376
- Slick-C Formatting Options dialog 353
- Slick-C Stack 67
- Slick-C Stack tool window 67
- SlickEdit command line 79
  - activating 79
  - common commands 82
  - history 79
  - keyboard shortcuts 80, 85
  - prompting 81
  - prompting (option) 566
  - shelling 81
- SlickEdit dialog 47
- SlickEdit File Manager 145
- SlickEdit regular expressions 697
  - examples 702
- smart next window 73, 575
- SmartPaste 252
- smooth 574
- snippets 257

- soft wrap 582
- Sort buffer 244
- sort commands 244
- sort dialog box 244
- Sort on selection 244
- Sort within selection 244
- sort (selection) 244
- sorting text 244
  - commands 244
- SortOrder element (Code Templates metadata file reference) 173
- sort\_on\_selection command 245
- Source file name (Code Templates Add File dialog box) 160
- space between 470
- Space inserts space (tagging tab) 594
- special characters tab 571
- specifying tabs for a file extension 252
- spell check 427
  - in multiple files 428
  - operations 427
  - options 429
  - running 427
- Spell Check Files dialog box 428
- Spell Check Menu 548
- spell options dialog box 429
- spelling dialog box 427
- spill file (and buffer cache) 141
- spill file (option) 577
- Split line comments (Comments tab) 587
- Split strings (Comments tab) 588
- SQL Server Formatting Options dialog 407
- ssh executable 480
- ssh tab 479
- stack class 180
- stand alone comments 346
- Standard Open dialog box 136
- Standard toolbar 68
- Start in column (Comments tab) 586
- start on file 573
- Start wrapping on line (Comment Wrap tab) 589
- starting FTP connection 432
- starting SlickEdit 47
  - for the first time 47
  - multiple instances 47
- state file (alternate) 652
- Statement Level Tagging 182
- statement navigation 214
- status line 60
- stopping FTP connection 433
- String editing (Comments tab) 588
- string editing (comments) 292
- Strings tab (Color Coding Setup) 616
- strip trailing spaces 600
- structure matching 212
  - setting match style 213
  - viewing/defining 212
- submenus 77
- Substitution parameters (Code Templates) 156
- subsystem 480
- Subversion 425
- summary of keys 97
- support (for SlickEdit) 51
  - contact info 52
- surrogate support 649
- surround selection 270
- Surround With 270
  - commands 272
  - modifying templates 271
- surround with block statement 267
- Surround With dialog 270
- surrounding text 267
- Symbol Browser Filter Options dialog box 231
- Symbol browser (auto-restore) 113
- symbol browsing 217
  - base/derived classes 230
  - Class tool window 217
  - Current Context tool window 220
  - Defs tool window 221
  - Find Symbol tool window 222
  - Preview tool window 223
  - References tool window 225
  - symbol uses/call tree 229
  - Symbols tool window 227
  - Tag Properties tool window 233
- symbol info (show under mouse) 596
- symbol navigation 211
  - between multiple instances 212
  - browsing symbols 217
  - Find Symbol tool window 212
  - more methods 212
- Symbol Properties tool window 233
- symbol use 229
- Symbol Uses/Calling tree dialog box 229
- Symbol view 227
- symbols
  - comparing in one file 414
  - comparing in two files 415
- Symbols tool window 227
  - base/derived classes 230
  - filter options 231
  - filtering symbols 227
  - options 228
  - uses/calling tree 229
- Symbols (Auto-Complete tab) 592
- sync backgrounds (colors) 623
- sync extension 343
- sync vertical line (comment wrapping) 590
- syntax expansion 265
- Syntax expansion (Auto-Complete tab) 592
- Syntax expansion (Overview) 265
- syntax indent 251, 345, 580
- syntax-driven search 300
- system command line invocation options 651
- system configuration files 664
- system requirements 43

## T

### tab

- advanced 476
- auto save 142
- begin-end style 343
- comments 346
- diffzilla files 555
- exit 576
- file filters 143
- general 475, 477, 565, 583
- indent 251, 579
- indenting 344
- load 140
- margins 90
- more 572
- other 347
- save 141
- schemes 348
- ssh 479
- stops 251
- virtual memory 576
- word wrap 581

