

Welcome to SlickEdit 2021

Welcome to SlickEdit 2021

SlickEdit Inc.
PO BOX 1953
Clayton, NC 27528 USA

1.919.473.0070
1.800.934.EDIT (USA)
1.919.473.0080 fax

www.slickedit.com [http://www.slickedit.com]

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. 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 1988-2021 SlickEdit Inc.

SlickEdit, Visual SlickEdit, Clipboard Inheritance, DIFFzilla, SmartPaste, Context Tagging, Slick-C, and Code Quick | Think Slick are registered trademarks 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.

SE-1301-061108

Introduction

This chapter describes how to install SlickEdit and provides information about new features, documentation, supported languages, product support, and performance tuning.

Getting 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.

Learn About Our 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 features we think are particularly cool, click **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

These strategies will help you write more code, faster than you ever have before:

- **Use workspaces and projects to manage your code** - By adding your files to a workspace and project, you can quickly open your files without specifying a path with the Open or Files tool window. The Projects tool window lets you organize files in folders based on file type or directory. (Pro only) SlickEdit uses Context Tagging® to build a database of the symbols in your source files. All of the files within a workspace are tagged, allowing for more accurate completion information and rapid navigation. For more information, see [Context Tagging Features](#).
- **Keep your hands on the keyboard** - Time is wasted each time you reach for the mouse. SlickEdit contains 14 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 keystrokes you type, including: completions, syntax expansion, aliases, macros, and code templates. For information about these features, see the topics in the [Editing Features](#) chapter.
- **(Pro only) Rapidly navigate code** - Instantly jump from a symbol to its definition or declaration 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](#) and [Symbol Browsing](#) for more information.

Note

The Standard edition support navigating to global symbols using CTags. See [Building CTags Based Tag Files](#) 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(Pro only), symbols(Pro only), 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(Pro only) are just a few of the automatic formatting features in SlickEdit. For more information, see the topics in the [Editing Features](#) chapter.
- **Time-saving utilities** - SlickEdit provides many utilities for working with your code, such as DIFFzilla®, 3-Way Merge(Pro only), 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 with 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.

Quick Start

SlickEdit offers a **Quick Start Configuration Wizard** to help configure the most commonly changed options. Use this to get up and running with SlickEdit as quickly as possible. It also helps you set up a workspace and project, which is essential for getting the most out of SlickEdit.

The User Guide also includes the [Quick Start](#) guide. It describes how to set the options featured in the **Quick Start Configuration Wizard** plus a few more common user preference settings. It also describes how to quickly create a workspace and project, if you want to do that without using the wizard.

Register Your Product

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 main menu, click **Help → Register Product**, then follow the steps indicated.

Get Maintenance & Support

(Not in Community edition)Subscribe to Maintenance and Support Service to receive the following benefits:

- Unlimited technical support via telephone or e-mail for 12 months.

New Features and Enhancements

- Access to new releases, upgrades, 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, click 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) [mailto:sales@slickedit.com]).

New Features and Enhancements

This section describes the new features and enhancements in SlickEdit®.

Language Support

- New Julia Language support
 - Color coding
 - Smart editing features including smart indenting, SmartPaste(TM), and reindenting on Tab
 - Support for translated backslash unicode abbreviation names to unicode characters
 - Defs tool window lists tags and supports displaying statements
 - Tab key translation support for backslash unicode character specifications ("\\alpha<Tab>")
 - Single and multi-file project support
 - Partial support for Context Tagging(TM)
 - Interactive REPL support (Read-Eval-Print-Loop)
- TypeScript Enhancements
 - New beautifier
- New Terraform Language support
 - Color coding
 - Smart editing features including smart indenting, SmartPaste(TM), and reindenting on Tab
 - Defs tool window lists outer block headings for navigation
- C++ and Objective-C
 - Tagging supports new syntax introduced in C++20.
 - Support for tagging Objective-C function blocks.
 - Numerous other improvements to Objective-C context tagging.
 - Added Advanced C/C++ Advanced Option to treat nested structs like they are scoped in ANSI-C. This can help when working with mixed code.
- Python Enhancements
 - Updated run-time tag file to Python 3.9.1

- Swift Enhancements
 - Interactive REPL support (Read-Eval-Print-Loop)

Project Support

- Enhanced wildcard support
 - Improved Visual Studio wildcard project support (i.e. csproj). Now automatically updates Project tool window, tag file, and file list in the background. No need to explicitly Refresh the Project tool window and/or rebuild the tag file.
 - When adding new files to the project, no explicit filenames are added to the project if they match existing wildcards.
 - Improved updating when new files matching wildcards are added while outside of SlickEdit.
 - The wildcard cache is saved when you close the workspace and restored when you open the workspace. For large wildcard projects, this can significantly reduce startup time when restoring the Project tool window. After opening the workspace, the wildcards are updated in the background.
 - Wildcards are updated in the background except when the wildcard cache is empty.
- Improved Quick-Start project wizard
 - Quick-Start configuration lets you directly open projects or workspaces from other tools.
 - Project > Open Workspace... adds a new smart dialog to help you discover that SlickEdit can open projects and workspaces from other tools.
- Improved New Project dialog
 - Project > New... dialog is now resizable.
 - Project > New... dialog adds ability to directly add a tree of source files, recursively or as a wildcard.
 - Project > New... will set the project working directory if the source tree added differs from the project directory.

Find and Replace

- Improved ability to cancel a foreground search/replace. If your search is taking too long (more than 5 seconds), press any key (i.e like Esc) and you will be prompted to cancel. This works great for very slow regular expressions. For a foreground search or replace, keys are not checked until 5 seconds have gone by (status line starts showing percentages). Checking for keys significantly slow down the search which is why keys aren't checked for initially.
- Improved ability to cancel a foreground multi-file search/replace. If your search is taking too long, press any key (i.e like Esc) and you will be prompted to cancel. This works great for very slow regular

expressions. Note that a multi-file search and replace is always done in the foreground. Also, when you do a background multi-file search, currently open files are searched in the foreground.

- Added threading support for "List current context" search option when searching multiple files.
Significantly faster if less than half the files have a match.

Files

- Open tool window now has an option to show indicator for files which are read-only on disk.
- Open tool window now has an option to show indicator for files which are writable on disk.

DIFFzilla

- Multi-file diff now has a size limit to initially enforce a byte-by-byte compare rather than an actual diff.
- You can still diff by clicking the Diff button in the multi-file diff dialog.
- Next Diff and Prev Diff are now much faster.
- Multi-file diff now much faster on extremely large files.
- Added support for horizontal scrolling using mouse.

Version Control Enhancements

- General version control enhancements
 - Much easier to set the version control system for default, current workspace, and project.
 - Old separate list of SCC providers removed.
 - New lists make it easy to see which systems SlickEdit has specialized support for, SCC support for, and old-style command line support for.
 - Version Control menu now at top level.
 - Version Control Configuration now in its own category in Options dialog.
 - Diff dialog now built in to Compare Workspace/Project/Directory with version control.
 - Shelving now supports adding a file to any existing shelf file, not just the ones listed in the menu.
 - Version Control Configuration setup re-tooled
- Git specific enhancements

- Support for the following now integrated into SlickEdit
 - Checkout
 - Support for the following now integrated into SlickEdit
 - Supports listing branches available locally, or listing all remote branches.
 - Has support to checkout a new branch.
 - Stash
 - Pop
 - Pull
 - Blame

Tagging

- New History category added to Auto-Complete which shows recent and frequently used completions you have used in the current scope.
- Auto-Complete now supports suggesting completions for keywords and symbols you have made minor typos in.
- Key parts of the Auto-Complete engine have been rewritten in native C++ for better performance.
- Added support for tagging source files in compressed tar archives.
- Add `symbol-info-help` command to bring up symbol information like mouse-over does for symbol under cursor (Ctrl+Alt+Comma)
- Add `codehelp-update-context-stats` for reporting local context tagging performance statistics
- Add `codehelp-tag-file-stats` for reporting tag file size statistics for debugging performance issues

Highlight Tool Window

- New feature for highlighting word patterns
- Allows you to quickly add words, strings, or regular expressions to highlight
- Allows you to define multiple Highlight profiles
- Allows you to select several style options for how to do the highlighting
- Word patterns can be case-sensitive, case-insensitive, or language dependent

- Word patterns can be restricted to a particular language mode
- Works seamlessly with very large files
- Uses extended color palette which adapts to the current color profile
- Has option to show scroll bar markup for highlighted items

Apearance Enhancements

- Application Theme
 - Automatically detect Windows 10 Dark app mode and use our Dark theme
 - Improved Dark application color theme scroll bar colors to make it more visible
- Fonts
 - Added per-language configuration for editor, minimap, and diff font and size
 - The Window > Font... dialog is now resizable
 - Font ligature support added for Windows 32-bit
 - Font ligature support added for Linux 64-bit (Qt5 version only, Request Qt5 Linux installer)
 - Improved Appearance > Fonts configuration dialog
- Colors
 - Add ability to have a color scheme inherit settings from a predefined color scheme
 - Added new color schemes: Charcoal, Ash, Stone, and Onyx
 - Added ability to change color profile per-buffer (View > Color...)
 - Added ability to change color profile for mini-map per-buffer (right-click > Color...)
 - Added per-language configuration for editor and minimap color profiles
 - Enhanced Color dialog to preview all colors in the list of colors
 - Added setting to Color dialog for default mini-map color profile
 - Improved code for resizing the Color configuration dialog
 - Added ability to import color themes from Visual Studio, Xcode, VSCode, and Eclipse
- Symbol coloring
 - Each color profile now has colors defined for symbol coloring rules. This mitigates the need for a

relationship between color profiles and a compatible symbol coloring scheme, as all built-in symbol coloring schemes are now compatible with all built-in color profiles.

- Each color profile also has a palette of symbol coloring colors, selected as color-safe foreground colors tuned to the background canvas color of each color profile. These can be used when adding new symbol coloring rules instead of defining colors directly in the symbol coloring scheme (which may or may not be compatible with all color profiles).
- Improved code for resizing the Symbol Coloring configuration dialog.
- Add option to "Override Library Symbols" so that identifiers normally color coded as "Library Symbol" or "User Defined Keyword" can be colored by Symbol Coloring if their definition is found by Context Tagging.
- Tree controls and tool windows
 - Tree controls now support smooth horizontal scrolling. Examples include the Defs tool window, Class, Projects, Symbols, Bookmarks, Annotations, References, and most Debug tool windows.

Printing

- Added ability to select a color profile for printing.
- This makes it easier to print color or grayscale even if you are using an editor color profile with a dark background.

F1 Edit Window Help Enhancements

- When you press F1 on a function name or member name, SlickEdit will attempt to determine the package/namespace/class that the current word belongs to. The class (possibly with namespace/package) is added to the search URL so you are much more likely to get help on the right member. This is very helpful for highly typed languages like C#, Java, and C++.
- Custom help URLs can be configured per Language. Multiple URLs can be configured based on what is found at the cursor.

Build Window Enhancements

- Added option to hide VSICKERRORPATH lines to reduce clutter in build window. These lines are hidden by default.
- Added option to hide change directory commands to reduce clutter in build window. These lines are shown by default.
- Improved Terminal and Build window completion support for tar, jar, and unzip commands.

- Build commands with "Clear build window" set now queue up the clear command along with the build command. That way, you can still see all the output from the previous build command until it's finished.

Color Coding Engine improvements

- Previously, you could only specify the color for the entire start and/or end text that was matched. Now you can specify multiple colors using match groups and/or offsets and lengths.

General

- Much improved touch scrolling and touch support.
- By default, when creating a new file without specifying a filename sets the filename to something like "Untitled YYYY-MM-DD". You can turn this new feature off. This makes it easier to identify Untitled files and use backup history. Also this allows you to save untitled files without specifying a name. If you rename (Save As) an Untitled file, you will be prompted whether to delete the temp file.
- Added \$<VarName> color coding for strings which supports them (PHP, PowerShell, Kotlin, Scala, Perl, Dart).
- Improved multiple clipboard support. When you copy text to the clipboard, if the current clipboard is a text clipboard created by another application, that system clipboard is added to the multiple clipboards before your clipboard is added. This allows you to get back to older clipboards created outside of SlickEdit.
- Windows: Added support for pasting FileNameW format clipboard as text. This allows you to use Ctrl+C on a filename in the Windows File Explorer and then paste it into SlickEdit.
- Linux: Improved launcher support. Better support for Gnome launcher. Added menu item to create another instance of SlickEdit. Added commands for quickly editing the various .desktop launcher files (edit_desktop_shortcut, edit_app_shortcut, edit_diff_app_shortcut).
- Find File (not to be confused with Find in Files) now supports matching files, directories, or both.
- Improved multiple instance support. Added -sid <instance-id> invocation option which allows you to specify a specific instance. No instance id specifies the most recently activated instance.
- Bookmarks will now attempt to restore the window the bookmark was set in when using One file per window mode. The Bookmark window id is saved and restored when you restart SlickEdit.
- Added Softwrap line break cache. This helps scrolling performance in situations where there are long lines that are softwrapped.
- Window > Font selection form is now resizable.
- When using multiple cursors, the Tab key can now be used to align multiple lines of text at a common tab stop, making it very easy to align tabular data.

- The copy-to-clipboard command (Ctrl+C) will now detect if there is no selection and the current line is the first line of a block statement, and prompt if you want to copy the entire block, just the current line, or the outer-most surrounding statement. This feature closely mirrors the behavior when using the cut-line or delete-line commands on the first line of a block statement, where you are prompted to just delete the current line, delete the entire block, or unsurround the block.
- New copy-code-block command can be used with the cursor on the first line of a code block to copy the contents of a code block to the clipboard.
- New copy-surround-block command can be used with the cursor on the first line of a statement block to copy the outer-most surrounding statement. This statement can then be pasted automatically triggering Dynamic Surround mode. It is also a very simple way to copy and paste an outer statement.
- XMLDoc comment expansion will skip <returns> for classes, packages, constructors, destructors, and functions with void return types.

Documentation

Documentation for SlickEdit® consists of:

- A fully integrated Help system, including context-sensitive Help, a searchable index, and categorized lists of C and Slick-C® macro functions. See [The Help System](#) for more information.
- (Not in Community edition) The *SlickEdit® User Guide*, which provides the same information as the Help system in PDF format for viewing and printing as a manual.
- (Not in Community edition) Emulation charts in PDF format for the following editors: BBEdit, Brief, CodeWarrior™, CodeWright®, CUA (SlickEdit's default emulation for all platforms but macOS), Epsilon, GNU Emacs, ISPF, SlickEdit (Text Mode edition), Vim, Visual C++® 6, Visual Studio® default, Xcode®, and Eclipse.
- (Not in Community edition) Slick-C® reference material in PDF format, which includes the *Slick-C Macro Programming Guide* and *Slick-C Macro Conventions and Best Practices for End Users*.

PDF documents are located in the `docs` directory on the root of the product CD. After SlickEdit is installed on your computer, these documents are located in the `docs` subdirectory of [`SlickEditInstallDir`].

Documentation for SlickEdit is also available on the SlickEdit Web site at

www.slickedit.com/products/slickedit/product-documentation

[<http://www.slickedit.com/products/slickedit/product-documentation>] .

Documentation Updates/Feedback

In-product documentation is current as of the build date of the product. Revisions to the product documentation are made regularly with the most current version being made available on the SlickEdit Web site (www.slickedit.com [<http://www.slickedit.com>]).

We welcome your comments and suggestions regarding the documentation. Please send feedback to docs@slickedit.com [<mailto:docs@slickedit.com>].

Other Resources

The following additional resources are available:

- **The SlickEdit Community Forums** - Learn more about SlickEdit products and interact with other users at <http://community.slickedit.com>.
- **The SlickEdit® book** - After years of developing code with SlickEdit, expert John Hurst brings his wealth of knowledge to readers in *Professional SlickEdit* (Indianapolis, Wiley Publishing, 2005, ISBN - 978-0470122150).

Documentation Conventions

CUA is the default editor emulation mode for all platforms except macOS which defaults to macOS. CUA emulation and macOS emulation are almost identical. Therefore, key bindings and shortcuts listed in the documentation follow the CUA emulation. Key sequences and mouse clicks are described using the actions performed on a typical Windows or Linux computer. For information on how to perform those actions on other platforms see [Platform-Specific Notes](#).

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

Menus and Dialogs

Instructions for navigating to items accessed from the main menu are written in the form: **MainMenuItem** → **SubMenuItem**.

For example, the text "click **File** → **Open**" indicates that you should first select **File** from the main menu, then select **Open** from the submenu. Brackets are used to indicate that the menu item is a variable. Some menu paths include tree nodes in dialogs. For example, **Tools** → **Options** → **Editing** is a quick way to write "select **Tools** → **Options** from the main menu, then in the Options tree, expand the **Languages** node, the category for the language you want and the language you're using, then select the **Editing** node".

Instructions for using the product make up the bulk of our documentation, 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, variables, and text to be typed by the user are shown in **bold** type.
- User-input variables and placeholders are shown in **bold italic** type.
- File extensions and environment variables 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.

The Help System

The searchable Help system is installed with the product. In addition to information about the SlickEdit application, it provides categorized lists of C and Slick-C® macro functions.

The Help system can be accessed in several ways:

Supported Languages and Environments

- To view a list of all topics in the Help system, from the main menu, click **Help → Contents**. This is useful for browsing topics depending on your needs.
- To look up specific keyword(s) in the Index, click **Help → Index**. Use the Index to find specific information, for example, "changing emulations" or "toolbars". You can also use the index to find information about a specific option in SlickEdit by looking up the label, for example, "One file per window". When you type in the Index, the list of terms is searched incrementally.
- To search the entire Help system for every instance of a word used, click **Help → Search**. This can be particularly useful to discover information about API functions or more obscure topics. Use quotation marks around the term if it contains spaces.
- To invoke the Help entry for toolbars, tool windows, menus, and dialog boxes, press **F1**.

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.

Language	Version	List Members	Auto List	Parameter Compatibility	Code Navigation	Syntax Expansion	Syntax Indent	Smart Paste	Code Beautification	Selective Display	Code Blocks	Expand/Collapse
ActionScript	2.0	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Ada	95	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	
Ant	*	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	
ANTLR	v2	No	No	Yes	No	No	No	No	No	Yes	No	
AppleScript	*	No	No	No	No	No	No	No	No	No	No	
Assembly Language	Windows Nasm, Unix as	No	No	Yes	No	No	No	No	No	No	No	
AWK	UNIX System V, nawk	No	No	Yes	Yes	Yes	Yes	No	No	No	No	
Bourne shell scripts	bash	No	No	Yes	No	No	No	No	No	No	No	
C, C++	K&R C, C++	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	

**Supported Languages and File
Types**

Language	Version	List Members	Auto List	Parameter Compatibility	Code Informati	Syntax	Syntax	SmartPa	Code	Selective	Code	Expand/Collapse
		Parameter	Compatibility	Code	Navigation	Expansion	Indent	Parameter	Navigation	Indent	Beautifie	Display
	ANSI-C, C90 and C99, ISO C++ 98											
C#	1.0, 1.2, 2.0	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
C Shell	BSD, tcsh	No	No	Yes	No	No	No	No	No	No	No	No
CFScript	*	Yes	No	Yes	Yes	Yes	No	No	No	Yes	No	
Ch	*	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	
CICS	2.x	Yes	No	Yes	No	No	No	No	No	Yes	No	
Clojure	*	No	No	No	No	Yes	No	No	No	No	No	
COBOL	74, 85, 2002	Yes	No	Yes	Yes	Yes	No	No	No	Yes	Yes	
D	1.0	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
DB2	*	No	No	Yes	No	No	No	No	No	Yes	Yes	
Diff Patch	Universal or Context diffs	No	No	No	No	No	No	No	No	No	No	
DTD	ISO 8879:19 86 SGML	Yes	No	Yes	Yes	No	No	No	No	Yes	No	

Supported Languages and File Types

Language	Version	List Members	Auto List	Parameter Compatibility	Code Informati	Syntax	Syntax	Code	Selective Code	Expand/Collapse
		Parameter	Compatibility	Code	Navigation	Expansion	Indent	Smart Paste	Beautify	Display
Erlang	*	No	No	No	Yes	Yes	No	No	No	No
Google Go	*	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
Google Protocol Buffers	*	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
Groovy	*	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
F#	*	No	No	Yes	No	No	No	No	No	No
Fortran	Fortran 77, Fortran 90	No	No	Yes	Yes	Yes	No	No	Yes	Yes
Haskell	*	No	No	Yes	No	No	No	No	No	No
High Level Assembler	*	Yes	No	Yes	No	No	No	No	Yes	Yes
HTML, CFML	HTML 4.0	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
IDL	OMG IDL	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
INI, config files	*	No	No	No	No	No	No	No	No	No
InstallIScript	*	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes

Supported Languages and File Types

Language	Version	List Members		Auto List		Parameter Compatibility		Code	Syntax	Syntax	Code	Selective Code	Expand/Collapse
		Information	Parameters	Compatibility	Code	Navigation	Expansion	Indent	Smart Paste	Beautify	Display	Blocks	
J#	2.0	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes		
Java	5.0	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
JavaScript	ECMAScript ECMA-262 (1997)	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
JCL	*	No	No	Yes	No	No	No	No	No	Yes	No		
JSP	*	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No		
Kotlin	*	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes		
Lex	System V and BSD Unix	No	No	Yes	No	No	No	No	No	Yes	No		
Lua	*	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No		
M4	*	No	No	No	No	No	No	No	No	No	No		
Makefile	Unix make, nmake, and gmake	No	No	Yes	No	No	No	No	No	No	No		
Matlab	*	No	No	Yes	Yes	No	Yes	Yes	Yes	No	Yes		
Modula-2	Wirth/ PIM	No	No	Yes	No	No	No	No	No	Yes	Yes		
Objecti	*	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes		

**Supported Languages and File
Types**

Language	Version	List Members	Auto List	Parameter Compatibility	Code Informati	Syntax	Syntax	Code	Selective Code	Expand/Collapse
		Parameter	Compatibility	Code	Navigation	Expansion	Indent	Smart Paste	Beautify	Display
Visual C										
Pascal	ETH and Delphi	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
Perl	Perl 5	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
PHP	5	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
PL/I	ANSI 1976	Yes	No	Yes	No	No	Yes	No	Yes	No
PL/SQL	*	No	No	Yes	Yes	Yes	No	No	Yes	Yes
PowerN P Assembler	*	No	No	Yes	No	No	No	No	Yes	No
Progress 4GL	*	No	No	Yes	No	No	No	No	No	No
PV-WAVE	*	Yes	No	Yes	Yes	Yes	Yes	No	Yes	No
Python	2.0, 3.0	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No
QML		Yes	No	Yes	No	Yes	Yes	No	Yes	Yes
R	*	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
REXX	ANSI X3.274 1996, Object REXX	No	No	Yes	Yes	Yes	No	No	Yes	No

**Supported Languages and File
Types**

Language	Version	List Members	Auto List	Parameter Compatibility	Code Informati	Syntax	Syntax	Code	Selective Code	Expand/Collapse
		Parameter	Compatibility	Code	Navigation	Expansion	Indent	Smart Paste	Beautify	Display
Ruby	1.8	Yes	No	Yes	Yes	Yes	Yes	No	Yes	No
SAS	*	No	No	Yes	Yes	Yes	No	No	Yes	No
Scala	*	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Slick-C	*	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Swift	1.0, 1.1, 2.0	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
System Verilog	IEEE 2009	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No
Tcl	8	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes
Transact-SQL	*	No	No	Yes	Yes	Yes	No	No	Yes	No
TypeScript	*	No	No	No	Yes	Yes	Yes	No	Yes	No
VBScript	2.0	Yes	No	Yes	Yes	Yes	No	No	Yes	No
Vera	*	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes
Verilog	2005	Yes	No	Yes	Yes	Yes	No	Yes	Yes	No
Visual Basic	VBA, QBasic, VB 6.0	No	No	Yes	Yes	Yes	No	No	Yes	Yes
Visual Basic .NET™	9.0	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes

Supported Languages and File Types

Language	Version	List Members		Auto List		Code	Syntax	Syntax	Code	Selective Code	Expand/Collapse
		Parameter Information	Compatibility	Code	Navigation	Expansion	Indent	Smart Paste	Beautify	Display	Blocks
VHDL	IEEE 1076, VHDL-2006	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No
Windows batch files	*	No	No	Yes	No	No	No	No	No	No	No
Windows Powershell	2.0	No	No	Yes	No	No	No	No	Yes	No	
x86 Assembly	Nasm	No	No	Yes	No	No	No	No	Yes	No	
XML, XSD	1.0, 1.1	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	
Yacc	System V and BSD Unix	No	No	Yes	No	No	No	No	Yes	No	
YAML	*	No	No	No	No	Yes	No	No	No	No	

Versions for languages marked with "*" indicate that best efforts are made to keep the language up to date but no specific version is supported.

Special Features for macOS

The following features are available for programmers using macOS:

- Xcode project support

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

See [macOS Notes](#) for information on macOS 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: 10/28/97 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.gif";
        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 profile 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:

- BBEdit
- Brief
- CodeWarrior
- CodeWright
- CUA (SlickEdit's default for all platforms but macOS)
- Epsilon
- GNU Emacs
- ISPF
- macOS (SlickEdit's default for macOS)
- SlickEdit (Text Mode edition)
- Vim
- Visual C++ 6
- Visual Studio default
- Xcode
- Eclipse

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
- Git
- Mercurial
- MKS Source Integrity®
- Perforce
- PVCS®
- RCS
- SCC
- StarTeam®
- Subversion
- TLIB
- Visual SourceSafe®

Installation

Installing SlickEdit®

Installation files are available on the SlickEdit website on your Registered Projects page. Visit www.slickedit.com [<http://www.slickedit.com>], enter your user name and password, then click the **Login** button to login. Once logged in, select **My Account → Registered Products**. The products you have purchased are listed along with an icon to download the installer and an icon to download a license file. Use the information below to install SlickEdit® on your Windows, Linux, UNIX, or macOS platform.

Note

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

Windows

Complete the following steps to install SlickEdit® on a computer running Microsoft Windows.

1. Download the `.msi` file from your Registered Products page (see above).
2. Double-click the `.msi` to begin the installation.

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 v14.0.0 and SlickEdit v14.0.1 installed, but you cannot install two copies of SlickEdit v14.0.1.

Linux/UNIX

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

1. Download the `tar.gz` file from your Registered Products page (see above).
2. Untar the file using a command like, "`tar -xvfz installfile.tar.gz`". If the tar command does not support the z option, you will need to uncompress the file first.
3. Start the installation at the prompt: # `./vsinst`. You will be prompted for the directory into which SlickEdit will be installed.

Mac

Complete the following steps to install SlickEdit® on a computer running macOS:

Licensing (Pro and Standard edition only)

1. Download the .dmg file from your Registered Products page (see above).
2. Double-click `slickedit.dmg`. A disk image is mounted, with Finder™ displaying the contents.
3. Drag the SlickEdit icon into the **Drag Here** folder, which is a shortcut for the Applications folder. This begins the installation.

The default installation path is /Applications.

Unattended Installation

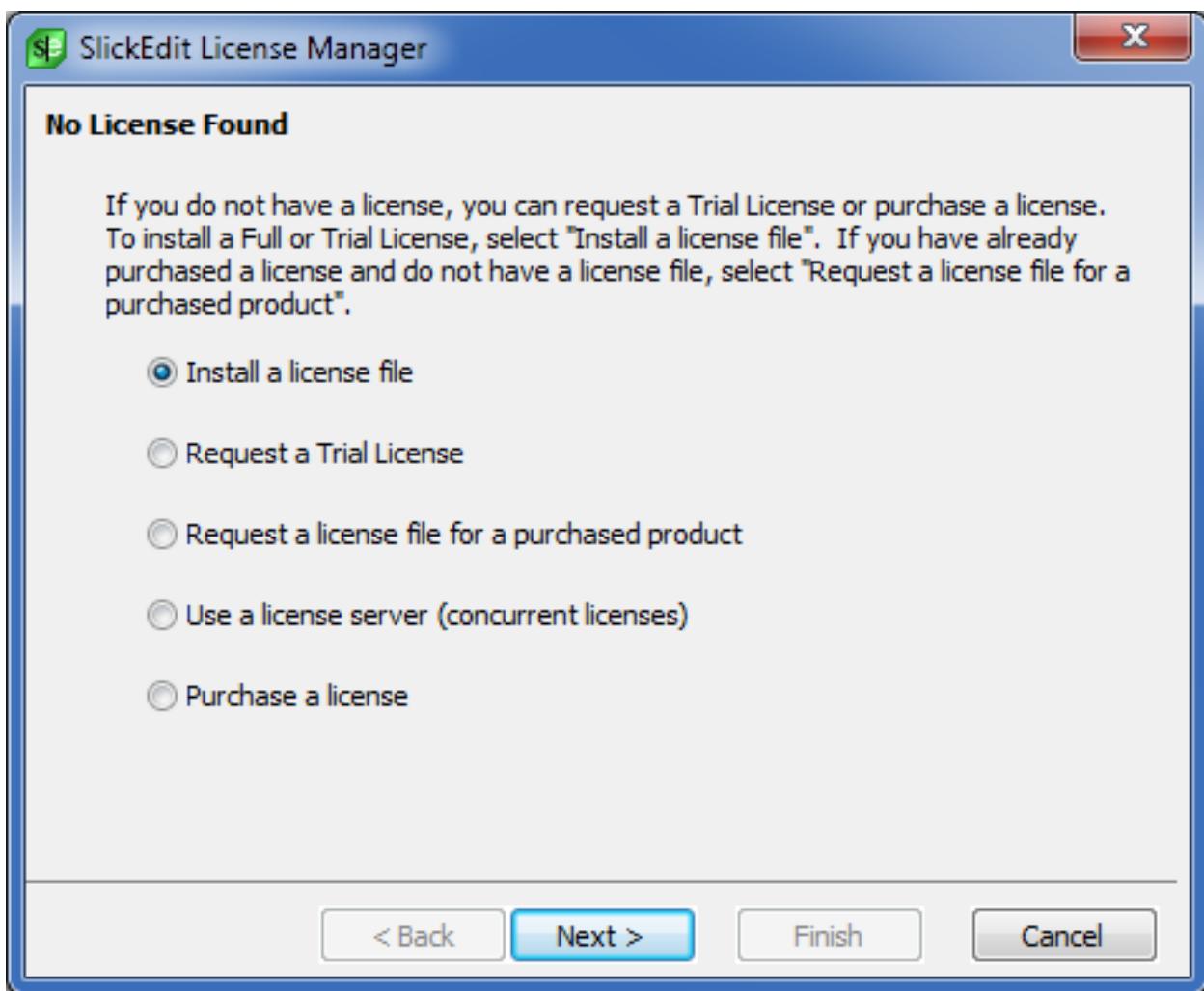
Using command line switches and arguments, SlickEdit® can be installed in an unattended manner. This is useful for network administrators to deploy SlickEdit on multiple client machines with standardized settings. Instructions can be downloaded in PDF format from www/slickedit.com/unattended [http://www/slickedit.com/unattended].

Licensing (Pro and Standard edition only)

SlickEdit uses its own licensing system to manage licenses for both Named User and Concurrent User licenses. Named User licenses are stored on the local machine. Concurrent User licenses use a license server to access a license.

License Manager

When SlickEdit is run, it checks for a license. If one can't be found, the SlickEdit License Manager wizard is run. You can also manually run the SlickEdit License Manager by selecting **Help → Licensing → License Manager**.



The SlickEdit License Manager provides the following options:

- **Install a license file** - Copies a downloaded license file to the location SlickEdit uses to store the file.
- **Request a Trial License** - To try out SlickEdit, click the option to obtain a Trial License. This will take you to a Web page where you can register for a trial. A license key will be e-mailed to you. A trial can be converted to a Full License by entering a Full License key at any time in the SlickEdit License Manager, or by downloading a Full License file.
- **Request a license for a purchased product** - If you have already purchased a license for SlickEdit, you can download a license file to this computer from the SlickEdit Web site.
- **Use a license server (concurrent licenses)** - Use this to configure concurrent licenses using a license server.
- **Purchase a license** - To buy a Full License, visit the SlickEdit Web site at www.slickedit.com [http://www.slickedit.com] or select this option, which will take you directly to the product page for SlickEdit.

Concurrent User Licenses

Licensing (Pro and Standard edition only)

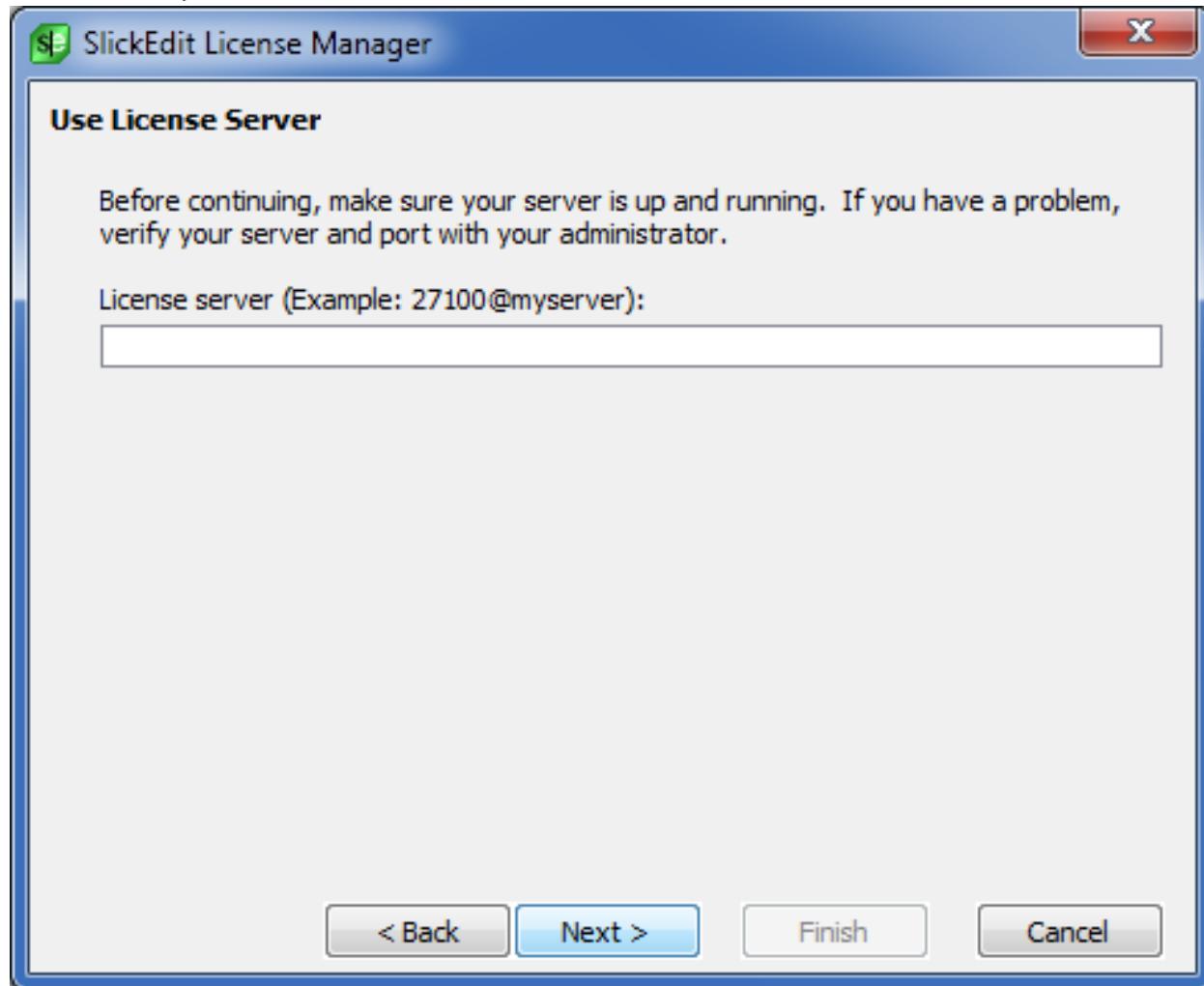
Concurrent License users will find complete instructions for setting up a license server at www/slickedit.com/selicense [http://www/slickedit.com/selicense].

When you run SlickEdit, if a license is not found, the SlickEdit License Manager wizard is run. Select **Use a license server (concurrent licenses)** to configure access to a concurrent license server.

Note

SlickEdit 2011 (v16) will not work with the Flex license servers used in previous releases. A new license server must be set up for this and subsequent versions.

You are prompted to enter the port address for the license server. Enter the port@hostname for the server running the SlickEdit License Server. Your system administrator should be able to provide this information for you.



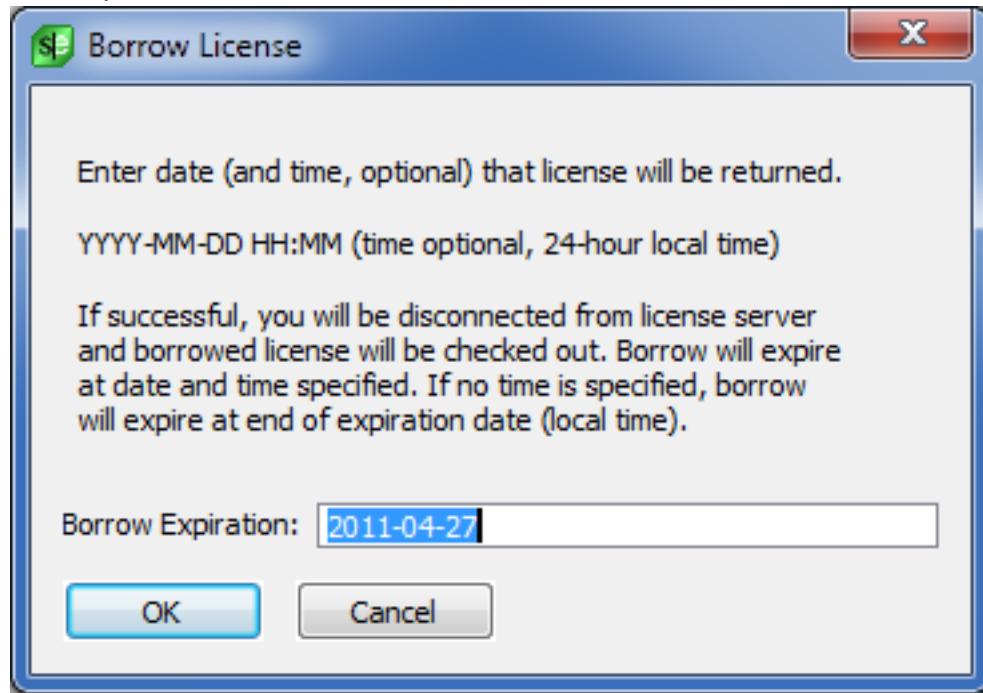
The next screen should say, "Successful license checkout". If so, you are finished. If not, try the following:

- Make sure that your machine can access the server. Confirm this using the ping utility.

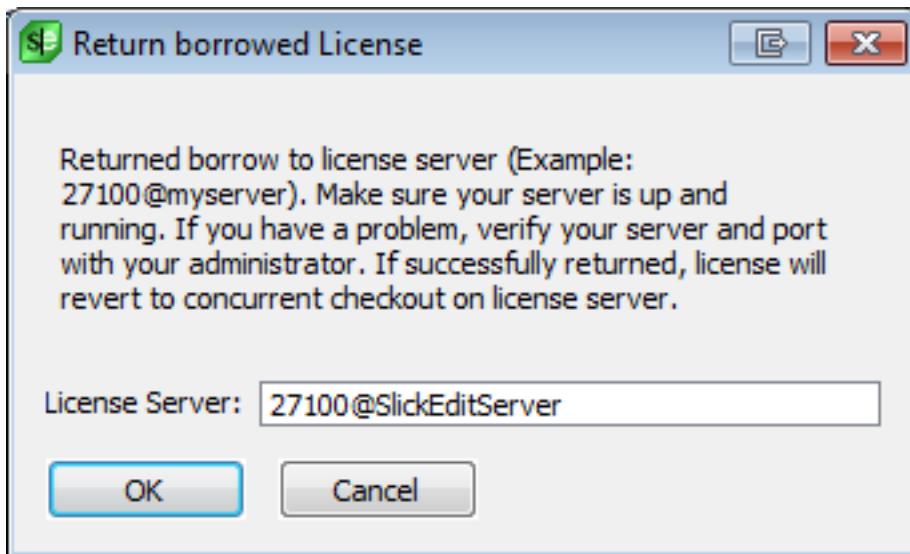
- Make sure that the specified port is allowed through all firewalls. The default port is 27100.
- Contact your IT or License Administrator before contacting SlickEdit support.

Borrowing a License

If you need to use SlickEdit during a period when you will not be able to access the License Server you can borrow a license. To borrow a license, select **Help → Licensing → Borrow License** from the main menu. You are prompted to enter the time this borrow will expire. By default, the maximum borrow period is 30 days.



You can return a borrowed license prior to the expiration date. Select **Help → Licensing → Return License** from the main menu. The port address of the license server you borrowed from is already filled in. Change that only if that license server is no longer in use.



Named User Licenses

SlickEdit uses a license file to authenticate your license. A Full License file is not bound to a particular machine and may be used on as many machines as allowed by the End User License Agreement. A Trial License file is bound to a single machine.

In most cases, the location of the license file is managed by SlickEdit and depends on your platform. However, if you are setting up a portable installation of SlickEdit, to run on a USB drive for example, you should manually copy the license file to the `win` subdirectory of your SlickEdit installation on Windows or the `bin` subdirectory on all other platforms.

Upgrading SlickEdit®

We recommend keeping up with SlickEdit releases and updates to get the latest features and bug fixes. Major versions contain many new features and enhancements to SlickEdit. Minor versions are usually released subsequently to a major version and contain additional enhancements. Hot fixes are published frequently and contain a smaller set of changes to address a specific problem with the previous release.

Upgrading to a New Version

Checking for Updates

SlickEdit® automatically checks and notifies you when a new major or minor version is available. The Update Manager displays an Update Notification that describes the update and how to apply it. For more information see [Notifications](#).

You can set the frequency of how often SlickEdit checks for these updates and change proxy settings through the Update Manager Options dialog. To access these options, from the main menu, click **Help** → **Product Updates** → **Options** (or click the **Options** button on the Update Manager dialog).

You can also check for updates manually. From the SlickEdit main menu, click **Help** → **Product Updates** → **New Updates**. Or, visit the Support Web site at www.slickedit.com/support

<http://www/slickedit.com/support>] and click **Updates**.

Migrating Settings

SlickEdit® creates a versioned subdirectory in your configuration directory corresponding to each version of SlickEdit you have installed. SlickEdit migrates your settings to a new subdirectory corresponding to the new version the first time the new version is run. However, as a precaution, we recommend that prior to installing an update, exit SlickEdit and make a backup of your user configuration directory. For information about the user config directory's location and the files it contains, see [User Configuration Directory](#).

Note

Custom forms that you have created are not migrated as part upgrading your configuration.

Keeping the Previous Version

You can continue to run a previous version of SlickEdit® by installing the new version into a new directory. Because the config directories are versioned, each version of SlickEdit will locate the matching settings.

Applying 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.

When a hot fix is loaded, the changes are stored in the user's configuration. No change is made to the files in the original installation of SlickEdit. Therefore, a hot fix must be applied for each user. If a user has multiple configuration, controlled by the -sc option when launching SlickEdit, then the hot fix must be applied for each configuration.

There are two ways to load a hot fix:

- Manually, using the menu. This loads the hot fix into the current configuration.
- Automatically, by placing the hot fix in a directory specified by **Admin.xml**. This will load the hot fix for any user who shares this install of SlickEdit. This method is particularly useful for multi-user systems or enterprises with many workstations to update. Instead of copying the hot fix to multiple workstations, you can configure each workstation to look in a single directory for hot fixes. You can then deploy a hot fix to all users by copying the hot fix file to that directory.

Manually Installing Hot Fixes

Note

If a user manually 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, use the automatic method, described below.

To manually install a hot fix, complete the following steps:

1. Save the ZIP file to any location on your computer.
2. From the SlickEdit menu, click **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.

Automatically Installing Hot Fixes

Automatic installation of a hot fix will install this hot fix for any user who shares this installation of SlickEdit. When SlickEdit is run, it checks the location specified in **Admin.xml** for any new hot fixes and loads them. To configure automatic installation of hot fixes, complete the following steps:

- Edit the **Admin.xml** file located in the **sysconfig/options** subdirectory of your SlickEdit installation. Find the node that looks like the following:

```
<AutoHotFix menuitem="1" prompt="1" directory=" " />
```

- **menuitem** enables/disables the hot fix menu item under Help > Product Updates (Load Hot Fix). Set to 1 to make menu items visible (default). Set to 0 for them to disappear.
- Set the value for **directory** to the full path for the directory to check for new hot fixes. The **Admin.xml** file contains an example showing where you should put the path.
- Change the value for **prompt** to "0" to suppress a prompt asking if the user would like to apply the hot fix.

To apply a hot fix. Copy the hot fix file to the location **Admin.xml** file. It will be loaded by each user who shares this installation the next time they launch SlickEdit. There is no need to remove hot fixes files after they have been loaded. SlickEdit keeps track of which hot fixes have been loaded.

Hot fixes are handled differently depending on whether the prompt has been suppressed:

- **When prompt="1"** - an Update Notification will be displayed informing you that a hot fix is available. It contains a link you can click to install the hot fix. The notification will be displayed periodically until the hot fix is applied. See [Notifications](#) for more information.
- **When prompt="0"** - no Update Notification is displayed. The hot fix will be loaded automatically when you quit SlickEdit.

Listing Installed Hot Fixes

To see the list of hot fixes installed, from the SlickEdit® menu, click **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.

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 macOS, open the Application folder, and drag the **SlickEdit** icon to the Trash, then empty the Trash.