Tab cycles choices (Auto-Complete tab) 593

tab groups 63

Tab inserts longest (Auto-Complete tab) 593

tab key 251

tab key reindents 580

tab size 345

tabs 581

- enable indent with tabs 251
- indent 581
- set for current file 252
- setting spacing 251
- tab indents selection 252

tabs dialog box 252

tag files 351

- auto-create 183
- auto-updated 186
- building 183
- categories 186
- for libraries 184
- rebuilding 187
- search order 187

tag navigation 214

tagging 179

- see Context Tagging

tagging cache 577

tagging libraries 184

tagging tab

- List members 594
- Parameter information 595
- Show symbol info 596
- Update after idle 596

tagging (minutes before retagging) 611

Tags tab (Color Coding Setup) 621

Target file name (Code Templates Add File dialog box) 160

Tcl Formatting Options dialog 353

Template file (Template Manager dialog box) 159

Template Manager dialog box (Code Templates) 158

Template Options dialog box (Code Templates) 160

TemplateContent element (Code Templates metadata file reference) 174

TemplateDetails element (Code Templates metadata file reference) 175

Templates list (Code Templates Add New Item dialog box) 161

Templates list (Template Manager dialog box) 159

test parsing configuration 320

testing regular expressions 435

text box editing keys 85

text compare 413

text editing 235

- block insert mode 236
- cut, copy, move 242
- inserting characters 245
- lines 243
- selections 235
- sorting 244

Text Mode Print dialog box 472

text selection 581

third-party workspaces 124

Threads tool window 70

three way merge editing 417

timeout 476, 478

toggle 60

Tokens tab (Color Coding Setup) 614

tool tip delay 574

tool tips 574

tool windows (overview) 63

Toolbar Control Properties dialog 506

Toolbar Customization Categories Tab 504

Toolbar Customization dialog 503

Toolbar Customization Options Tab 505

Toolbar Customization Toolbars Tab 503

toolbars 63

- displaying 63
- docking 63

Toolbars (Customizing) 63

Tools Menu 545

Tools Options Menu 563

Tools tab (Project Properties) 513

Tools toolbar 68

tools (for project configuration) 128

Tornado workspaces 124

transfer type 475

truncation 584, 598

two up 470

types of code annotations 279

types of projects 124

## U

UCN 649

unattended installation 44

unbinding a key 105

uncommenting 289

undo replacement 305

- undo (multi-file) 305
  - Unicode 647
    - category specs for reg expressions 706
    - character blocks for reg expressions 708
    - converting to UCN 649
    - file recognition 648
    - implementation 650
    - limitations 649
    - opening files 648
    - surrogate support 649
  - uninstalling SlickEdit 44
  - Unit testing 357
  - Unit Testing tool window 67
  - UNIX regular expressions 693
    - examples 697
  - unix temp environment 577
  - UNIX (printing on) 91
  - UNIX (recommended fonts) 110
  - Unlist Search dialog box 146
  - unload hot fix 52
  - Unlock Dialog Box 422
  - Unsurround Block 273
  - update 579
  - Update after idle (Auto-Complete tab) 593
  - Update after idle (tagging tab) 596
  - update manager 51
  - updates (to SlickEdit) 51
  - upload filename case 476, 478
  - url mappings 372
  - URL Mappings dialog 626
  - URL (opening) 137
  - use block cursor 575
  - use child directory 602
  - use firewall/proxy 476
  - use hanging indent (comment wrapping) 590
  - use smart merge 555
  - Use style (Comments tab) 587
  - use undo 599
  - user configuration files 663
  - user id 475, 479
  - User interface 59
  - user preferences 95
  - User templates (Locating) 162
  - Using Version Control 421
  - UTF-8 647
- ## V
- Value (Code Templates Add Parameter dialog box) 161
  - Variable editor 450
  - Variable Editor dialog box 539
  - VBScript Formatting Options dialog 405
  - version control 421
    - advanced settings 423
    - configuration 422
    - CVS 425
    - overview 421
    - PVCS 424
    - SCC 423
    - setting up command line systems 423
    - Subversion 425
      - using 421
      - workspaces and projects 121
    - version control command setup dialog box 423
    - Version Control Menu 546
    - Version Control Setup dialog box 422
    - vertical scroll bar 107
    - VHDL Formatting Options dialog 405
    - view and filter code annotations 282
    - View Menu 499
    - view special characters 327
    - view text symbols 327
    - View text symbols (Special Characters tab) 571
    - viewing accessible members 232
    - viewing symbol properties 229
    - viewing symbol references 225
    - viewing symbol uses 229
    - virtual memory tab 576
    - visible lines only 470
    - Visual Basic Formatting Options dialog 405
    - Visual C++ Setup dialog box 192
    - Visual Studio projects 125
    - Visual Studio workspaces 124
    - VLX file 671
      - creating a lexer name 671
      - modifying to change color definitions 671
    - VPW extension 115
    - vsbuild to compile files 192
    - VSLICK 655
    - VSLICKALIAS 655
    - VSLICKBACKUP 655
    - VSLICKBIN 655
    - VSLICKBITMAPS 655
    - VSLICKCONFIG 655
    - VSLICKLOAD 656
    - VSLICKMACROS 655
    - VSLICKMISC 655
    - VSLICKPATH 655
    - VSLICKRESTORE 655
    - VSLICKTAGS 655
    - VSLICKXNOBLINK 656
    - VSLICKXNOPLUSNEWMSG 656
    - VSLICKXTERM 656
    - vslick.sta 651
    - vslick.stu 651
    - vsmktags 186
    - VST 656
- ## W
- Watch tool window 70
  - Web Browser Setup dialog box 374
  - web development 363
  - welcome 29
  - what is key 80
  - where is command 80
  - wildcards 122
  - window font dialog box 631
  - window left margin 575
  - Window Menu 631

- window ordering 575
- windows temp environment 577
- Windows (printing on) 89
- Windows 3.1 575
- word boundary 583
- word chars 591
- Word completion (Auto-Complete tab) 592
- word help 49
- word help file name 669
- word navigation 214
- word wrap 582
- word wrap tab 581
- workspace files (auto-restore) 113
- workspace files (listing) 531
- workspace history list 661
- Workspace Properties dialog box 508
- workspaces 119
  - creating 123
  - managing 122
  - managing projects 123
  - opening and closing 122
  - sharing projects 123
  - third-party 124
- workspaces and projects 119
- wrap 493
- wrap at beginning/end 568
- wrap line length 599
- wrap long lines 582
- wrapping (comments) 292
- Write List dialog box 135
- Write Selection dialog box 135

## X

- Xcode build methods 193
- Xcode workspaces 124
- XEDIT Line Commands 690
- Xft font support (disable) 651
- XML 363
  - beautifying 365
  - DTD caching 372
  - toggle between begin/end tags 373
  - turn off auto-validate 662
  - XMLdoc Editor 364
- XML Beautifier Advanced tab 371
- XML Beautifier Attributes Values tab 369
- XML Beautifier Comments tab 369
- XML Beautifier dialog box 365
- XML Beautifier Indent tab 365
- XML Beautifier Schemes tab 371
- XML Beautifier Tags tab 366
- XML encoding 647
- XML Formatting Options dialog 363
- XML toolbar 363
- XMLdoc comments 291
- XMLDOC Editor dialog box 364
- xml/html formatting 387
  - content wrap 393
  - enabling 387
  - enabling for current document 388

- schemes 388
- settings 392
- tag layout 395
- tags 391
- XML/HTML Formatting dialog 387
- XML/HTML formatting menu 387
- XML/HTML Formatting Scheme Configuration dialog 625
- XOR operator 438
- X11 installation 43

## Symbols

- .bak 601

## Numerics

- 3 Way Merge Setup dialog box 554