Startup and Exit

Running SlickEdit

You can launch SlickEdit in a variety of ways, depending on the operating system you are using:

- **Desktop Icon** - On Microsoft Windows and macOS operating systems, use the **SlickEdit** icon displayed on your desktop. On Windows, you can customize the command used by the icon to set [Invocation Options](#).
- **Start Menu** - Some operating systems, including Microsoft Windows, provide a menu for launching installed applications.
- **Command Line** - Use the operating system command line to invoke the **vs** binary. This is the primary method for launching SlickEdit on Linux and UNIX. The command line accepts [Invocation Options](#) that can be used to change the behavior of SlickEdit.

The **vs** executable is stored in the **win** subdirectory (Unix: bin Mac: MacOS) of your SlickEdit installation.

Tip

- 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 SlickEdit for the First Time

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

- If SlickEdit cannot locate a valid license for this version, the License Manager will run. It provides options to install or download a license file. For more information, see [License Manager](#)
- The Quick Start Configuration Wizard will guide you through the steps of setting up the most common options. It also allows you to view the release notes and other useful information. For more information, see [Quick Start Configuration Wizard](#).

Running Multiple Instances

When launching SlickEdit from the operating system command line, append **+new** to the invocation command to get a new instance.

Examples:

```
# Invoke a new instance of SlickEdit. A unique numeric instance id >=1 is used.
vs +new

# Invoke a new instance of SlickEdit with specific instance id "s1".
vs +new -sid s1

# Open file in instance "s1" or new instance "s1" of SlickEdit.
vs -sid s1 /home/username/file.txt

# Open file in oldest instance (least recently activated -- not that useful).
vs -sid 0 /home/username/file.txt

# Open file in most recent (most recently activated) or new instance of SlickEdit.
vs /home/username/file.txt

# Open file, go to line 124 column 5, and search for word "command"
# in most recent or new instance of SlickEdit.
vs /home/username/file.txt -#124 "-#goto-col 5" "-#/command"

# Open file, go to start, perform case sensitive replace without prompting, and save and close file
# in most recent or new instance of SlickEdit.
vs /home/username/file.txt -#top-of-buffer "-#c/wrd/word/e*" "-#file"
```

If you are already running SlickEdit, right click on the running instance icon in the task bar and select **New Instance** to create another instance. Not all Linux desktops support the "New Instance" menu item.

See [Sharing Your Configuration with Multiple Instances](#) for more information.

Invocation Options

SlickEdit can be invoked with a variety of options to control key editor behaviors. This allows you to specify things like a file or multiple files to edit and a different location for your SlickEdit configuration. A full list of invocation options is listed in the table, below.

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.

Tip

On macOS, you can use the shell **open** command to invoke SlickEdit by giving the application bundle name and passing arguments as follows.

```
open -a SlickEditPro2020 --args {options} file1 {options} file2
```

Invocation Options

Invocation options can also be stored in the **VSLICK** environment variable. When SlickEdit is invoked, it inserts the value of this variable before the options typed on the command line. See [Environment Variables](#) for a list of variables you can use with SlickEdit.

The table below shows a list of available invocation options.

Invocation Option	Description
-? or -h[elp]	Display a summary of command line switches and their usage.
<i>file1 file2</i>	Files to edit. File names may contain ant-like wildcards (**, *, and ?). For example, "vs **/*.html" will recursively open all HTML files under the current directory.
filename.vpw	Auto Restore from workspace file. If you specify .vpj, SlickEdit Auto Restores the project. If you specify .sln, SlickEdit Auto Restores from the solution file. Auto Restore from workspace file. If you specify .vpj, SlickEdit Auto Restores the project. If you specify .sln, SlickEdit Auto Restores from the Visual C++ solution file.
-fn filename	Treat <i>filename</i> as a file to edit and not a workspace or switch. For example, vs -fn -e edits a filename called "-e". Note that you can specify the -fn option without a filename in the VSLICK environment variable. When this is done, workspace and project files are opened for editing.
+ or -new	Indicates whether a new instance of the editor should be created or if the existing instance should process the command line parameters. +new creates a new instance. Opening the same workspace in multiple instances is not supported. Default is -new .
+ or -newi	Same as +new option except it always avoids restoring your previous files or workspace. Typically, this option isn't needed when sharing the same configuration directory because +new is converted to +newi if another instance is detected using the same configuration directory. One reason

Invocation Options

Invocation Option	Description
	to use this option is simply to avoid restoring your previous files or workspace for the first instance. Default is -newi .
+ or -newr	Same as +-new option except it always restores your previous files or workspace if you auto-restore options allow it. Default is -newr .
-sid <i>instance_id</i>	Used to refer or give a name to an instance of SlickEdit. See Running Multiple Instances for more information.
-sc <i>config_path</i>	Specifies the configuration directory. This directory will be used to find and save configuration files. Sets the SLICKEDITCONFIG environment variable to <i>config_path</i> .
-migrate	If there is no matching version-specific configuration directory, create a new, default configuration instead of migrating settings from an earlier version's configuration directory.
-sr <i>restore_path</i>	Specifies the directory containing auto-restore files. Sets the VSLICKRESTORE environment variable to <i>restore_path</i> .
-snoconfig	Do not write any files into the configuration directory. This option is needed for utilities which run the editor with the main window hidden which we do not want to save any state information.
-snorestore	Do not read vrestore.slk on editor invocation. This option is used to simplify starting the editor when you have a corrupt vrestore.slk, and is also used by utility programs that launch the editor and do not require any saved state, like vsmktags.
-q[uiet]	Do not display the standard version message on editor startup.
-mdihide	Specifies that the main window should be hidden on startup. This is normally used with the -p <i>macro_name</i> option when running a utility macro under the editor.

Invocation Options

Invocation Option	Description
-si <i>kmax_var_space max_Nofvars</i>	<i>kmax_var_space</i> specifies the amount of memory in kilobytes allocated for storing the contents of interpreter variables (default is 1000 kilobits). <i>max_Nofvars</i> specifies the maximum number of local, static, and global variables there may be (default is 10000).
-st <i>state_ksize</i>	(Pro only) Specifies the maximum amount of swappable state file data in <i>vslick.sta</i> 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.
-sm <i>max_file_size</i>	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.
+supty	(Unix only) Enables Pseudo TTY support in the Build window. This off by default because it typically is not as stable as a standard pipe.
+forcero	(Experimental) Specifies that all existing files be opened in read-only mode. This allows for an instance of SlickEdit to be for browsing source code only.
-sumotif -sucde -sukde -suwindows -sugtk	(Unix only) Use Motif, CDE, KDE, Windows, or GTK style widgets. The default is to use KDE style (Qt Plastique style) widgets.
-graphicssystem native	(Unix only) SlickEdit attempts to automatically choose the correct Qt graphics system at start up. If you notice unusually slow performance, try this option. SlickEdit can get confused when ssh is used to start an X Terminal session. This option is best when running remotely through an X server.
-graphicssystem raster	(Unix only) SlickEdit attempts to automatically choose the correct Qt graphics system at start up. This option is best when running locally on your display and not through an X server.

Invocation Options

Invocation Option	Description
-sprimarydisplay	Specifies that the editor should open all windows in the primary display.
-summ "[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, x and y are not necessary. The following two examples are equivalent because the monitors are in a left-to-right configuration:</p> <ul style="list-style-type: none">• -summ "1024 768,1024 768"• -summ "0 0 1024 768,1024 0 1024 768" <p>Specifying the -summ 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> <ul style="list-style-type: none">• -summ "1600 1200,1024 768"• -summ "0 0 1024 768,0 768 1024 768"• -summ "1024 768,1024 768,1024 768" <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>user.cfg.xml</code> file so you don't need to specify this for every invocation. For more information, see Setting Environment Variables in user.cfg.xml.</p>
-sul	(Unix only) Disables the byte-range file locking that SlickEdit normally performs. Enable this option when receiving an "access denied" error with remote files.

Invocation Options

Invocation Option	Description
-x <i>pcode_name</i>	Alternate state file (.sta) or pcode file (.ex).
-m <i>menu_file</i>	Name of menu resource to use for the SlickEdit menu bar.
-p <i>cmdline</i>	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.
-r <i>cmdline</i>	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.
-#command	Execute command on active buffer. For example, vs test.c -#bottom-of-buffer places the cursor at the end of test.c. Use double quotes if the command has spaces (vs test.c "-#goto-col 50").
+ or -L[C]	Turn on/off load entire file switch. The optional C suffix specifies counting the number of lines in the file.
+ nnn	Load binary file(s) that follow with a record width <i>nnn</i> .
+T [buf_name]	Start a default operating system format temporary buffer with name <i>buf_name</i> .
+TU [buf_name]	Start a UNIX format temporary buffer with name <i>buf_name</i> .
+TM [buf_name]	Start a Macintosh format temporary buffer with name <i>buf_name</i> . Classic Mac line endings are a single carriage return (ASCII 13).
+TD [buf_name]	Start a DOS format temporary buffer with name <i>buf_name</i> .
+ or -E	Turn on/off expand tabs to spaces when loading file. Default is off.

Invocation Option	Description
+ or -ssymlink	(Windows only) Turn on/off resolving of symbolic links for files and directories. This option can slow down performance but is highly optimized. This option can take a lot of memory for caching the resolved filenames (needed for performance). Symbolics are always resolved on non-Windows platforms.
+ or -ssymlinkdirs	(Windows only) Turn on/off resolving of symbolic links for directories only. This option can slow down performance but is highly optimized. Symbolics are always resolved on non-Windows platforms.

Exiting the Program

To safely exit SlickEdit®, from the menu, click **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.

Tip

- 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, click **Tools → Options → Application Options → Exit**. See [Exit Options](#) for descriptions of these options.

Product Support

Product Support is provided to customers with a current Maintenance and Support agreement. Limited support is also available to new customers and trial customers to help them get started. For more information, please visit the SlickEdit Product Support web page at www.slickedit.com/support [http://www.slickedit.com/support].

The Product Support Web site provides a list of frequently asked questions and answers as well as information about upgrades and hot fixes. You can launch this site in a browser by clicking the menu item **Help → Contact Product Support** or by going directly to www.slickedit.com/support [http://www.slickedit.com/support]. You can also access the FAQ page directly by clicking the menu item **Help → Frequently Asked Questions**.

SlickEdit has an active user community supported by forums, where users can post questions and get answers. Though created as a means for users to help other users, the SlickEdit team monitors the forums and answers selected questions. Visit the forums at <http://community/slickedit.com>.

See [Documentation](#) for more help resources.

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, which helps us to better answer your questions. If SlickEdit won't run, you can report problems via the web at: www.slickedit.com/support [http://www.slickedit.com/support].

For problem reports, please provide the following information:

- A description of the problem.
- The language you are working in (C/C++, Java, etc.).
- SlickEdit program information, which is automatically provided if you use **Help → Contact Product Support**. If you initiate a report from the website, select **Help → About SlickEdit**, then select the **Program Information** tab, click **Copy To Clipboard**, and paste the information into the problem report.
- A code snippet to help us reproduce it (if possible).

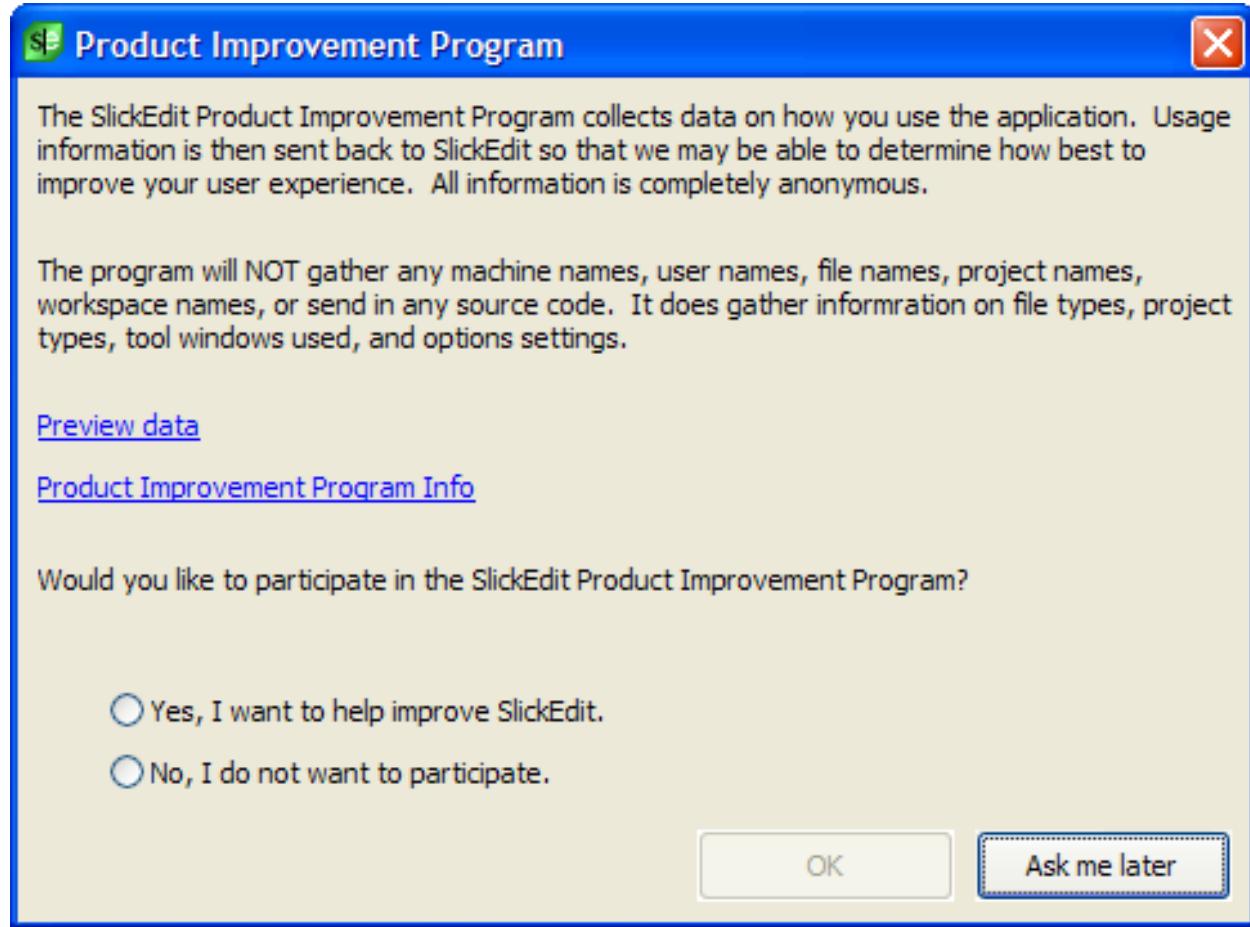
To speak to a member of our Product Support team, call the Support line at 1.919.473.0070. Telephone support is only available during business hours for customers with a valid Maintenance and Support Service Agreement.

Product Improvement Program

SlickEdit strives to meet the demands of its customers. Without knowing what features or languages our customers are using, it is a challenge to determine which areas of the product need our attention most. To close this information gap, we developed the Product Improvement Program. The program runs in the background, logging events as they happen in your daily usage.

We gather a variety of data, including information about command invocation, toolbars and tool windows, settings, project and workspace usage, error messages, and file types. We do not make any record of file, project, or workspace names, directory structures, or source code.

Periodically, the compiled data is sent to SlickEdit. The first time the program tries to send data, you are prompted.



The dialog contains a link to preview the data that was compiled before deciding to send it to SlickEdit. You can also get more information by clicking the **Product Improvement Program Info** link, which takes you to an informational page on SlickEdit's website.

Select the **Yes** radio button to send the data and continue your participation. If you do not wish to have any data logged or sent to SlickEdit, select the **No** option. After making your selection, click **OK**. You can also click the **Ask me later** button to postpone the data transmission. No data will be sent at that time, and you will be prompted again later. You can also change your participation status at any time by going

to **Tools → Options → Application Options → Product Improvement Program**. For more information about these options, see [Product Improvement Program Options](#).

After this first prompt, if you agree to participate in the program, all data transmissions are completely silent. Transmission only occurs at start-up of the application or when the editor has been idle for a period of time, thus causing the least possible disruption to your work.

All data is completely anonymous. We do construct a source ID to enable us to determine how many data sources we have. This ID is a concatenation of machine name, user name, and serial number that is then hashed. The hash result is used as the ID. Therefore, even with a unique ID for each user, we are unable to associate an ID with any particular person or source.

Performance Tuning

SlickEdit was designed with speed in mind. Most operations perform nearly instantaneously. However, the size and location of your codebase can affect SlickEdit performance along with various settings within SlickEdit. This guide will help you to make sure that you get the best performance possible.

First Steps

In some cases, Symbol Coloring can cause delays while typing. If you are experiencing performance problems while typing, please turn off that feature to see if the problem is fixed. For more information, see [Symbol Coloring](#).

Virus checkers also might be a cause for bad performance. Many do real-time checking each time a file is read. When trying to diagnose the cause of a performance problem, please turn off any such checking. Some virus checkers give you the option of exempting specific file types from these checks. If so, you can achieve better performance by exempting SlickEdit workspace files (.vpw), project files (.vpj), and tag files (.vtg). You may also wish to exempt your source files from these checks.

File Locations

Whenever possible, make sure that your source code files, workspace and project files, and configuration files are stored locally. SlickEdit is subject to normal file latency. When files are stored remotely they take longer to access.

Source Files

Storing your source files remotely will increase the amount of time it takes to open and save files. Additionally, it will increase the amount of time it takes to tag your files. Tagging is the process of building a symbol database, which is used for many advanced operations in SlickEdit. On a fast, reliable network you may find that storing your source files remotely does little to harm performance. On a slow network, these operations will likely take unacceptably long to complete.

Workspaces and Project Files

Even if you store your source files remotely, you should still either try to store your workspace and project files locally or more importantly configure your workspace **Tag Files Directory(Pro only)** to a local drive (see [Workspace Properties Dialog](#)). By default, tag files are stored in the same location as your workspace file. Tag files are large and complex databases, that are used for operations like symbol completions that happen while you type. Storing your tag files remotely often introduces unacceptable latency into this access, slowing down SlickEdit's response time.

SlickEdit Configuration Files

Your SlickEdit Configuration files should also be stored locally. This is where SlickEdit stores a great deal of information about your options and the state of SlickEdit. Having these files located remotely will introduce latency at unpredictable times.

By default, SlickEdit stores your config files in \\My Documents\\My SlickEdit Config on Windows, in \$HOME/.slickedit on UNIX, and in \$HOME/Library/Application Support/SlickEdit on Mac. These are typically on a local drive. You can specify a different location for your config using the -sc option when SlickEdit is launched:

```
vs -sc /dev/seconfig
```

If necessary, use this option to specify a new location for your config files that is on a local drive.

Memory and Caching (Pro only)

Along with making sure that your tag files are stored locally, you should make sure that SlickEdit has enough memory to hold all of your tag files in memory. When it doesn't, it has to page sections of the tagging database in and out of the cache.

To increase the size of your tag file cache, select **Tools** → **Options** → **Editing** → **Context Tagging** and change the value for **Tag file cache size (KB)**. Try to make it large enough (within reason) so that we can get your entire workspace tag file and extension specific tag files into memory. To determine that size, open **Tools** → **Tag Files**. This lists all of the tag files in SlickEdit. Not all of them are used at any one time, though. You may also want to adjust the value for **Tag file cache maximum (KB)**. This setting controls the maximum amount of memory that can be dedicated to the tag file cache depending on the amount of memory available on your machine at the time that SlickEdit starts. If you have a machine with lots of memory available, setting this maximum to a large value is the simplest way to get good tag file performance without having to worry about adding up the total sizes of your tag files as described below.

For a given workspace, you need to add the size of your workspace tag file, listed at the top of the tree, to the size of the extension-specific tag files used in that workspace. If you are only using a single language, then it will just be the one extension-specific tag file. If you are using a mixture of languages, you will need to add the tag file for each language. If you have tagged multiple tool chains in a given language, like GNU C/C++ and Microsoft Visual Studio, you need only factor in the one used by that workspace. The Tag Files dialog will tell you the location of the tag files. Use the operating system to determine the size of the files. Add them together, and use that value for the tag file cache size.

The tag file cache size is a global value that is used for all workspaces, so you should set this value for your largest workspace. If that workspace is atypical or infrequently used, set it based on the tag file sizes used by a more typical workspace.

It is possible that you could hit a threshold where increasing the cache size reduces performance. This is likely to be the case if the tag file cache size exceeds the amount of free memory available on your system. So, once you've set this value check your operating system and make sure it isn't being forced to do a lot of paging. If it is, you should decrease the tag file cache size. Like most performance tuning, this could be an iterative process until you find a value that provides the best speed for your codebase and system.

On systems with large amounts of memory, it may also improve performance to dedicate an independent tagging block cache to each tag file rather than having all the open tag files share the same cache. Using this option reduces cache contention and thrashing. To change this option, select **Tools** → **Options** → **Editing** → **Context Tagging** and change the value for **Use independent database file caches**.

Tuning Context Tagging (Pro only)

After you've checked the items above, the next optimizations to try are the various control settings for Context Tagging. SlickEdit's Context Tagging system provides many of the advanced features that make using SlickEdit so great. Context Tagging creates a database of all the symbols in your code and where they are located. This is used to provide rapid navigation from a symbol to its definition or declaration, for all kinds of completions, and for rapid symbol searches. All of that information is great, but it does you no good if you have to wait too long to get it.

As mentioned above, Symbol Coloring can cause performance problems while it attempts to identify and resolve symbols. If you are having a performance issue while typing, the first thing to do is to shut off Symbol Coloring. For more information, see [Symbol Coloring](#).

To configure Context Tagging, open **Tools** → **Options** → **Editing** → **Context Tagging**. This screen contains a number of parameters you can use to control the performance of Context Tagging.

Background Tagging (Pro only)

If you are experiencing sporadic pauses in SlickEdit, the first thing to check is that **Background tagging of other files** is off. It's generally fine to leave **Background tagging of open files** on. We recommend that you turn that off only after you've applied all other tuning approaches. Likewise, you should leave **Tag file on save** enabled. This ensures that the tag database is always current by tagging a file when it is saved.

The context tagging engine is single threaded with SlickEdit, and background tagging has been known to introduce random periods of unresponsiveness. Generally, you don't need to tag other files in the background. Once you've tagged your workspace, you only need to tag files that are being changed, and SlickEdit does this automatically if you leave the other two values on.

The exception to this is if you fetch updated files from a source code repository. Then, other developers may have changed files or added new ones. SlickEdit won't know about those changes until you retag the workspace. For normal size projects, SlickEdit can tag the workspace in a few minutes. On extremely large projects, this can take over an hour. Your strategy for how and when to tag depends on the size of your codebase.

For a normal codebase, you can open the Project tool window and right-click on the workspace entry, and select **Retag Workspace**. You will have to wait while SlickEdit retags your workspace. Retagging is generally faster since it only has to look at new or modified files.

For extremely large codebases, you may want to script this process. You could set up a nightly process that fetches all new and updated files from source control, adds the new files to appropriate projects, and then runs the tagging engine on them.

To configure Background Tagging, open **Tools** → **Options** → **Editing** → **Background Tagging**. This screen contains a number of parameters you can use to control the performance of Background Tagging.

Context Tagging Maximums

These tuning options for Context Tagging set maximum values for specific tagging operations. You can

change these values when a specific operation is found to be too slow. For example, if you type in a function call, like

```
foo( );
```

After typing the open parenthesis, SlickEdit will look for a list of local variables that match the parameters in foo. The value, **Maximum candidates for list parameters** determines the upper limit in that search. By default it is set to 200. Once that number is reached, it will stop looking for matches. If you find that SlickEdit is taking too long in this situation, you can decrease that number to, say, 100. You have to weigh the tradeoff between completeness and responsiveness.

We won't go into each of the values in that list. When you select an item in the Options dialog, help is provided that will guide your decision on whether to change that value.

Warning

You can easily degrade the performance of SlickEdit by changing the Context Tagging defaults. You should compare your changes to the performance using a default configuration. To create a default configuration, use the -sc options on the command line:

```
vs -sc config
```

This will launch SlickEdit putting the configuration in a "config" directory below your SlickEdit install directory. Be sure to use a new location or delete that directory before launching SlickEdit in this manner, or it will use the config that was already in place.

References

The Context Tagging options also contain a group for References. If you are experiencing performance issues with reference lookup (when using **Ctrl +/** or **push-ref**), then you may want to change some of the values in this group. Turning on **Build workspace tag file with references** makes reference look-ups faster, but it makes creating tag files take longer. For normal sized codebases the slow-down is negligible, so we often turn this on.

If you have a large codebase, you may want to turn on **Find references incrementally (faster)**. When set to **True**, reference queries are faster because SlickEdit does not open each candidate file to eliminate invalid references. So, you get your answer more quickly, but it may not be fully accurate.

Add as Wildcard

When you set up a project, you can use **Add as wildcard** checkbox on the **Add Tree** dialog (accessed from the **Project Properties** dialog) to specify directories to search for new files. This capability is useful when other team members are not using SlickEdit. In that case, other programmers will be adding files without updating and checking in the project files.

Each time SlickEdit is launched, projects that were configured using **Add as wildcard** search the

specified directories for new files. This is an exhaustive search and can take a long time, particularly on large projects or if source directories are stored on network resources.

If you are experiencing delays when launching SlickEdit, you may want to redefine your projects, adding your files in a one-time tree traversal, rather than as wildcards (dynamic tree traversal). When new files are added, you will have to use Add Tree to find and add them to your projects. Which approach is better for you depends on how frequently you need to manually new files and whether you see any performance problems due to the size of codebase.

For more information on **Add as wildcard** see [Add as wildcard](#).

Profiling

SlickEdit includes a profiler to measure the amount of time spent in different functions. This tool can be very helpful to track down performance problems. To run the profiler, do the following:

- Start the profiler from the SlickEdit command line by typing the following: **profile on**. Then press Enter.
- Perform the operations to be measured. Try to include only the steps necessary to produce the problem.
- Stop the profiler. From the command line, type the following: **profile save "<filename>"**, where <filename> is the name of the file to save to. For example, you could type: profile save "profile.txt".

You can then send the file into Product Support to be analyzed.

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 Configuration Wizard, and you'll be up and running with SlickEdit in no time.

Quick Start Configuration Wizard

The **Quick Start Configuration Wizard** helps you to set up common options and shows you where these options are usually accessed. Each set of options is accessible outside of the wizard, through the normal SlickEdit Options dialogs, by selecting **Tools** → **Options** from the main menu. Several pages have **Customize** links to their normal Options location.

The Quick Start Configuration Wizard has two methods of navigation. You can use the **Previous** and **Next** buttons to visit the adjoining parts of the wizard. You can also jump directly to a section by clicking on the section name in the tree on the left side of the wizard. You can run the wizard again later by selecting **Tools** → **Quick Start Configuration** from the main menu.

The following items are configurable through the Quick Start Configuration Wizard:

- [Emulation](#) - select which other editor SlickEdit will emulate.
- [Colors](#) - set your color profile.
- [Fonts](#) - set your font for Unicode and non-Unicode languages.
- [Coding](#) - set common coding preferences, like indentation, brace styles, and the use of syntax expansion.
- [Associate File Types](#) - determine which file types should automatically be loaded in SlickEdit.
- [Workspaces & Projects Setup](#) - allows you to quickly set up a new project and workspace.
- [Context Tagging](#) - build tag files for common compiler libraries to aid in code navigation.
- [More Information](#) - allows you to export your newly configured options, as well as see the Release Notes and some Cool Features of SlickEdit.

Emulation

CUA is the default emulation for all platforms except macOS which defaults to macOS. These emulations provide key bindings familiar to Microsoft Windows and macOS users. Emulations are provided for other popular editors including Vim, GNU Emacs, Brief, and more. If you are already an experienced user of one of these other tools, you will find that these emulations will help you get up and running quickly. Otherwise, you may find that the CUA emulation or macOS is best.

You can change your emulations at any time by selecting **Tools** → **Options** from the main menu. Then expand **Tools** → **Options** → **Keyboard and Mouse** → **Emulation**. For more information, see [Emulations](#).

Emulation

- CUA
- Vim
- Mac OS X
- GNU Emacs
- Visual Studio default
- Epsilon
- Visual C++ 6
- ISPF
- SlickEdit (text mode edition)
- CodeWarrior
- Brief
- Xcode
- CodeWright
- BBEdit
- Eclipse

The CUA (Common User Access) keyboard emulation uses a command set similar to that used in Microsoft Word and Notepad.

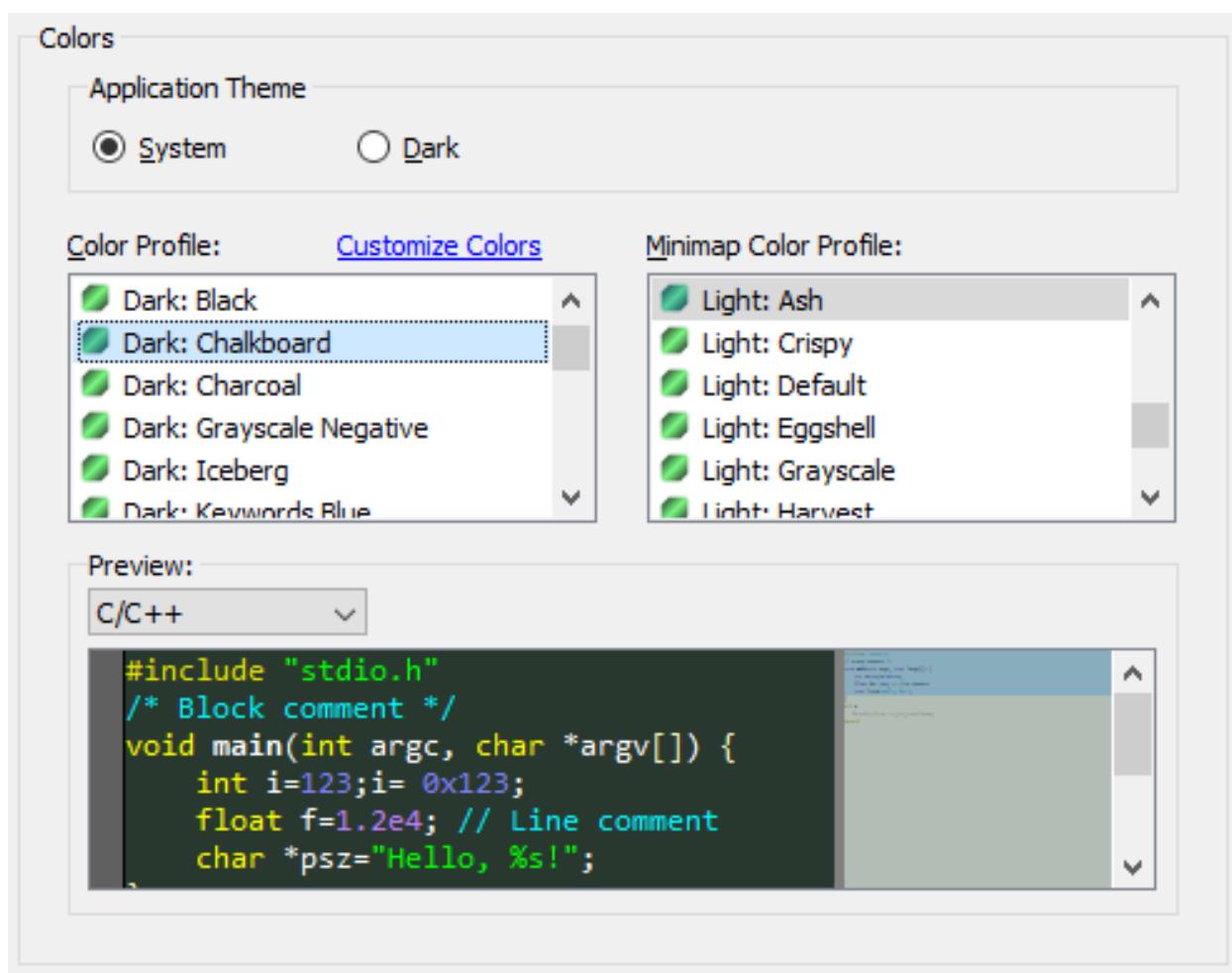
[Customize Emulation](#)

Colors

Many users are particular about the colors they use. On this form, you can select an application theme, as well as default global color profiles for all editor windows and minimap windows. Preview your selections in different languages using the preview window and the language combo box.

To change your colors later, select **Tools** → **Options** → **Appearance** → **Colors** from the main menu. For more information on setting your colors, see [Colors, Color Coding, and Symbol Colors](#).

For more information on selecting the application theme, see [Application theme](#).

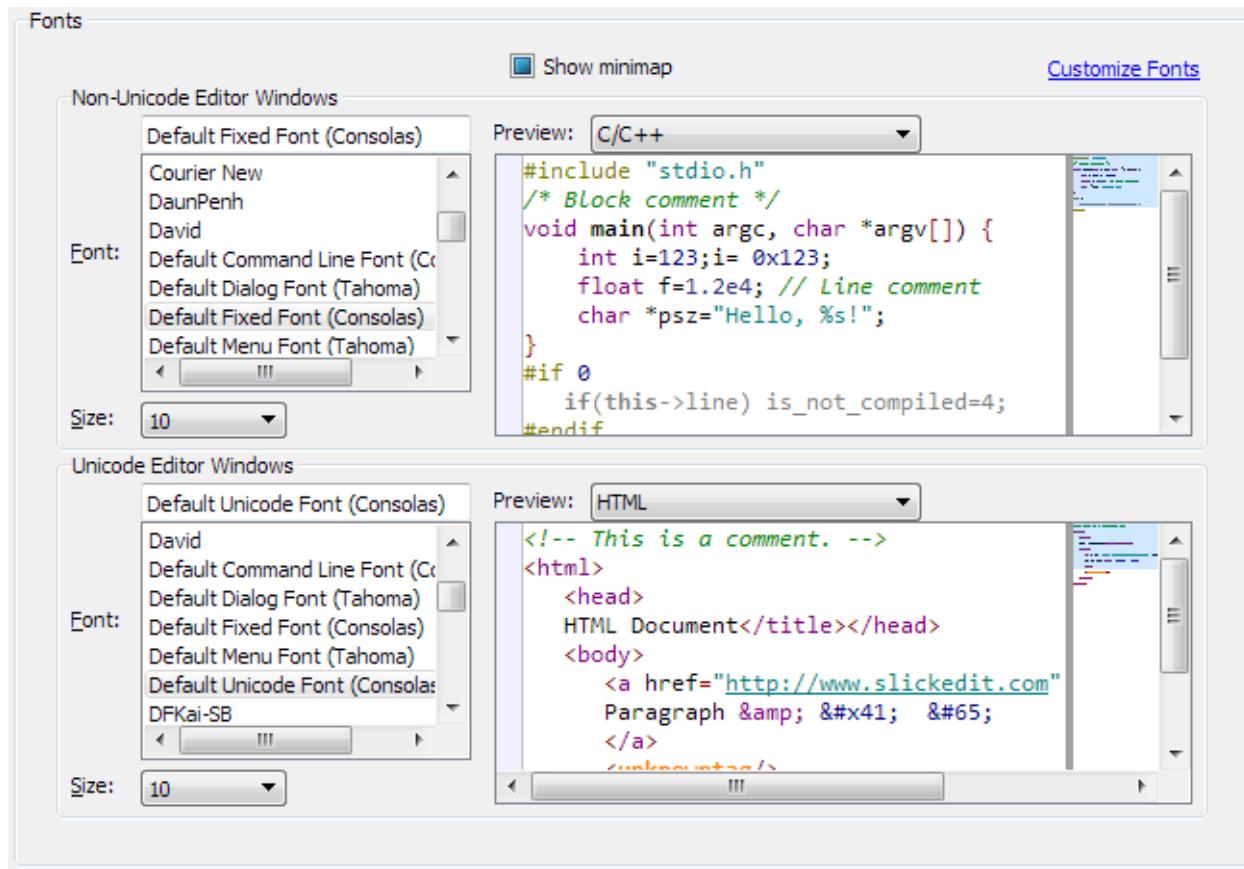


Fonts

Fonts and viewing of the minimap window are another matter of personal preference. The Quick Start Configuration Wizard lets you choose different font styles for Unicode (HTML, XML, etc) editor windows and Non-Unicode editor windows. The **Show minimap** checkbox determines whether the minimap window is displayed for all languages. Use the preview windows to view your selections in the languages of your choice.

To set fonts later, select **Tools** → **Options** → **Appearance** → **Fonts**. For more information on setting fonts, see [Fonts](#).

You can select whether the minimap window is displayed per Language as well just for the current buffer. See [Using the Minimap](#) for more information on minimap settings.



Coding

SlickEdit features many options to control your editing experience. On this form you can set three important ones for all language modes. Normally these are set one language at a time. You can set the following options:

- Indent settings - controls your indent amount and tab size, as well as whether you would like to indent using tab or space characters.
- Brace style - controls the location of braces in C-style languages.
- Syntax Expansion - specifies whether or not you want SlickEdit to automatically expand block structures like *if* or *for* for all languages. This option uses a tri-state checkbox. A check indicates that Syntax Expansion will be turned on for all languages. Unchecked indicates that it will be turned off for all languages. When it is grayed in, the individual language settings will be retained and no changes will be made.
- Line Numbers - controls display of line numbers for all languages. This uses a tri-state checkbox. A check indicates that Syntax Expansion will be turned on for all languages. Unchecked indicates that it will be turned off for all languages. When it is grayed in, the individual language settings will be retained and no changes will be made.

To see where these options are normally configured, click the **Customize** link next to each setting.

Associate File Types

Coding

Set indent settings for all languages [Customize Indent](#)

Indent with spaces Indent amount:

Indent with tabs Tab size:

When Indent with tabs is selected, tab characters are used for the leading indent of lines. You can also specify your tab size. To use spaces for the leading indent of lines, select Indent with spaces and specify how many spaces should be used to indent a line.

Set brace style for all languages [Customize Brace Style](#)

`if () { if () if ()
 ++i; { {
} ++i ++i
 } } }
 Style 1 Style 2 Style 3`

In C-style languages, SlickEdit will insert your braces for you when you type certain keywords as part of Syntax Expansion. This option controls the style of braces entered.

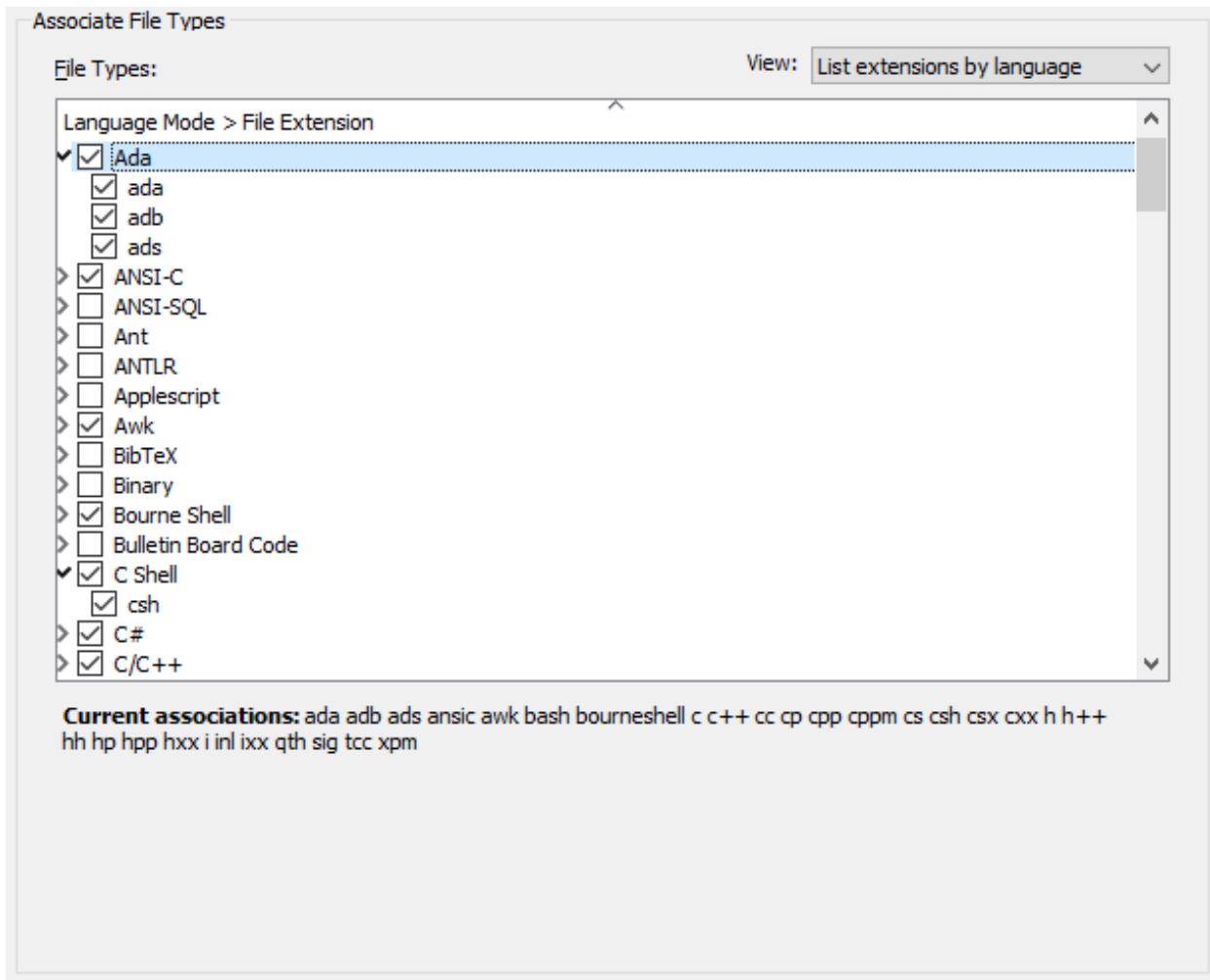
Use Syntax Expansion for all languages [Customize Syntax Expansion](#)

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. However, many users who are accustomed to their own coding styles may not want automatic Syntax Expansion.

Display line numbers for all languages [Customize Line Numbers](#)

Associate File Types

This screen lets you select the file types that will be automatically opened in SlickEdit from Windows Explorer. For more information see [Associate File Types Options](#).



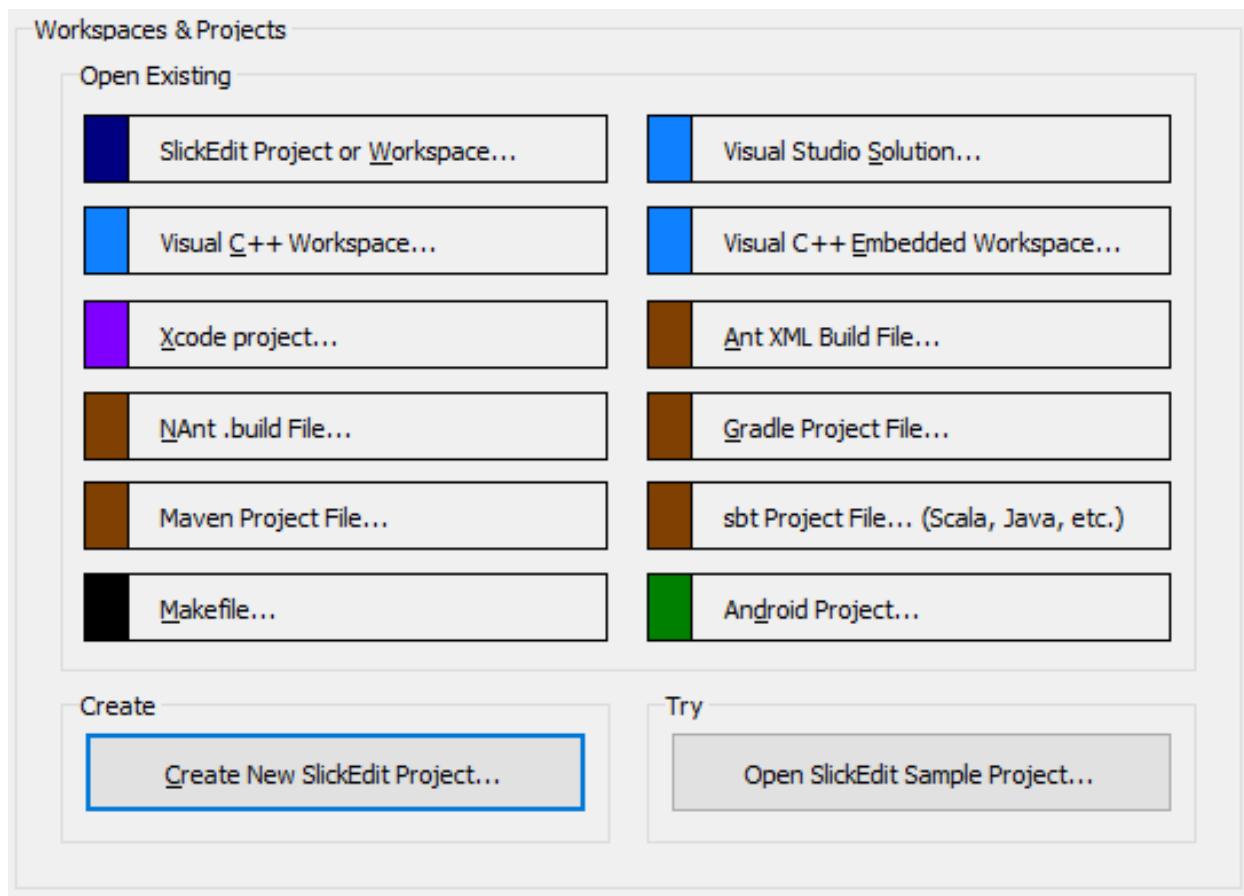
Current associations: ada adb ads ansic awk bash bourneshell c c++ cc cp cpp ppm cs csh csx cxx h h++ hh hp_hpp hxx i ixl ixx qth sig tcc xpm

Workspaces and Projects Setup

SlickEdit can open projects from many other tools, or you can create a new workspace and project directly and add your source files to it.

By adding your files to a workspace and project, you can quickly open your files without specifying a path with the Open or Files tool window. The Projects tool window lets you organize files in folders based on file type or directory. (Pro only) To get the most out of SlickEdit symbol analysis features, your source code files need to be tagged. This is done automatically for files that are part of a project. This lets you use powerful features like SlickEdit's [Symbol Navigation](#) to quickly jump from a symbol to its definition or declaration or see a list of references.

(Pro only) It is critical that you use the correct project type. Click **Create New Project** to create a new project. You can perform this action later by selecting **Project → New** from the main menu.



Context Tagging (Pro only)

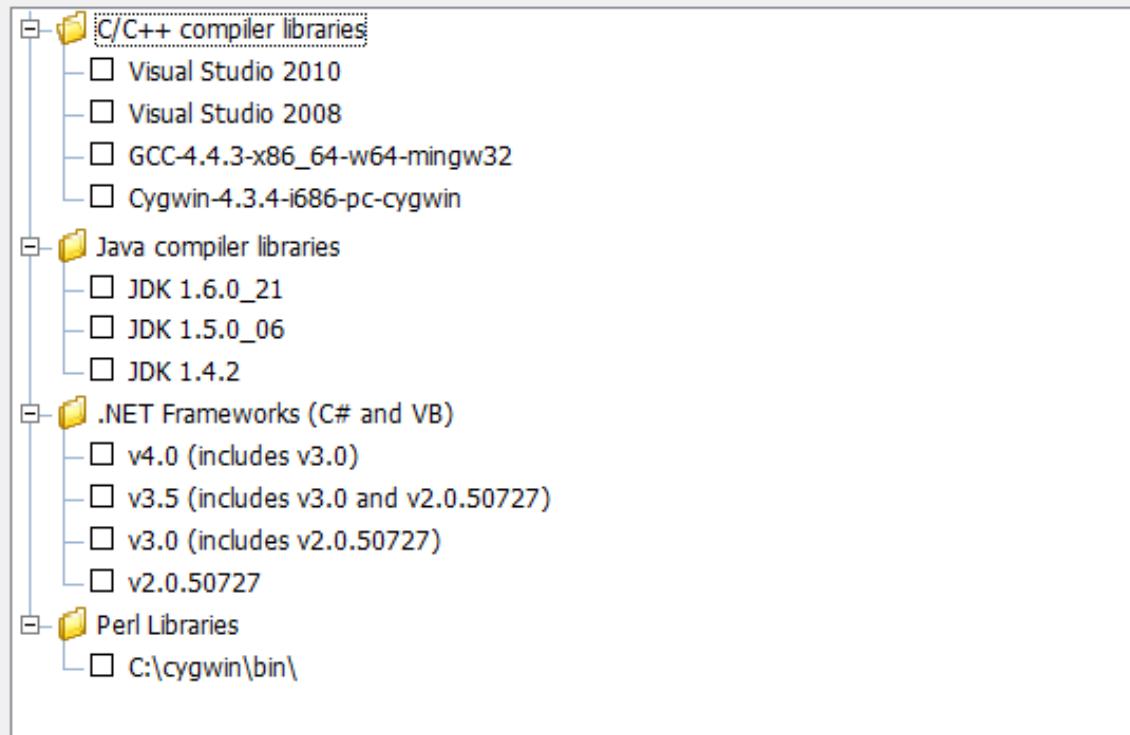
Context Tagging creates a database for all the symbols in your code. This allows SlickEdit to rapidly jump from a symbol to its definition, its declaration, or show a list of references. Other features, such as completions, also use this information. To properly work with your code, SlickEdit will need to tag the libraries associated with the compiler you are using.

You can choose to tag as many compiler libraries as you want. You can choose to build the tag files in the background.

If you choose to skip this step now, SlickEdit will automatically tag your compiler libraries if you are using Microsoft Visual Studio, GNU C/C++, or Java. For other compilers, you can tag them later by selecting **Tools → Tag Files** and then clicking the **Auto Tag** button. For more information about Context Tagging, see [Context Tagging Features](#)

Context Tagging

Context Tagging performs expression type, scope and inheritance analysis as well as symbol look-up within the current context to help you navigate and write code. It parses your code and builds a database of symbol definitions and declarations - commonly referred to as tags. Context Tagging works with your source code as well as libraries for commonly-used languages such as C, C++, Java, and .NET. To tag these compiler libraries now, click the button below. You can also access this feature from the main menu under **Tools > Tag Files** and pressing the **Auto Tag** button.



More Information

The final step in the Quick Start Configuration Wizard, allows you to do three things:

- Export Options - this is useful to save your settings. You can do this to share your settings with others or to restore your settings later.
- View Cool Features - this is a list of the key features that set SlickEdit apart from other editors. Look through this list to learn how you can become more productive.
- View the Release Notes - this contains a list of known limitations and other useful information pertinent to this release.

Additional Settings

The Quick Start Configuration Wizard helps you set the most commonly changed settings in an editor. This section lists some additional settings you may want to alter. SlickEdit contains a vast number of settings to allow you to work your way. It can be very helpful to browse through the options hierarchy and see what else is available.

Options are changed using the Options dialog, which is displayed when you select **Tools** → **Options** from the main menu. Option settings are divided into two categories: [General Options](#) and [Language-Specific Options](#).

General Options

General options affect all languages.

- **Clicking past the end of a line** - To have the ability to place the cursor past the end of a line, select **Tools** → **Options** → **Editing** → **Cursor Movement**, then set the option **Click past end of line** to **On**.
- **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, click **Tools** → **Options** → **Editing** → **Cursor Movement**, then set the option **Cursor up/down places cursor in virtual space** to **On**.
- **Changing the line insert style** - In code, a line of text is a meaningful unit of functionality. 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, click **Tools** → **Options** → **Editing** → **General**, then set the **Line insert style** option to **Before** or **After**.
- **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, select **Tools** → **Options** → **Keyboard and Mouse** → **Advanced**, then set the value of **Selective Display, Expand/collapse** to **Expand on single click**.

Language-Specific Options

Language-specific options are configured for each language that you work with in SlickEdit. These options are accessed from the Options dialog (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]**). All menu instructions below are relative to this path.

Tip

A quick way to access language-specific options for the current buffer is to use the **Document** → **[Language]** → **Options** menu item (or the **setupext** command).

- **(Pro only)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 preferably navigates to one or the other, select the language-specific **Context Tagging** category, then select one of the **Prioritize navigation to symbol** options. The **push_altag** command (**Ctrl+Alt+Dot**, or **Search → Go to Declaration**) will prioritize navigation conversely.
- **(Pro only)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 language-specific **Context Tagging®** category, then clear the option **Show info for symbol under mouse**.
- **Configuring C/C++, SystemVerilog, or Verilog preprocessing** - For C/C++, SystemVerilog, or Verilog, 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 language 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

The Quick Start Configuration Wizard contains a step to launch the New Project Wizard. This helps you select the correct project type(Pro only) based on your programming language and compiler. This section will help you set up a new project and workspace if you choose not to use the wizard.

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(Pro only) and execute(Pro only) as a unit. For each project you can specify the set of files it contains, a working directory, a set of commands(Pro only) to build(Pro only) and execute(Pro only) the project, compiler options(Pro only), and dependencies(Pro only) between other projects. A tag file(Pro only) for each project's source files is automatically created and maintained, enabling SlickEdit's advanced navigation(Pro only) and unique Context Tagging®(Pro only) lookup features. By adding your files to a project, you can quickly open your files without specifying a path with the Open or Files tool window.

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

Typically, you create a new workspace by creating the first project for that workspace (see [Create a New Project](#)). To create a new workspace without a project, complete the following steps:

1. From the main menu, select **Project** → **New**. This will display the New Project Wizard. Click **Cancel** to close the wizard.
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 main menu, click **Project** → **New**.
2. It is important that you select the correct project type. The New Project Wizard will help you choose the correct project type based on your language and compiler. If you already know the correct project type to use, you can click **Cancel** to close the wizard. Then select the type of project that you want to use from the list box on the left side of the dialog.

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. If the new project is for an existing workspace, select **Add to current workspace**. If this is the first project in a new workspace, select **Create new workspace**.
7. (Pro only) 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 main menu, click **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 options. For information, see the [User Preferences](#) chapter.

User Interface

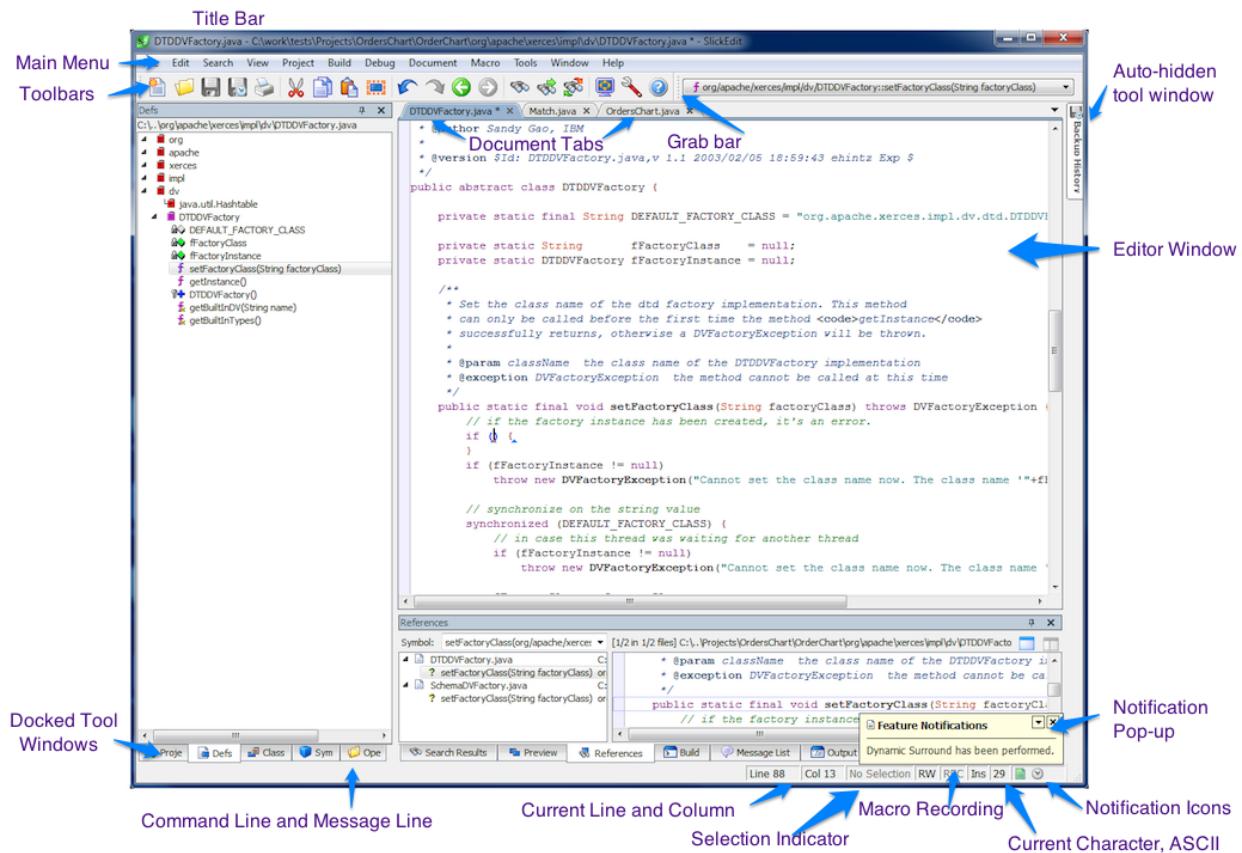
This chapter describes the SlickEdit user interface. Much of the power of SlickEdit comes from using the keyboard to invoke operations. See [Using the Mouse and Keyboard](#) for more information.

Screen Layout

The SlickEdit Interface

SlickEdit® uses a Multiple Document Interface (MDI), which opens all documents for the application within the application's main window, called the main window group or MDI frame. To open a file in a different window, you can run another instance of SlickEdit. For more information see [Running Multiple Instances](#).

The screenshot, below, shows a representative SlickEdit session and identifies common items on the screen:



Note

These pictures show some tool windows that are only available in the Pro edition 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 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 [Files, Buffers, and Editor Windows](#) for more information.
- **Document tabs** - For each window you create (not buffer), you will see a document tab. Click on the document tab to select the buffer you want to edit. Document tabs may be drag and dropped outside the main window group to create a floating window group. Tool windows may be duplicated and docked to floating window groups. Right-click on the Document tab to display a context menu. See [Document Tabs](#) for more information on document tabs and the context menu.

Note

If you want to see a document tab per buffer (probably because you are using "Multiple files share window"), try using the **File Tabs** tool window. See [File Tabs](#) for more information.

You may want to hide the Document tabs when you only have only one edit window. To do this, set the **Zoom (hide tabs) when one window** to **Always** at **Tools → Options → Editing → Editor Windows**.

- **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** button 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 [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.
 - **Selection 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 [Selection Indicator](#) 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, click **Tools** → **Options** → **Editing** → **General**, then set the option **Protect read-only mode** to **True**. Now 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, click **Tools** → **Options** → **Editing** → **General**, then set the **Start mode** option.

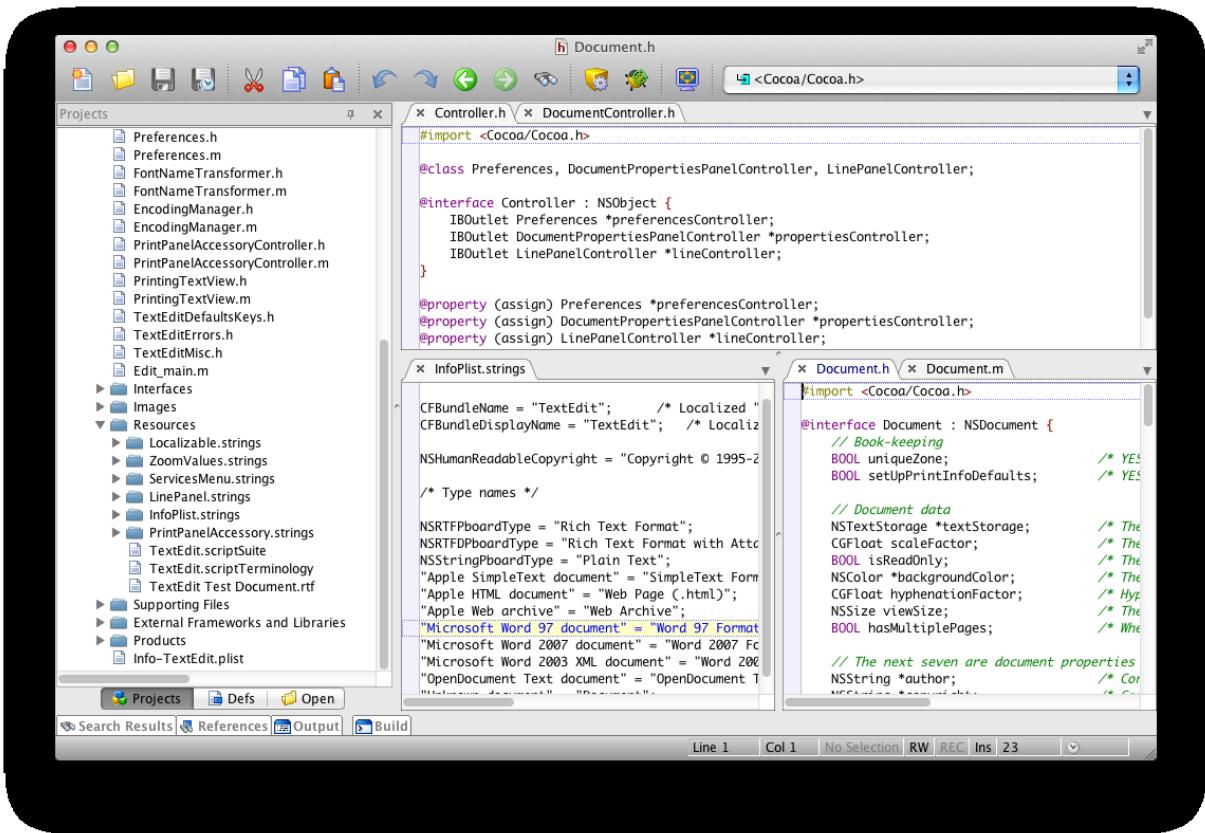
- **Current character indicator** - When editing an SBCS/DBCS mode file or non-Unicode file, the current character is displayed in hexadecimal format. If the current character is a double byte character (DBCS), then two bytes and its Unicode equivalent are displayed in hexadecimal (95 74 U+4ED8). When editing a Unicode file, the current composite character is displayed in hexadecimal. The indicator field is blank when the cursor is past the end of the line. See [Encoding](#) for more information about Unicode.
- **Alert Icons** - Used to display information about operations within SlickEdit. It can contain icons for things like Feature Notifications (see [Notifications](#)) and Background Tagging (see [Creating Tag Files for Compiler-Specific Libraries](#)).

Editor Windows

SlickEdit uses a Multiple Document Interface (MDI) that allows you to manage several editor windows within the application window. Windows can be arranged into multiple groups of document tabs, and tab groups can be detached (floated) separately from the main application window.

- **Multiple Tab Groups** - Open documents can be arranged into multiple groups of tabbed document areas. Tab groups are created by splitting windows or using the New Tab Group family of commands on the document tab context menu. Tab groups can also be created via drag-and-drop operations.

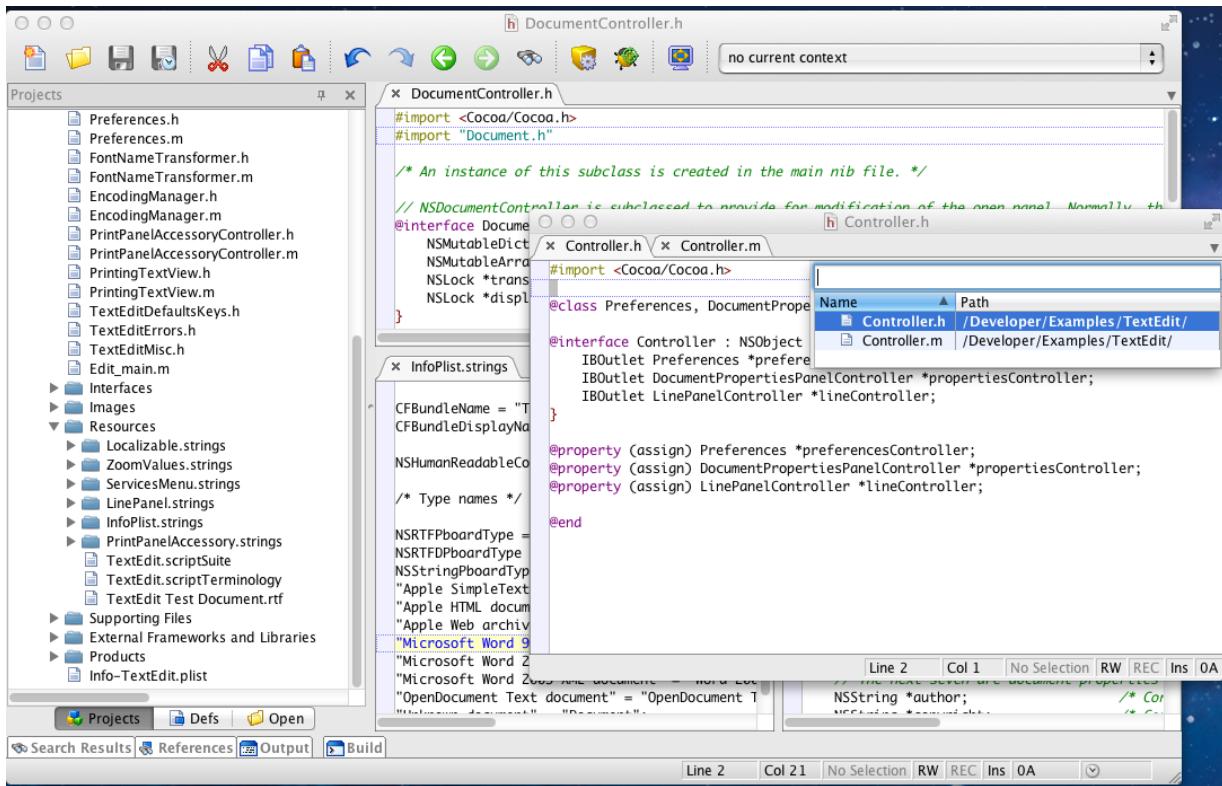
Editor Windows



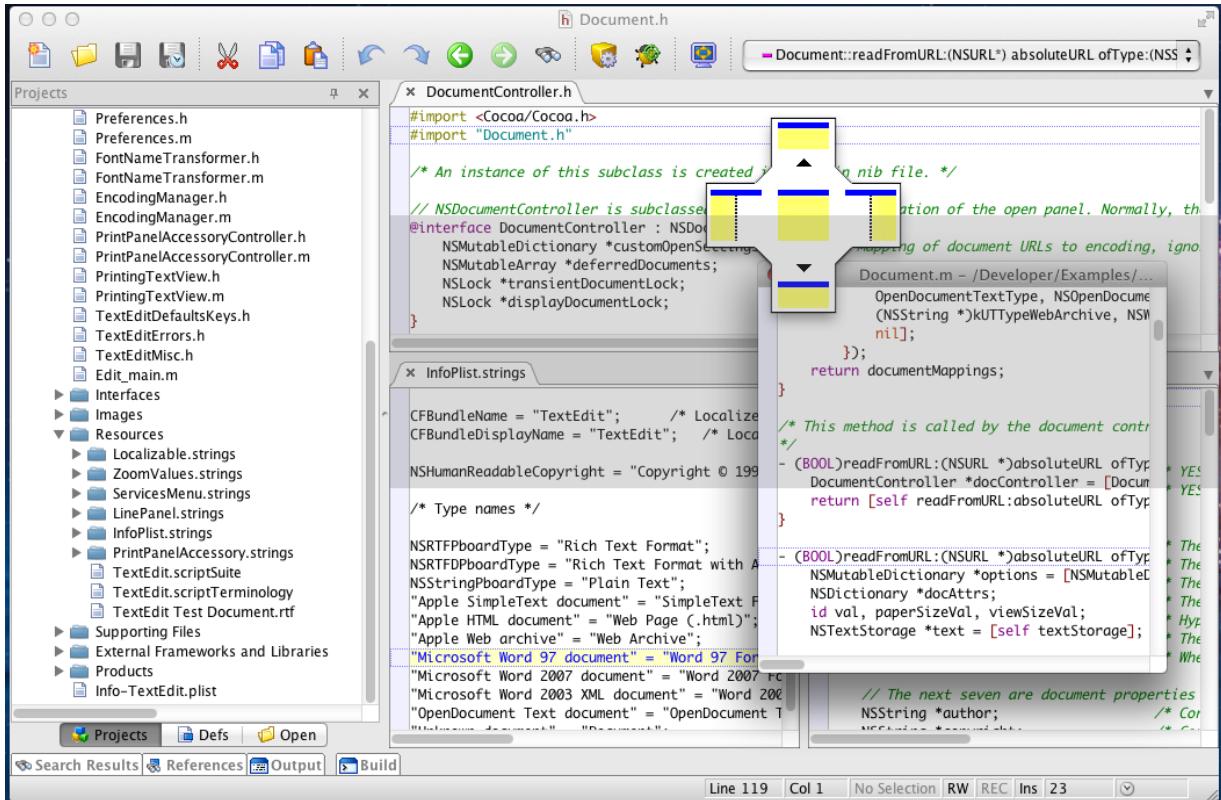
- Floating Tab Groups - Individual files can be dragged outside the main application window to create a floating tab group. You can also float entire tab groups from the main MDI area using the **Float All** command on the document tab context menu.

Document Selector Menu - The top-right corner of each tab group displays a drop-down arrow. Clicking that arrow presents a listing of all the documents contained in that tab group. The listing can be filtered by entering partial file names.

Editor Windows

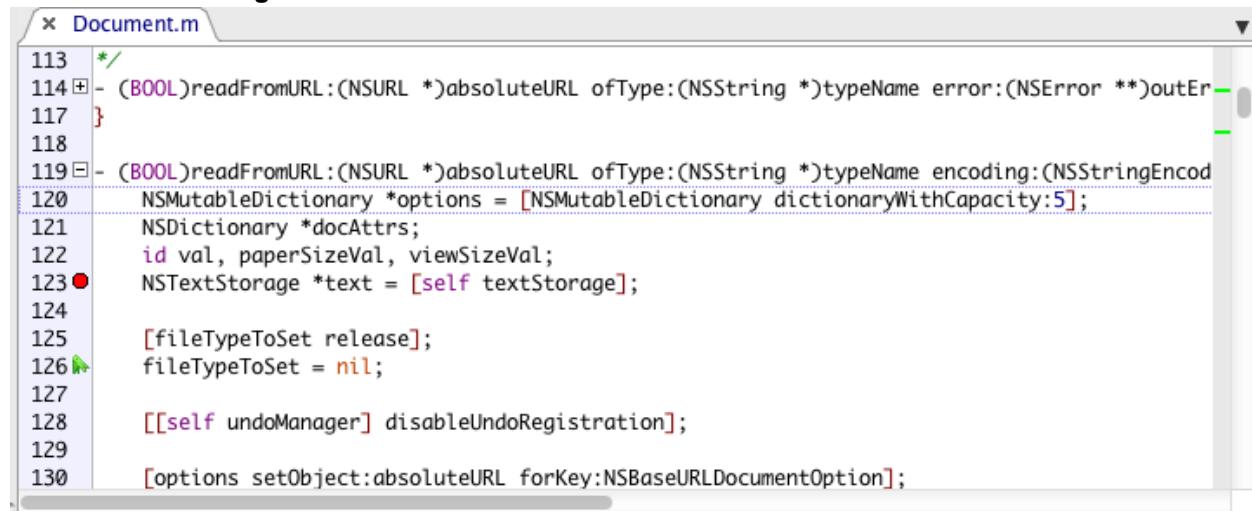


- Drag and Drop Guides - When using the mouse to drag a document tab you will see an overlay window that acts as a drag target, allowing you to easily specify where the document window should be placed. A grey prediction rectangle is also shown as a preview of what the new placement will look like.



Margin Icons

Each editor window contains a margin on the left, which is used to display icons and line numbers. The size of this margin is set by selecting **Tools** → **Options** → **Appearance** → **General** and setting the value for **Window left margin**.

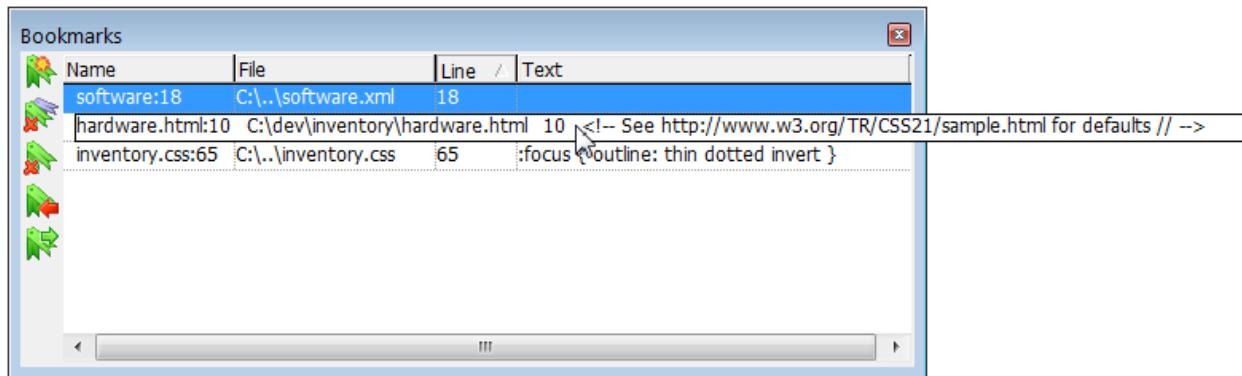


The window left margin may display any of the following icons:

Icon	Purpose
	Named bookmark (see Named Bookmarks).
	Pushed bookmark (see Pushed Bookmarks).
	(Pro only)Used to indicate an error on this line of code. This can be set to show the location of an error after doing a build or as part of Live Errors.
	Along with a minus icon, used to expand/collapse sections of code (see Selective Display).
	(Pro only)Breakpoint, see Setting Breakpoints .
	(Pro only)Disabled breakpoint.
	(Pro only)Watchpoint (see Watches and Watchpoints).
	(Pro only)Disabled watchpoint.
	(Pro only)Indicates an exception during debugging.

Tabular Lists

Several dialogs and tool windows in SlickEdit®, such as the [Bookmarks Tool Window](#) and the [Key Binding Options](#) screen, present data in a tabular list.



To sort the data, click on a column header to sort in ascending order, or click again to sort in descending order. An arrow in a column header indicates that the data is sorted by this column, and the direction of the arrow indicates the ascending (up arrow) or descending (down arrow) sort order.

You can resize columns by clicking and dragging the column separators. If a file path column is not large

enough, the text is elided so that you can always see the file name. If a text column is not large enough, the contents are abbreviated. You can always hover over a row with the mouse to see a tooltip that shows the entire contents of each cell.

Toolbars and Tool Windows

Toolbars are groups of buttons (called toolbar controls) that allow you to perform specific operations. Tool windows are used to display and manipulate various kinds of information. Both toolbars and tool windows can be docked. For documentation purposes, the terms "toolbars" and "tool windows" are sometimes 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](#), [Section searchingmemberssearchingclassesClass](#), [Open](#), [Projects Tool Window](#), and [Symbols](#) tool windows are docked into a tab group on the left side of the editor. The Standard toolbar contains commonly appearing icons, docked at the top 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 clicking **View → Toolbars** or **View → Tool Windows**, then selecting the item to display.

Note

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

You can also control the display of tool windows using the **activate-toolwindow** and **toggle-toolwindow** commands, replacing "toolwindow" with the name of the tool window to control, like **toggle-preview**.

- The activate commands make the tool window visible and switches the focus to it. If the tool window was part of a docked group, it becomes the top item in that group. Otherwise it is brought up floating.
- The toggle commands are used to bring up a tool window and then close it. This command behaves differently, depending on whether the tool window is part of a tab group:
 - **The tool window is not part of a tab group** - if it is already open, then the toggle command closes it; if it is not already open, then the toggle command opens it.
 - **The tool window is part of a tab group** - if the tab group is visible, the toggle command hides it; if the tab group is not visible, the toggle command makes it visible and brings the associated tool window to the front. The toggle command cannot be used to bring a tool window to the front in a tab group that is already visible; for that, use the associated activate command.

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.
- **Float** - This undocks (or floats) a Toolbar or tool window.
- **Dock** - This docks a floating Toolbar or tool window.
- **Hide** - Closes a Toolbar or tool window.
- **Duplicate** - If present, allows you to create another instance of a tool window. Most tool windows support this option. This is very useful if you want to dock a tool window to a floating window group but want the tool window to remain where it is.
- **Floating** - This is the default setting when a toolbar is first displayed. It allows you to click on the toolbar's title bar and drag it around within the editor pane. When this option is selected, the toolbar 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** button 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, click **View** → **Toolbars** or **View** → **Tool Windows**.

Caution

If you undock 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 Options](#). To display these options, from the main menu, click **Tools** → **Options** → **Appearance** → **Toolbars**. Or, right-click on any toolbar's background and select **Customize**.

To add a button to a toolbar, drag the button from the Toolbar Customization dialog to the intended toolbar (see [Toolbar Customization dialog](#)). To remove a button, drag it off the toolbar.

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 button image, as well as the command key binding and auto-enable properties.

The command may invoke an internal Slick-C command or external program. The following % options

allow you to use some information from the current editor window if there is one.

- **%F** - Replaced with the current editor filename. For example, notepad "%F". The double quotes are often needed for filenames with spaces.
- **%W** - Replaced with the current word in current editor window.
- **%L** - Replaced with the current line number in current editor window.

Customizing Tool Windows

Tool Windows can be customized using the [Tool Windows Options](#). To display these options, from the main menu, click **Tools** → **Options** → **Appearance** → **Tool Windows** or right-click in the tool windows title bar and select **Customize**.

Available Toolbars and Tool Windows

The display of toolbars and tool windows can be toggled on and off by clicking the items in the **View** → **Toolbars** or **View** → **Tool Windows** menus.

Toolbars

The available Toolbars are described below. They are listed in the **View** → **Toolbars** menu.

Debug (Pro only)

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 [Basic 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 (Pro only)

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 buttons for commands 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.

Current Context toolbar (Not in Community edition)

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 Toolbar](#) for more information.

Context Tagging® (Pro only)

Contains several buttons 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](#), [Files Tab](#), [Using the Calculator and Math Commands](#), [Spell Checking](#), and [Hex Mode Editing](#).

XML

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

Tool Windows

The tool windows listed below are used for various operations. Some tool windows are only available while debugging. They will only appear in the **View → Tool Windows** menu while you are debugging. Those tool windows are listed separately, in [Debug Tool Windows](#).

Backup History (Not in Community edition)

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 [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 clicking **Search → Bookmarks → Bookmarks Tool Window**. For more details, see [Bookmarks](#).

Breakpoints (Pro only)

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 (Pro only)

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, click **Tools → Options → Editing → General**, then set the option **Auto exit build window** to **True**.
- **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, click **Tools → Options → Editing → General**, then set the option **CR w/o LF erases line in build window** to **False**.

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

Class (Pro only)

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.

Clipboards

Allows you to preview and manage clipboards. Shows a list of recently used clipboards and lets you insert a clipboard into the current buffer. The tool window contains a Preview area that lets you see the entire contents of a clipboard with color coding. In addition to the **View → Tool Windows → Clipboards** menu item, there are several other ways to display the tool window:

- Using the menu item **Edit → List Clipboards (Ctrl+Shift+V or list_clipboards command)**

- Using the **activate_clipboards** command
- Toggle the window with the **toggle_clipboards** command

See [Clipboards](#) for more information.

Code Annotations (Pro only)

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.

Highlighting

Highlighting allows you to define a set of word patterns to highlight using an array of colors. The word patterns can be simple word matches, substring matches, or regular expressions. There are a variety of options for how to display the highlighted matches. You can also define Highlight profiles, custom sets of word patterns. See [Highlighting](#) for more information.

Code Annotations (Pro only)

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. information.

Defs (Not in Community edition)

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 (Pro only)

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

Feature Notifications

Displays a list of notifications about features that have altered the contents of your file. For example, a feature notification is displayed when **Syntax Expansion** expands an **if** or **for** statement. For more information see [Notifications](#).

File Tabs

Displays 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 on managing the File Tabs, see [File Tabs](#).

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 clicking **Document** → **List Open Files** (**Ctrl+Shift+B**), or by using the **list_buffers** command. For more information, see [Document Dialogs and Tool Windows](#).

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 clicking **Search** → **Find**. See [Find and Replace](#) and [Search Dialogs and Tool Windows](#) for more information.

Find Symbol (Pro only)

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 (Not in Community edition)

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 (Not in Community edition)

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.

Message List (Pro only)

Automatically displays output messages from processes running in SlickEdit®, such as build warnings and errors. This tool window is docked into the bottom tab group of the editor by default. In addition to the menu item, you can activate or toggle this window by using the **activate_messages** or the **toggle_messages** command, respectively. See [Message List](#) 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, see [Open Tool Window](#). For more information about opening files, see [Opening 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 (Pro only)

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 (Pro only)

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 clicking **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 [Search Dialogs and Tool Windows](#). 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 (Pro only)

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 (Pro only)

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 (Pro only)

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

Terminal (Pro only)

Manages one or more terminals which are run in an editor window. Type a shell command and press Enter to start an empty terminal window. To start a new terminal, right click on the Terminal tab and select "New". You can use the **sgrep** program to search for a string in one or more files. Traverse the matches found with the **next-error** command (Ctrl+Shift+Down).

Interactive (Pro only)

Manages one or more Interactive REPL shells which are run in an editor window. Shells are preconfigured for many languages including Clojure, CoffeeScript, C#, Groovy, Haskell, Lua, PHP, Perl, PowerShell, Python, R, Ruby, and Scala. See [Interactive Tool Window](#) for more details.

Debug Tool Windows (Pro only)

The tool windows listed below are used for debugging. In debugging mode, the **View → Tool Windows** menu displays these items along with the other tool windows. See [Debugging](#) for more information about working with these features.

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.

Loaded Classes

(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, click **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.

Menus

The SlickEdit main menu, displayed at the top of the editor, provides a way to access most of the editing features of SlickEdit. Context menus are available in the editor windows and tool windows with operations specific to those windows. 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 customize menus by adding items, deleting items, or changing the command that is executed. See [Creating and Editing Menus](#) for more information.

Right-Click Context Menus

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

Language-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, from the main menu, click **Tools → Options → Languages**, expand your language category and language, then select **General**. Choose 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 language-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 an 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 key bindings using the same **Alt** key combination.

Alt Menu

(Not supported on Mac) 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 (**Tools → Options → Keyboard and Mouse → Advanced**), 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 enable this option, set the **Alt menu hotkeys** option (**Tools → Options → Keyboard and Mouse → Advanced**) to **True**.

Caution

When this option is enabled, 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 enable this behavior, set the option **Short key names** (**Tools** → **Options** → **Appearance** → **Advanced**) to **Condensed**.

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

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.

Note

For information about passing arguments to SlickEdit from the operating system command line, see [Invocation Options](#).

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

- BBEdit - **Esc**
- Brief - **Esc**
- CodeWarrior - **Esc**
- CodeWright - **F9**
- CUA (SlickEdit's default emulation for all platforms but macOS) - **Esc**
- Epsilon - **Alt+X** or **F2**
- GNU Emacs - **Alt+X** or **F2**
- ISPF - **Esc**
- macOS (SlickEdit's default emulation for macOS) - **Esc**
- SlickEdit (Text Mode edition) - **Esc**
- Vim - **Ctrl+A**

- Visual C++ - **Esc**
- Visual Studio default - **Esc**
- Xcode - **Esc**
- Eclipse - **Esc**

See [Emulations](#) for more information.

Command Line History

The SlickEdit® 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. This history is stored in **vrestore.slk**, under your configuration directory. For more information about configuration files, see [Configuration Directories and Files](#).

Command Line Completion

As you type a command on the SlickEdit® command line, 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 options page (**Tools** → **Options** → **Keyboard and Mouse** → **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, click **Tools** → **Options** → **Appearance** → **General** and set the option **List command line completions** to **False**. Note that this option does not apply to the Vim command line.

Using Shortcuts Inside the Command Line

The SlickEdit® 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 options page (**Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings** or `gui_keybindings` command) to see a list of command/key binding associations. See [Key Binding Options](#) 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. Click **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. Click **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.

Command Line Switches

In addition to setting options through the graphical interface, you can specify or override some options on the SlickEdit® command line for immediate, one-time use. This way, you don't need to constantly open the Options dialog to change an option every time you want to enable or disable it. For example, when

using the **save** or **save_as** command, you can specify many of the [Save File Options](#), such as **Expand tabs to spaces**, for just this one operation.

Switches are described in the documentation when the switch is helpful or applicable. To use a switch, type it between the command and file name. Depending on whether you want to enable or disable the option, type a plus (+) or minus (-) sign before the switch character. No matter the default setting, the specified switch will be used.

For example, perhaps you have the Save File Option **Expand tabs to spaces** set to **False**. This means that when you save a file, tabs are not expanded to spaces. However, you may want to quickly save a file with tabs expanded. To enable the option just this once, use the **save** command with the **E** switch, as follows:

```
save +E
```

Starting a Program from the Command Line (Shelling)

You can use the SlickEdit® 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, click **Tools → OS Shell**.

Command Line Prompting

Many commands that display dialog boxes have equivalent commands that prompt for arguments on the SlickEdit® command line. For example, the **gui_open** command (**File → Open** or **Ctrl+O**), which displays the Open file dialog, corresponds to the **edit** command, which is used to open files via the SlickEdit command line. If you frequently use key bindings to open dialogs, a faster method of entering arguments is to use Command Line Prompting. When this feature is enabled, you are prompted on the command line for arguments that you would otherwise select as options on a dialog. For example, instead of displaying the Open file dialog when you press **Ctrl+O**, the SlickEdit command line is invoked, so you can type the name of the file to open and any other desired arguments. To enable Command Line Prompting, from the main menu, select **Tools → Options → Keyboard and Mouse → Advanced** and set the **Command line prompting** option to **True**.

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

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

Common SlickEdit® Commands

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

e <i>filename</i>	Edit a file
sa <i>filename</i>	Save file as
<i>number</i>	Go to line number
f <i>symbol</i>	Find a symbol
/ <i>search_string</i> / <i>options</i>	Search for a string
c/ <i>search</i> / <i>replace</i> / <i>options</i>	Replace a string
gt/ <i>search</i> / <i>options</i>	Substring search for a symbol
sb <i>name</i>	Set a bookmark
gb <i>name</i>	Jump to a bookmark
help <i>topic</i>	View Help on topic

Screen Management

man <i>command</i>	Show UNIX man page
cd <i>directory</i>	Change directory
dir <i>directory</i>	Show directory in the File Manager
list <i>wildcards</i>	Show directory tree in the File Manager
del <i>filename</i>	Delete file
pushd <i>directory</i>	Push directory
popd	Pop directory
set <i>env=value</i>	Set environment variable
dos <i>command</i>	Execute command outside of editor
math <i>expr</i>	Evaluate expression. There are also mathx, mathb, and matho which output in hex, binary, and octal.
o <i>filename</i>	Opens a file. On Windows, this uses WinExec to open the file. On Unix, this either edits the file or runs the program that is associated using SlickEdit File Extension Manager. This command is useful for running an external program to open a graphic resource file.

Screen Management

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

Full Screen Mode

To get the largest possible view of your code, use Full Screen Mode. This hides all of the toolbars and tool windows, expanding the editor frame to fill the application window. This is useful when you need screen real estate for editing large files and/or when tool windows and toolbars are not needed.

To activate full screen mode, from the main menu, click **View → Full Screen**, or use the **fullscreen** command.

Multiple Monitor Support

SlickEdit® supports the use of multiple monitors on Windows, UNIX, and macOS 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 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.

Using the Mouse and Keyboard

SlickEdit® provides four ways to launch operations: commands, menu items, key bindings, and buttons. For example, to bring up the **Find and Replace** dialog, you could use any of the following methods:

- Type the **gui_find** command on the SlickEdit command line.
- Click **Search → Find** in the main menu.
- Press the key binding **Ctrl+F**.
- Click the binoculars button on the Standard toolbar.

The command forms the basis of each method. 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 [SlickEdit® Command Line](#) for more information about commands, and [Key and Mouse Bindings](#) for more information about bindings.

Emulations

SlickEdit has the capability of emulating other editors. An emulation controls the key sequences used to invoke operations and many of the behaviors of the editor. For more information, see [Emulations](#).

Platform-Specific Notes

macOS Notes

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 macOS operating system. The documentation contains specific information for the macOS operating system where relevant. Mouse and keyboard shortcuts in the documentation are written for Microsoft Windows, but can be adapted for macOS 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 macOS 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.

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

Microsoft Windows	macOS
Right-click with the mouse	Ctrl+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)

Key Shortcuts in Text Boxes

Most keyboard shortcuts for basic text operations can be used inside any text box in SlickEdit®, including the SlickEdit command line, which is also a text box. The table below shows a list of these shortcuts, based on the CUA emulation.

Note that even if you are not using the CUA emulation, by default, you can still use the common Cut/Copy/Paste keyboard shortcuts inside text boxes (**Ctrl+X**/**Ctrl+C**/**Ctrl+V**, respectively). To disable this capability, so that you can use your emulation's Cut/Copy/Paste shortcuts, from the main menu, click **Tools** → **Options** → **Editing** → **General**, then set the option **CUA text box** to **False**.

Text Box Editing Operation	Key Shortcut
Append Cut	Ctrl+Shift+X
Append to Clipboard	Ctrl+Shift+C
Copy Word to Clipboard	Ctrl+K
Copy	Ctrl+C
Cut Line	Ctrl+Backspace
Cut to End of Line	Ctrl+E
Cut Word	Ctrl+Shift+K

Redefining Common Keys

Text Box Editing Operation	Key Shortcut
Cut	Ctrl+X
Delete Character Under Cursor or Selection	Delete
Delete Previous Character or Selection	Backspace
Expand Alias	Ctrl+Shift+O
Expand Partially Typed Parameter or Insert Space	Spacebar
Extend Selection to Mouse Position	Shift+Click
Insert Mode Toggle	Insert
List Clipboards	Ctrl+Shift+V
List Matches to Partially Typed Parameter	?
Lowercase Word	Ctrl+Shift+L
Move Cursor Left	Left arrow
Move Cursor Right	Right arrow
Move Cursor to Beginning of Line	Home
Move Cursor to End of Line	End
Next Word	Ctrl+Right arrow
Paste	Ctrl+V
Previous Word	Ctrl+Left arrow
Select Line	Triple-click
Select Text Between Cursor and Beginning of Line	Shift+Home
Select Text Between Cursor and End of Line	Shift+End
Select Word	Double-click

Redefining Common Keys

Many users have a preference for the functions of the keys **Backspace**, **Delete**, **Enter**, **Tab**, and **Home**. Options are available for changing the function of these keys. To access these options, from the main menu, click **Tools** → **Options** → **Keyboard and Mouse** → **Redefine Common Keys**.

For a description of each option, see [Redefine Common Key Options](#). For more information on changing Tab key functions, see [Indenting with Tabs](#).

Using the Minimap

By default, a minimap (miniaturized view of your source file) is displayed to the right of your edit window. It allows you to see more of your source file.

Note

By default, the minimap uses the same font as your edit window in a smaller size. You can choose a specific font name for the minimap Window (**Tools** → **Options...** → **Appearance** → **Fonts**). On Windows, the default font height is typically 3 pixels. If you choose Lucida Console for the minimap font name (there are others that are typically 2 pixels), you can reduce the pixel height to 2 pixels to display more lines.

On Windows (which defaults to a 1 point font), you can try turning anti-aliasing off to make the font look brighter. For macOS, we recommend leaving anti-aliasing on since small fonts typically look quite a bit better when anti-aliasing is on. Use the Minimap Context Menu to toggle anti-aliasing on/off for a particular minimap font size.

Viewing the Minimap

You can turn the minimap on/off one of the following ways:

- **View** → **Minimap**. This effects the current buffer only. If you, exit the editor, auto restore will restore your minimap windows to their previous state.
- **Tools** → **Options** → **Languages** → **All Languages** → **View** and set the Minimap check box. This will turn the minimap on/off for all languages.
- **View** → **[Language] View Options...** and set the Minimap check box. This will turn the minimap on/off for the current edit buffers language.

Minimap Context Menu

The menu items provide the following:

- **Zoom In** - Increases the size of the minimap font.
- **Zoom Out** - Decreases the size of the minimap font.
- **Unzoom** - Restores the minimap font to the default setting.
- **Color** - Select color profile for minimap windows. For more information, see [Window Color dialog box](#).
- **Move Cursor On Click** - Determines whether clicking on a line outside the minimap slider moves the cursor to a line just scrolls the line into view.

- **Set Minimap Default Font Size.** Sets the default minimap font size to the current minimap font size.
- **Anti-aliasing for Current Font Size.** Toggles anti-aliasing on/off for the current minimap font size.
- **Show → Vertical Lines.** Determines whether vertical lines are drawn in the minimap window.
- **Show → Modified Lines.** Determines whether modified and inserted line colors are displayed in the left margin of the minimap window.
- **Show → ToolTip.** Determines whether a tooltip window is displayed which shows lines under the mouse pointer in a larger font.
- **Show → Symbol Coloring.** Determines whether symbol color highlighting is displayed in the minimap window.
- **Delayed Minimap Updating for Faster Scrolling.** Determines whether modifications in the edit window are immediately displayed in the minimap window. On macOS, which has slower graphics than Windows or Linux, you may find it better to use delayed minimap updating for faster scrolling in the edit window.
- **Width Defaults → Set Fixed Width (Percentage Width not used).** Sets the default width for all minimap windows to the width of the current minimap window. Drag the minimap vertical size bar to change the width of the minimap window.
- **Width Defaults → Set Maximum Width (uses Percentage Width).** Sets the default maximum width for all minimap windows to the width of the current minimap window. The current default Percentage Width gets applied to all the minimap windows as well. If the default percentage width is less than the maximum width setting, then the minimap window will get smaller. Drag the minimap vertical size bar to change the width of the minimap window.
- **Width Defaults → Percentage Width Up To Maximum Width → [Percentage].** Sets the default percentage width for all minimap windows to the percentage selected. For example, if the width of the edit window area is 1000 pixels and 25% is chosen, then the minimap window will be 250 pixels. However, the maximum width still gets taken into account. If the maximum is 300 pixels, the minimap window will be no larger than 300 pixels. However, if the maximum width is only 150 pixels, then the minimap window will be no larger than 150 pixels. Using a percentage width and a maximum width is useful when the width of the edit window gets smaller so the minimap window doesn't take up as much space.

User Preferences

This chapter describes how to set key options that control the look and behavior of SlickEdit. For a complete description of all options screens, see [Options Dialog](#).

Introduction to User Preferences (Options)

SlickEdit is one of the most configurable editors available. User preferences, also called options, can be set to change the appearance and control the behavior of most editing features.

User preferences are set using the Options dialog, which can be displayed from the main menu by clicking **Tools** → **Options**. For more information including a detailed breakdown of options, see [Options Dialog](#).

SlickEdit contains two kinds of preferences:

- [Global Options](#) - affect all languages.
- [Language Specific Options](#) - affect only the specified language.

Tip

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

Global Options

Global options affect all languages and include the following:

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

Other global preferences, such as default search options and selection styles are also available for setting through the Options dialog (**Tools** → **Options**). These options are described in the documentation on a contextual basis. For a listing of options categories, see [Option Categories](#).

Language-Specific Options

The behavior of the editor can be customized for files based on specific languages. indent, word wrap, comment, auto-complete, Context Tagging®(Pro only), and other code-style settings are all language-specific. These settings are located on the Options dialog (**Tools** → **Options** → **Languages** →

Saving, Restoring, and Backing-up User Preferences

[Language Category] → [Language]). The options are described in the documentation on a contextual basis. For a flat listing of the language-specific options, see [Language Options](#).

Tip

A shortcut method to access language options for the current buffer is to use the **Document** → **[Language] Options** menu item (or the **setupext** command). This will open the Options dialog to the **General** language-specific option screen for that language.

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

Saving, Restoring, and Backing-up User Preferences

SlickEdit stores all option settings in your configuration directory. Most options are stored in `user.cfg.xml`. User created macros, forms, menus, and toolbars are cached in the state file for faster editor launch. It is safe to delete the state file (`vslick.sta`) since it will be recreated if needed. There are other option files. For more information, see [Configuration Directories and Files](#).

If you've made many configuration changes, you should make a backup of your configuration directory or export all your option settings (see below). This will ensure that if your machine has a hardware failure, you still have a copy of the your configuration. You may find that just backing up `user.cfg.xml` is sufficient.

Options Export and Import

SlickEdit also provides the capability to Export and Import your option settings. You can create an export package of all or part of your SlickEdit options. You can use this to:

- Backup and restore your option settings.
- Share selected options with other team members.
- Transfer your options from one machine to another.

Note

Moving options to a machine with a different operating system will mostly work. Options that are per platform (Windows, Mac, Unix), will be imported to the other platform (stored in `user.cfg.xml`) but will have no effect (like fonts, filenames, and paths). Options exported from one version of SlickEdit and then imported into a newer version is only fully supported for the previous release. While this typically works, there may be some issues when importing from very old releases of SlickEdit.

For more information see [Export/Import Options](#).

Sharing Your Configuration with Multiple Instances

By default, SlickEdit supports running multiple instances which share one configuration directory. Be sure this feature is turned on by setting "**Save configuration**" to "**Share config - Save configuration immediately**" (**Tools** → **Options** → **Application Options** → **Exit** → **Save configuration**). For example, if you start two instances of SlickEdit, change your edit window color profile, then switch to the other instance, you will notice that this configuration change is transferred. This sharing feature supports almost all configuration options (emulation, key bindings, theme, recorded macros, fonts, etc.). See "Limitations with sharing your configuration with multiple instances" below for information on limitations.

This makes it easy to launch two different instances of SlickEdit and open a different workspace in each. Use an additional instance of SlickEdit for a quick edit or use SlickEdit as the git comment editor. There's no need to maintain multiple configuration directories.

Limitations with sharing your configuration with multiple instances

The following are limitations with sharing your configuration with multiple instances:

- Sharing will not work on Unix or macOS if NFS file locking doesn't work or has been disabled with the `-sul` invocation (`fcntl(fh,F_SETLK64,...)`).
- Loading a hot fix does not support shared configurations. If you've been running multiple instances of SlickEdit, exit all instances of SlickEdit and restart SlickEdit before loading a hot fix.
- Auto Restore data (`vrestore.slk`) is not shared. (i.e. File History, Project History, Tool window layout, command retrieval, Dialog history/position, and multiple clipboards). The last instance to exit will overwrite `vrestore.slk`. The project information under "Project>All Workspaces" is shared because it is not stored in `vrestore.slk`.
- Per file auto restore data (`perfile.xml`) is not shared. (i.e. last file position, encoding override, softwrap override, language mode override, hex mode, selective display). The last instance to exit will overwrite `perfile.xml`.
- Language specific tag files may cause problems with multiple instances. These can be generated when you first type in source for a specific language, when you use the "Tag Compiler Libraries" dialog, and when you manually add one. If you add a language specific tag file, make sure to wait until the current instance completes creating the tag file before switching to the other instance. Otherwise failures will occur. The plan is to protect against this issue in the near future.
- Opening the same workspace and/or project in two different instances may cause problems with tag files and settings.
- Single file project data is not saved until you switch to a different file. If you want your current single file project data transferred to another instance, switch to a different file or close the file. Also make sure the other instance isn't already editing this file before switching to it. Switching files in the other instance usually allows for updating the single file project settings.
- If an options dialog is already displayed in another instance, the options displayed will not be updated. Applying any options may overwrite the auto-reloaded options.

- Deleting a form, menu, or toolbar currently doesn't work. It won't be deleted in the another instance when these options are transferred. The deleted form, menu, or toolbar may get regenerated by another instance and not be deleted for newly launched instances either.
- If you're using custom macros:
 - Make sure your macros can easily be found and automatically loaded at startup. If you put your macros in the unversioned configuration directory and load them from there, SlickEdit will be able to easily find them without the VSICKPATH environment variable being modified.
 - If your additional macros can not be automatically loaded at startup, you will get failure loading macro errors any time you run an additional instance of SlickEdit.
 - If you have many additional macros and they take a long time to load, running an additional instance of SlickEdit will be slow.
 - If you modify and reload a macro, your changes will not be synced with additional instances of SlickEdit. However, modified and reloaded recorded macros will sync with additional instances.
 - installing or uninstalling a plugin will sync with additional instances of SlickEdit. Modifying and reloading a macro from a plugging won't sync.

Support for .editorconfig Files

SlickEdit supports .editorconfig files. The indent_style, indent_size, tab_width, end_of_line, and trim_trailing_whitespace properties are supported. You may find this useful for defining options for a specific project or directory of code.

In order to enable .editorconfig file support, you must turn this feature on (**Tools → Options → Editing → General → Apply .editorconfig file settings**). For documentation on .editorconfig files, go to <http://editorconfig.org/>.

Emulations

Emulation is the process of imitating another program. SlickEdit® provides emulations of key bindings for 14 editors so that you can use the style to which you are accustomed, making your coding experience as efficient as possible.

The Key Bindings option screen 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. For a list of key bindings that open the SlickEdit® command line in each emulation, see [Activating the Command Line](#).

Note

If you are a Windows user running the Mac version of SlickEdit with a non-Mac like emulation, you may want to reconfigure the macOS window manager which takes over a number of keys like `Ctrl+Left` and `Ctrl+Right`.

- **BBEdit**
- **Brief** - This emulation relies heavily on **Alt** key bindings.
- **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 for all platforms but macOS.**
- **Eclipse**
- **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](#)).
- **macOS** - Very similar to CUA emulation. Certain keys were changed due to macOS standards such as **Command+Comma**, **Ctrl+A**, and **Ctrl+E**. A number of keys were unbound or changed due to conflicts with the macOS window manager such as **Ctrl+Left**, **Ctrl+Right**, **F9-F12**, and **F4**. **This is the default**

emulation mode used by SlickEdit for macOS.

- **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 and a comma (,) indicates sequential 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 lowercase **w**, release, then immediately press **Shift** plus **w** to enter the uppercase **W**. Another example is the key binding **gP**, which pastes the text before the cursor. Press the **G** key (to enter a lowercase **g**), release, then press **Shift** plus **p** at the same time (to enter the uppercase **P**).

Tip

SlickEdit supports the **vimtutor** command. This opens a practice file in the editor that you can actually edit as you learn Vim commands. See [Vim Tutorial](#) for more information.

- **Visual C++ 6**
- **Visual Studio default** - The key bindings provided for the Visual Studio default 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 using the Emulation Options. To access these options, from the main menu, click **Tools** → **Options**, expand **Keyboard** in the tree, and select **Emulation**.

Emulation

- CUA
- Mac OS X
- Visual Studio default
- Visual C++ 6
- SlickEdit (text mode edition)
- Brief
- CodeWright
- Eclipse
- Vim
- GNU Emacs
- Epsilon
- ISPF
- CodeWarrior
- Xcode
- BBEdit

Restore to default key bindings

The CUA (Common User Access) keyboard emulation uses a command set similar to that used in Microsoft Word and Notepad.

Custom key/mouse bindings for the current emulation are always saved before switching emulations. This ensures that 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, click the **Restore to default key bindings** button on the Emulation options page.

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 option screen 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** - Any single key on the keyboard.
- **Key combination** - Two or more keys pressed simultaneously, for example, **Ctrl+O** (in CUA emulation, associated with the **gui_open** command, **File** → **Open**, and the **Open** button 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**.
- **Key sequence** - A series of one or more keys or key combinations, for example, **Ctrl+X,R** (in Vim emulation, this binding is associated with the **redo** command, **Edit** → **Redo**, and the **Redo** button 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.
- **Key binding range** - A command bound to a range of keys. For example, the **alt_bookmark** command is bound by default to the key combination range of **Ctrl+0** through **Ctrl+9**. Press **Ctrl+0** to create a bookmark named "0", **Ctrl+1** to create a bookmark named "1", etc.

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 are for the CUA emulation. To change the emulation mode, click **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 option screen. This 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 screen 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 option screen, from the main menu, click **Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**, or use the **gui_keybindings** command.

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

Key Bindings

Search by command: Search by key sequence: OR

Command	Key Sequence	Mode	Recorded
+			No
-			No
--ex-bufdo			No
-SetSymbolColoringSchemeName			No
/			No
0			No
abort			No
actionscript-begin			No

Moves the cursor the specified number of lines down.

Parameters:
NofLines - Number of lines

AppliesTo:
Edit_Window, Editor_Control

Categories:
CursorMovement_Functions

Import... Export... Save Chart... Remove Add...

Click Add to add binding to current command. Click Remove to unbind current selected command.

Bindings are based on the editor emulation mode (CUA is the default). To change the emulation mode, click **Tools** → **Options** → **Keyboard and Mouse** → **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

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 Mode](#) for more information.

The bottom of the screen 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 following sections describe different ways to use the Key Bindings option screen:

- [Viewing and Filtering Bindings](#)
- [Creating Bindings](#)
- [Editing Bindings](#)
- [Removing Bindings](#)
- [Exporting and Importing Bindings](#)
- [Saving a Bindings Chart](#)
- [Running a Command/Macro using the Key Bindings Dialog](#)
- [Resetting Default Bindings](#)
- [Working with Key Binding Ranges](#)

Viewing and Filtering Bindings

You can filter the data on the Key Bindings screen 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 find a mouse event, place the focus in the **Search by key sequence** box (by tabbing or using the mouse) and click the mouse event you want to find. If the mouse event involves the scroll wheel, click the **Mouse Event** button () to the right of the field. This displays the Select Mouse Event dialog containing a list of all mouse events. If the event involves pressing a modifier key or keys, such as **Ctrl**, **Alt**, **Shift**, **Cmd**, **Ctrl+Alt**, etc., in conjunction with a mouse click, for example, **Ctrl+RButtonDn**, press the modifier key(s) when clicking the **Mouse Event** button. Then the Select Mouse Event dialog shows a list of modifier-prefixed mouse events. After selecting the mouse event you want to look up, click **OK**.

The option screen updates to show only those commands that are bound to that mouse event.

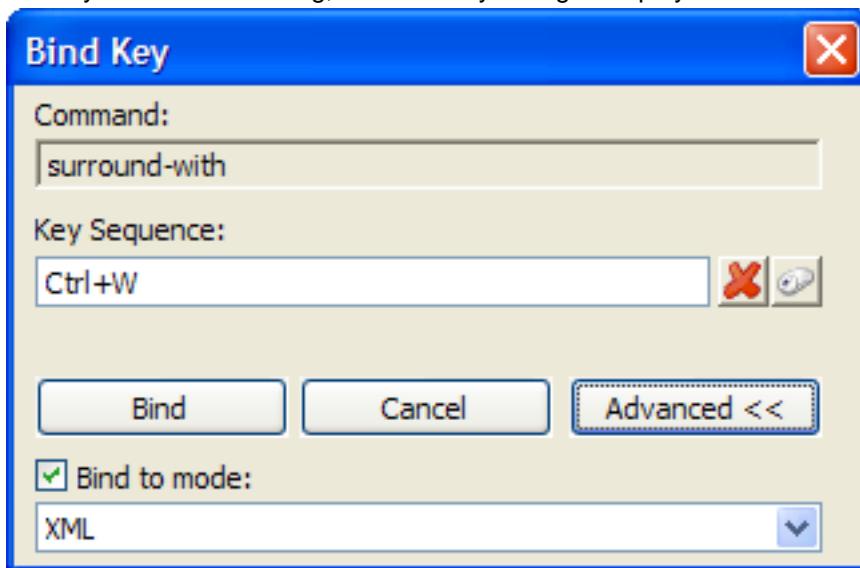
To clear either field, click the red X button to the right of each box. This is especially handy for the key sequence search, due to the fact that the field recognizes any keyboard/mouse input including **Backspace**.

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 options screen (**Tools → Options → Keyboard and Mouse → Key Bindings**), 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.
3. 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 next to the **Key Sequence** field, and select the mouse

event you want to use from the Select Mouse Event dialog. For example, to bind **surround_with** to the right-click mouse event, select **RButtonDn** and click **OK**.

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.

Tip

- 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.

4. 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.

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](#). Key binding changes are stored in `user.cfg.xml`. See [Editing a Key Binding Profile](#) for more information.

Removing Bindings

To remove a binding:

1. Using the Key Bindings options screen, 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.

If a command is bound to a range (see [Working with Key Binding Ranges](#)), for example, **Ctrl+0** through **Ctrl+9**, the entire range is unbound.

Exporting and Importing Bindings

Key and mouse bindings can be exported out of SlickEdit® and imported in. This can be useful for sharing just your key bindings with other team members. Copying your `user.cfg.xml` from one machine's configuration directory to another is a better way to copy most of your configuration changes (not just key bindings).

Exporting Bindings

When exporting, 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 option screen. 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.cfg.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 option screen. 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 option screen 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 option screen 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, click **Tools** → **Options** → **Keyboard and Mouse** → **Emulation**, then select the **Restore to default key bindings** option on the Emulation options screen. See [Emulations](#) for more information.

Working with Key Binding Ranges

A key binding range is a command that is bound to a range of keys. For example, the **alt_bookmark** command is bound by default in CUA emulation to the key combination range of **Ctrl+0** through **Ctrl+9**. Press **Ctrl+0** to create a bookmark named "0", **Ctrl+1** to create a bookmark named "1", etc. Key binding ranges are displayed in the **Key Sequence** column on the Key Bindings option screen. For example, the range for **alt_bookmark** is displayed as **Ctrl+0 -> Ctrl+9** in CUA emulation. Key binding ranges are also shown when using the **Export** and **Save Chart** features.

You cannot remove a single key combination from within a range, but you can rebind the key range to a different command. If you unbind a command that is bound to a range, the entire range is unbound. This is a limitation of the dialog and not the key binding system. The **unbind-key** command can be used from the SlickEdit command line to unbind a single key combination from a key range. It unbinds the next key sequence pressed.

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, click **Tools** → **Options** → **Keyboard and Mouse** → **Advanced**, then set the **Key message delay** for the amount, in tenths of a second, to delay before a prefix key. 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.

Cursor, Mouse, and Scroll Settings

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, click **Tools** → **Options** → **Appearance** → **General**, then select **Use block cursor** from the **Cursor style** drop-down list.

Hiding the Mouse Pointer

To hide the mouse pointer when typing, from the main menu, click **Tools** → **Options** → **Appearance** → **General**, then set the option **Hide mouse pointer** to **True**. 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, from the main menu, click **Tools** → **Options** → **Appearance** → **Advanced**, then set the option **Show tool tips** to **False**. To change the amount of time before tool tips are displayed, change the value of the option **Tool tip delay**. 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, click **Tools** → **Options** → **Appearance** → **General**, then set the options **Horizontal scroll bar** and/or **Vertical scroll bar**. When these options are set to **True**, 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, click **Tools** → **Options** → **Appearance** → **General**, then set the **Scroll styles** settings that you want 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:

- **Smooth horizontal scroll** - When set to **True**, editor windows scroll column-by-column when the cursor moves out of view. When set to **False**, the cursor is centered and the text is scrolled one-fourth the width of the window when the cursor moves out of view.
- **Smooth vertical scroll** - When set to **True**, editor windows scroll line-by-line when the cursor moves out of view. When set to **False**, the cursor is centered and the text is 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.

Fonts

This section describes how to set the fonts used in various screen elements.

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.

Setting Fonts for Screen Elements

To configure font settings for screen elements, use the Fonts options screen (**Tools** → **Options** → **Appearance** → **F**onts). For a description of each option, see [Font Options](#).

Tip

The "Regular" (non-Unicode) editor window font is set by the **SBCS/DBCS Source Windows** element.

The "Unicode" editor window font is set by the **Unicode Source Windows** element.

Setting Fonts for Screen Elements

Fonts

Element	Font Name	Size	Style
Editor Windows			
SBCS/DBCS Source Windows	Consolas	10	
SBCS/DBCS Minimap Windows	Consolas	1	
Hex Source Windows	Consolas	10	
Unicode Source Windows	Consolas	10	
Unicode Minimap Windows	Consolas	1	
File Manager Windows	Consolas	10	
Diff Editor SBCS/DBCS Source Windows	Consolas	10	
Diff Editor Unicode Source Windows	Consolas	10	
HTML Controls			
Parameter Information	Tahoma	9	
Parameter Information Fixed	Consolas	9	
HTML Proportional	Times New Roman	12	
HTML Fixed	Courier New	12	
Application			
Command Line	Consolas	10	
Status Line	Tahoma	10	
Selection List	Courier	10	
Dialog	Tahoma	8	
Document Tabs	Tahoma	8	

Description

Editor windows that are displaying non-Unicode content (Windows: plain text).

Sample

== Line before ==
Aa_Bb_Cc = (11 + 00);
== Line after ==

Use fixed spacing for bold and italic fixed Unicode fonts
 Use anti-aliasing

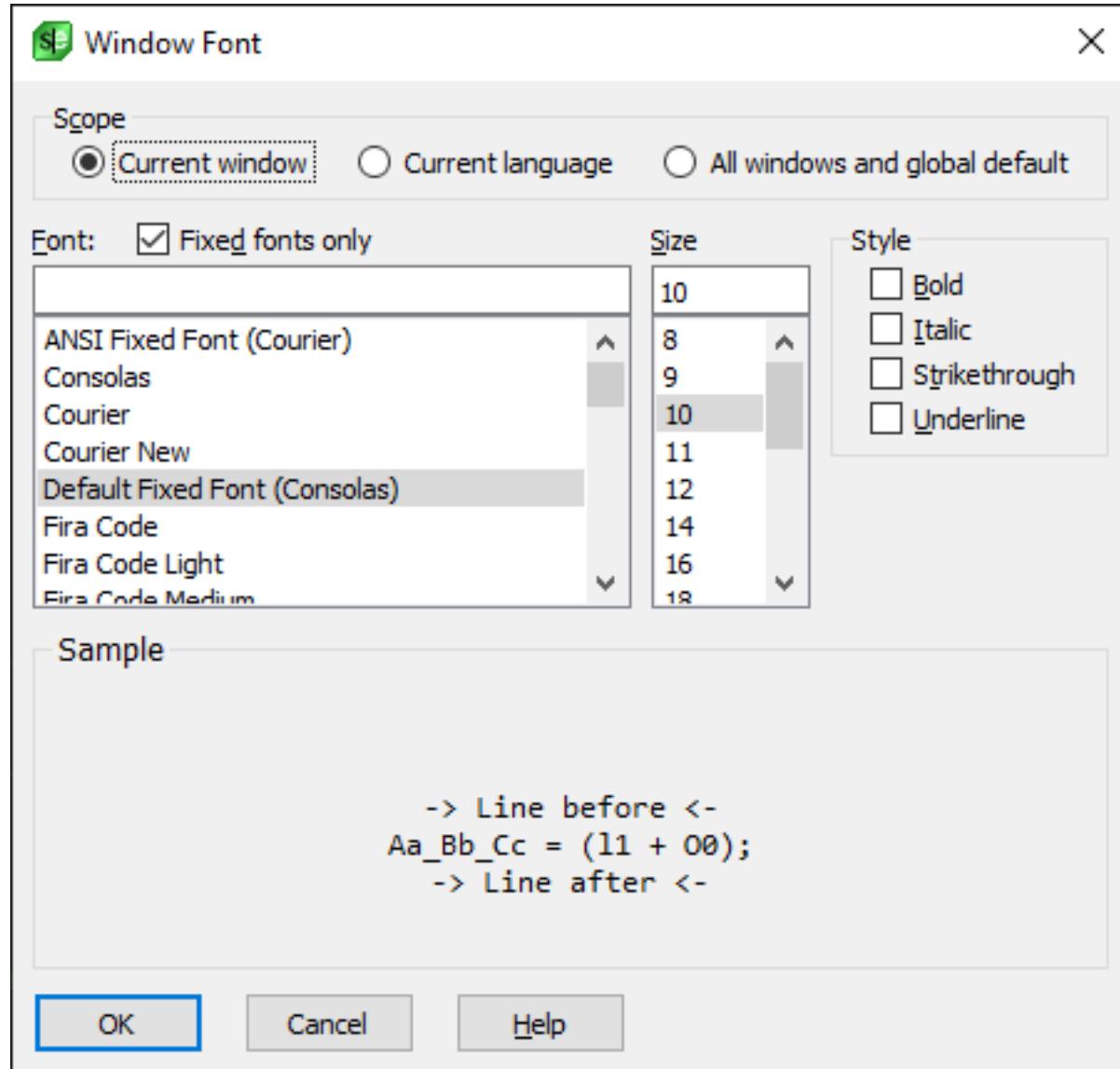
Some font names are portable font names which are translated into other fonts. This allows Slick-C® macros and dialog boxes to be portable across Windows and UNIX.

Default Unicode Font is the default font for the Unicode Source Windows element. When this font is selected on Windows, 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

You can set the font for editor windows by selecting the **SBCS/DBCS Source Windows** element or the **Unicode Source Windows** element in the **FONT** dialog, as described above. You can also change the font for editor windows by selecting **Window → Font**, or by using the **wfont** command. The Window Font dialog is displayed. Font, size, and style options are the same as those on the [Font Options](#) screen.



Tip

Use the **Window Font** dialog to set the font for a single editor window.

Colors, Color Coding, and Symbol Colors

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 Dialogs and Tool Windows](#).

Colors, Color Coding, and Symbol Colors

SlickEdit provides a great deal of control over the colors in the editor, using two complimentary coloring systems. With Color Coding you can color your code based on syntactic information about the elements: keywords, strings, operators, etc. Using Symbol Coloring, you can define rules to color symbols based on scope, visibility, and other detailed properties. Symbol Coloring provides more detailed information for identifiers that would otherwise be colored the same using Color Coding.

Note

Colors for the SlickEdit application window are controlled by the operating system. This includes the font and background color for tool windows and dialogs. The colors in editor windows are controlled by SlickEdit.

This section is divided into the following subsections:

- [Colors](#) - describes how to set the colors for various entities in the editor window. These colors are applied to items identified by the Color Coding engine.
- [Symbol Coloring](#) - describes how to define rules to color symbols based on scope, visibility, etc.
- [Color Coding](#) - provides information about the Color Coding engine.

Colors

Use the Colors option screen (**Tools** → **Options** → **Appearance** → **Colors** or the **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. Non-editor window colors and backgrounds are set using the application theme or using the facilities provided by the operating system. See [Application theme](#) for more information.

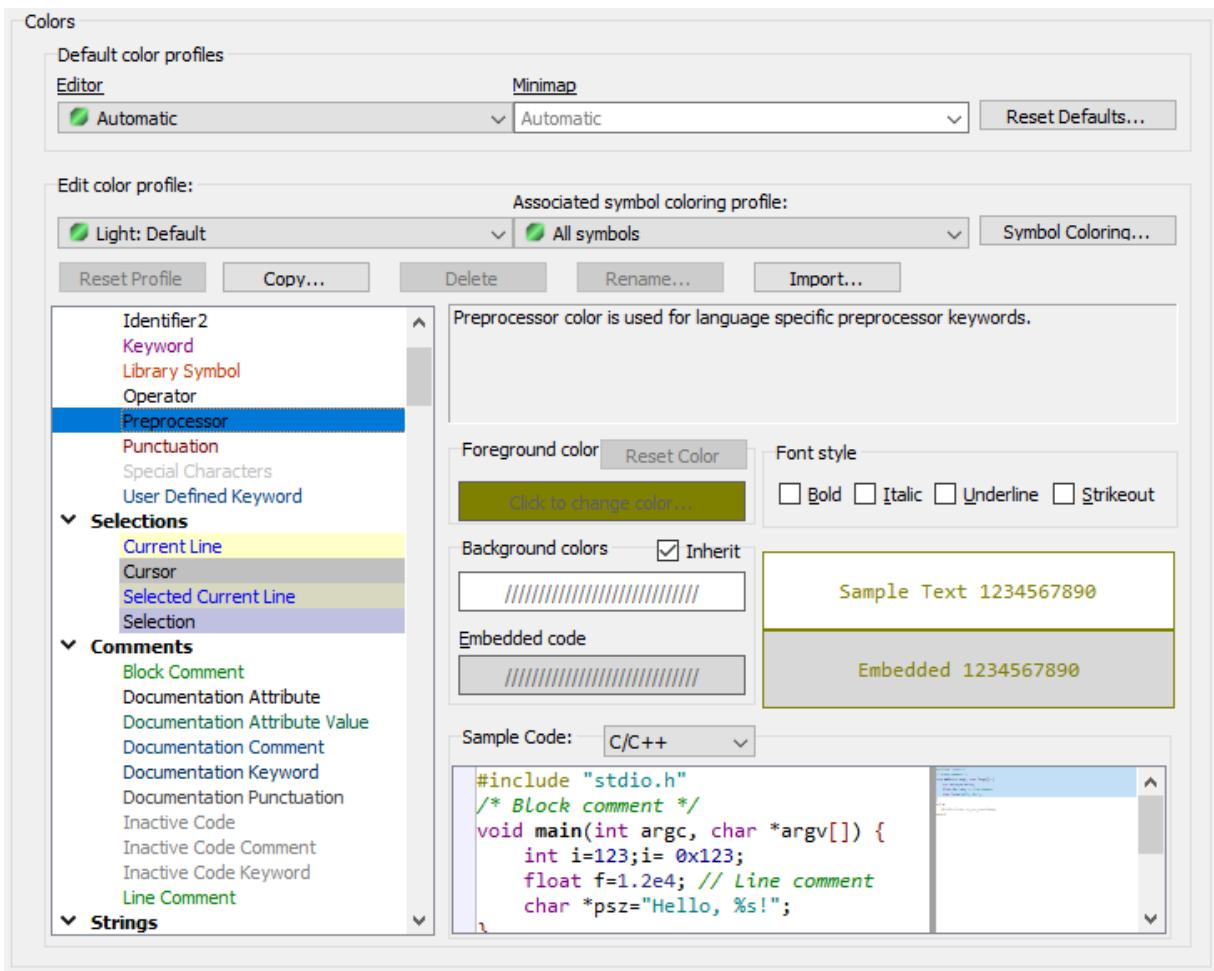
Setting Colors for Screen Elements

Colors can be set either individually or by editing a profile. To change the default colors, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Appearance** → **Colors** (or use the **color** command).
The Colors option screen is displayed.

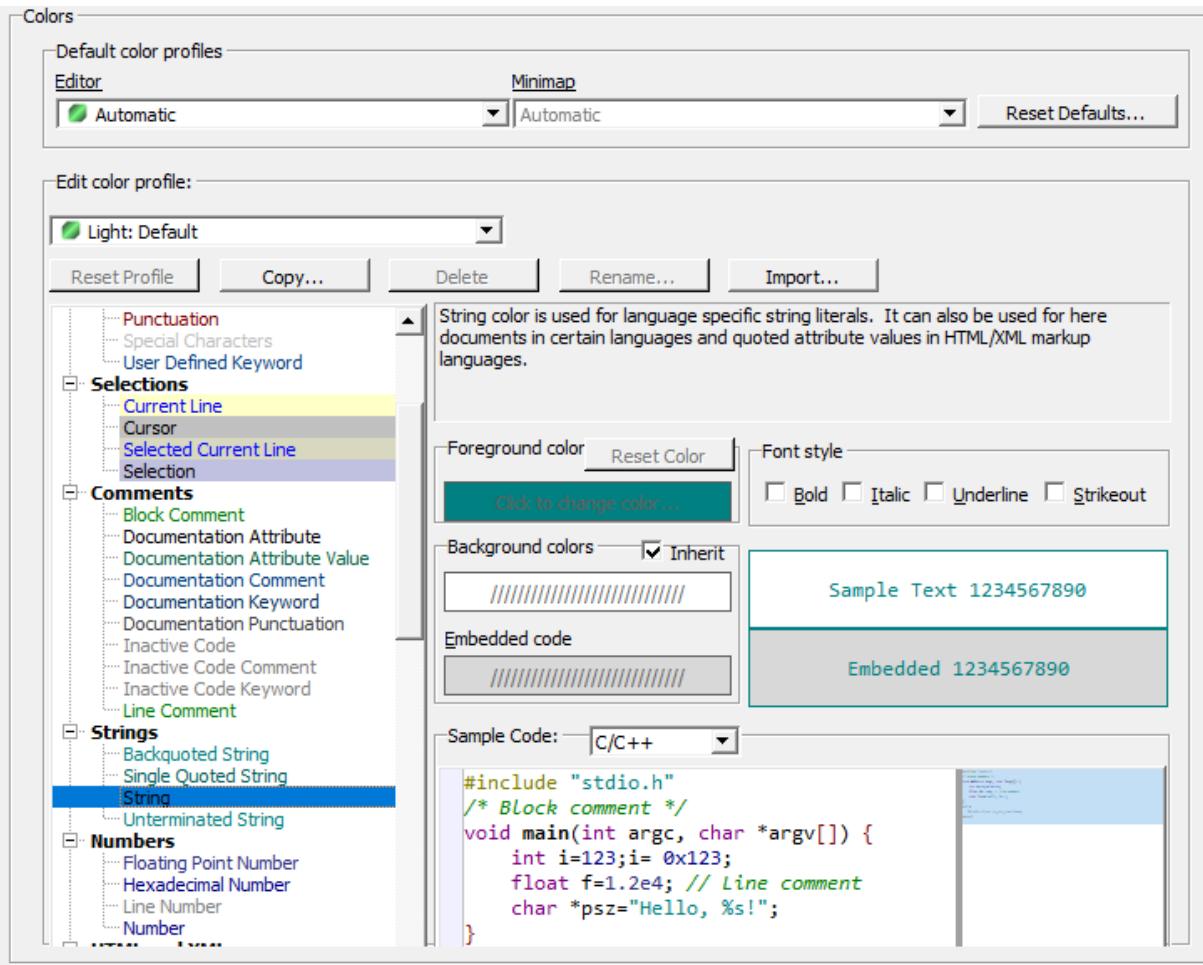
Pro:

Colors



Standard and Community:

Colors



2. Select the element you want to change from the list of customizable items. The items are categorized by their purpose. For descriptions of the individual color elements, see [Color Options](#).

Tip

The element selected in the list matches the symbol at the cursor position when this screen was opened. You can use this to determine what kind of symbol SlickEdit thinks it is. If you're not sure which screen element to pick, close the options screen and put the cursor in the symbol you want to color, and then reopen the options screen.

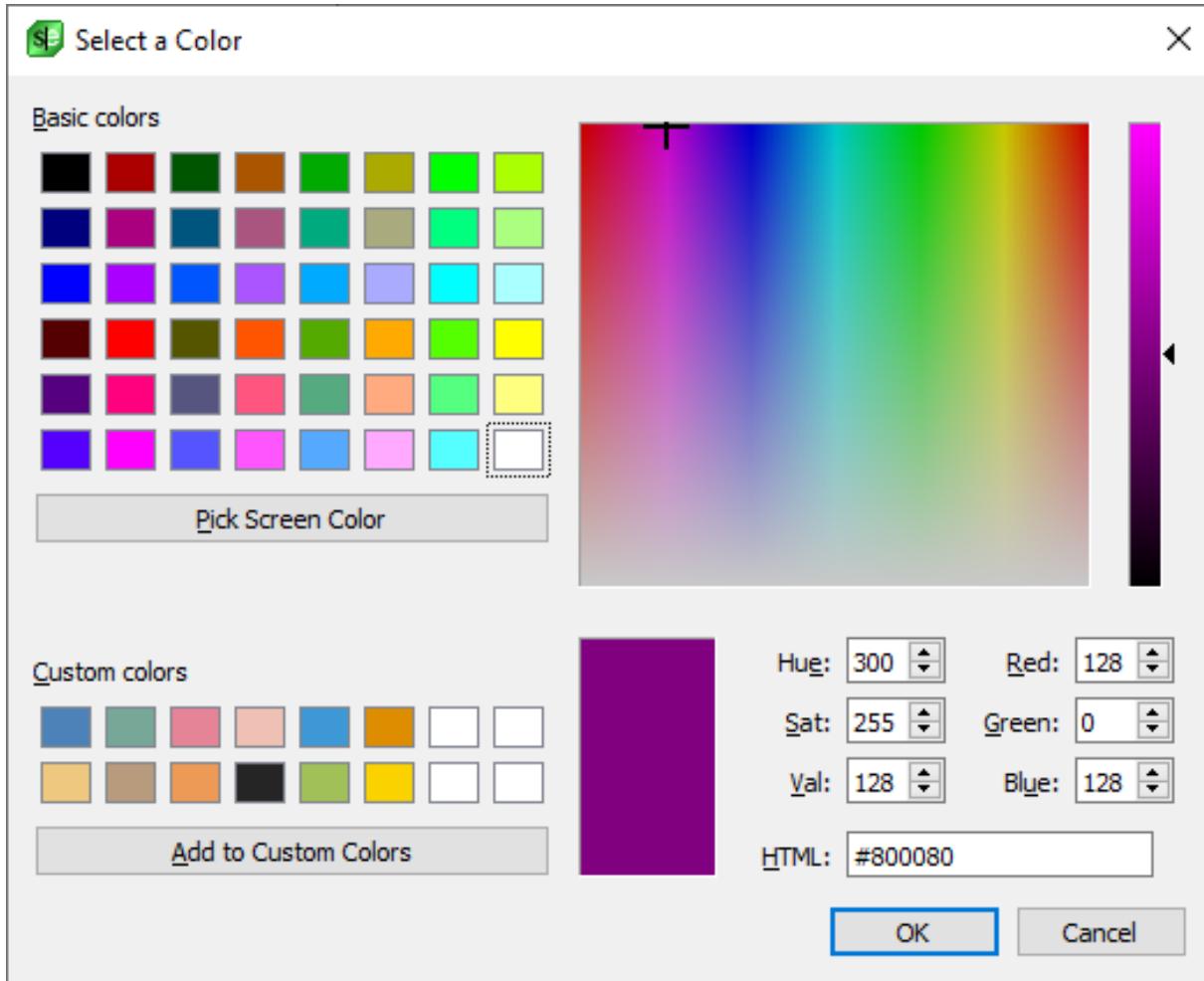
Note

If you have chosen the **Selection** or **Current line** screen element, note that SlickEdit will attempt to render the text 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 foreground color for the underlying text element and selected background color 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 text is still easy to read.

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

Several colors for syntactic screen elements such as comments, keywords, and numbers inherit their background color information from the **Window Text** color. This allows you to change the background color for an entire color profile merely by changing the background color for **Window Text**.



4. If you want, choose a **Font Style** for the text.

For a complete list of all of the options available, see [Color Options](#).

Using Color Profiles

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

To use a color profile, from the **Default color profiles** section, select a color profile for editor windows or for minimap windows and click **Apply** or **OK**.

To define a new color profile, click the **Copy...**, set your colors for the various screen elements, and click **OK**. User-defined color profiles are stored in `user.cfg.xml` file located in your configuration directory. You can change the name of a profile by clicking **Rename...**. You can also import editor color themes assembled for other products, including Visual Studio, Xcode, VSCode, and Eclipse. For more information on importing color profiles, see [Importing Color Profiles](#).

Setting an Embedded Language Color

Colors for editor screen elements also have an embedded background color. This color is used as the background when in embedded code. It is best to select an embedded background color that is only a slight tint from your standard background color. This makes it easier to select common foreground colors that will display with enough contrast in both embedded and normal code.

Embedded Language color is used 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 profiles 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
```

Symbol Coloring (Pro only)

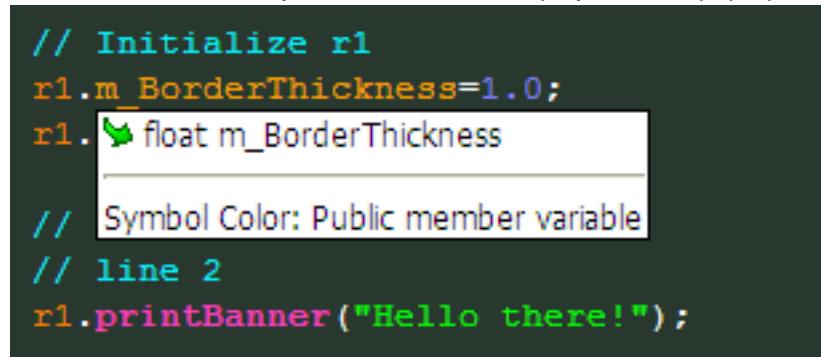
Use the Symbol Coloring options screen (**Tools** → **Options** → **Appearance** → **Symbol Coloring**) to set the color for symbols identified by Context Tagging®. This includes function declarations, function definitions, variables, class names, package names, type names, defines, enumerated types, constants, as well as undefined symbol names.

Note

Symbol Coloring is turned off by default. Symbol Coloring can be enabled on a per-language basis by going to **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **View** and checking **Symbol Coloring**. You can also enable Symbol Coloring for a specific file by selecting **View** → **Symbol Coloring** → **Enable Symbol Coloring**. This will override the language specific setting, but only for the current file.

Symbol Coloring is different from Color and Color Coding. The base color profile, along with the Color Coding profile, are used to identify and color lexical elements found in source code, such as comments, strings, numbers and keywords. Symbol coloring augments the base coloring by overlaying additional color information for identifiers based on the corresponding symbol's name, type, and attributes. This allows you to define symbol coloring rule sets for focusing in on certain symbols or groups of symbols. It also makes it easier to distinguish between different symbol types, such as local variables and constants. See [Color Coding](#) and [Colors](#) for more information about lexical color coding and color configuration.

When you point at a symbol with the mouse cursor, SlickEdit displays a pop-up that includes information from Symbol Coloring about what rule was applied. If that symbol is not colored by Symbol Coloring, no information about the symbol color will be displayed in the pop-up.



A screenshot of the SlickEdit editor interface. A tooltip is displayed over the code line `r1.m_BorderThickness=1.0;`. The tooltip contains the text `float m_BorderThickness` with a small green icon to its left. Below the tooltip, the code continues with `// Symbol Color: Public member variable`, `// line 2`, and `r1.printBanner("Hello there!");`. The background shows other parts of the code in various colors.

Symbol Coloring under some circumstances can cause SlickEdit to pause momentarily while you type. If you experience these pauses, please turn off Symbol Coloring by unchecking **View → Symbol Coloring → Enable Symbol Coloring** which will turn off symbol coloring for the current file. If these pauses happen in all files for that language, you can turn Symbol Coloring off for that language by going to **Tools → Options → Languages → [Language Category] → [Language] → View** and unchecking **Symbol Coloring**.

Note

The standard symbol coloring profiles shipped with SlickEdit® are very thorough and attempt to assign a color to nearly every symbol type. While this is useful, it may be more information than is necessary. The profiles are this way because it is easier to edit or remove rules than it is to add new rules or create a new profile from scratch. You can use the standard symbol coloring profiles as templates that you prune down to create your own, more focused, symbol coloring profiles suiting your specific needs.

Symbol Coloring can be used to highlight unidentified symbols. These are symbols for which the SlickEdit Context Tagging engine can not find a definition or declaration. If you are working without a workspace or your libraries are not fully tagged, you would see a lot of unidentified symbols. Because of this, the capability to highlight unidentified symbols is turned off by default, even if you enable Symbol Coloring. To enable highlighting of unidentified symbols, select **Tools → Options → Languages → [Language Category] → [Language] → View** and put a check in **Highlight unidentified symbols**. The **Symbol coloring** checkbox must be checked for this control to become active.

Symbol Coloring Profiles (Pro only)

A symbol coloring profile is a set of rules defining what color to assign to a symbol with a specific name, type, and attributes. You can think of a profile as a colored lens for looking at your code that highlights the specific symbols you are interested in. Since you can quickly switch symbol coloring profiles, it is very easy to use a special lens for specific tasks, like refactoring out global variables or identifying where your code uses preprocessing.

A symbol coloring rule consists of the following elements:

- **A Rule name.**
- **Symbol types** -- A matching symbol's type must be one of the specified types. The special ***SYMBOL NOT FOUND*** type is used to identify symbols which Context Tagging® can not locate. See [Symbol types](#) for detailed descriptions of each symbol type.
- **Symbol attributes** -- The attributes can be either required, ignored, or disallowed. A matching symbol must have all the required attributes, and none of the disallowed attributes. See [Symbol attributes](#) for detailed descriptions of each symbol attribute.
- **Class name** -- A matching symbol must belong to a class matching the regular expression.
- **Symbol name** -- A matching symbol's name must match the regular expression.
- **Color and font attributes** -- The color definition includes foreground color, background color, and font attributes. This is the color the symbol will be highlighted using. A color definition can base its color on another rule, for example, in order to inherit background color and font attributes for consistency.

Symbol coloring rules are matched in order from the top to bottom of the list of rules in the symbol coloring profile. For a symbol to match a rule, it must be the first rule in the symbol coloring profile that matches all of the requirements above.

Unidentified Symbols

An unidentified symbol is one for which the context tagging engine cannot locate the type information. This could be because the code is incomplete, the source file for that definition has not been tagged or is out of date, or the definition wasn't located before a specified timeout or limit was hit. We use the term "unidentified" instead of "undefined" because the symbol may be defined even though the tagging engine doesn't know it. Unidentified symbols are found using the ***SYMBOL NOT FOUND*** symbol type.

If a symbol is configured in Color Coding to display using the **Library Symbol** or **User Defined Keyword** color, it will not be colored as a symbol coloring error.

Symbol Coloring contains a profile, Unidentified Symbols Only, that can be used to spot these symbols. You can select that profile via the Symbol Coloring options page, **Tools** → **Options** → **Appearance** → **Symbol Coloring**. You can also select that profile from the View menu, **View** → **Symbol Coloring** → **Unidentified Symbols Only**. Lastly, you can toggle the view of unidentified symbols from the view menu using **View** → **Symbol Coloring** → **Highlight Unidentified Symbols**. This will work with any profile, even if it doesn't contain a rule for unidentified symbols.

Color Profile Compatibility (Pro only)

Symbol coloring augments the standard lexical color coding for keywords, comments, strings, numbers,

and other items. Since the symbol colors will be overlayed and typically inherit background color information from the base color profile, it is important for the selected foreground color to be chosen such that the symbol name is still visible and readable against the editor window background.

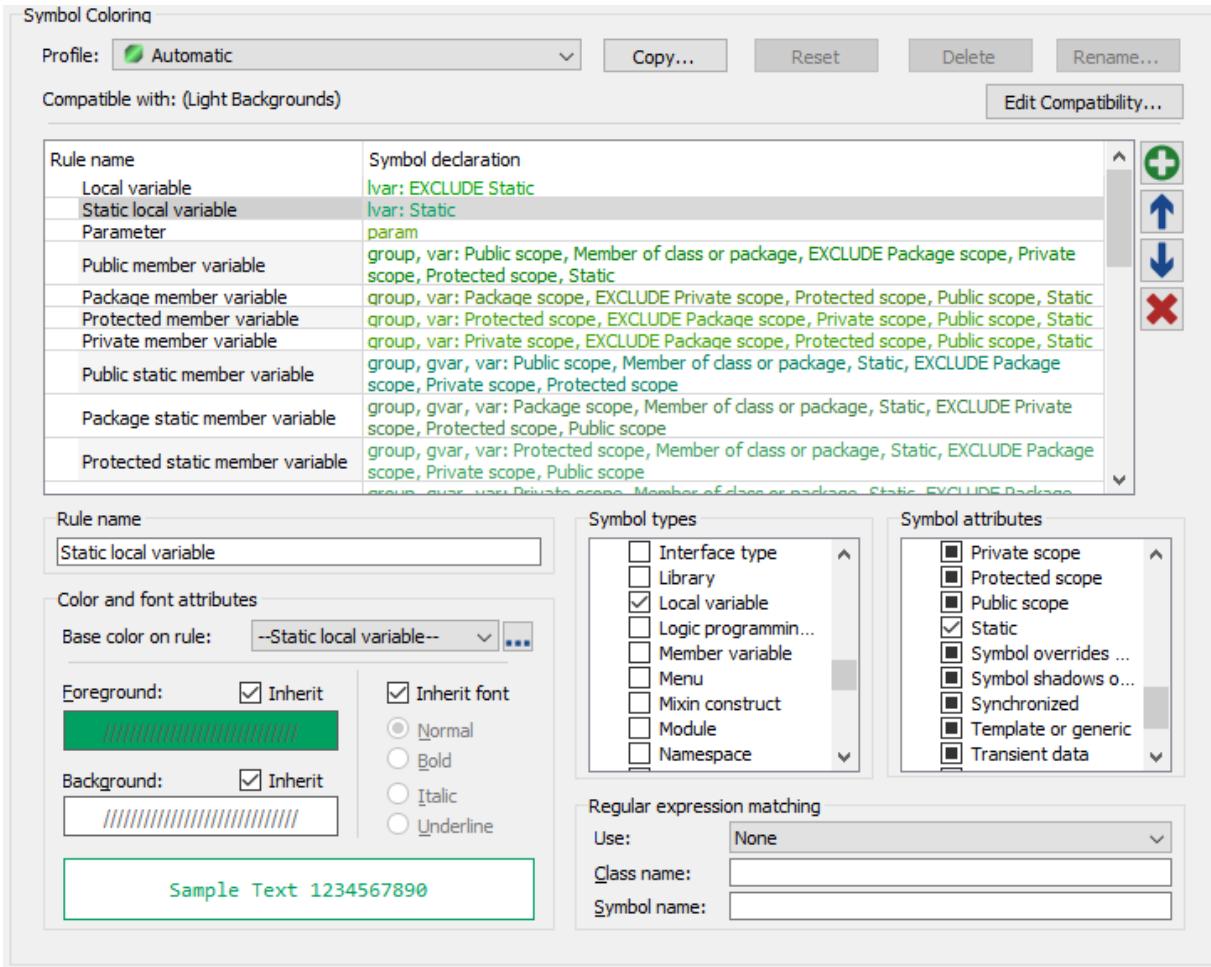
The standard symbol coloring profiles shipped with SlickEdit are marked with the standard base color profiles they are compatible with. Some profiles, such as **Protected and Private** are compatible with all color profiles. Others are fine-tuned to work best against a dark background, a light background, or a specific color profile. User-defined color profiles can specify which base color profiles they work best with.

In addition, each base color profile has a designated, default symbol coloring profile preferred for that profile. This allows you to switch color profiles and automatically get a corresponding symbol coloring profile which is compatible. See [Colors](#) for more information.

Selecting a Symbol Coloring Profile (Pro only)

Symbol coloring rules can be set either individually or by editing a profile. To change the default symbol coloring profile, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Appearance** → **Symbol Coloring**. The Symbol Coloring options screen is displayed.



2. Using the **Profile** combo box, select a profile name. If your current profile is modified, those modifications will be automatically saved to that profile. If you select a profile which is incompatible with your current base color profile, you will be prompted to confirm that you really want to use the selected color profile. See [Color Profile Compatibility](#) for more information.
3. The list of rules will be shown for the selected symbol coloring profile. You can get a quick overview of the profile from the list and see detailed information about each rule by selecting the rule.
4. Press **OK** or **Apply** to commit the changes. The change will be applied to all open files using the default symbol coloring profile.

Editing a Symbol Coloring Profile (Pro only)

To edit the current symbol coloring profile, from the main menu, click **Tools** → **Options** → **Appearance** → **Symbol Coloring**. The Symbol Coloring options screen is displayed.

Select a rule from the list of rules. Note that the rule list not only displays the name of the rule, but also a brief summary of the rule settings. You can add a new rule after the currently selected rule by clicking on the plus icon. Likewise, you can remove the current rule by clicking on the delete icon. Rules can be moved up or down in the rule order by clicking on the up or down arrows.

See [Color Rules](#) for detailed descriptions of each of the standard symbol coloring rules shipped with SlickEdit®.

The current rule can be renamed by clicking in the text box under **Rule name**, modifying the name, then hitting **Enter**.

The display color and font choices for a rule allow you to inherit color and font information from another rule or from certain items from the base color profile. By default a rule will inherit from the **Window Text** color defined in your base color profile.

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. You can also select **Inherit** in order to specify that the rule use the same color as it's parent rule. Set the font attributes by clicking on **Normal**, **Bold**, **Italic**, or **Underline**. Select **Inherit Font** to specify that the rule should use the same font attributes as it's parent rule.

Select a set of symbol types from the list of symbol types supported by the Context Tagging® engine. A symbol must be one of the selected symbol types in order to match the rule. You can select as many symbol types as you want. Select the special ***SYMBOL NOT FOUND*** symbol type to define a rule for what to do with symbols that could not be found using Context Tagging®.

Caution

Not all symbol types apply to every language.

Symbol attributes can have three states. The default state is a grayed state which says we don't care if this attribute is set or not for this rule. If an attribute is checked, it must be set in the matching symbol.

Note

If an attribute is unchecked, it must not be set. Some attributes, such as **Public**, **Protected**, and **Private**, are mutually exclusive by nature. If you configure a rule that checks both **Public** and **Private**, that rule will never be matched. You should instead either define two rules, or one rule with **Protected** and **Package** unchecked.

In addition to the symbol type and attribute specifications, you can further refine a symbol coloring rule by adding a **Class name** or **Symbol name** regular expression, using the regular expression syntax of your choice. The class name regular expression is matched against the name of the scope (class, package, struct) which a symbol is defined in. Do not confuse this with the name of the scope in which the symbol is used. The symbol name regular expression is matched against the name of the symbol. For example, a Wildcards expression of "vs*" would match all symbols starting with the characters "vs". Case sensitivity for the regular expression matching is regulated by the language's case-sensitivity. See [Color Coding](#) for more information.

Creating a New Symbol Coloring Profile (Pro only)

New symbol coloring profiles can be created by selecting an existing profile and adding or subtracting rules, then saving the profile under a new name. To create a new symbol coloring profile from scratch, select the **Unidentified Symbols Only** profile, click the **Copy...** button, give the new profile a name, remove the one existing rule, and then add your own custom rules.

Selecting a Symbol Coloring Profile for the Current File (Pro only)

From the main menu, click **View** → **Symbol Coloring**. This will bring up the Symbol Coloring view menu.

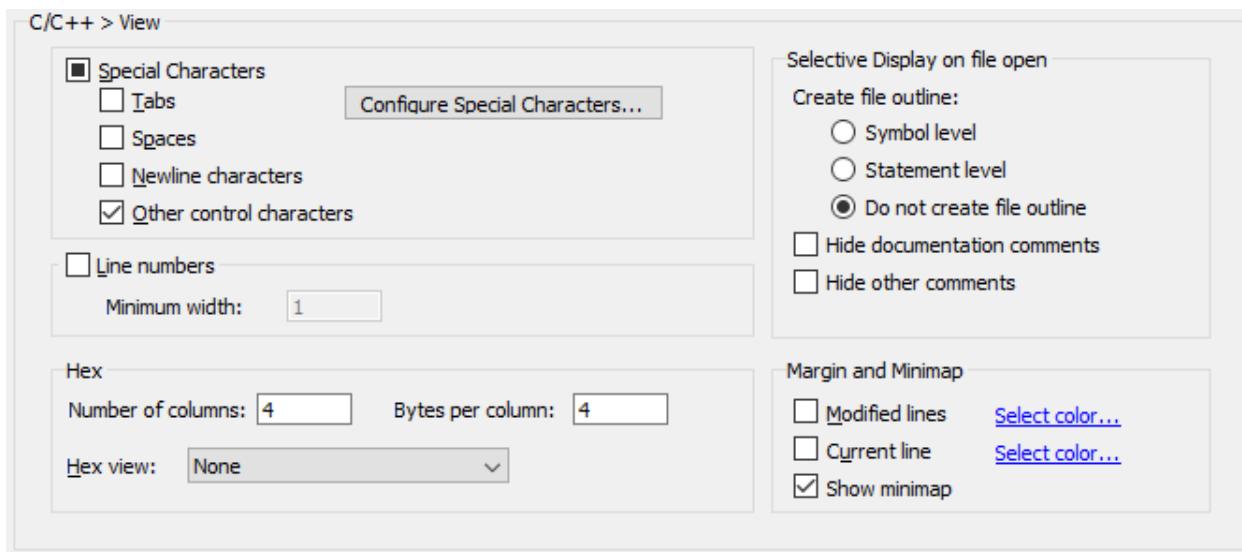
The menu will list only symbol coloring profiles known to be compatible with the current base color profile. Select a profile to switch to that profile for the current file. Select the **(None)** profile to disable symbol coloring for the current file.

Selecting a specific profile for the current file will not change the symbol coloring profile for any other files, nor will it change the default symbol coloring profile. The selected symbol coloring profile will be saved in your file history so that the next time you open that file, it will return to using the same symbol coloring profile you selected, as long as your base color profile does not change.

Language-Specific Symbol Coloring Settings (Pro only)

Certain Symbol Coloring features can be disabled on a per-language basis. To edit language-specific symbol coloring options, from the main menu, click **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **View**. The Language View options are displayed.

Symbol Coloring (Pro only)



From this dialog, you can configure the following on a per-language basis.

- Turn off Symbol Coloring entirely for the language. This would be a good idea if symbol analysis was particularly slow or ineffective for a language, such that symbol coloring was only slowing you down.

Note

Symbol coloring is automatically disabled for HTML and other XML variants. It is also automatically disabled in all modes which do not have any Context Tagging® implemented. Finally, Symbol Coloring is disabled in all embedded language contexts. This means that Symbol Coloring is disabled for all PHP code, since PHP is always embedded in an HTML or XML processing instruction (<?php).

- By default, Symbol Coloring will bold the name part of symbol declarations and definitions. This is particularly useful for languages which allow implicit local variable declarations. It is also helpful when the declaration syntax is not always visually distinct from the rest of the code. This bolding behavior can be turned off by unchecking **Use bold for symbol names in definitions and declarations**.
- Symbol Coloring is able to select the **Symbol not found** rule for symbols which are not found by Context Tagging®. This can serve effectively as a live error checker with respect to spelling and capitalization of symbols.

However, in certain languages, especially scripting languages that allow variables to be declared implicitly, Context Tagging® can be rather ineffective, simply because the code can not be analyzed statically. In this case, you might see an unusually large number of symbols highlighted as unidentified symbols. This can also happen if you do not have Context Tagging® configured correctly for the code and libraries you are working with.

For this reason, this feature is *disabled* by default. You can enable highlighting of unidentified symbols by checking **Highlight unidentified symbols**.

- By default, Symbol Coloring uses fairly strict language specific symbol lookups in order to identify

symbols. In some languages, it is necessary to relax the rules in order to find symbol definitions. This can, for example, be useful in heavily templated or preprocessed C++ code which is too complex for Context Tagging®. Selecting **Use relaxed symbol lookups** instead of the default of **Use strict symbol lookups** will tell Symbol Coloring to revert to a more flexible symbol lookup, ignoring scope and visibility rules, if the strict symbol lookup does not yield results. In other larger, more complex code bases, the strict symbol lookup algorithm may require too much time to be practical to use. Sometimes a more simplistic approach of looking up the symbol based on the symbol's name alone, ignoring context, usage, and scope is adequate. Select **Use simplistic symbol lookups** to enable the fast, simple symbol lookup algorithm. Note that using the simplistic symbol lookup algorithm can drastically decrease the accuracy of Symbol Coloring, especially with respect to detecting misspelled symbols.

Symbol Coloring Performance Settings (Pro only)

Symbol Coloring requires the editor to do symbol lookup and analysis for every symbol visible on the current page of the current file. This can be expensive, especially for extremely large files or large, complex code bases. Because of this, it attempts to only color the symbols which are currently visible, not the entire file. Furthermore, instead of immediately painting like basic syntax driven color coding does, symbol coloring works on a delay timer. This way you should never have to wait for symbol coloring to finish working except under extreme circumstances. Besides coloring the current page, symbol coloring will also look ahead slightly to surrounding lines. This makes it possible, in the typical case, to page up one page and not have to wait for symbol coloring to draw because the information was already prefetched.

Note

The most effective way to increase Symbol Coloring performance is to tune your workspace and tag files configuration so that you tag everything you need and do not tag a lot of extra code.

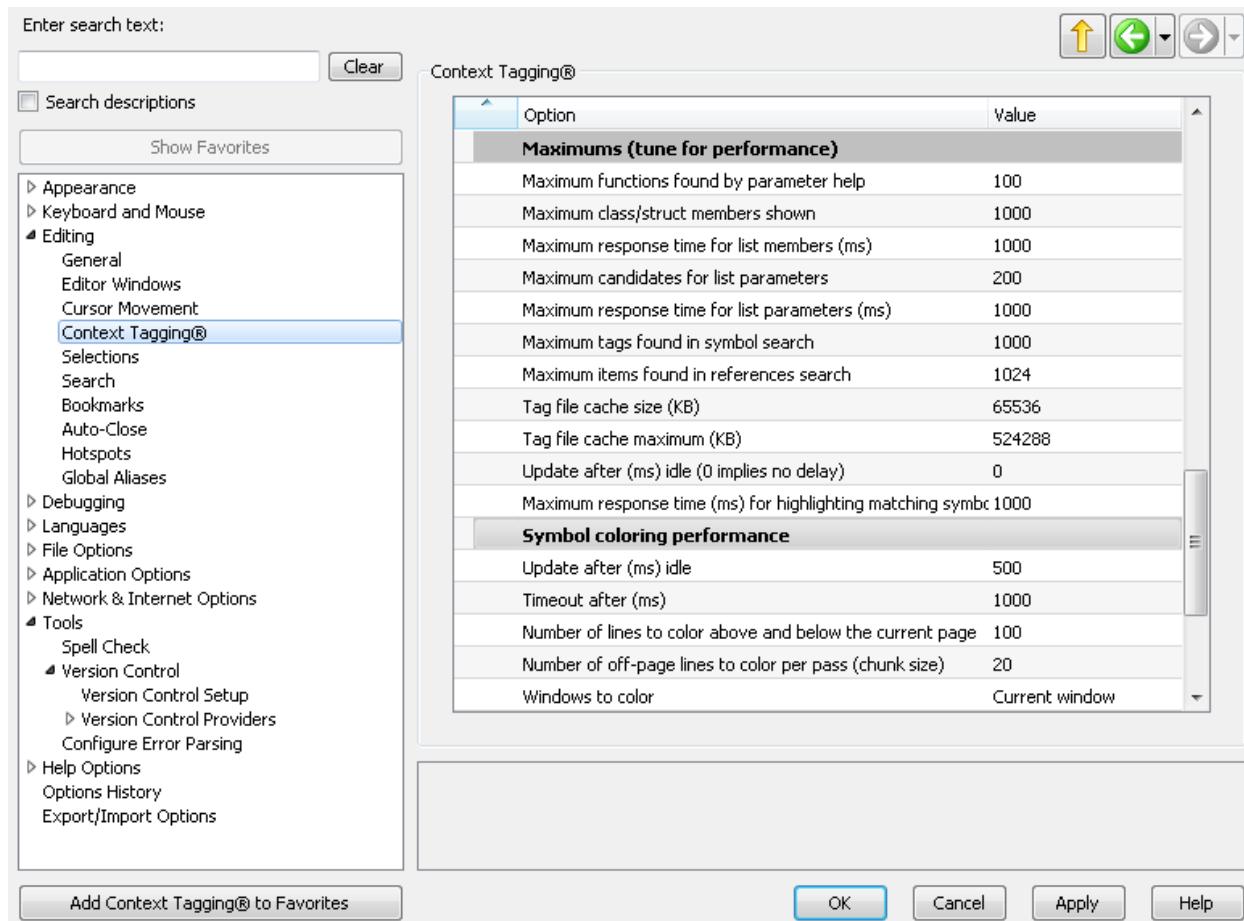
The second best way to increase Symbol Coloring performance is to make use of the **Use simplistic symbol lookup** option. See [Language-Specific Symbol Coloring Settings](#) for more information.

Caution

For typical users, the default performance settings will be good enough. Tinkering with these settings without regard to the implications could result in very poor performance and/or annoying drawing behavior.

Symbol Coloring performance can be fine-tuned through the user interface. To edit symbol coloring performance options, from the main menu, click **Tools** → **Options** → **Editing** → **Context Tagging**. The Context Tagging® options are displayed.

Symbol Coloring (Pro only)



From this dialog, you can configure the following settings.

- **Update symbol coloring after (ms) idle** -- This is the amount of idle delay symbol coloring should wait before updating the Symbol Coloring for the current page. Increasing this value can prevent interruptions to your normal typing due to symbol coloring updating, however, it will cause symbol coloring updates to lag further behind your editing. Decreasing this value too much can have the effect of making symbol coloring updates behave nearly synchronously and can create very bad editor response times. A good setting is four times your average keypress gap, which you can estimate by looking at how many words per minute you type when coding.
- **Timeout after (ms)** -- This is the maximum amount of time that symbol coloring should spend trying to do symbol analysis before giving up and trying to finish in the next pass. Increasing this time can cause increased intrusiveness. Decreasing this time too much can cause symbol coloring to not have enough time to paint the entire page. As a result, you would see lines get colored as symbol coloring makes subsequent passes to finish coloring the page.
- **Number of lines to color above and below the current page** -- This is the amount of prefetch symbol coloring should do for pages surrounding the current visible page of code. Setting this very high can have the effect of forcing symbol coloring to color the entire file in one shot. Setting it to 0 will force symbol coloring to only color the visible page and not do any prefetch at all.
- **Number of off-page lines to color per pass (chunk size)** -- When prefetching symbol coloring for off-

page lines, this is the number of lines to prefetch per pass. Setting this to a large number can make symbol coloring performance more intrusive. Setting this to a small number, such as 1, will force symbol coloring to make many passes before it can color all the off-page lines it is supposed to. As a result, a **Page Up** might reveal a page which is only partially colored.

- **Windows to color** -- Maybe be one of the following:
 - **Current window** -- (Default) Only symbol color the current window
 - **Visible windows** -- Symbol color visible windows
 - **All windows** -- Symbol color all windows including windows that are not currently visible (like an inactive Document tab)

Color Coding

For information on how to set up colors for various element types in the editor, see [Colors](#). This section describes how to configure the Color Coding engine, which pairs syntactic elements with various colors.

Creating Color Coding for a New Language

To create color coding support for your language, complete the following steps:

1. Display the Color Coding dialog for your language. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Click **New...** and enter the new profile name. Usually this is a language name such as **JavaScript** or **Groovy**. Click **OK**.
3. Select the [Color Coding General Tab](#).
4. On the **General** tab, set the **ID start characters**. These are valid characters which can be the start of an identifier. Use a "-" to specify a range of characters (ex a-zA-Z).
5. Set 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. Use a "-" to specify a range of characters (ex a-zA-Z0-9).
6. Now that you've defined the basics for your new color coding profile, you can add keywords, comment definitions, string definitions, and other types. See How-To's below. See [Language-Specific Color Coding Options](#) for more information on the Color Coding dialog.
7. Click **OK** on the Color Coding options screen to save your settings.

How to add new color coding words (keywords, library symbols, operators, punctuation etc.)

To add color coded words to an existing color coding profile, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Click the [Color Coding Tokens Tab](#).
3. Click **Add Words....**
4. Enter the new words separated with a space character. If the word contains a space, double quote the word.
5. Choose the color element **Type** (Keyword, Library Symbol, Operators, Punctuation, etc.).
6. Click **Add**.
7. Click **OK** on the Color Coding options screen to save your settings.

How to add a line comment

To add a line comment to an existing color coding profile, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Click the [Color Coding Tokens Tab](#).
3. Click **Add Other....**
4. Set the **Type** to **Comment** or **Doc Comment**. When **Type** is Doc Comment, supported documentation keywords (JavaDoc, XML Doc, and/or Doxygen) get nicer color coding. See [Color Coding Language Tab](#).
5. Set the **Start delimiter** to text you want matched. For example, a common line comment start delimiter is `//`. See [Tips on using regular expressions matching in color coding](#) for information on regular expression matching.
6. Set **Color to end of line** to **Comment** or **Doc Comment**.
7. Click **OK** on the Color Coding options screen to save your settings.

How to add a multi-line comment

To add a multi-line comment to an existing color coding profile, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Click the [Color Coding Tokens Tab](#).
3. Click **Add Other....**
4. Set the **Type** to **Comment** or **Doc Comment**. When **Type** is Doc Comment, supported documentation keywords (JavaDoc, XML Doc, and/or Doxygen) get nicer color coding. See [Color Coding Language](#)

[Tab](#)

5. Set the **Start delimiter** and **End delimiter** to the start and end text you want matched. For example, a common multi-line comment is /* and */. A more complicated example for a Lua multi-line comment requires the SlickEdit regular expression --[\{=@\}\] and plain text search]#0]. The #0 is replaced with the zero or more equal signs found in the start delimiter. To use a regular expression, change **Plain text search** to **SlickEdit regex** or **Perl regex**. See [Tips on using regular expressions matching in color coding](#) for information on regular expression matching.
6. Set **When end delimiter is not on same line** to **Color to end across multiple lines**.
7. Click **OK** on the Color Coding options screen to save your settings.

How to add a string

To add a string to an existing color coding profile, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Click the [Color Coding Tokens Tab](#).
3. Click **Add Other....**.
4. Set the **Type** to **String**.
5. Set the **Start delimiter** and **End delimiter** to the start and end text you want matched. For example, a common double quoted string uses " for both the start and end delimiters. A more complicated example for a Lua multi-line string requires the SlickEdit regular expression \[\{=@\}\] and plain text search]#0] for end. The #0 is replaced with the zero or more equal signs found in the start delimiter. To use a regular expression, change **Plain text search** to **SlickEdit regex** or **Perl regex**. When a string has one or more prefix words, it's more efficient to use a regular expression than to add multiple strings (ex (**s|f|raw**)" is used for Scala) .
6. For a multi-line string, set **When end delimiter is not on same line** to **Color to end across multiple lines**. Otherwise, set **When end delimiter is not on same line** to **Color to end of line**.
7. To add interpolation to this string definition, see [How to add interpolation to a string](#)
8. Click **OK** on the Color Coding options screen to save your settings.

How to define color coding for numbers

To define color coding for numbers for an existing color coding profile, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Click the [Color Coding Numbers Tab](#).
3. Select the variables number options on this tab. It covers the needs of most languages.

4. If there is a number construct not covered by the Color Coding Numbers tab, continue with the steps 5-7. Otherwise, skip to step 8.
5. Click **Add Other...**.
6. Set the **Type** to **Number**, **Float**, or **Hexadecimal Integer**.
7. Set the **Start delimiter** to text you want matched. Since you will likely need to be a regular expression, change **Plain text search** to **SlickEdit regex** or **Perl regex**. For example, a SlickEdit regular expression for a Visual Basic hexadecimal integer is **&[hH][0-9a-fA-F]#** (Perl regex is **&[hH][0-9a-fA-F]+**).
8. Click **OK** on the Color Coding options screen to save your settings.

How to add a interpolation to a string

To add a interpolation to a string of an existing color coding profile, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Click the [Color Coding Tokens Tab](#).
3. Click on the **String** item in the tree list which you want to add interpolation to.
4. Click **Add Sub Item....**.
5. Set the **Type** to **String**.
6. Set the **Start delimiter** and **End delimiter** to the start and end text you want matched. To avoid automatic word boundary checking for the start delimiter, change Plain text search to SlickEdit regex or Perl regex. For Scala interpolated strings, the start delimiter is **\\$`** and uses a SlickEdit regex. The end delimiter is **`** and uses a Plain text search (definitely no identifier characters in the end delimiter so plain text search is ok). See [Tips on using regular expressions matching in color coding](#) for information on regular expression matching.
7. If the interpolation expression can span multiple lines, set **When end delimiter is not on same line** to **Color to end across multiple lines**. Otherwise, set **When end delimiter is not on same line** to **Color to end of line**. This setting will have no effect if the parent string can't span multiple lines.
8. Click the [Color Coding More Tab](#).
9. Set the **Start color** and **End color** to **Punctuation**. You can choose another color if you like it better for these.
 - 1 Click the [Color Coding Embedded Tab](#).
 - 0.
 - 1 Set **Embedded profile** to the name of the color coding profile you are modifying. The profile name is displayed to the right of the **Profile** label near the top of the dialog.
 - 1 Check **Embedded end delimiter is token**. This option is only needed if the interpolation expression can contain tokens like strings which need to be ignored when searching for an instance of the **End**

delimiter.

- 1 Set **Embedded color style** to **Don't color as embedded if possible**. This isn't a requirement but
3. maintaining the same background color looks better.

- 1 Click **OK** on the Color Coding options screen to save your settings.
- 4.

Tips on using regular expressions matching in color coding

Using regular expressions for the Start delimiter, End delimiter, and/or embedded profile provides a lot of interesting possibilities for recognized and color certain syntax. Some things are obvious but some are definitely not. This section documents some of the less obvious things you will want to know about using regular expression to match start or end delimiters.

- Currently, using regular expressions to match a start or end delimiter does not support matching across line boundaries. This may be added later. The start and end delimiters do not need to be on the same line as long as **When end delimiter is not on same line** is set to **Color to end across multiple lines**.
- Sometimes you need to use a regular expression match for the start or end delimiter to avoid automatic word break logic performed by a plain text search. For example, if the **Start delimiter** is \${ and the identifier start or follow characters contain a dollar sign, you may need to use a regular expression so that \${ is still considered a match in a text string like abc\${.
- Variable length look behind assertions can be very valuable. This is a feature that is NOT supported by other regular expression engines. Probably because it's very complicated to implement. Take the member expression "foo.member". If you want to color code all member references in a particular color, a variable length look behind assertion will work nicely. For example, the SlickEdit regular expression to match this is roughly (#<=[a-zA-Z][a-zA-Z0-9]@.)([a-zA-Z][a-zA-Z0-9]@. The equivalent Perl regular expression to match this is (?<=[a-zA-Z][a-zA-Z0-9]*\.)[a-zA-Z][a-zA-Z0-9]*. It's important to note that the text found in the look behind assertion will not be colored because it's not considered part of the match.
- When the **Start delimiter** is a regular expression, tagged expressions and escapes are processed in the **End delimiter** even if the end delimiter is not a regular expression. When the **Start delimiter** and **End delimiter** are both regular expressions, things get complicated. First, tagged expressions and escapes are processed. Then the result is compiled as a regular expression. This means you may need to escape a literal character twice (ex instead of just "\\\" you need "\\\\\""). Note that when the tagged expressions are replaced, special characters are escaped so that the tagged expression replacements are considered literal text. See [Section Replacing with Regular Expressions](#) for information on tagged expressions and special characters in the replace string.

Lua multi-line string example:

```
If see [[, want end to match ]]
If see [=[, want end to match ]=]
If see [==[, want end to match ]==]
etc.
```

Start delimiter (SlickEdit Regex): \[\{@\}\[\[

```
End delimiter (Plain text search): ]#0]
```

```
Start delimiter (Perl Regex): \[(=*)\[
End delimiter (Plain text search): ]$1]
```

The above example is pretty simple but now lets change the brackets to be a backslash.

Make believe example:

```
If see \\, want end to match \\
If see \\=\\, want end to match \\=
If see \\==\\, want end to match \\==\
etc.
```

```
Start delimiter (SlickEdit Regex): \\{@}\\
End delimiter (Plain text search): \\#0\\
```

```
Start delimiter (Perl Regex): \\(*\\)\\
End delimiter (Plain text search): \\$1\\
```

Notice that the End delimiter needs two backslashes to represent one. This is exactly how the replace string is handled for a regular expression search and replace.

Now lets tweak the above example more so the End delimiter needs to be a regular expression.

Harder make believe example:

```
If see \\, want end to match \\<optional-whitespace>\\
If see \\=\\, want end to match \\=<optional-whitespace>\\
If see \\==\\, want end to match \\==<optional-whitespace>\\
etc.
```

```
Start delimiter (SlickEdit Regex): \\{@}\\
End delimiter (Plain text search): \\#0[ \\t]@\\
```

```
Start delimiter (Perl Regex): \\(*\\)\\
End delimiter (Plain text search): \\$1[ \\t]*\\
```

Noticed that escaping is only sometimes done twice. A literal backslash requires four backslashes but only two backslashes are needed to specify the tab character (\t). When the start delimiter is found, \\#0[\\t]@\\ is translated to \\<zero-or-more-equals>[\\t]@\\. Then the regular expression \\<zero-or-more-equals>[\\t]@\\ is compiled.

Note that when the tagged expression #0 or \$1 is replaced, it is replaced as literal characters so you are guaranteed the result can be compiled as a regular expression.

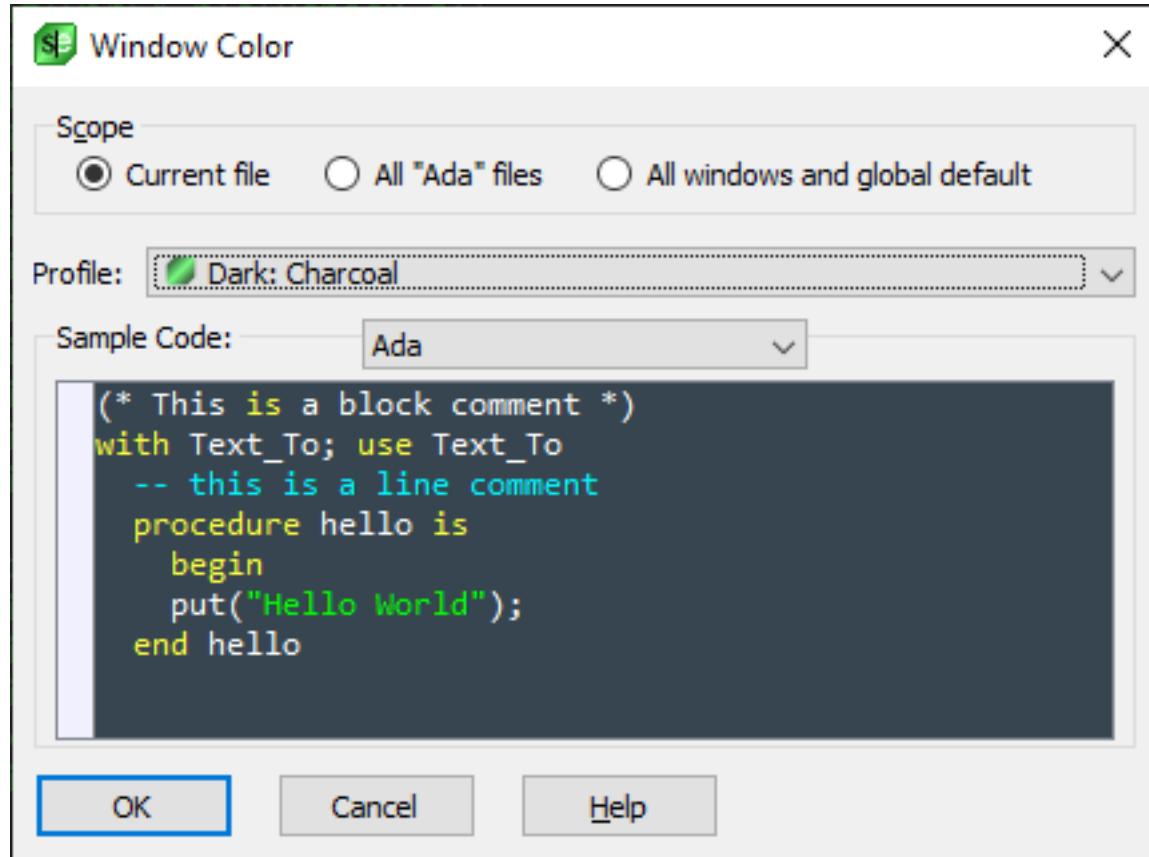
- It would be really handy to be able to specify different colors for tagged expressions. While this feature is not yet supported, it most likely will be added in the future.

Advanced Color Coding Configuration

User defined Color Coding profiles are stored in `user.cfg.xml` as well as other settings. For information about the format of these profiles, see [Color Coding Profile Format](#).

Setting Editor Window Colors

You can set the default color profile for editor windows by selecting a profile in the **Colors** dialog, as described above. In addition, you can set a default color profile on a per-language basis. Finally, you can set a color profile for an individual file by selecting **Window → Color**, or by using the **window-color** command. The Window Color dialog is displayed. As well, you can set a color profile for the minimap window by right-clicking and selecting **Color....**



From the Window Color dialog, choose the **Scope** that you wish to affect. Select the **Current file** option if you only want to change the current window's color. Select **All [language] files** to set the color profile for all editor windows displaying source files in the current language. Select the **All windows and Default** option to set the color profile for all editor windows that are open as well as newly-created windows.

Tip

Use the **Window Color** dialog to select a specific color profile for a single editor window (the current file).

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 control which elements of your SlickEdit environment that are automatically restored, from the main menu, click **Tools** → **Options** → **Application Options** → **Auto Restore**. See [Auto Restore Options](#) for more information.

Setting File Associations

Files of certain types can be associated with SlickEdit®, so that when those files are opened from Windows Explorer, they run in the SlickEdit application. To set up these associations, use the Associate File Types Options. To access these, from the main menu, click **Tools** → **Options**, expand **File Options**, then select **Associate File Types** (or use the **assocft** command).

Associate File Types

File Types: View: List extensions by language

Language Mode > File Extension

- > Applescript
- > Awk
- > BibTeX
- > Binary
- > Bourne Shell
- > Bulletin Board Code
- > C Shell
- > C#
- > C/C++
 - c
 - c++
 - cc
 - cp
 - cpp
 - cppm
 - cxx
 - h
 - h++
 - hh
 - hp
 - hpp
 - hxx
 - i
 - inl
 - ixx
 - qth
 - sig
 - tcc
 - xpm
- > CFML
- > CFScript

Current associations: ada adb ads ansic as awk bash bourneshell c c++ cc cfc cfml dj djc djs djx cp cpp ppm cs csh csx cxx h h++ hh hpp hxx i inl ixx qth sig tcc xpm

Check the file types you wish to associate in the **File Types** tree. You can organize the list by using the **View** combo box. To see extensions organized by the language they are associated with, select **List extensions by language**. To see the file types in one alphabetic list, select **List extensions only**. To add or remove file extensions, click the **Manage File Extensions** link.

Note

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](#)
- [Working with Files](#)

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, click **Tools** → **Options** → **Application Options** → **Auto Restore**. 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(Pro only) and execute(Pro only) as a unit. For each project you can specify the set of files it contains, a working directory, a set of commands(Pro only) to build(Pro only) and execute(Pro only) the project, compiler options(Pro only), and dependencies(Pro only) 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`.

Pro:

When you create a project, you select the project type based on the language and tool chain you are using. A tool chain is the combination of compiler and debugger used. Selecting the right project type is essential to configure SlickEdit to build, run, and debug 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 [Managing Projects](#).

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 Features](#).

Viewing your Files in the Projects tool window

You can use the Files tool window, Open tool window, or the Projects tool window for quickly opening files from your project or workspace. Both the Files tool window and Open tool window display a flat list of files and provide excellent search capabilities for easily searching opening project files. The Projects tool window allows you to create various folder views of your files without changing their location on disk.

Choose the viewing style that meets your needs:

- [Wildcard Directory Folder](#)
- [Folders for File Types \(.cpp, .h, .png, .rc, etc.\)](#)
- [Folders for each Package or Namespace](#)
- [Folders for Each Directory](#)
- [Customized Folders for Directories](#)

Wildcard Directory Folder

The Good: Very easy to setup. Allows you to use your existing source code directory organization for navigating and opening files in your project. When you add files to your source tree, the files will be automatically added to SlickEdit.

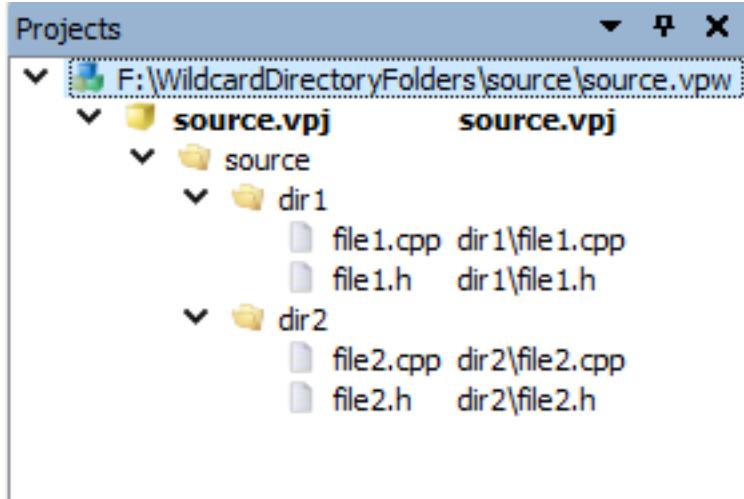
The Bad: Since this uses wildcards which need to be scanned at various times, you may see performance degradation depending on the size of the directory tree. When performance is a problem, it's best not to use wildcards. If you are using SlickEdit to build your source code, your wildcard may include files you don't want to build.

How to add a wildcard directory folder:

- Make sure your Project is in Custom View. Right click on the project file in the Projects tool window to display the context menu, choose **Folder View>Custom View**.
- Right click on the project file in the Projects tool window and choose **Add>Tree...**
- Set the base path (Path), file types to list (Include filespecs), and any exclusions you want (Exclude filespecs).
- Check the **Recursive**, **Add as wildcard**, **Show subfolders**, and **Create parent for directory** check boxes.

Note

(Close the any modal dialog like the Project Properties dialog first) You can drag/drop a directory from your operating system file explorer onto SlickEdit to perform an Add Tree to the active project. If no workspace/project is open, you will be prompted to create one.



Folders for File Types (.cpp, .h, .png, .rc, etc.)

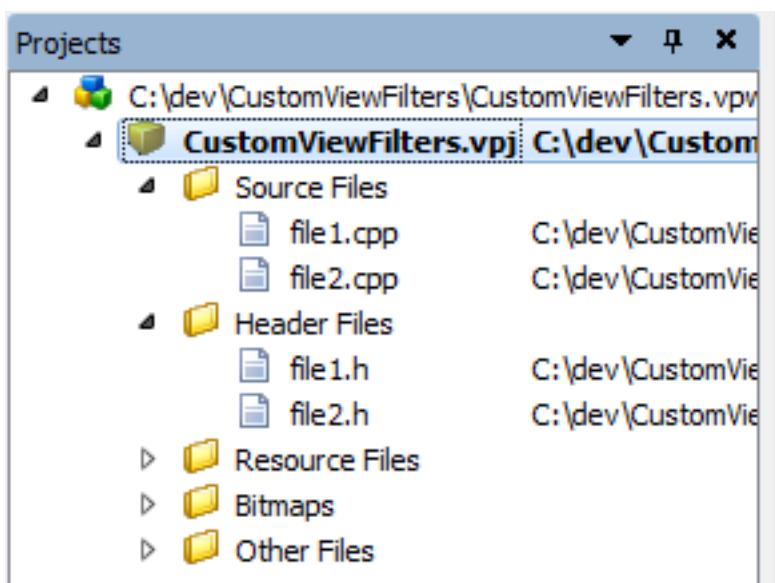
The Good: Great for small projects. Easy to setup. (Pro only) You can create custom project templates (**Project>New>Custom...**) that you can use later.

The Bad: The files of the same type are displayed in a flat list. Once you have many files, you have to scroll the list.

How to add a folder for a file type:

- Make sure your Project is in Custom View. Right click on the project file in the Projects tool window to display the context menu, choose **Folder View>Custom View**.
- Right click on the project file in the Projects tool window and choose **Add>Folder...**
- Type the name for the folder and the wildcards for the folders.

Overview of Workspaces and Projects



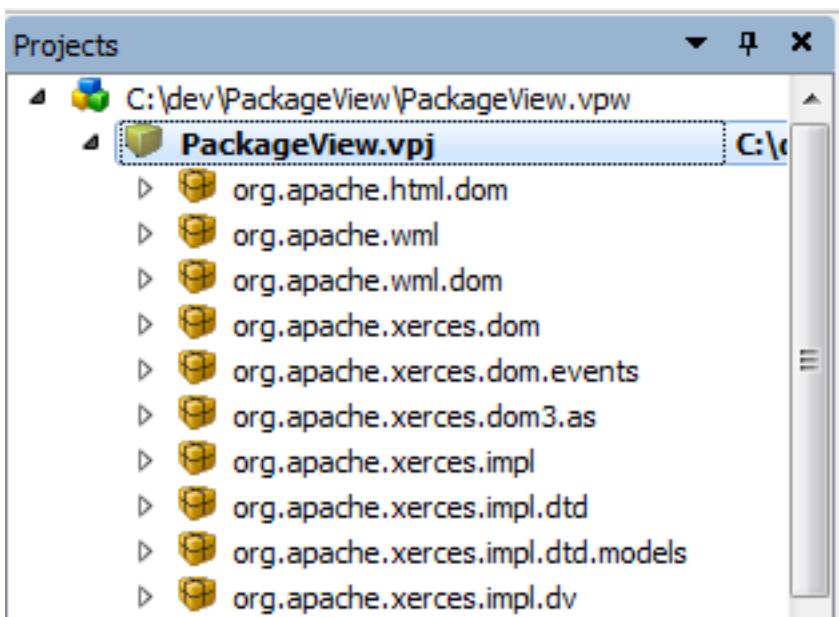
Folders for each Package or Namespace

The Good: Great for any size OO project. Very easy to setup.

The Bad: Not good for non-OO projects. Can't add any custom folders. All folder names are automatically chosen.

How to turn on package view:

- Right click on the project file in the Projects tool window to display the context menu, choose **Folder View>Package View**.



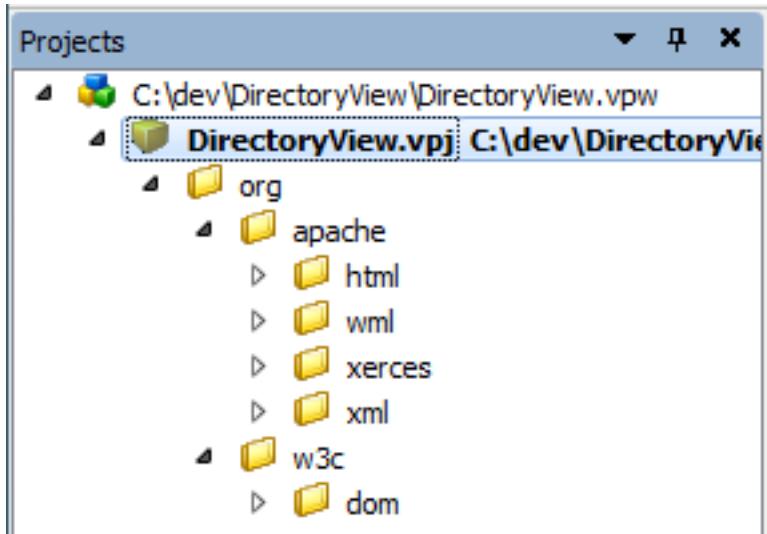
Folders for each Directory

The Good: Great for any size project which has a very useful directory structure. Very easy to setup.

The Bad: You may get odd folder names if your source files are not beneath your project file. Can't add any custom folders. All folder names are automatically chosen.

How to turn on directory view:

- Right click on the project file in the Projects tool window to display the context menu, choose **Folder View>Directory View**.



Customized Folders for Directories

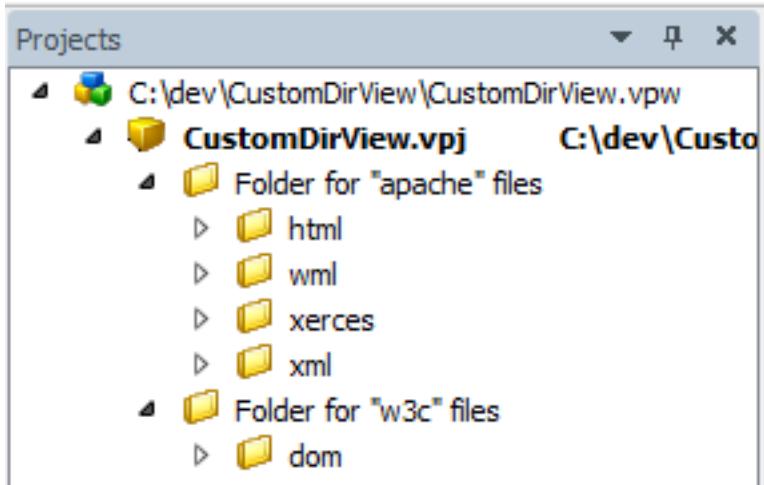
The Good: Great for any size project. Most flexible.

The Bad: Not obvious how to set this up. You must manually create every root folder. As long as you don't have that many root folders, it's not a big deal.

How to add a folder and add files to it:

- Make sure your Project is in Custom View. Right click on the project file in the Projects tool window to display the context menu, choose **Folder View>Custom View**.
- Right click on the project file in the Projects tool window and choose **Add>Folder...**
- Type the name for the folder. Make sure you leave the wildcards text box blank.
- Right click on the folder you just created in the Projects tool window and choose **Add>Tree...**
- Set the base path (Path), file types to list (Include filespecs), and any exclusions you want (Exclude filespecs).
- If you want a folder created for each directory as shown in the picture below, turn on the "Show subfolders" check box. You may also optionally choose the "Add as wildcard" option. Adding wildcards can cause performance issues for large source trees.

Overview of Workspaces and Projects



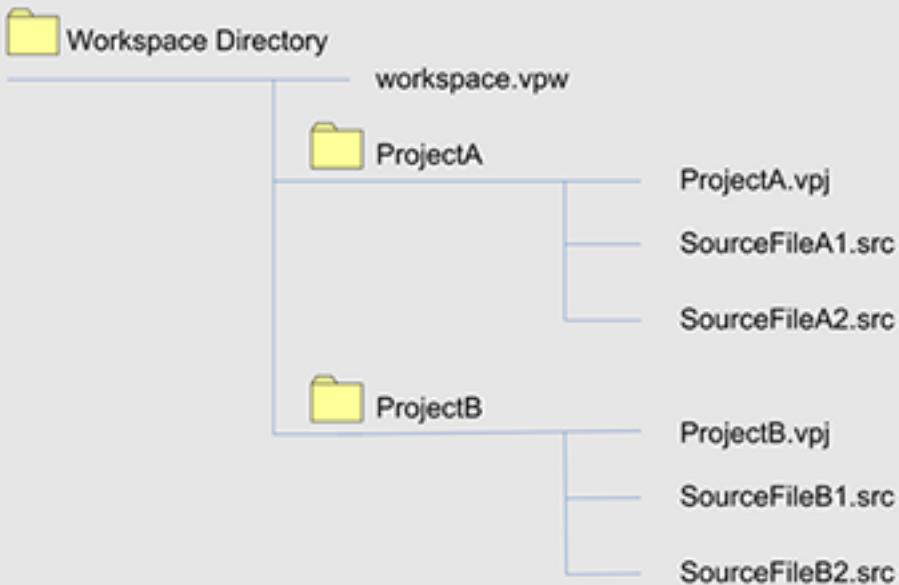
Organizing Workspace (.vpw) and Project (.vpj) Files

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:

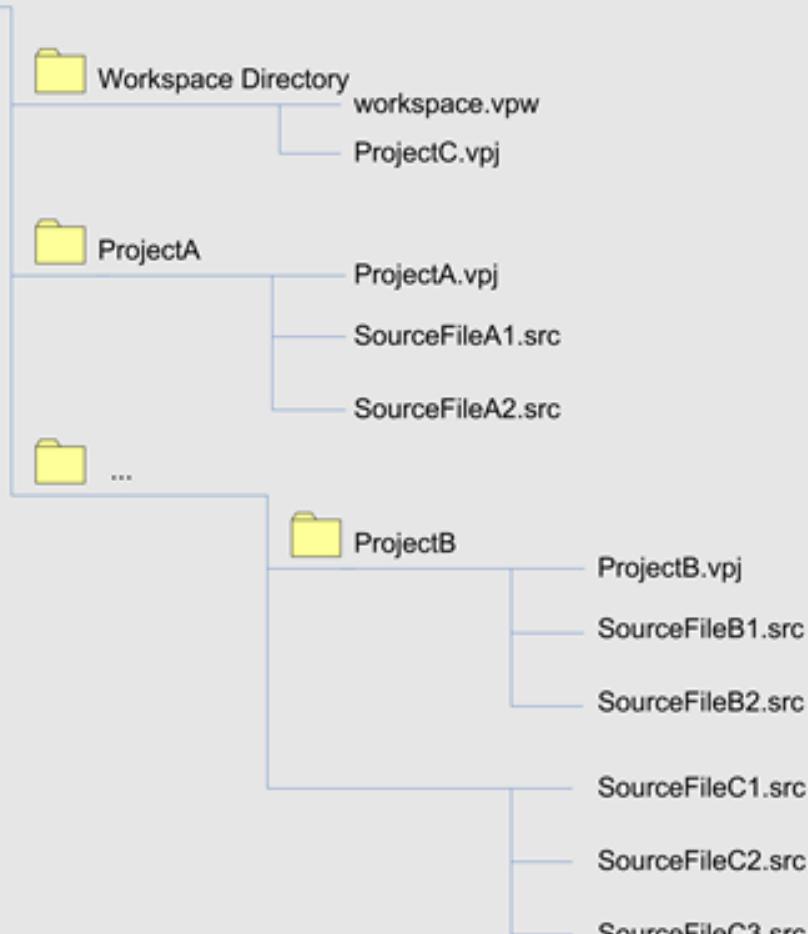
- **Single Root Approach** - All files are stored under a single root directory. In this approach, the workspace .vpw file is typically located in the root folder and there is a subfolder for each project. Each project folder contains the project .vpj file and the source files.

Single Root Approach



- **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.

Multiple Root Approach



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 either set up your workspace locally or more importantly configure your workspace **Tag Files Directory** to a local drive (see [Workspace Properties Dialog](#)). By default, tag files are stored in the same location as your workspace file. SlickEdit reads and writes workspace files (especially tag files) frequently, so storing them remotely will reduce performance.

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(Pro only), 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](#).

Project Wildcards

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 can use **Add as wildcard** when adding files to your projects in order to automatically pick up newly 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 (Pro only)

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 clicking **Tools** → **Tag Files** and then clicking 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 Language-Specific Tag Files](#) for instructions on how to tag libraries.

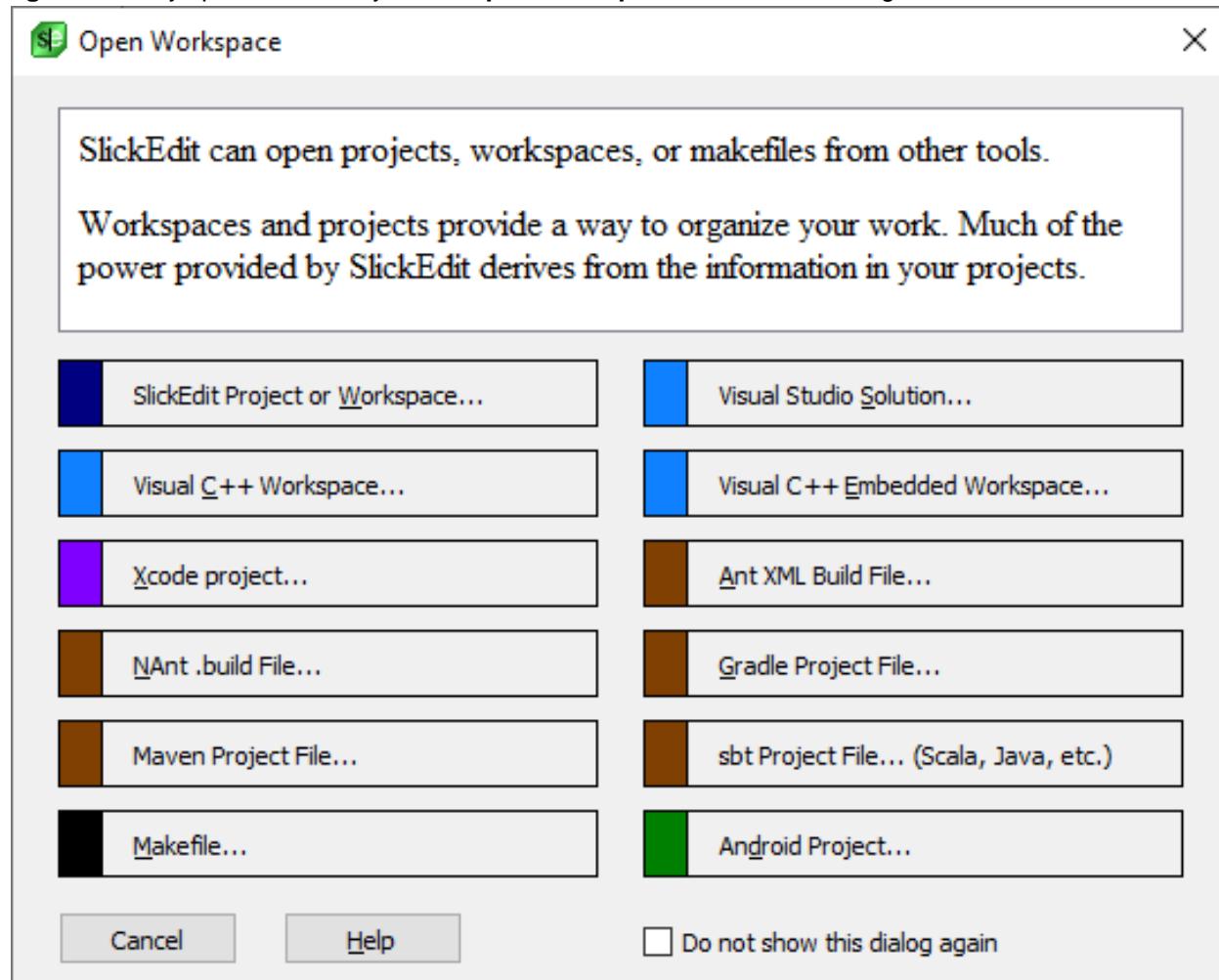
Managing Workspaces

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 clicking **View → Tool Windows → Projects**.

Opening and Closing Workspaces

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

To open a projects, workspaces, or makefiles from other tools, click **Project → Open Workspace** and then click on one of the project types you would like to open. This can also be done using the **Project → Open Other Workspace** menu. See [Open Other Workspace Menu](#). Check **Do not show this dialog again** to always proceed directly to the **Open Workspace** file chooser dialog.



To close a workspace, click **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](#).

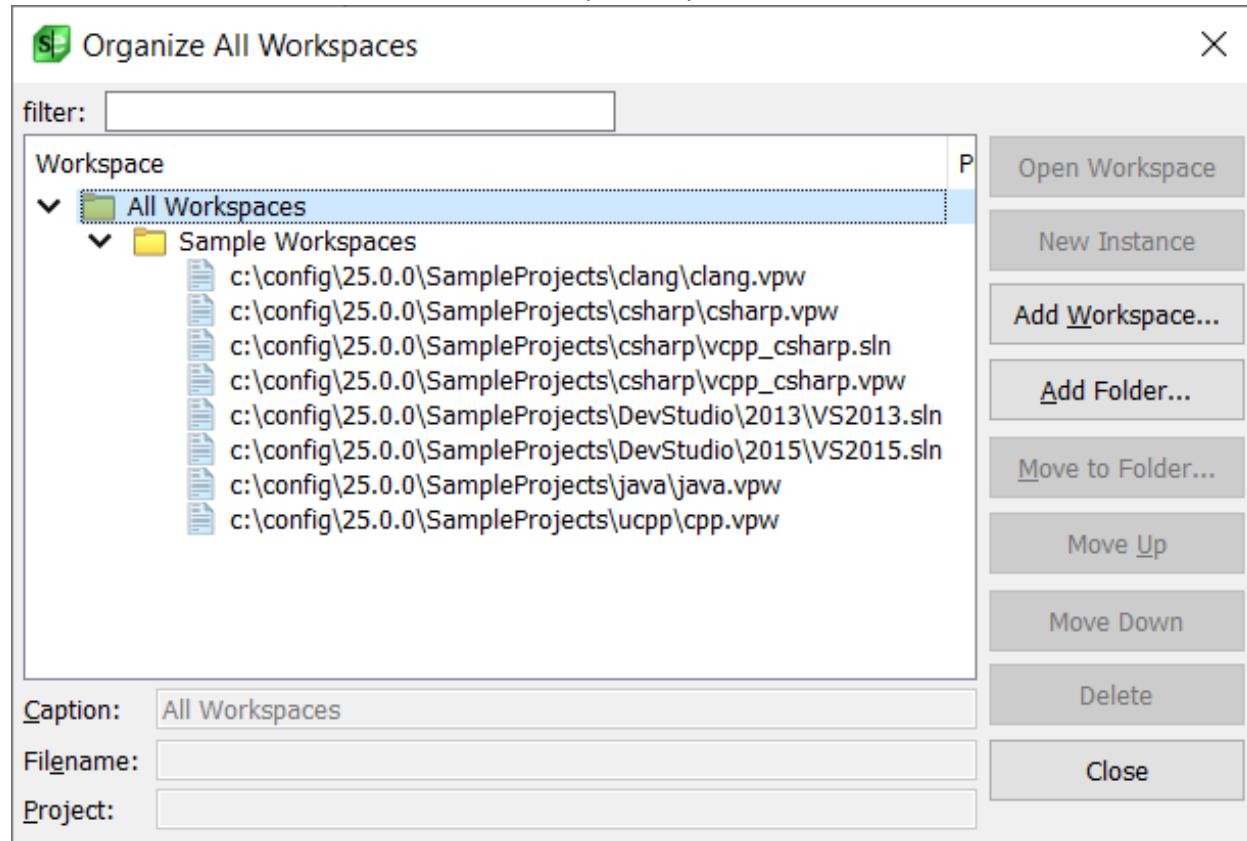
Creating a workspace without creating a project is useful when you plan to import existing projects or if you want a workspace for editing files that aren't part of a project, like shell script files.

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

1. From the main menu, click **Project** → **New** and select the [Workspace 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.

Organizing Workspaces

The Organize All Workspaces dialog can be found by going to **Project** → **Organize All Workspaces...**. This dialog enables you to organize the list of workspaces which appear in the Project Menu (**Project** → **Organize All Workspaces...**). You can add or remove workspaces from the list, as well as sort them into folders. You cannot move or delete the All Workspaces top-level folder.



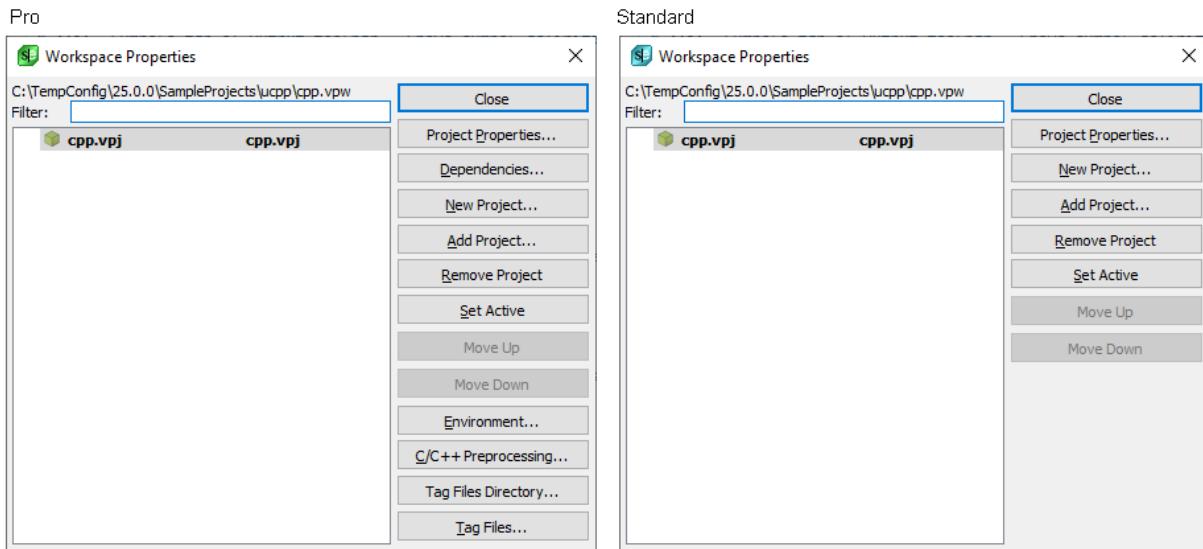
The dialog features a tree which shows the organizational structure of the All Workspaces menu. The current workspace is in bold. By selecting a workspace in the tree, the **Caption** and **Filename** fields will be filled in. You can modify the **Caption** field, and your changes will be reflected in the **Project → Organize All Workspaces...** menu.

The Organize All Workspaces dialog has the following buttons:

- **Open Workspace** - opens the workspace currently selected in the tree.
- **New Instance** - opens the workspace currently selected in the tree in a new instance of SlickEdit.
- **Add Workspace...** - add a new workspace to the list. New workspaces will be added to the bottom of the top-level list.
- **Add Folder...** - add a new folder to the list. This folder will show up as a submenu on the All Workspaces menu. You will be prompted for the new folder's name.
- **Move to Folder...** - move the selected item(s) to an existing folder in the tree. You will be prompted to select the destination folder.
- **Move Up** - move the selected item up one spot. This button is only enabled if the option to sort the All Workspaces menu is disabled, otherwise, moving items up or down in the list would be ineffective. See [History Options](#) for more information.
- **Move Down** - move the selected item down one spot. This button is only enabled if the option to sort the All Workspaces menu is disabled, otherwise, moving items up or down in the list would be ineffective. See [History Options](#) for more information.
- **Delete** - delete the selected workspace or folder from the list. You will be prompted to delete the workspace itself, as well as all associated files. You cannot delete the currently open workspace or the All Workspaces folder. If you delete a folder from the tree, then all workspaces inside that folder will be removed from the list as well. Just as if you had deleted the workspace itself from the list, you will be prompted to delete the workspace files.
- **Close** - close the Organize All Workspaces dialog, applying any changes to the Project menu.

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 clicking **Project → Workspace Properties**.



For more information on using this dialog, see [Project Dialogs and Tool Windows](#).

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, click **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. If you have difficulty opening a Visual Studio 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.
- **Xcode**
- **Cargo (Cargo.toml)**

Managing Projects

Projects are used to hold a set of related files. In compiled languages, a project typically represents the files for a single build unit, either an executable or a library. For interpreted languages, you can use projects to aggregate files into logical groups, though it is common to have a single project.

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

Project Types (Pro only)

SlickEdit provides a variety of project types that match commonly available languages, tool chains (defined by the compiler and debugger you are using), and types of programs. Some project types create main programs, starting you off with a fully compilable program.

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

Warning

It is not possible to configure the **Other** project type to work with supported tool chains. For example, if you are using the GNU compiler or debugger, you need to use the **GNU C/C++ Wizard**. The **Other C/C++** project type supports the GNU debugger but does not contain the configuration options for the GNU compiler.

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 or other project properties each time you set up a new project. You can customize the **(Other)** 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 clicking **Project → Project Properties**.

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

Microsoft Visual Studio

SlickEdit® cannot create Visual Studio solutions or projects. Visual Studio users should create a solution in Visual Studio, and define the projects it contains using Visual Studio.

Solutions can be opened in SlickEdit by clicking **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 Visual Studio project type on the **New Project** dialog, but it will simply warn you that SlickEdit cannot create a project of that type and that you need to do that in Visual Studio.

For C#, a tutorial is available that describes how to build a simple C# console application with SlickEdit, no Visual Studio required. See [Hello World Tutorial \(C#\)](#).

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 Debugger** → **Debug Executable (GDB)**. (See [C and C++](#) for more information about C/C++ features in SlickEdit®.)

C/C++ Compiler Compatible with LLDB (macOS and 64-bit Linux only)

The LLVM compiler suite (clang, clang++) are compatible with the LLVM toolkit debugger LLDB. GNUC, as well as other compilers, are also compatible with LLDB since they use a common ABI and can also use the GDB remote protocol (gdbserver) for remote debugging. For these, you should start with the CLang 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 Debugger** → **Debug Executable (GDB)**. (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 **Other 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.)

Mono

SlickEdit® provides support for a broad selection of .NET / Mono project types. SlickEdit detects the installed Mono interpreter on your system and configures the build, run, and debug commands. (See [Mono](#) for more information about SlickEdit's Mono development features.)

Perl, PHP, and Python

SlickEdit has project types for Perl, PHP, and Python. These are needed so that SlickEdit knows how to run and debug programs for these languages. If you are using one of these languages, pick the associated project type when you create a new project.

Other Dynamic Languages, Including 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 **(Other)** project type and add your files there. You can configure run and debug commands by clicking **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. For more information on creating new projects, see [Project Tab](#).

1. From the main menu, select **Project → New**.
2. (Pro only) Select the type of project that you want. It is critical that you select the correct project type. See [Managing Projects](#) 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**.

Tip

(Close the any modal dialog like the Project Properties dialog first) You can drag/drop a directory from your operating system file explorer onto SlickEdit to perform an Add Tree to the active project. If no workspace/project is open, you will be prompted to create one.

You can also create a project by importing a makefile. To do this, from the main menu, click **Project → Open Other Workspace → Makefile** (or use the **workspace_open_makefile** command). See [Importing Makefiles](#) for more information.

Creating Custom Project Types (Pro only)

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. Click **Project → New**, then click the **Customize** button.
2. On the Customize Project Types 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 the **(Other)** 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 Project Types 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. Click **Project → New**, then click the **Customize** button.
2. On the Customize Project Types 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 Project Types 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 `usrprjtemplates.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

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

(Pro only) Each workspace contains one project that is the active project. The active project is the one that is built when you click **Build → Build**. If the active project depends on other projects, those projects will be built first.

Defining Project Dependencies (Pro only)

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. Click **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 (Pro only)

Projects can have multiple configurations, each with different values for project settings. The most common use of project configurations is for creating 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. You can change the active configuration by selecting **Build → Set Active Configuration** from the main menu.

SlickEdit creates a "Debug" and "Release" configuration for each new project. These configurations are identical, until you change the project settings associated with them. When you select **Project → Project Properties**, the drop-down list at the top lets you select the project configuration you are modifying. The "All configurations" value is selected by default. By selecting a configuration you can change things that are specific to that configuration, like setting compiler flags or changing the list of source files.

Note

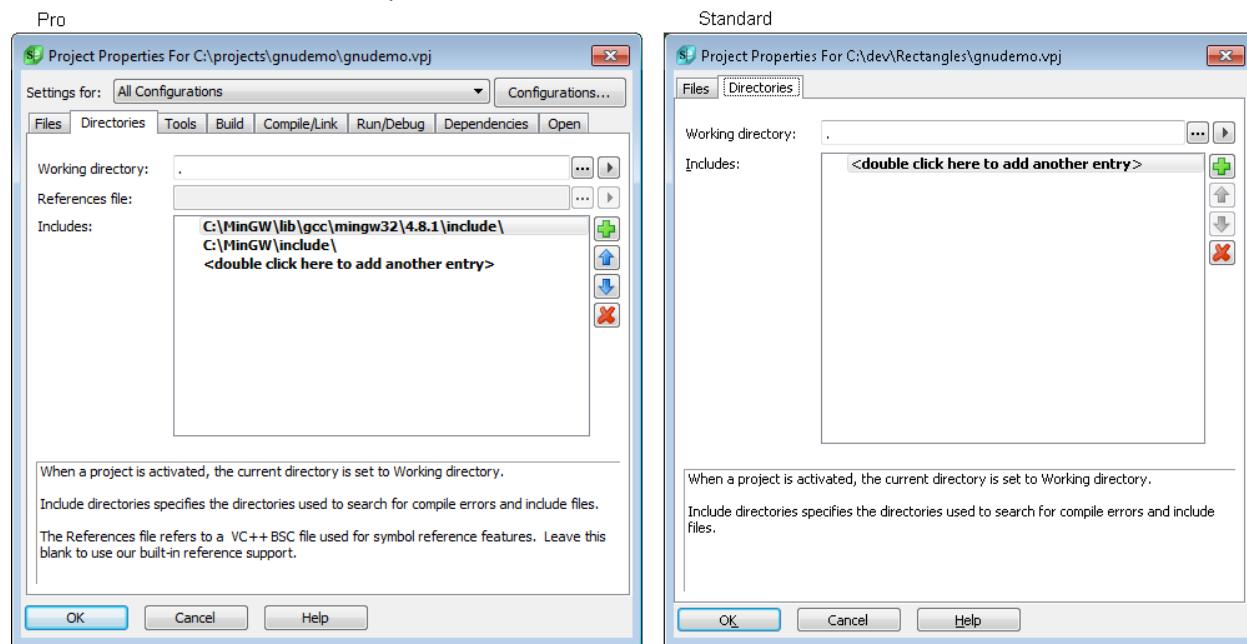
- **Visual C++** - If you open a Visual C++ v5.0 or later workspace, the configurations are automatically retrieved from the Visual C++ project. Some typical configurations for Visual C++ v5.0 or later are "CFG=MyApp - Win32 Debug" and "CFG=MyApp - Win32 Release." Use Visual C++ to change the configurations.

- **macOS** - 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.

For information on using Project Configurations in builds, see [Project Configurations in Builds](#).

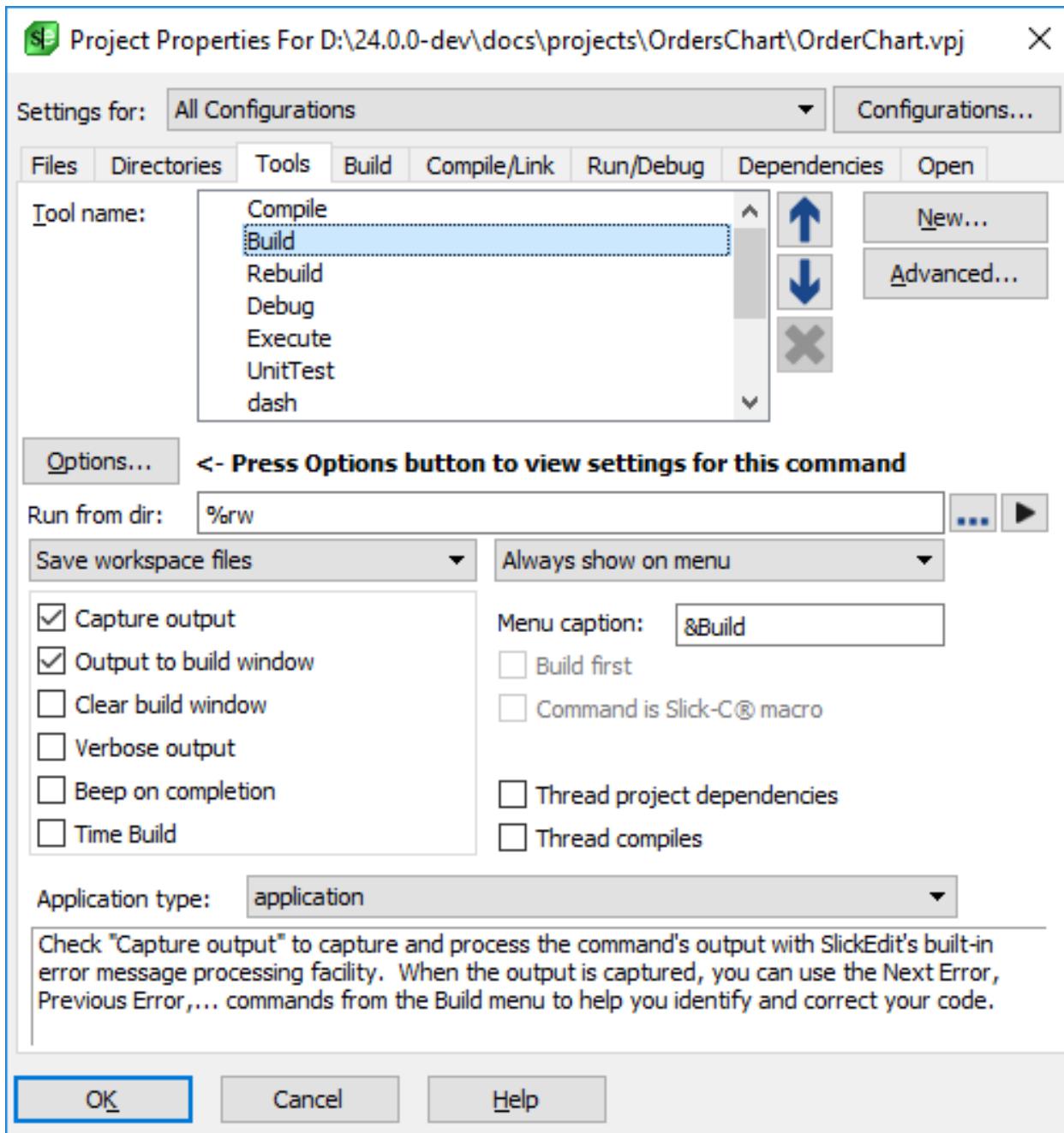
Configuring Project Directories

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 (Pro only)

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 button 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 Language-Specific Options

The **Options** button on the Project Properties **Tools** tab is only available for selected tools that support language-specific options. Click the **Options** button to display options specific to the language with which you are currently working. From there you can make settings for the command that gets executed for the tool specified in the **Tool name** combo box. For more information on changing language options, see the topic for your 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).

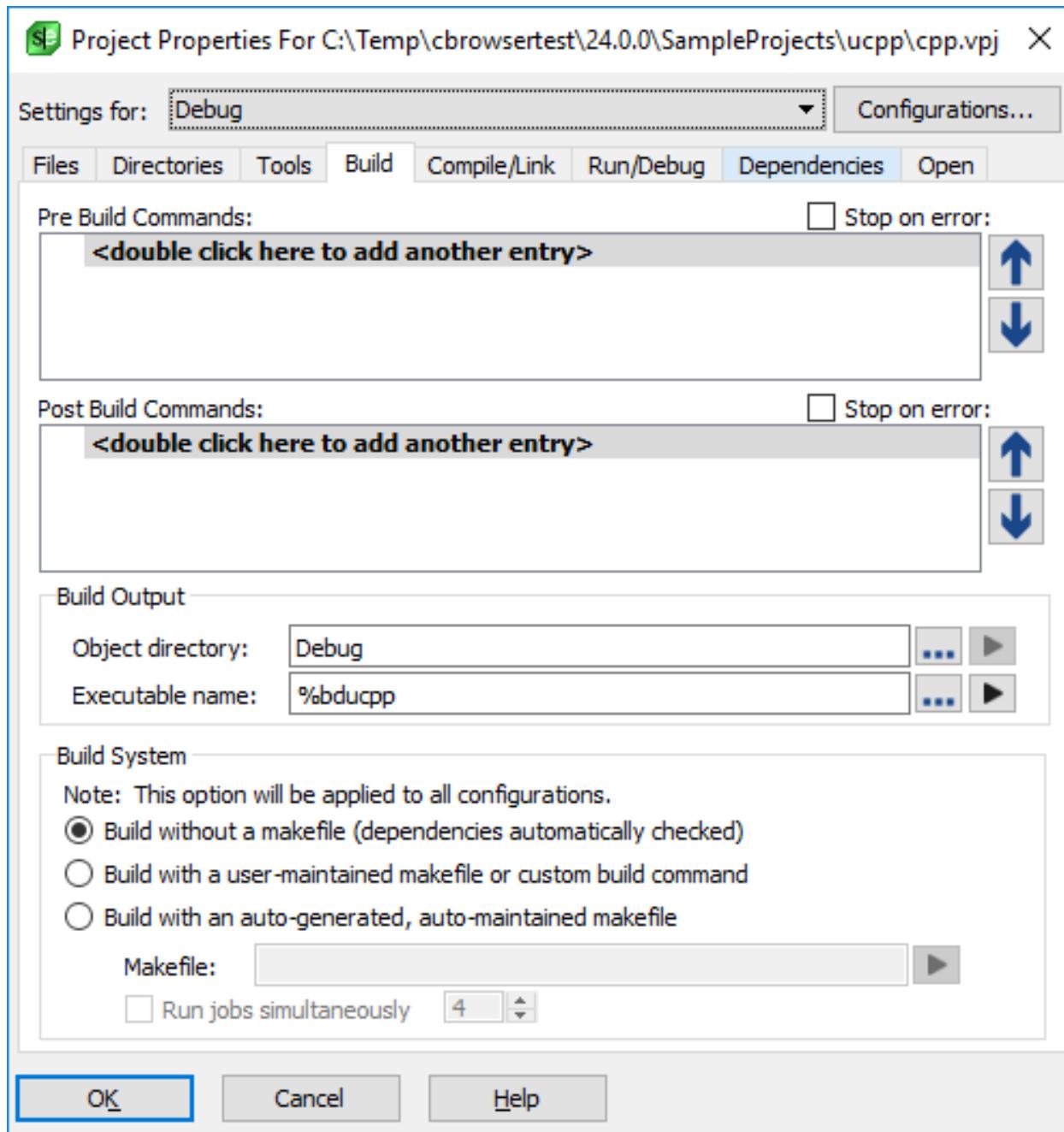
All of the options and settings on the Project Properties **Tools** tab are outlined in the section [Tools Tab](#).

Configuring Build Settings (Pro only)

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**) allow for 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 unavailable 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 Build and Compile Operations](#)) 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 Output Options

The build output options on the **Build** tab allow you to configure where object files are placed by the build, as well as the name of the executable name being created.

- **Object directory** - This is the directory where object files and the executable are placed by the build process. When building using a user-defined command or a custom makefile, set this to the directory where you expect the build process to place the object files and executable. When building using SlickEdit's build system or an auto-generated Makefile, this is the directory where object files will be placed. This setting is referenced elsewhere in the Project Properties dialog using the **%bd** project escape sequence.

If the **Object directory** is not specified, the default is to use a subdirectory matching the configuration name under the project directory.

- **Executable name** - This is the name of the item created by this build process. For a program, this is typically the name of the executable. For a library, this is the name of the library, including the library suffix (for example, .dll, .so, .a, .lib, .dylib).

This setting can be either an absolute path or just a file name or path relative to the **Object directory**. This setting is referenced elsewhere in the Project Properties dialog using the **%o** (output file name) project escape sequence, as well as the related escape sequences **%on** (output file name only), **%oe** (output file extension), and **%op** (output file path).

Build System Options

The build method options on the **Build** tab apply to 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 Build and Compile Operations](#)) 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 Build and Compile Operations](#)). 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 Language-Specific Projects

(Pro only) Language-specific projects are based on the language of the current file. The working directory is ignored for these projects. All language-specific projects are stored in the file `project.vpe` (UNIX: `uproject.vpe`).

(Pro only) If you are building something that only contains one source file and no project, you can define a language-specific project and configure project tools such as build and/or execute.

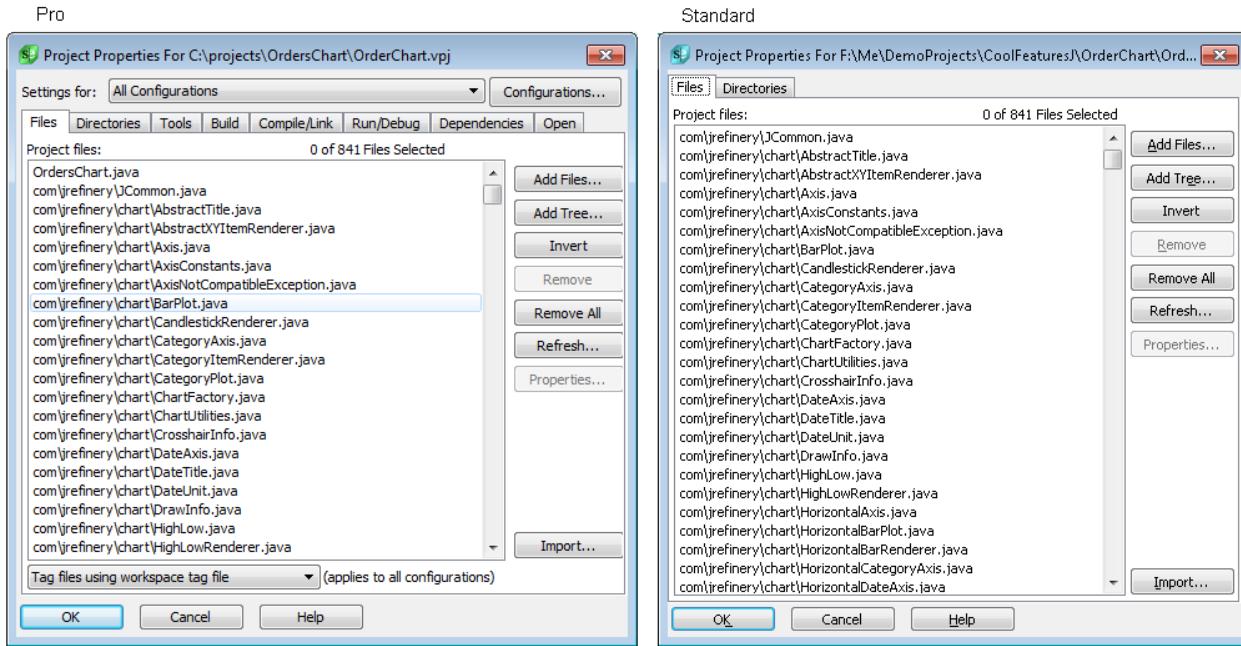
To define a language-specific project, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Single File Projects**.

Managing Source Files

Adding and Removing Files

To get the benefits of SlickEdit's Context Tagging features or to use the SlickEdit build system, SlickEdit must know about the set of files you are working on. The **Project Properties** dialog contains a **Files** tab that lists the files associated with this project. Files can be added to the project explicitly or implicitly.



You explicitly add files to the project using **Add File** or **Add Tree**. Files can also be explicitly added to your project when you select **File** → **New** (added by default) or when you select **File** → **Save As** (file not added by default). Explicitly added files are listed on the **Files** tab.

Note

By default SlickEdit displays the **Project Properties** dialog with **All Configurations** selected. To invoke the **Project Properties** dialog with the current active configuration, use **Project Properties for Config** found on the **Build** menu.

You can add files implicitly using the **Add as wildcard** option available on the **Add Tree** dialog. With this mechanism, you define a filespec which SlickEdit will use to search for matching files. This search is performed each time the editor is launched. Using this method, only the filespec is listed in the **Files** tab, not the matching files.

These two approaches can be mixed. You can have some files in your project picked up by wildcard and others that are explicitly added. In that case you will see both filespecs and actual files listed. You should be careful not to explicitly add any files in directories that will be matched by a wildcard, or the file will be picked up and added to the project twice.

SlickEdit works the same whether the file was added explicitly or implicitly. In both cases, the file will be tagged and can be built using the SlickEdit build system. Which mechanism you choose depends on how you work and whether your whole team is using SlickEdit. The following questions govern how to set up your projects:

- Is the whole team working with SlickEdit?
- Do you want to use the SlickEdit build system?
- Do your directories often contain scratch files that should not be included into a project?

- Do you have a large or remotely stored codebase?

If the whole team is working with SlickEdit, then you can easily manage the list of files explicitly. You use **Add Tree** and **Add File** when first setting up your projects. Then newly added files are picked up when you create them, using **File → New**. By checking in your workspace and project files along with your source files other team members will see newly added files when they update from the repository. This approach is particularly useful if you plan to use the SlickEdit build system.

If some team members are not using SlickEdit, then they will be creating new files without updating the workspace and project files. You can use **Add as wildcard** (found on the **Add Tree** dialog) to create the file list, and it will pick up these newly added files, assuming the files were added in a location specified by one of the filespecs.

One drawback in using **Add as wildcard** is that it may pick up files that you don't want to include in the project. This is typically the case if you create scratch files with the same extension and leave them in the same directories as your source files. While this is not a problem for Context Tagging, except for the occasional unneeded symbol, it can pose a bigger problem for the build system. Since SlickEdit uses the same file list for both, you need to consider your work habits when using **Add as wildcard** with the SlickEdit build system.

Performance also needs to be considered before choosing **Add as wildcard**. If you have a normal sized project and all of the source files are stored locally, the **Add as wildcard** check will only take a few seconds. In testing, a project with over 1,000 source files took less than 10 seconds to build the initial file list and less than a second to scan for updates. If you have a very large project or your source files are stored on a network share, this can take significantly longer.

How to Add or Remove Files From a Project

To add or remove files from a project, complete the following steps:

1. From the main menu, click **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 other options to use.

Note

(Close the any modal dialog like the Project Properties dialog first) You can drag/drop a directory from your operating system file explorer onto SlickEdit to perform an Add Tree to the active project. If no workspace/project is open, you will be prompted to create one.

- **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.

Add Files

Use **Add Files** to add a single file or the files from a single directory to your SlickEdit project. When you click the **Add Files** button, SlickEdit displays the **Add Source Files** dialog. With this, you can browse to a directory and select one or more files to add. You can filter the list of displayed files by selecting from the **Files of type** drop down list.

Note

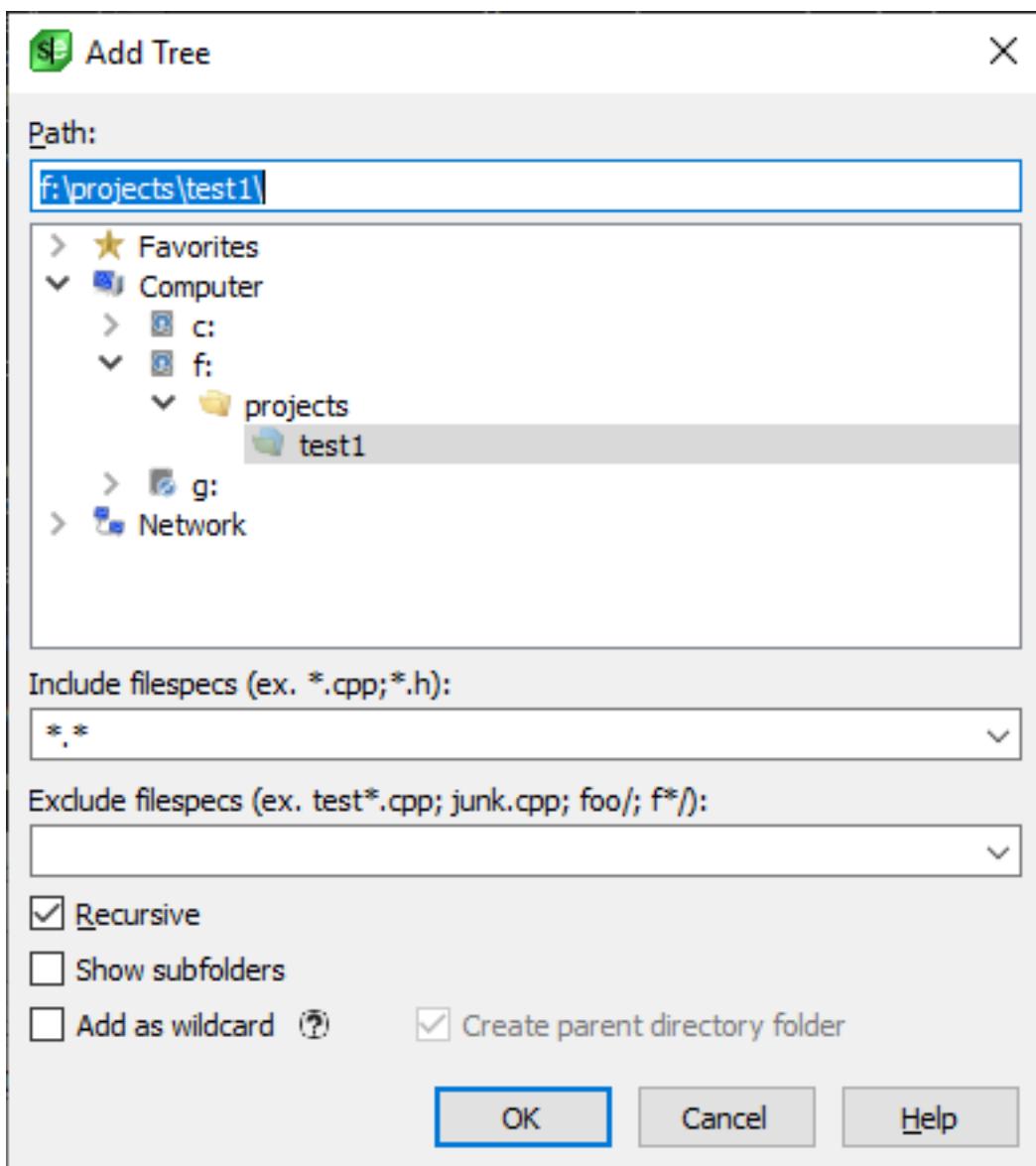
By default, the **Add Source Files** dialog will not list files in the directory being viewed that are already included in the project. This makes it easier to locate files to add that are not yet part of the project. This feature can be turned off by changing the [Open File Options](#).

Add Tree

Note

(Close the any modal dialog like the Project Properties dialog first) You can drag/drop a directory from your operating system file explorer onto SlickEdit to perform an Add Tree to the active project. If no workspace/project is open, you will be prompted to create one.

Use **Add Tree** to add all or some of the files under a specified directory. SlickEdit displays the **Add Tree** dialog, which allows you to select a directory from which to add files. Select a value or enter a new value in the **File types** combo box to specify which kinds of files are added. To enter your own file types, use * to match any characters and separate multiple file types using a semicolon. For more information on this dialog, see [Add Tree Dialog](#).



- **Path** - Path to search
- **Include filespecs** - Semicolon delimited list of ant-like wildcards. For example, enter "`*.c;*.cpp;*.h`" to include all files with .c, .cpp, and .h extensions.
- **Excludes filespecs** - Semicolon delimited list of ant-like wildcard file specifications to be excluded. For example, `junk*;test*` will exclude all files with names beginning with "junk" or "test". To exclude a subdirectory with a particular name, put a slash at the end of the name. For example, enter `".svn/` (short hand for `**/.svn/**`) to exclude all subdirectories named ".svn" wherever they occur. A more advanced ant-like wildcard can be used like `"c*"/` to exclude any directory that starts with "c". For more examples, see [Exclusion Examples](#).
- **Recursive** - Search subdirectories. To limit the search to the selected directory, uncheck **Recursive**.
- **Show subfolders** - Creating project folders for each child directory when adding files recursively.

Project folders are seen in the Projects tool window when the project is in Custom View.

- **Add as wildcard** - Specify that you want this tree to be periodically checked for new files.
- **Create parent directory folder** - Turn on the **Create parent directory folder** checkbox if you want a project folder created for the parent directory of a wildcard. For example, if the **Path** for this wildcard is `f:\project\source`, a **source** project folder will be created in the Projects tool window when in Custom View.

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`.

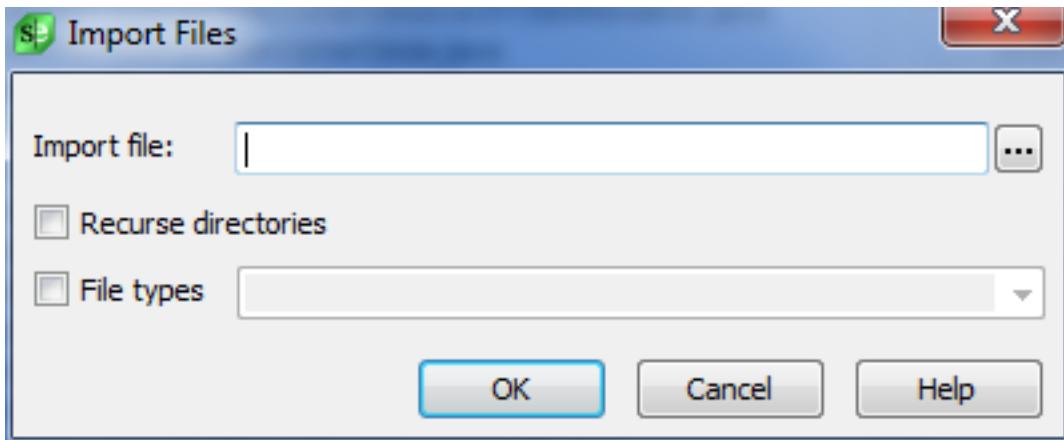
To create a new source file using the second approach, complete the following steps:

1. From the main menu, click **Project** → **New** and select the **File** tab.
2. Select the document mode 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 clicking **File** → **New Item from Template**. See [Code Templates](#) for more information.

Import Files

The Import Files 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.



The dialog has the following fields:

- **Import file** - the import file containing a list of the files and directories to be added to the current project. Use the browse button to navigate to and select the file. Each line in the file should contain the full path of a file or a directory to be added to the project. For directories, the application will add the files within that directory to the project.
- **Recurse directories** - Check this box to recurse into the subdirectories of directories specified in the import file.
- **File types** - Check this box to include only specific file types when adding files from directories specified in the import file. Select a file type from the combo box or add your own. If this box is not checked, then all the files found will be added (*.*). The file type restriction does not apply to individual files listed in the import file. They will be added to the project regardless of this setting.

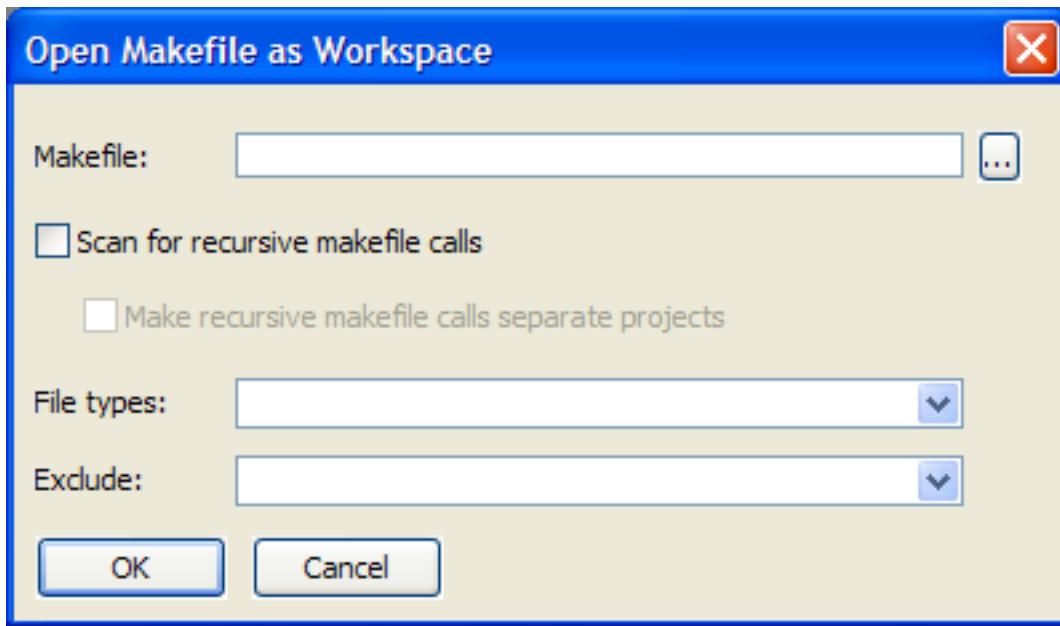
Importing Makefiles

You can create a project by importing a makefile. SlickEdit® parses the targets, finds all referenced source files, and adds them to the project. When you import a makefile, SlickEdit creates a new workspace and adds the project to it. If the same makefile is ever imported again, the corresponding workspace can be opened without creating a new redundant workspace.

The new project automatically imports all files that are referenced by the makefile. All of the make targets are also added and made available for execution from the main menu under **Build → Execute Makefile Target**. For makefiles that contain invocations of other makefiles, the other makefiles can be optionally added to the workspace as separate projects, or all their files added into one project.

The Build option for the makefile project is set to **Build with a user-maintained makefile or custom build command** (**Project → Project Properties, Build tab**) and the build command is set to **make**. See [Build System Options](#) for more information.

To create a new project by importing a makefile, from the main menu, click **Project → Open Other Workspace → Makefile** (or use the **workspace_open_makefile** command). The Open Makefile as Workspace dialog is displayed.



In the **Makefile** field, specify the makefile to import. Use the **Browse** button to browse for the makefile. Check **Scan for recursive makefile calls** if you want to also scan for invocations of make on other makefiles and to include them in the project. In this case, you can also check **Make recursive makefile calls separate projects** if you want a new, separate project created for each of the referenced makefiles. Then, specify any file types to include or exclude by using the drop-down lists or by typing the file extensions separated with semicolons. The "*" wildcard is permitted. For example, ***.ch;*.chf;*.chs;*.cpp;*.h** can be used to include or exclude all referenced files with those extensions.

Loading Project Files for Editing

The Files tool window can be used to open one or more files from the current project or the current workspace. To display the tool window, from the main menu, click **Project**, then select **Open Files from Project** or **Open Files from Workspace** (**project_load** command). The files that are shown in the tool window depend on the menu item selected. A button on the tool window also lets you toggle between viewing current project files and current workspace files.

To open a selected file, press **Enter**. Use **Ctrl+Click** to select more than one file. See [Document Dialogs and Tool Windows](#) for more information about operations in this window.

Single File Projects (Pro only)

Overview

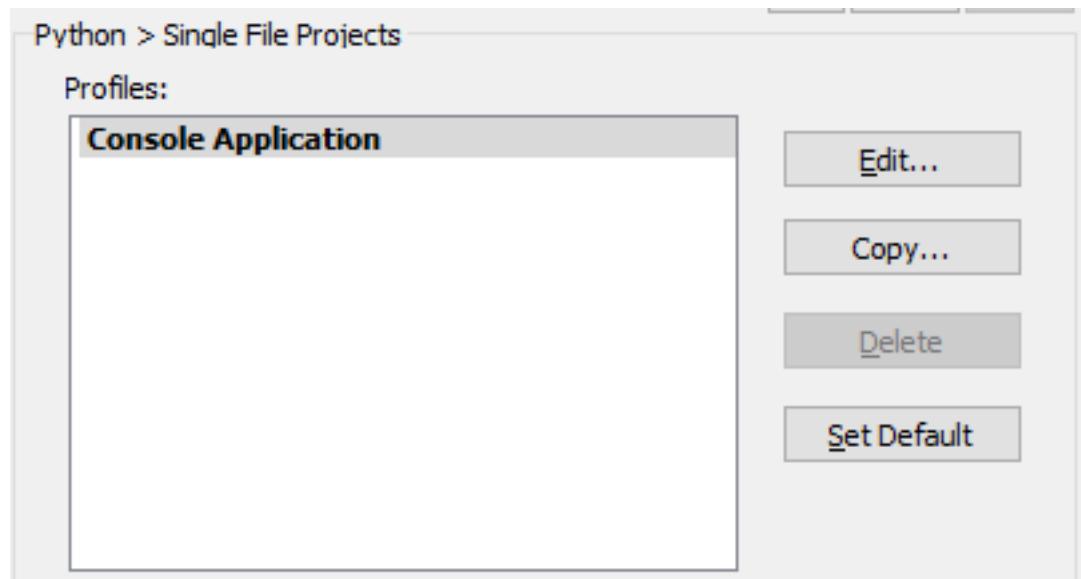
Single file projects are perfect for building, running, and debugging single file programs. If your program requires more than one source file to build, you need to create a workspace and project. Single file project profiles are provided for many languages including Python, Perl, Ruby, Groovy, Google Go, Java, C#, and C++. You can create or customize your own single file project profiles on a per language or per file basis. To activate the single file project for the current file you are editing, close your workspace and it will be active. When the single file project is active, the Build menu will be updated. JUnit testing and WinDbg (for C++ debugging on Windows) are not supported by single file projects.

Configuring Single File Project Profiles

Go to Document → [Language] Options → Single File Projects to configure your single file project profiles for a specific language.

Note

Once you modify a single file project using the Project Properties dialog from the Project menu, a copy of the language specific profile is created. Once this happens, modifying the default single file project in the language specific settings won't have any effect for that file. File-specific single file projects and customized single file project profiles are stored in the configuration directory in user.cfg.xml.



- **Edit...** - Displays the Project Properties dialog for modifying the selected single file project profile.

- **Copy...** or **New...** - Copies the settings from the selected single file project profile or creates a new profile.
- **Delete** - Deletes the selected single file project profile. System profiles can't be deleted.
- **Set Default** - Sets the default single file profile used by files for this language.

Working with Files

Overview

SlickEdit® provides familiar operations for creating, opening, and working with files. There are many ways to perform these tasks and many options available for specifying your preferences.

Topics in this section are:

- [The Working Directory](#)
- [Creating Files](#)
- [Opening Files](#)
- [Saving Files](#)
- [File Backups](#)
- [Closing Files](#)
- [The SlickEdit® File Manager](#)

Here are a few tips to keep in mind as you read through this section and work with files:

- SlickEdit supports file names up to 200 characters long.
- If you're using the command line and need to edit a file whose name contains space characters, place double quotation marks around the name.
- Options are available for specifying actions that automatically occur when a file is opened or saved, including adjustments for the line format. When you specify a file to edit, the format is automatically recognized and necessary adjustments are made, based on whether the file type is Windows/DOS(CRLF), Unix/macOS(LF), or Classic Mac(CR). An EOF character will not be appended to Unix/macOS(LF), or Classic Mac(CR) text file formats 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 destination buffer.

A Word of Caution for Binary Files

The default Load and Save options are safe for editing binary files since all reformatting is turned off. As an extra precaution to ensure that no reformatting takes place when opening and saving binary files, set the encoding to **Binary** when opening the file.

While the default options are safe, it is NOT safe to edit binary files with the following non-default settings, due to the risk of unintentionally modifying the files. These options are located on the **Load** and **Save** option screens (**Tools** → **Options** → **File Options**):

- **Show EOF character** - False
- **Expand tabs to spaces** - True
- **Append EOF character** - True
- **Remove EOF character** - True
- **Strip trailing spaces** - True

The Working Directory

SlickEdit® uses the current working directory for various operations that involve file navigation, and to save you from tediously browsing and typing long path names when navigating files. For example, when you use the **e** command to open a file for editing (see [Using the e Command](#)), you don't need to specify the complete path if the file is in the current working directory.

To see the current working directory at any time, use the **pwd** command on the SlickEdit command line.

SlickEdit automatically changes the working directory to save you time in various situations. You can also manually set the working directory at any time. Note that the working directory in SlickEdit is different from the working directory used by the Build tool window. See the following topics:

- [Automatic Changes to the Working Directory](#)
- [Manually Changing the Working Directory](#)
- [Working Directory in the Build Window](#)

Automatic Changes to the Working Directory

SlickEdit® automatically sets or changes the working directory in the following situations:

- By default, when you start SlickEdit, the working directory is automatically set to the directory from the previous editing session. You can turn this behavior off with the option **Auto restore working directory** (**Tools** → **Options** → **Application Options** → **Auto Restore**).
- The working directory is automatically set or changed when you switch projects, according to the specified project directory or the directory specified in the **Working directory** field on the **Directories** tab of the Project Properties dialog (**Project** → **Project Properties**).
- By default, the working directory is automatically set or changed if you specify a directory with the **open** or **save_as** commands or specify a directory location in the Open, Save Copy As, and Save As dialogs (**File** → **Open**, **File** → **Save Copy As** and **File** → **Save As**). You can change this behavior by setting the **Change directory** option (**Tools** → **Options** → **Appearance** → **General**) to **False**.
- The working directory changes to match directories you select in the Open tool window. This window is docked to the left tab group of the editor by default, and can also be displayed by clicking **View** → **Tool Windows** → **Open** from the main menu.

- The working directory can be automatically set to the corresponding directory when you navigate to a file, if the configuration variable `def_switchbuf_cd` is modified (see [Configuration Variables](#)). This is similar to the **Change directory** option mentioned in the preceding bullet, except it also works for any file navigation operation (such as `e` or `edit` or when SlickEdit opens a file automatically, such as with [Symbol Navigation](#)). This feature is on by default in GNU Emacs emulation and off in all other emulations.

Manually Changing the Working Directory

You can change the working directory at any time with the Change Directory dialog (from the main menu, click **File** → **Change Directory** or use the `gui_cd` command). If you want to also change the directory in the Build window, select the option **Change directory in Build Window**. To use an alias for the directory name, select **Expand alias**. Click **Save Settings** to preserve your preferences. The settings here are also used when you use the command line version of this feature.

Changing Directories From the SlickEdit® Command Line

To change the directory from the SlickEdit command line, use the `cd` command as follows, where the `p` option corresponds to the **Change directory in Build Window** option on the Change Directory dialog, and `a` corresponds to **Expand alias**:

```
cd {+p|-p} {+a|-a} drive/and/path/to/directory
```

You can also change the working directory with the `pushd` and `popd` commands. The `pushd` command adds the current working directory to the top of the directory stack, and makes the specified directory the new working directory. For example:

```
pushd drive/and/path/to/directory
```

If no arguments are given, the current working directory is swapped with the directory at the top of the directory stack. After using `pushd`, you can use the `popd` command to remove the top directory from the directory stack and make it the new working directory.

Working Directory in the Build Window (Pro only)

The working directory in the editor is different from the working directory in the **Build** tool window. The Build window is a shell that displays build and compile output, and also allows you to type operating system commands and see the results. It is docked to the bottom tab group of the editor by default. It can also be displayed by clicking **View** → **Tool Windows** → **Build** from the main menu.

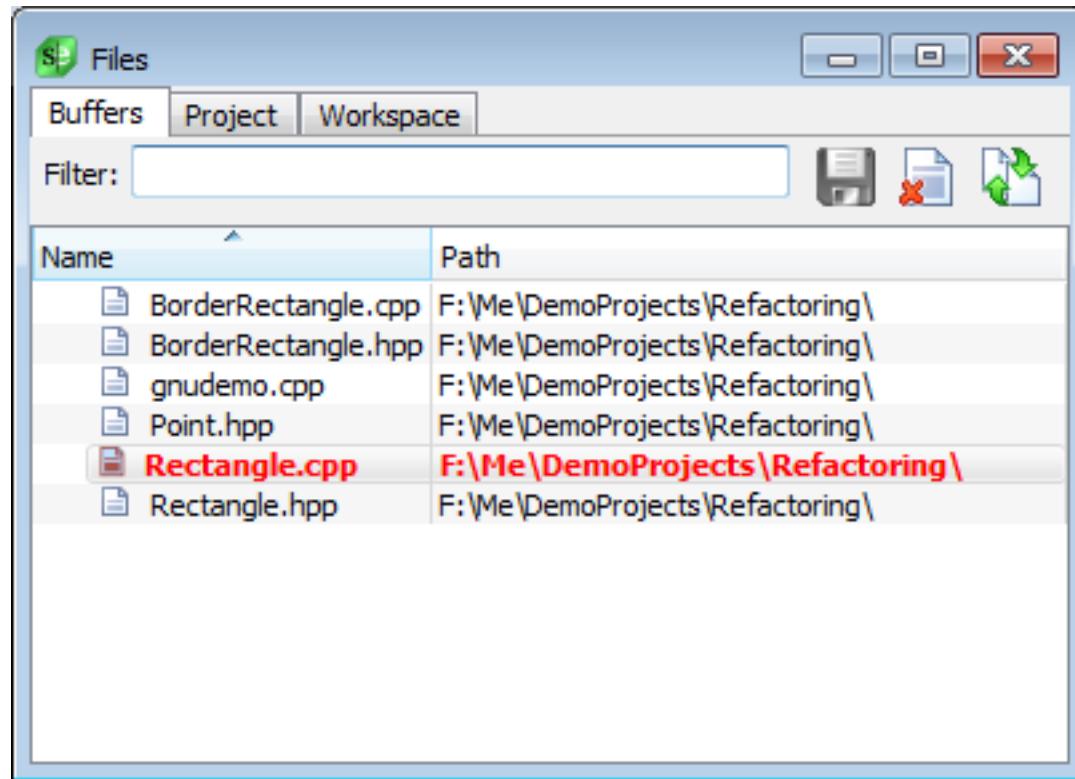
The working directory is shown in the prompt inside the Build window. When you first launch SlickEdit®, the window is blank. Type a command in the window for the prompt to be displayed.

By default, when the directory is changed in the editor, it is also changed in the Build window. However, the reverse is not true: changing the directory in the Build window does not change the directory in the editor. To make the working directory completely independent so that a change directory does not affect the Build window, make sure the **Change directory in Build Window** option on the Change Directory dialog is off (not selected) and save the settings (see [Manually Changing the Working Directory](#)).

The Files Tool Window

The Files tool window has three tabs to display:

- the list of open buffers.
- the list of files in the current project.
- the list of files in the workspace.



See [Document Dialogs and Tool Windows](#) for more information.

Creating Files

SlickEdit® provides four methods for creating new files:

- [Using the e Command](#)
- [Using the New File Dialog](#)
- [Using Code Templates](#)
- [Using Write Selection](#)

Using the e Command

One of the quickest ways to both create and open files in SlickEdit is to use the **e** command on the SlickEdit command line. Press the ESC key to open the command line (in most emulations), then type **e file**, where *file* is the name of the file. The **e** command is just shorthand for the **edit** command, so you can use **edit** instead, if you prefer.

Tip

Many people prefer to use the **e** command to create new files. Typing **e myfile.ext** is the fastest way to create a new file in SlickEdit. Including the extension on the filename is all that is needed to let SlickEdit know what file type you are editing so it can use the correct language mode.

Depending on your options, SlickEdit will match the characters you type against the files in the current working directory, open files, and files in the current workspace. To configure this option, select **Tools** → **Options**, expand **File Options** → **Open** and set **e/edit command Smart Open** to one of these values:

- **Smart Open off** - matches only against files in the current working directory.
- **Smart Open open documents** - matches files in the current working directory and those that are already open. Matching open files is useful if you have a lot of files open.
- **Smart Open workspace files and open documents** - matches against files in the current working directory, open files, and files in the current workspace.
- **Smart Open workspace files** - matches against files in the current working directory and files in the current workspace.
- **Smart Open files in same directory** - matches against files in the same directory as the current file.

If you're opening a file that is not in the current working directory, use the full path. [Command Line Completion](#) will help entering file names and paths.

If the file already exists, the file will be loaded in a new buffer. If the specified file does not exist, a new buffer by that name will be created. For new files, be sure to specify the file extension so SlickEdit knows what language mode to use.

Using the New File Dialog

If you prefer working with the GUI, use the menu item **File** → **New** to create a new file. This displays the New dialog open to the **File tab**. First select the **Document Mode** for the new file (recently chosen modes are listed at the top). The **Add to Project** box determines whether the new file will be added to the active project. Its value is retained from the last time this dialog was used. Specify a **Filename**, the **Location**, and **Encoding**, **Line format**, then click **OK**. See the section on the New dialog's [New Dialog](#) for more information about these fields.

You can create a new, untitled buffer by leaving the file name blank and then hitting OK or double-clicking on a language in the **Document Mode** list.

Using 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. See [Code Templates](#) for more information and details about this feature.

To instantiate a predefined Code Template, complete the following steps:

1. From the main menu, click **File → New Item from Template** (or use the **add_item** command). The Add New Item dialog is displayed.
2. In the **Categories** list, expand **Installed Templates**, then click through the tree to select the language and category of the file you want to create.
3. In the **Templates** box, select the template you want to use.
4. In the **Name** field, type the name of the new file.
5. In the **Location** field, type the directory path where you want to store the new file, or click **Browse**. This field is prepopulated with the active directory.
6. Optionally, select **Add to current project**.
7. The type of template you selected may generate more than one file. If you want to view a confirmation list of the files that will be created after you click **Add**, select **Confirm files before adding**.
8. Click **Add**.

Using Write Selection

SlickEdit also lets you create a file from a selection. Select the text you want to write, then from the main menu, click **File → Write Selection** (or use the **gui_write_selection** command). The Write Selection dialog is displayed, with fields for specifying the file name and location (similar to the [Save As Dialog](#)). The selected text is written to the specified file.

Opening Files

SlickEdit provides many ways to open files. With most editors, you must explicitly open each file you want to view or edit. To do that you must know the filename and location of the file. In SlickEdit, you often open files implicitly, by browsing into them. With these mechanisms, you don't have to know the filename or where the file is located to open it.

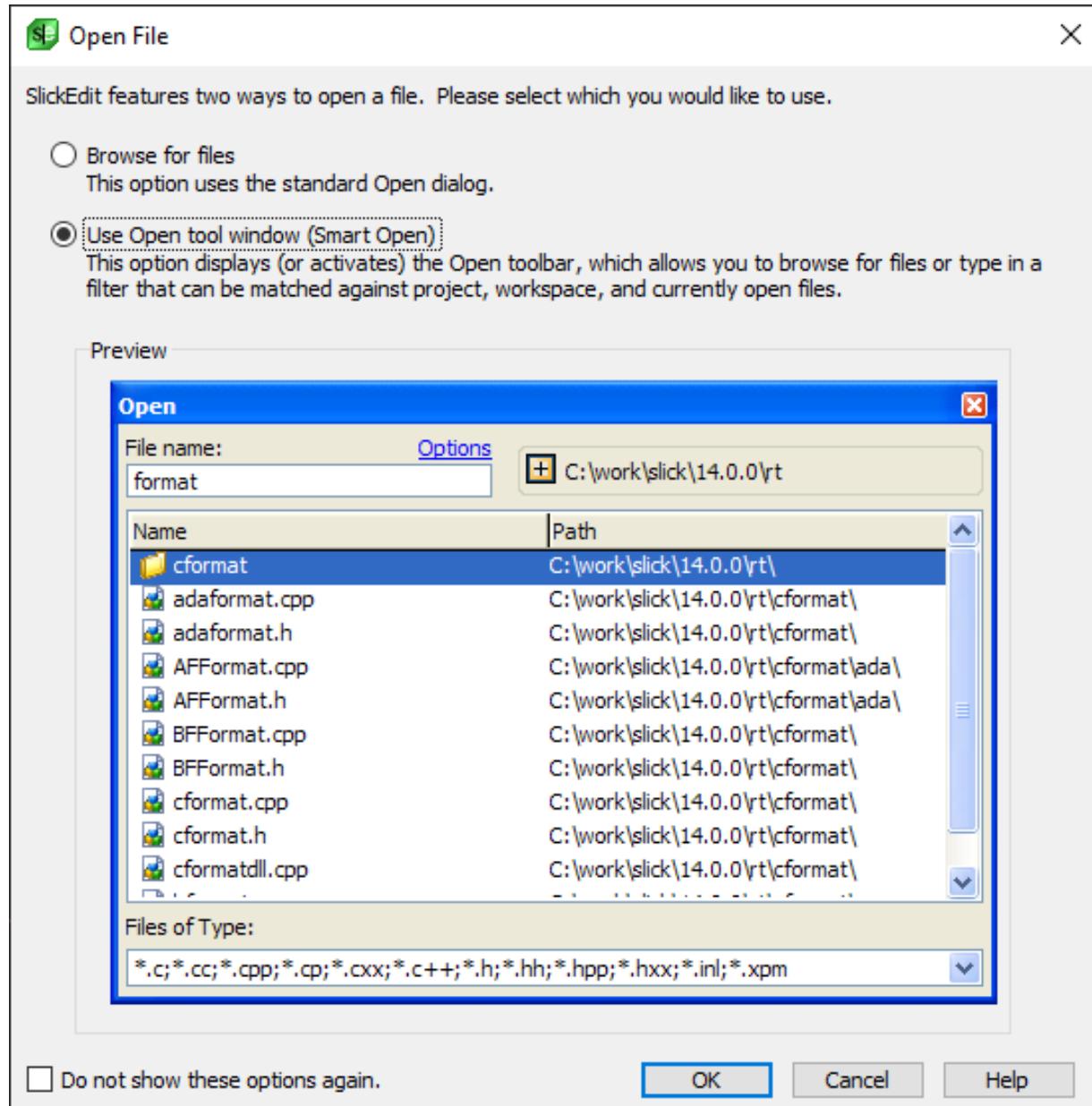
Explicitly Opening Files

To open a file by name, use any of the following mechanisms:

- One of the quickest ways to open a file is to use the **e** command (in the syntax **e filename**), which can also be used to create files. SlickEdit uses completions to match the filename you type against the current directory, open files, and files in your workspace (Not in Community edition). If you are entering the full path to a file, completions will assist you with that as well. See [Using the e Command](#) for more

information.

- If you prefer to open files with the GUI, select **File → Open** (or use the **gui_open** command). SlickEdit will prompt whether to open files using a standard file browser or using the Open tool window



- If you choose **Browse for files**, SlickEdit uses the standard Open dialog (see [Standard Open Dialog](#) for more information).
- The Open tool window allows you to browse through directories and view a list of files. You can type a filename and the list of displayed files will be filtered to match those from the active directory and, optionally, open files and files in your workspace. See [Open Tool Window](#) for more information.
- The [Document Dialogs and Tool Windows](#) displays a list of files in the current project or workspace.

You can open a file by typing a portion of the filename and then selecting a matching file from the list.

- On Windows platforms, you can open a file by double-clicking on it in the Windows Explorer. This assumes that you have associated the SlickEdit application with the extension of that file. See [Setting File Associations](#) for more information.
- The Projects tool window displays a list of the files contained in the projects in this workspace. You can open a file by double-clicking on it in the tree.

Implicitly Opening Files

You open a file implicitly by using information from another window or process rather than explicitly opening the file by name and location.

- SlickEdit's [Symbol Navigation](#) allows you to jump from a symbol to its definition or declaration or from a symbol to a reference.
- The [Preview Tool Window](#) displays the definition for the symbol at the cursor. You can open that file for editing by double-clicking in the Preview window.
- For C and C++, you can quickly open the header or source file that is associated with the current file. From the main menu, select **Document** → **Edit Associated File** (or use the `edit_associated_file` command).
- You can quickly navigate from an error message in the Build tool window to the associated code location. See [Navigating from Build Errors to Source Locations](#).
- You can jump from an item in the Message List to the corresponding code location. See [Message List](#).
- SlickEdit contains a number of search mechanisms that allow you to search based on the contents of the file or part of the filename. For more information, see [Find and Replace](#) and [Finding Files](#).

Recently Opened Files

SlickEdit keeps two lists of recently opened files: a list of recently opened files on the File menu and a list of recently opened workspaces on the Project menu. To manage these lists, select **Tools** → **Options**, expand **Appearance** and select **History**. You can see the history of items recently opened from the File or Project menu. See [History Options](#) for more information.

Finding Files

SlickEdit® provides a Find File feature so you can search for one or more files to open. To access this feature, from the main menu, click **Search** → **Find File** (or use the `find_file` command). The Find File dialog is displayed. [Files Tab](#) for information about using this dialog.

Opening URLs

In addition to opening files on your computer, SlickEdit® lets you open files on the Web for editing. To open a file located at a particular Web address, from the main menu, click **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. The URL will be opened in a new editor window. For a description of this dialog, see [Open URL Dialog](#).

Tip

URLs in SlickEdit editor windows are treated as links. You can open these links in a Web browser by hovering over the link with your mouse pointer and using the binding **Ctrl+Click** (or **Cmd+Click** on the Mac). See [Navigating to URLs](#) for more information.

Inserting Files

The Insert File feature can be used to insert a file at the cursor location in the current buffer. To access this feature, from the main menu, click **File → Insert a File** (or use the **gui_insert_file** command). The Insert File dialog is displayed (similar to the standard Open dialog), which lets you specify the file to insert.

Invoke and Edit

Using your operating system command line, you can invoke SlickEdit® with a file(s) loaded for editing. The syntax is:

```
vs {options} file1 {options} file2
```

If the file that you want to edit is not found, an empty buffer is created with that name.

The table below shows some invocation examples. See [Invocation Options](#) in the Appendix for more options and information.

Invocation Example	Description
vs project1.vpw	Auto Restore from workspace file. If you specify .vpj, SlickEdit Auto Restores the project.
vs project1.sln	Auto Restore from a Visual C++ solution file.
vs autoexec.bat config.sys	Edit two files.
vs "this is.c"	Edit a file with a space character.
vs test.c -#1000	Edit test.c and go to line 1000.
vs orders -#bottom-of-buffer -#/invoice/-	Edit orders file, go to bottom of buffer, and search backward for "invoice".
vs +70 test.exe	Edit binary file test.exe in record width 70.

Invocation Example	Description
vs *.c	Edit all C source files in the current directory.
vs -r list \	Start the Windows file manager and list the entire drive contents.
vs -x rich.sta autoexec.bat	Specify a different state file (<code>rich.sta</code>) and edit the file <code>autoexec.bat</code> .

Options for Opening Files

SlickEdit provides several options related to opening files:

- [Activating Change Directory](#)
- [Setting Global Load Options](#)
 - [Working with Large Files](#)
 - [Different Options for Different Drives](#)
- [Setting Language-Specific Load Options](#)

Activating Change Directory

You can specify that the current directory is changed in the editor when the directory is changed in the Open, Save Copy As, Save As, and Change Directory dialogs. From the main menu, click **Tools** → **Options**, expand **Appearance** and select **General**, then set the option **Change directory to True**.

Setting Global Load Options

SlickEdit® provides many global Load options, such as specifying which file is active after multiple files are loaded at once, expanding tabs to spaces, size limits for files to be loaded, and Auto Reload preferences. These options affect the Open dialog box and any other command that uses the **e** or **edit** commands to open a file. In fact, you can override many of these global Load options by selecting the corresponding option on the Open dialog. To access global Load options, from the main menu, click **Tools** → **Options**, then expand **File Options** and select **Load**. See [Load File Options](#) for information about each option.

Working with Large Files

The default settings are optimized for excellent performance when editing very large files. You will notice that some features such as color coding, undo, and soft wrap are turned off when you edit a large file. Go to **Tools** → **Options** → **Editing** → **General** to configure these options.

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 will not 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 that 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.

Different Options for Different Drives

If you want to specify different global Load options for different disk drives, define an environment variable called **VSLICKLOAD** and specify each drive followed by the appropriate switch. For example, the following command specifies preloading files from jump drives M and N:

```
set VSLICKLOAD= m: +L n: +L
```

Tip

TIP See [Command Line Switches](#) for information about enabling and disabling switches on the SlickEdit® command line.

Setting Language-Specific Load Options

Some options for loading files, such as loading as binary and expanding tabs to spaces, can be set on a language-specific basis. Language-specific File options override all global File options as well as any other options set for that language. To access these, from the main menu, click **Tools** → **Options**. Expand **Languages**, the language category and language, then select **File Options**. See [Language-Specific File Options](#) for more information.

Saving Files

You can use the following methods to save files in SlickEdit®:

- **To save the current file or buffer:** From the main menu, click **File** → **Save** (**Ctrl+S** or **save** command). Alternately, right-right click on a file tab and select **Save [file]** from the context menu.
- **To save the current file or buffer with another name:** From the main menu, click **File** → **Save As** (**gui_save_as** command). The standard Save As dialog is displayed. Select the **Keep old file** option if you do not want the buffer name to be changed. See [Save As Dialog](#) for more information.
- **To save the current file or buffer with another name:** From the main menu, click **File** → **Save Copy As** (**gui_save_copy** command). The standard Save Copy As dialog is displayed. See [Save Copy As Dialog](#) for more information.

- **To save all open buffers:** Click **File** → **Save All** (`save_all` command). Alternately, right-click on any file tab and select **Save All** from the context menu.

If a Save operation fails, for example, because the file does not have write permissions, the Save Failed dialog box is displayed, presenting some alternate choices. See [Save Failed Dialog](#) for a description of this dialog.

Options for Saving Files

SlickEdit® provides many options regarding the saving of files:

- [Setting Global Save Options](#)
- [Setting Language-Specific Save Options](#)
- [Setting Backup Options](#)
- [Setting AutoSave Options](#)
- [Setting Files of Type Filter Options](#)

Setting Global Save Options

Global Save options are available to specify actions that automatically occur when a file is saved, such as appending/removing EOF characters, expanding tabs to spaces, and setting the line format. You can override some of these global Save options by selecting the corresponding option on the Save As dialog. To access global Save options, from the main menu, click **Tools** → **Options**, then expand **File Options** and select **Save**. See [Save File Options](#) for information about each option.

Setting Language-Specific Save Options

Some options for saving files, such as saving as binary and expanding tabs to spaces, can be set on a language-specific basis. Language-specific Save options override all global Save options as well as any other options set for that language. To access these, from the main menu, click **Tools** → **Options**. Expand **Languages**, the language category and language, then select **File Options**. See [Language-Specific File Options](#) for more information.

Setting Backup Options

When a file is saved, SlickEdit can automatically create a backup of the file. To enable Backup and configure Backup options, from the main menu, click **Tools** → **Options**, then expand **File Options** and select **Backup**. See [File Backups](#) for more information.

Setting AutoSave Options

SlickEdit® can save temporary versions of files, in the event of power failure or other unforeseen circumstances. To enable AutoSave and set your AutoSave preferences, from the main menu, click **Tools** → **Options**, then expand **File Options** and select **AutoSave**. When AutoSave is enabled, SlickEdit creates temporary files in the `autosave` directory of your user config. Temporary files are only created for modified files and are replaced or deleted when AutoSave runs subsequently. AutoSave files are

deleted when they are no longer needed. See [AutoSave File Options](#) for more information.

Setting Files of Type Filter Options

You can change the file specifications that appear as choices in the drop-down lists for the **Files of type** and **Save as type** fields on the **Open**, **Save Copy As** and **Save As** dialogs, respectively. By default, all files are listed; however, you might want a smaller list that displays only the source files you typically edit. To edit these file filters, from the main menu, click **Tools** → **Options**, then expand **File Options** and select **Files of Type Filters**. See [Files of Type Filter Options](#) for more information.

File Backups

SlickEdit provides two mechanisms for creating backup files:

- Backup History - Creates a backup each time you save a file, forming a version history that you can browse, compare, and restore.
- Single backup on save - Creates a single backup file of a files contents before the last save.

Backup History is on by default in the Pro and Standard editions. The Community Edition does not support Backup History and creates a single backup file of a files contents before the last save. To change how SlickEdit makes backup files, from the main menu select **Tools** → **Options**, expand **File Options** and select **Backup**.

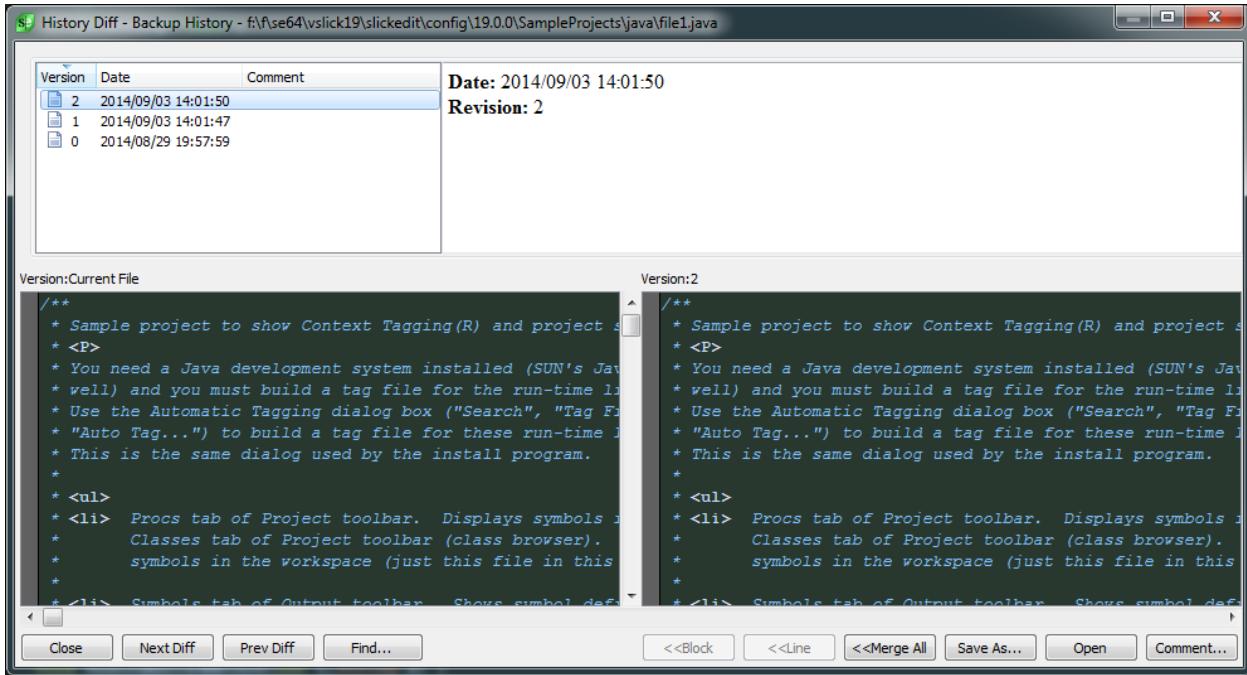
You can set other Backup options such as the location of the backup files, the number of backups to keep for each file, and a limit for the maximum size of a particular backup. For more information see [Backup File Options](#).

Backup History

Backup History maintains a version history for a file, creating a new version each time you save. This creates a more fine-grained version history that bridges the gap between checkins with your source control system. With Backup History you can compare the current version to previous versions, compare two previous versions, or open a previous version for editing. SlickEdit creates and stores deltas to conserve disk space.

To access Backup History for the current file, from the main menu click **File** → **Backup history for**. An icon is also available in the Standard toolbar to open the Backup History.

File Backups



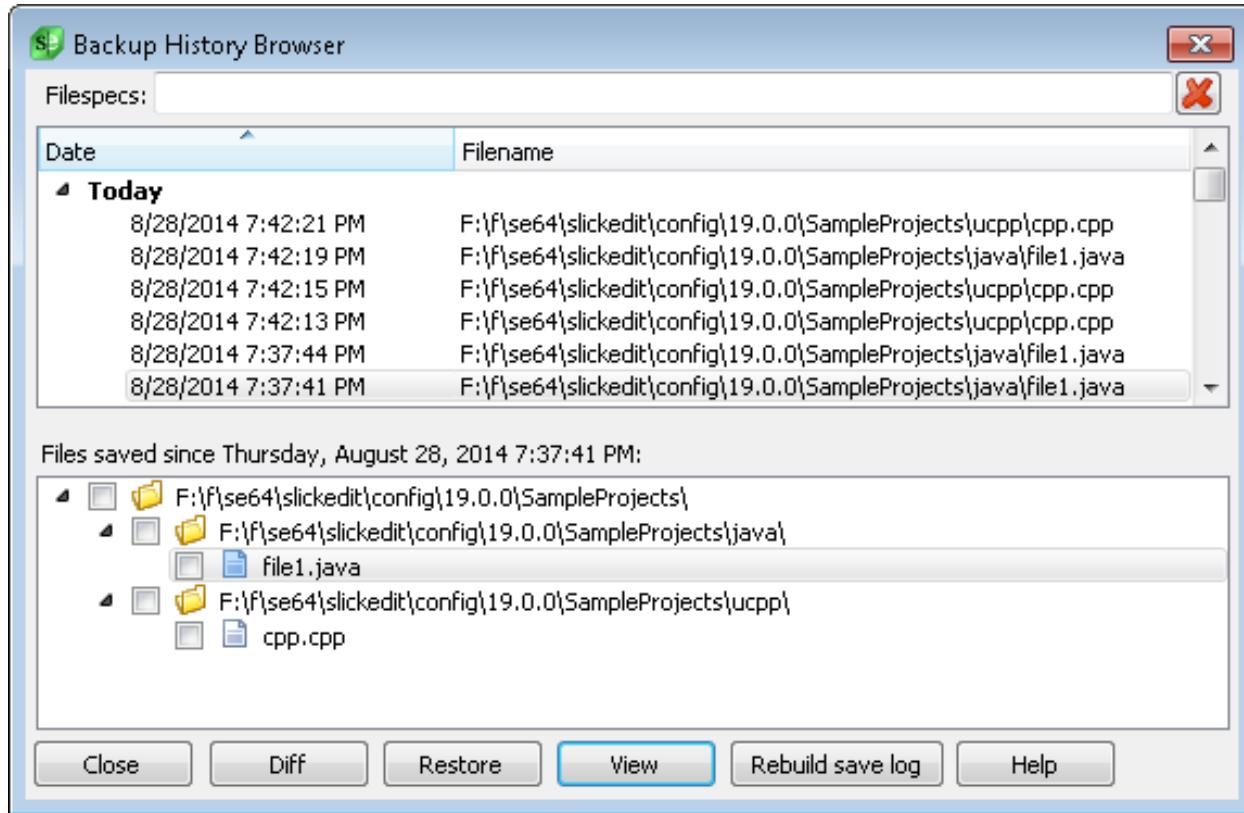
Click through the list of Versions to instantly diff your file against a previous version. To diff adjacent versions and see what changed between versions, right click in the Versions list to bring up the context menu and select the **Compare version X to version Y** menu item. This changes the mode of the dialog to always diff adjacent versions.

The less obvious buttons are as follows:

- **<<Merge All** - Reverts your file to the contents of the version displayed.
- **Save As...** - saves the selected version to the location of your choosing.
- **Open** - opens the selected version for editing. The file is opened as a copy with a number appended to the filename.
- **Comment...** - allows you to enter a comment on a version.

The Backup History Browser (**File → Backup history browser...**) is great for viewing your save history. In addition, it can be used to restore deleted files.

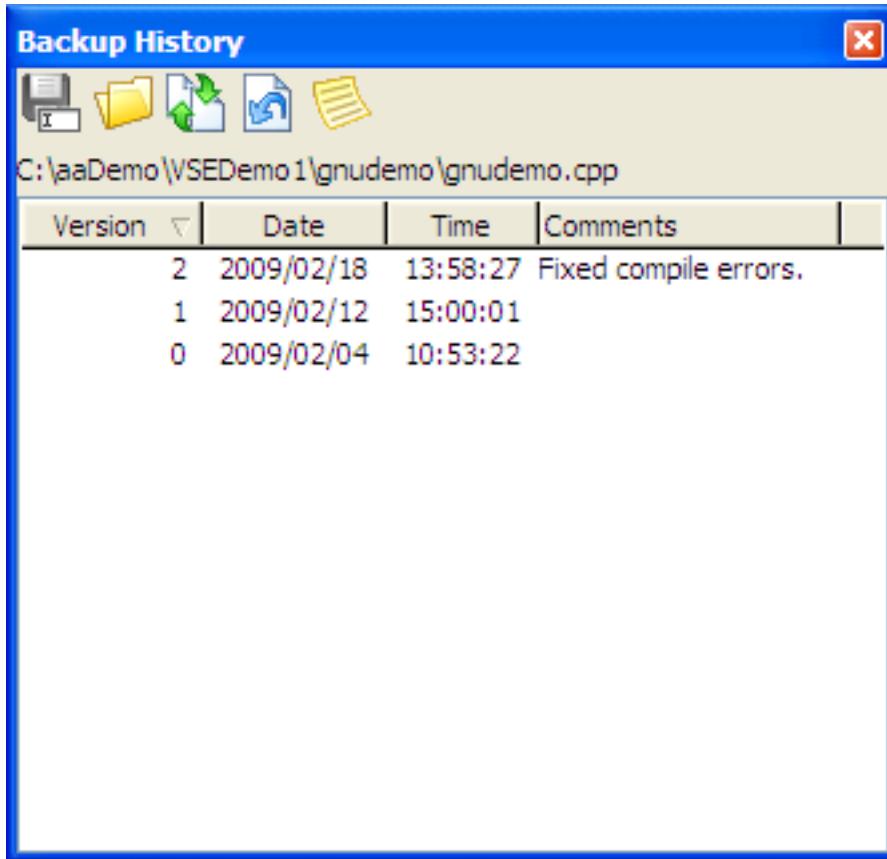
File Backups



The top list displays all saves for which a backup delta is available. Once an item in the top list is chosen, the bottom list is filled in with all files which were saved at or after the date selected. For example, if you choose a save date that was 5 days ago, the lower list will show you all files which you saved since then. This is a great way to know what files you've recently been working on.

- **Diff** - Diffs the selected version against the current version.
- **Restore** - Restores one or more deleted files.
- **View** - Displays the selected file versions.
- **Rebuild save log** - You probably won't need to use this button because there's a lot of automation to walk you through this. When you upgrade and you've been using Backup History, SlickEdit will detect that you are missing the save log file and prompt you to rebuild your save log. If you choose not to build the save log or if some how your log is corrupt/inaccurate, use this button to rebuild it.

There is also a Backup History (**View → Tool Windows → Backup History**) which is shown below.



Once opened, use the context menu or icons to perform the following operations on the Backup History:

- **Save Selected Backup As** - saves the selected version to the location of your choosing.
- **Open Selected Backup** - opens the selected version for editing. The file is opened as a copy with a number appended to the filename.
- **Run Diff on Selected Backup** - compares the selected version to the current version. You can also select two versions and compare them to each other. If only one version is selected, the Backup History dialog is displayed and your file is diffed against the selected version.
- **Revert to Selected Backup** - loads the selected version for editing as the most recent version of this file.
- **Add Comments to Selected Backup** - allows you to enter a comment on a version.
- **View Source Control History** - displays the Backup History dialog for the selected file. Adjacent versions are diffed which will show you what's changed between versions.

For more information on diffing files and using DIFFzilla, see [DIFFzilla®](#).

Closing Files

Files can be closed in SlickEdit® with the methods below. Prior to closing, you will be prompted to save

any changes.

- **To close the current file:** Click **File** → **Close** (F3 or q command), or right-click on the file tab and select **Close [file]** from the context menu. You can also use **Shift+LeftClick** or the middle mouse button on a file tab to close a file, even if the focus is not on the file to be closed. Note that the middle mouse button must be configured to send a Middle Button event.

Tip

The q command is a shortcut for the quit command. Use q (or quit) in the syntax q [file], where [file] is a file or 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:** Click **File** → **Close All** (close_all command) or right-click on the file tab and select **Close All** from the context menu.
- **To close all open files except for the one currently in focus:** Right-click on the file tab and select **Close Others** from the context menu.

See [Opening and Closing Workspaces](#) for information about closing workspaces and projects.

The SlickEdit® 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 change file attributes, copy, move, delete, or back up the selected files.

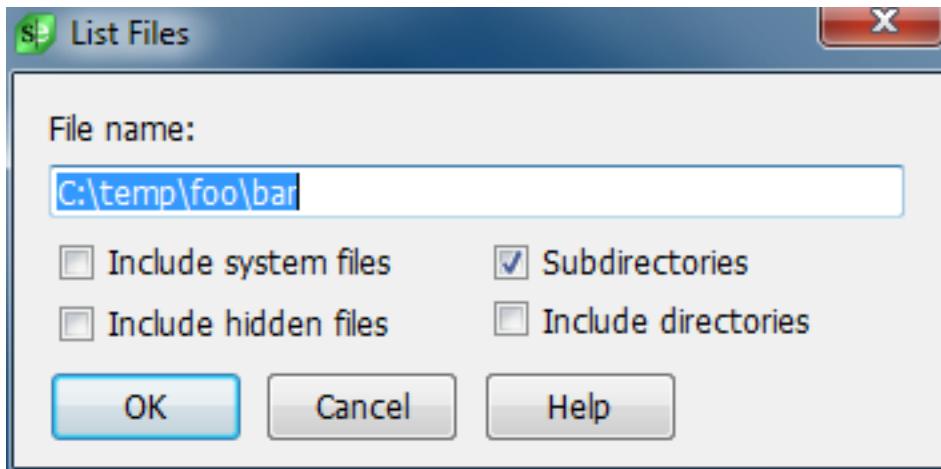
File Manager operations and options can be accessed from the main menu by clicking **File** → **File Manager**.

Topics in this section:

- [Creating a New File List](#)
- [Selecting Files in the File List](#)
- [Operating on Selected Files](#)

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, click **File** → **File Manager** → **New File List**. The List Files dialog is displayed.



The dialog has 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 [Standard Open Dialog](#)).
- **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 [Standard Open Dialog](#)).
- **Subdirectories** - When checked, all subdirectories below each file specified in the **File name** text box are searched.
- **Include directories** - When checked, directories will be included in the listing.

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

You can append files to your current file list. From the main menu, click **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

TIP To modify and close a file list without being prompted to save each time, from the main menu, click **Tools** → **Options** → **Editing** → **General**, then set the **Throw away file lists** option to **True**.

Selecting Files in the File List

To select files in the file list, use the File Manager selection functions. To access these functions, from the main menu, click **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 specific attribute flags that you specify, including Read-Only, Hidden, System, Directory, and Archive. On UNIX systems, you can specify read/write/execute attributes.
- **Extension** - Selects all files with the specified 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 following operations are available on the File Manager menu (**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) Opens the selected files for editing.
- **Select** - Displays a menu of selection commands. See [Selecting Files in the File List](#).
- **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. Enter the extension without the dot character.
 - **Unlist Attribute** - (**unlist_attr** command) Removes files with a specific attribute from the list. Windows attributes include Read-Only, Hidden, System, Directory and Archive. On UNIX systems, you can specify read/write/execute attributes.

- **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 language, use the **Word chars** box on the language-specific **General** options screen (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **General**). See [Language-Specific General Options](#).
 - **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.

Context Tagging® (Pro only)

This chapter describes SlickEdit Context Tagging® system used for symbol analysis, the features that use Context Tagging, and how to manage tag files.

Context Tagging Features (Pro only)

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 Programming Interfaces). Symbol information is updated dynamically 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 (Pro only)

The Context Tagging database allows you to navigate your code, jumping from a symbol to its definition, declaration, or references. For more information, see [Symbol Navigation](#).

List Members (Pro only)

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 **Ctrl+Space** or **Alt+Dot** when the cursor is positioned after the member access character.

Tip

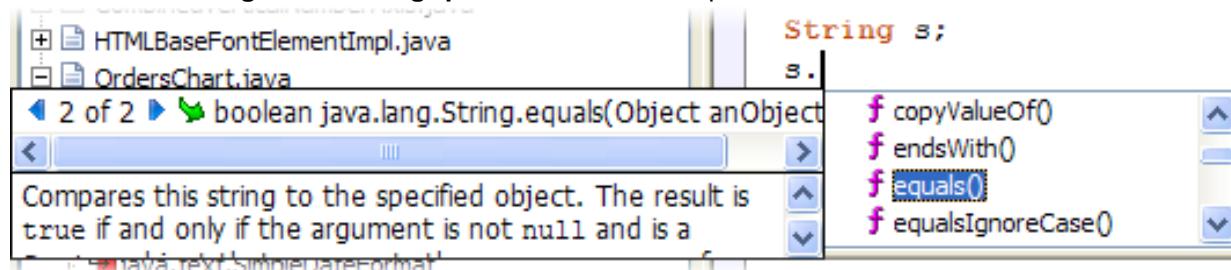
When the cursor is positioned after a member access character, like the dot in "foo.", **Alt+Dot** will display a list of members for that symbol. If the cursor is not positioned after a member access character, this key binding will display a list of symbols visible in the current context.

If you want to disable automatic listing and only list members on demand, turn List Members off, as follows:

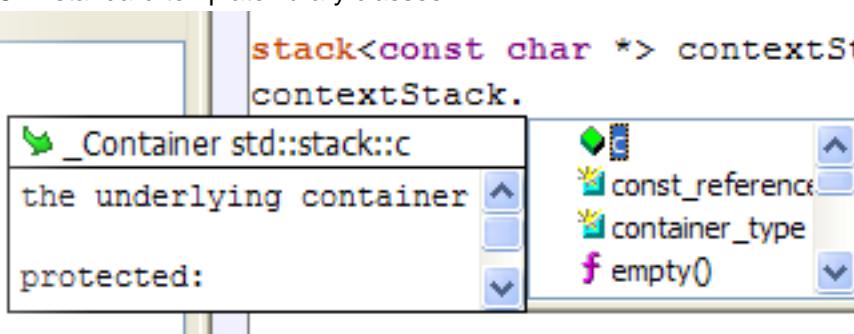
1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Auto-Complete**.

2. Clear the **Auto-list members** check box found in the **List-Symbols options** options group.

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.

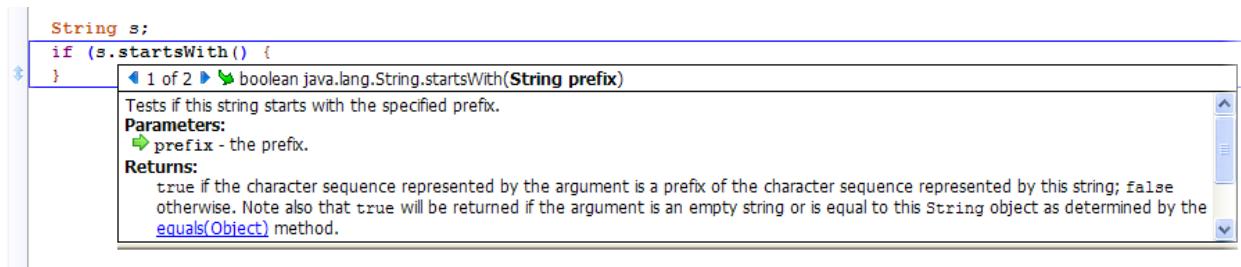


Parameter Information (Pro only)

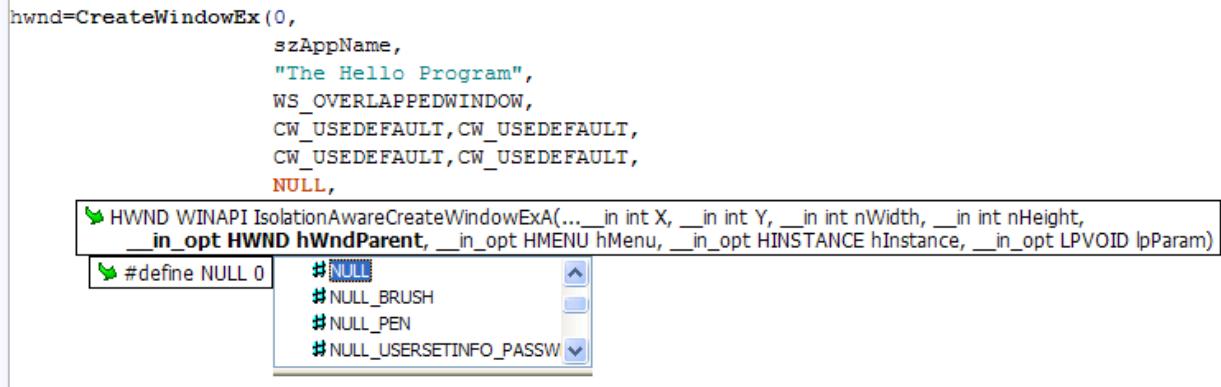
The prototype for a function is automatically displayed when typing a function operator such as the open parenthesis. This also highlights the current argument within the displayed prototype. When working with C++, parameter info is also automatically displayed when typing a template argument operator 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.

Auto List Compatible Parameters (Pro only)



The example below shows the result of pressing **Alt+Comma** inside the argument list of the WIN32 API function **CreateWindowEx**.



Auto List Compatible Parameters (Pro only)

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=<Alt+Comma>**) and when listing members of a class or struct. Keep in mind, not all possible variables and expressions are listed. Press **Alt+Dot** if the symbol that you want is not listed. To access Auto List Compatible Parameters on demand, press **Alt+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, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Context Tagging**.
2. Clear the **Auto-list compatible parameters** check box.

The following example displays the results of pressing **Alt+Comma** after an assignment operator. The **Rect**, **pRect**, and **argv** are not listed because their types do not match.

```
static int CalculateArea(RECT *pRect)
{
    return(pRect->w * pRect->h);
}
void main (int argc, char *argv[])
{
    RECT Rect;
    RECT *pRect=&Rect;
    int iArea;

    iArea=
```



Symbol Information

SlickEdit has several ways to gather and display information about a specific symbol or identifier in your source code.

- **Show info for symbol under mouse** - When you hover the mouse pointer over a symbol, SlickEdit will pull up information about that symbol, including documentation comments and the symbol's return type, and display it in a popup underneath the symbol. See [Preview and Highlighting](#) for more information.
- **Symbol information** - The same information displayed when you hover the mouse pointer over a symbol can be brought up on-demand by pressing **Ctrl+Alt+**, to invoke the **symbol_info_help** command. The popup will remain up until you move the mouse or type a character.
- **Preview tool window** - When active, the Preview tool window automatically displays the definition or declaration of the symbol under the cursor, as well as documentation comments. See [Information Displayed in the Preview Window](#) for more information.
- **Current Context** - The Current Context
<error>Toolbar is not defined.</error>
displays the logical location of the cursor within your code. In addition, the Defs tool window and Class tool window automatically highlight the symbol containing the current line under the cursor.

Completions (Pro only)

Completions save keystrokes as you are typing code by providing a way to automatically complete

partially-typed text. Press **Ctrl+Space** to invoke the SlickEdit® **codehelp_complete(Pro only)** command to automatically complete the rest of the symbol you are currently typing. If a unique match is not found, a list is displayed allowing the selection of the exact match. See [Completions](#) for more information.

Symbol completion can also be used to correct typographical errors such as the following: See [Find completions which fix minor typos](#) for more information.

- **Correcting symbol case errors** - In case-sensitive languages, if you type a symbol with the incorrect case, such as typing "getDictionary" instead of "GetDictionary", symbol completion can look up the correct symbol, and if it is unique, replace it with the correct case.
- **Correcting transposed characters** - If you mistype a symbol and transpose two characters, such as typing "getHlep" instead of "getHelp", symbol completion can look up the symbols visible in the current context, and match them against the symbol under the cursor. If only one unique symbol matches after correcting transposed characters, symbol completion will replace the symbol with the corrected version.
- **Correcting single mistyped character** - If you mistype a single character in a symbol, such as typing "getArrau" instead of "getArray", symbol completion can look up the symbols visible in the current context, and match them against the symbol under the cursor. If only one unique symbol matches after correcting a single character, symbol completion will replace the symbol with the corrected version.
- **Correcting single missing character** - If you miss a single character in a symbol, such as typing "getVectr" instead of "getVector", symbol completion can look up the symbols visible in the current context, and match them against the symbol under the cursor. If only one unique symbol matches after inserting the missing character, symbol completion will replace the symbol with the corrected version.
- **Correcting repeated character** - If you accidentally double type single character in a symbol, such as typing "settValue" instead of "setValue", symbol completion can look up the symbols visible in the current context, and match them against the symbol under the cursor. If only one unique symbol matches after removing a single repeated character, symbol completion will replace the symbol with the corrected version.

In general, symbol completion looks for symbols which start with the same characters as the identifier under the cursor, but it can also be configured to match patterns in order to locate, or complete shorthand for symbols which match an identifier pattern. Symbol completion will attempt to find all the symbols visible in the current context which match the pattern. If only one unique symbol matches, symbol completion will replace the symbol, otherwise a list is displayed allowing the selection of the exact match. See [Subword matching](#) for more information.

Symbol Browsing (Pro only)

SlickEdit® gives you the ability to browse and view symbols in your files or workspaces. There are several tool windows and dialogs 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 clicking **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 clicking **View → Tool Windows → 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 (Pro only)

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, Fortran, SQL, Visual Basic .NET, and many other languages. Along with definitions and declarations, 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**.

CTags Based Tagging Features (Standard only)

CTags Based Tagging is a feature set that integrates SlickEdit with the CTags(1) tool for building and using *tags* databases to locate symbol definitions and perform simple symbol navigation and completion operations.

The ctags program generates an index (or "tag") file for a variety of language objects found in a set of file(s). This tag file allows these items to be quickly and easily located by a text editor or other utility. A "tag" signifies a language object for which an index entry is available (or, alternatively, the index entry created for that object).

The CTags Based Tagging feature set includes:

- [Tag-Driven Navigation](#)
- [Completions](#)

Before you begin working with these features, some configuration is required. See [Building CTags Based Tag Files](#).

Tag-Driven Navigation (Standard only)

The CTags Based Tagging database allows you to navigate your code, jumping from a symbol to its definition. For more information, see [Symbol Navigation](#).

Completions (Standard only)

Completions save keystrokes as you are typing code by providing a way to automatically complete partially-typed text. Press **Ctrl+Space** to invoke the SlickEdit® **codehelp_complete(Pro only)** command, to automatically complete the rest of the symbol you're currently typing. If a unique match is not found, a list is displayed allowing the selection of the exact match. See [Completions](#) for more information.

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.

Tag File Categories (Pro only)

SlickEdit creates 4 kinds of tag files. The "Context Tagging - Tag Files" dialog (**Tools** → **Tag Files**) lists the tag files by category.

- **Workspace and Project tag files** contain the symbols in the files that are part of your workspace. This includes any file that has been added to a project that this workspace contains. Projects can, optionally, have their own tag file. In this case, the files that have been added to that project are not included in the workspace tag file, and the project tag file will also be listed under "Workspace and Project Tag Files". This list will change when you switch workspaces in SlickEdit.
- **Workspace Auto-Updated tag files** contain additional symbols from files that you want to have associated with your workspace, but are not technically part of the workspace. This is useful for things like third-party libraries that you use only with specific workspaces. The order of these tag files is significant because it defines the order that the tag files are searched by tagging.

Workspace Auto-Updated 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. These files are listed under "Workspace Auto-Updated Tag Files".

- **Compiler-specific tag files** are used only when the specified compiler is selected in Project Properties for the current project. The compiler tag files are listed with the name of the language in quotes followed by "Compiler Configuration Tag Files".

Compiler-specific tag files have a read-only checkboxes to indicate which tag file is currently active for the current workspace and project configuration.

- **Language-specific tag files** are used anytime you code in a particular language. This is useful for things like third-party libraries that you use a lot. These tag files are listed by language. For example, the C/C++ library tag files are listed under "C/C++" Tag Files'.

Language-specific tag files have checkboxes which can be turned off in order to keep a tag file in the list, but ignore it. This makes it easy to de-activate and re-activate a tag file that you need on occasion, but do not want to always use for the current language.

Note

For languages that have compiler-specific tag files, for example, C/C++ and Java, the language-specific tag files category is added immediately below the compiler-specific tag files category.

Building Tag Files (Pro only)

Each kind of tag file is built differently. Refer to the following sections for how to build each kind.

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.

Creating Tag Files for Workspace Files (Pro only)

Tag files for your workspace files are automatically created and updated as you edit. If you edit a source file with a different editor, you will have to retag the file or workspace to make sure that the symbol information is up to date.

To retag your workspace, do one of the following:

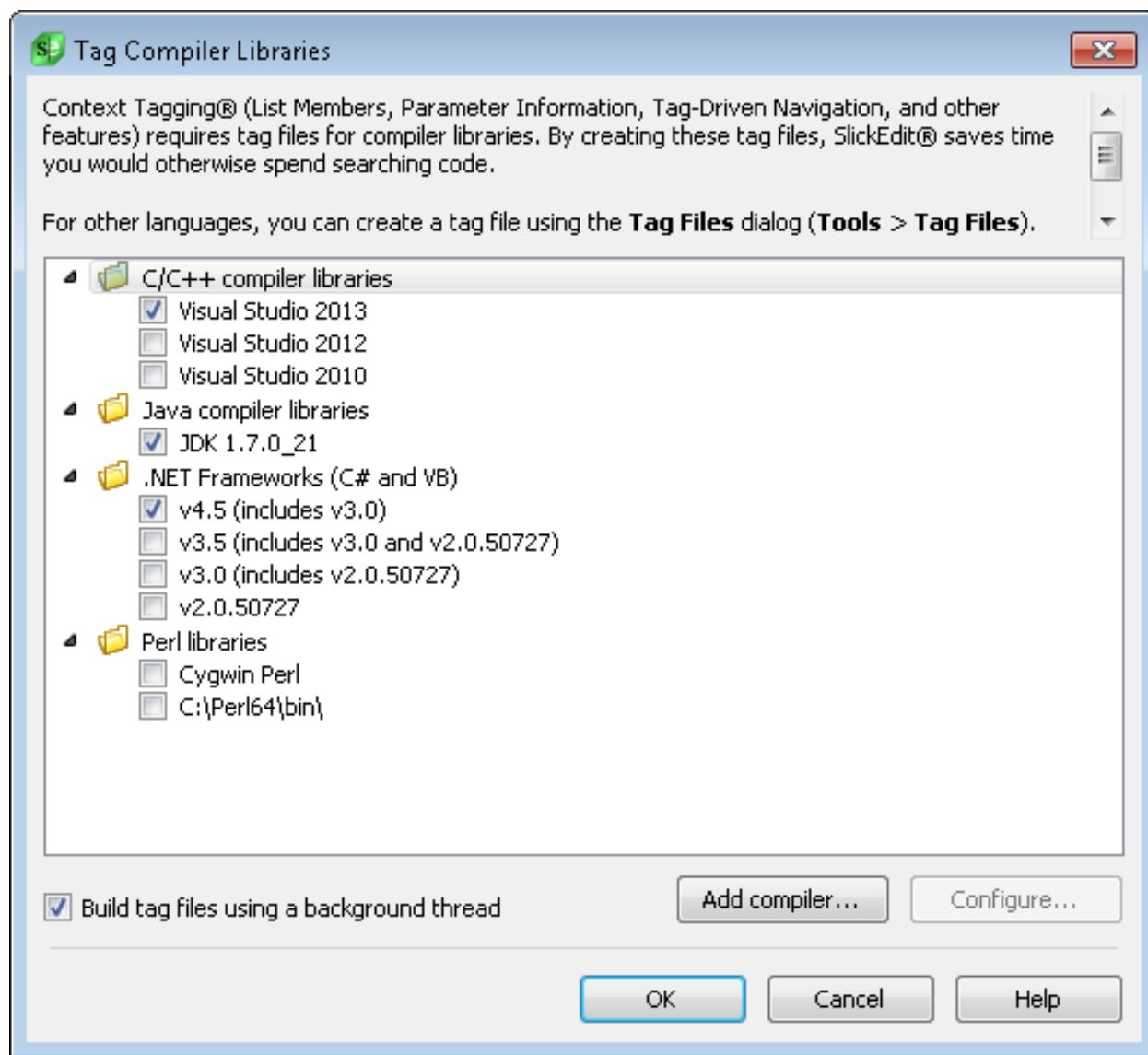
- Use the **Projects** tool window - right-click on the root workspace node and select **Retag Workspace**. This action will also rebuild any project-specific tag files.
- From the main menu, click **Project** → **Retag workspace**. This action will also rebuild any project-specific tag files.
- Use the **Context Tagging - Tag Files** dialog - select **Tools** → **Tag Files** from the main menu. Select the first tag file listed under **Workspace and Project Tag Files** and click the **Rebuild Tag File** button.

To retag the files in a project, do one of the following:

- Use the **Projects** tool window - right-click on the project node and select **Retag Project**.
- From the main menu, click **Project** → **Retag project**. This action will only retag the files in the current project.
- Use the **Context Tagging - Tag Files** dialog - select **Tools** → **Tag Files** from the main menu. Select the project tag file listed under **Workspace and Project Tag Files** and click the **Rebuild Tag File** button.

Creating Tag Files for Compiler-Specific Libraries (Pro only)

The **Tag Compiler Libraries** dialog is used to tag libraries associated with the most commonly used languages in SlickEdit. This dialog appears as part of the Quick Start Configuration Wizard, 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 from the [Context Tagging - Tag Files Dialog](#) (select **Tools** → **Tag Files**, then click **Auto Tag**).

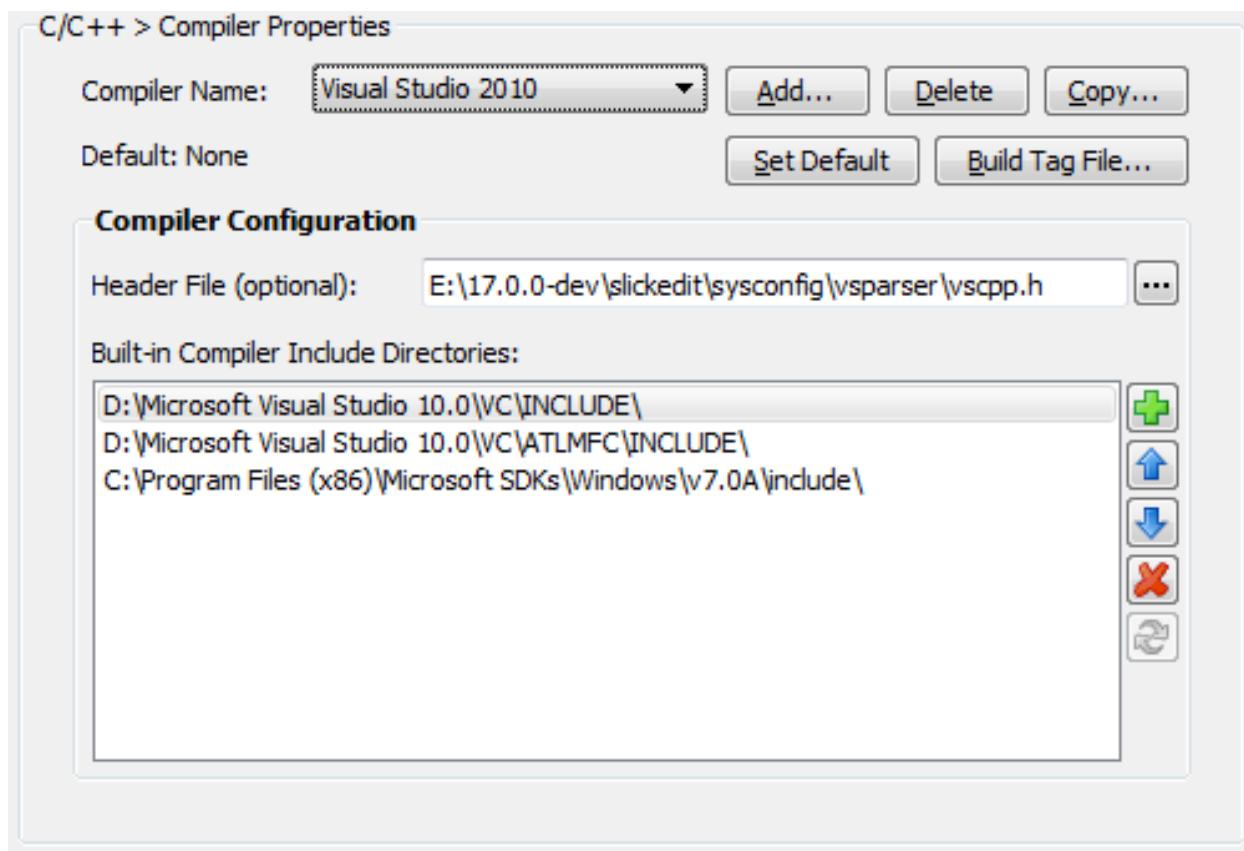


To create tag files for the languages listed, select the packages you want to build. If you want to have the tag files built in the background, select **Build tag files using a background thread**. Click **OK** to begin. If you have chosen to build your tag files in the foreground, then the Building Tag Files dialog box opens, showing the progress as the tag file is built.

If you have chosen to build in the background, the progress dialog shows the progress of queuing files for background tagging. You can then continue to edit code while your files are being tagged. To inform you of the progress of this task, an icon is displayed in the Alert area. While background tagging is being performed, the icon is highlighted.

You can configure some compilers by selecting them in the tree and then clicking the **Configure** button. This will open the Compiler Properties dialog for that language.

For languages not listed on the "Create Tag Files for Compiler Libraries" dialog, you can create language-specific tag files (see [Configuring Other Languages](#)).



In the Compiler Properties dialog, do the following:

1. Click **Add** to enter the name of the compiler you are configuring.
2. Click **Set Default** if this is the main compiler you use for this language.
3. Click the ... button (ellipses) next to the "Built-in Compiler Include Directories" field to specify an include directory. SlickEdit will tag all files in that directory and any subdirectories.
4. Click **Build Tag File** to build the tag file for this compiler.
5. Click **OK** to finish.

Creating Language-Specific Tag Files (Pro only)

Language-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.

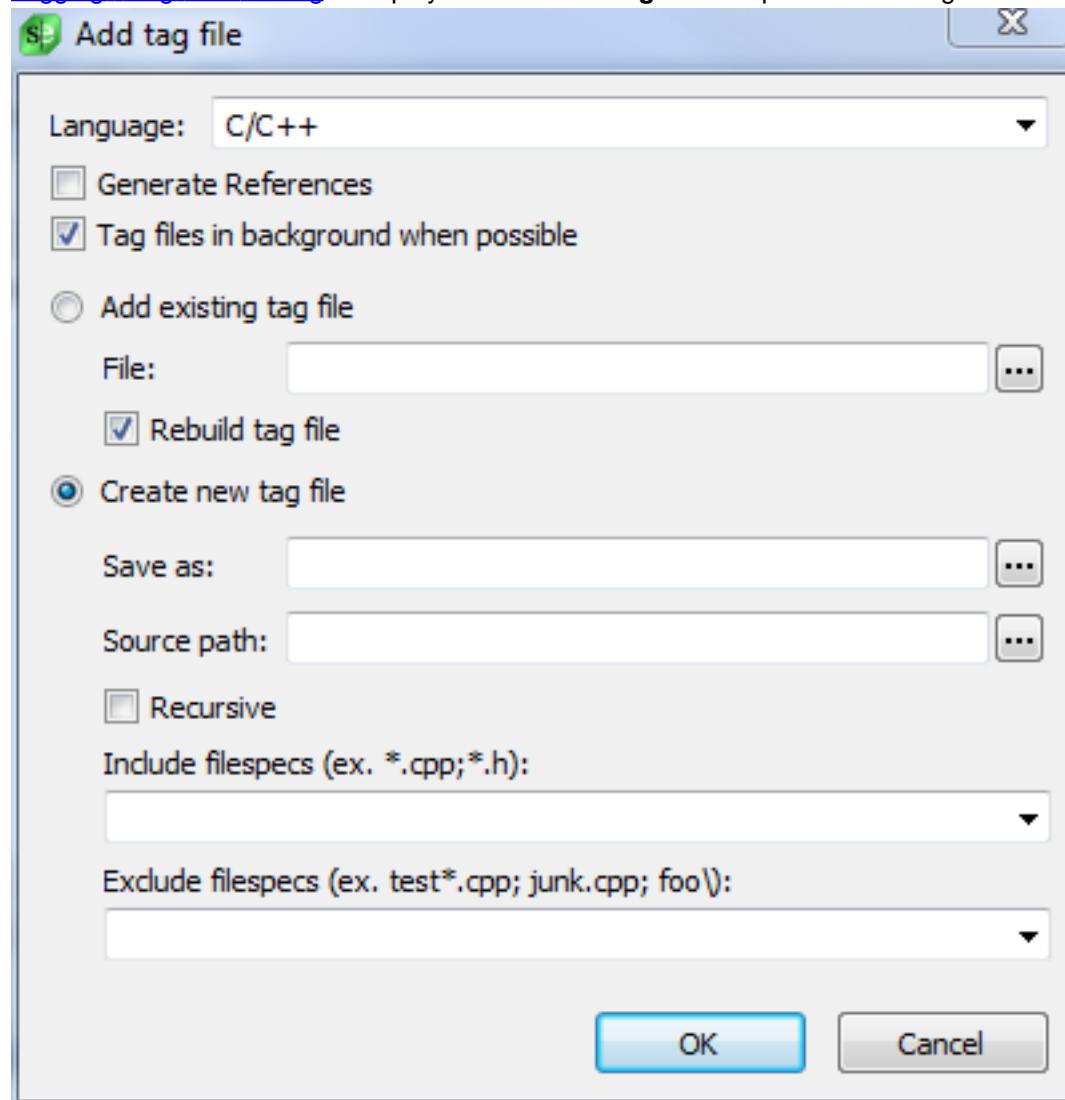
You should create a language-specific tag file for any library that is not a compiler-specific library or part of the codebase you are editing. For example, you may have local libraries that are reused from project to project.

Note

Language-specific tag files are used by all projects using that language. If you have a library that is used by one project and not another, the symbols in that library will show up as completions in both projects.

You should consider using workspace-specific auto-updated tag files for libraries that are only used by certain workspaces. This can also be helpful when you need to work with multiple versions of the same third-party library. See [Workspace Auto-Updated Tag Files](#) for more information.

To create a language-specific tag file, select **Tools** → **Tag Files** from the main menu. The [Context Tagging - Tag Files Dialog](#) is displayed. Click **Add Tag File** to open the Add Tag File dialog.



The Add Tag File dialog has the following fields:

- **Language** - Select the language type into which you want the tag file inserted.

- **Generate References** - Select this 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 your 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.

- **Tag files in background when possible** - Check this option to use background tagging when possible.
- **Add existing tag file** - If you are adding an existing tag file, select this option, which will enable the following fields:
 - **File** - The path to your tag file.
 - **Rebuild tag file** - Check this option to go ahead and rebuild the tag file when adding it.
- **Create new tag file** - Select this option to create a new tag file. These additional options will be enabled:
 - **Save as** - Where to save the new tag file. Give it a name that is representative of the library being tagged. For example, if you are tagging the Boost library, you would name the file "Boost.vtg". Tag files are required to have the extension **.vtg**.
 - **Source path** - The path to the directory from which to include files.
 - **Recursive** - If checked, the selected directory will be searched recursively.
 - **Include filespecs** - The **Include Filespecs** combo box lets you select from predefined wildcard specifications or you can type your own. Each file spec should be separated with semicolons. For example, to include only Java files, select ***.java** from the predefined list. To include all files in a directory, type the wildcard *****. To customize the items in this list, see the [Files of Type Filter Options](#).
 - **Exclude filespecs** - Use this combo box to exclude paths, files, or file types from the specified directory using ant-like wildcards. To specify multiple patterns, separate them with semicolons. No files are searched in a path that is excluded, including any files in sub-directories beneath. For examples, see [Exclusion Examples](#).

After filling in the fields, click OK to build your tag file. 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.

Configuring Context Tagging for COBOL (Pro only)

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.

Configuring Context Tagging for Other Languages (Pro only)

For languages other than C/C++, Java, or .Net, you can create language-specific tag files for the standard libraries that are part of those languages.

A tag file is automatically built for the run-time libraries of C#, Unity, JavaScript, Perl, PV-WAVE, Slick-C®, TCL, Visual Basic .NET, and InstallShield 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.

Building CTags Based Tag Files (Standard only)

In order for CTags tag navigation and completion to work, you must create a workspace and project (see [Creating Projects](#)). You can either generate the CTags tag file yourself or use SlickEdit to generate the *ctags* tag file for your workspace files (see [Creating Tag Files for Workspace Files](#)).

At this time, SlickEdit can handle generic *ctags*, tag files, but not emacs-style (exuberant) *etags* tag files. If you are building tag files manually, it is recommended to you use the standard style.

SlickEdit will look for a CTags based tag file in one of several locations. It will only use the first tag file found.

- First SlickEdit will look for a file in the current project directory with the project's name and the extension ".tags". Next it will look for a file named "tags" in the current project directory.
- Next it will look for a file in the workspace directory with the workspace name and the extension ".tags". Next it will look for a file named "tags" in the workspace directory.
- Finally it will look for a file named "tags" in the current directory.

Creating Tag Files for Workspace Files (Standard only)

Tag files for your workspace files can be manually created and will be automatically used by SlickEdit for symbol navigation and completion. You can either build the CTags tag files from within SlickEdit or build them outside of SlickEdit using the *ctags* tool directly.

To retag your workspace, do the following:

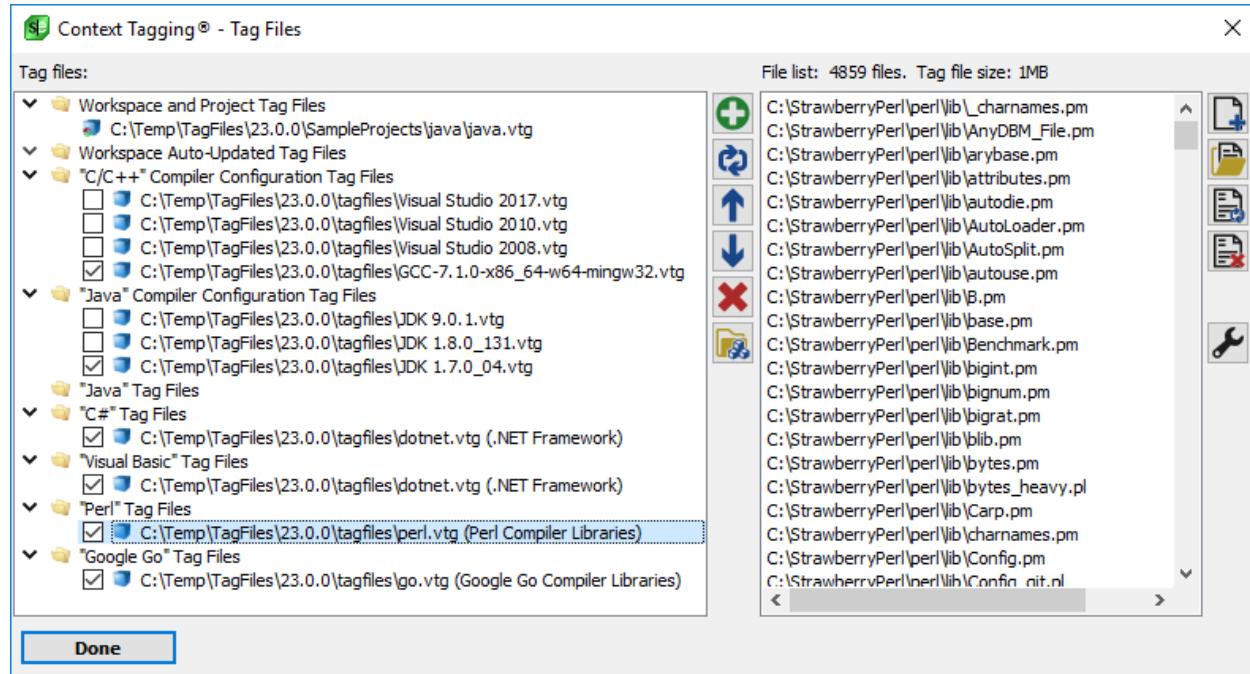
- The **ctags** executable must be in your PATH or at the location configured in options **Tools** → **Options** → **Tools** → **CTags Tagging**).
- Use the **Projects** tool window - right-click on the root workspace node and select **Retag Workspace**.
- From the main menu, click **Project** → **Retag workspace**. This action will also rebuild any project-specific tag files.

To retag the files in a project, do one of the following:

- Use the **Projects** tool window - right-click on the project node and select **Retag Project**.
- From the main menu, click **Project** → **Retag project**. This action will only retag the files in the current project.

Managing Tag Files (Pro only)

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** bitmap 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 Search Order (Pro only)

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: 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. Local variables in the current function or symbol, and other symbols in the current source file are searched first.
2. Workspace tag file, providing it contains other C/C++ source files.
3. Project tag files, providing they contain other C/C++ source files. The order that project tag files are searched is not defined.
4. Auto-updated tag files containing other C/C++ source files.
5. The "C" Compiler Configuration tag file corresponding to your default C compiler configuration as specified in your project (see [C/C++ Compiler Settings](#)), or global default.
6. Language-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.

Example: Java Tag File Search Order

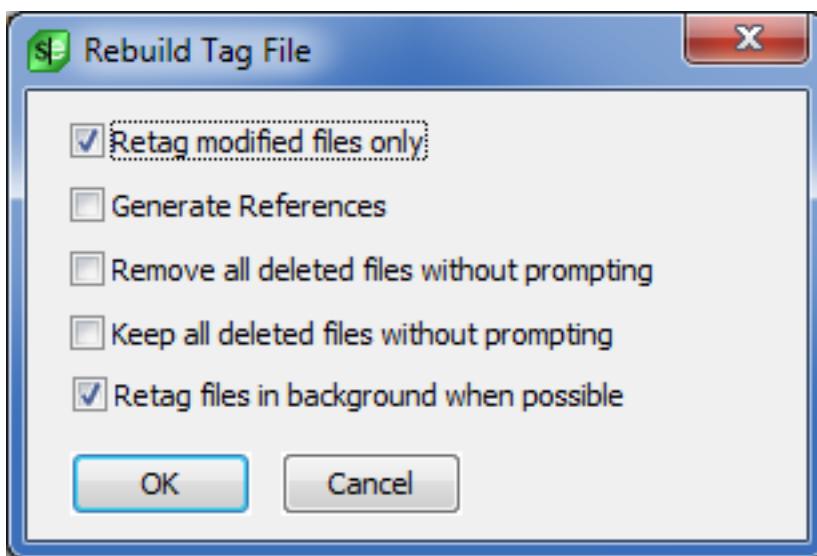
If a Java source file is open, when a tagging-related operation is performed, the tag files are searched in the following order:

1. Local variables in the current function or symbol, and other symbols in the current source file are searched first.
2. Workspace tag file, providing it contains other Java source files.
3. Project tag files, providing they contain other Java source files. The order that project tag files are searched is not defined.
4. Auto-updated tag files, containing other Java source files.
5. The "Java" Compiler Configuration tag file corresponding to your default Java compiler configuration as specified in your project (see [Java Compiler Properties Dialog](#)), or global default.
6. Language-specific Java tag files, in the order that they are listed in the [Context Tagging - Tag Files Dialog](#).

Rebuilding Tag Files (Pro only)

The Rebuild Tag File dialog box contains options for rebuilding the selected file. To display the Rebuild Tag File dialog, click select **Tools** → **Tag Files**. When the [Context Tagging - Tag Files Dialog](#) is displayed, select a file to rebuild, then click **Rebuild Tag File**.

Context Tagging® Options (Pro only)



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. This checkbox is not present when rebuilding the workspace tag file and project tag files since the list of files in the workspace's projects determine what files should be tagged.
- **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. This checkbox is not present when rebuilding the workspace tag file and project tag files since the list of files in the workspace's projects determine what files should be tagged.
- **Retag files in background when possible** - If checked the tag file is rebuilt in the background if background tagging is supported for these files.

Note

The options **Remove all deleted files without prompting** and **Keep all deleted files without prompting** are mutually exclusive. Selecting one will clear the other.

Context Tagging® Options (Pro only)

General Context Tagging® Options

Options are available for setting general parameters for the Context Tagging feature set. You can designate how tagging is done, how references function within the application, and tune the application to maximize performance. To display the options, from the main menu, select **Tools** → **Options** → **Editing** → **Context Tagging**. See [Context Tagging® Options](#) for descriptions of the options.

Tip

To improve tagging performance, you may need to adjust the tag file cache size (**Tools** → **Options** → **Application Options** → **Virtual Memory**). See [Virtual Memory Options](#) for more information.

Language-Specific Context Tagging® Options (Pro only)

You can activate and deactivate various Context Tagging features on a per-language basis. To access these options, from the main menu, select **Tools** → **Options** → **Languages**, expand your language category and language, then select **Context Tagging®**. See [Language-Specific Context Tagging® Options](#) for more information.

Building, Running, and Debugging (Pro only)

This chapter contains the following topics:

- [Building and Compiling](#)
- [Running and Debugging](#)
- [Working with Google Web Toolkit Projects](#)
- [Working with Android Projects](#)

Building and Compiling (Pro only)

Project Configurations in Builds (Pro only)

Each project may have a number of configurations defined. See [Project Configurations](#) for more information on creating and managing Project Configurations. The active configuration is used during the build process to determine the project settings to use. The manner in which the configuration affects a build depends on which build system you are using.

The build system is specified by selecting **Project → Project Properties** from the main menu and then selecting the Build tab. The first and third options, **Build without a makefile** and **Build with an auto-generated, auto-maintained makefile**, use the SlickEdit® build system. The second option, **Build with a user-maintained makefile or custom build command**, allows you to use a custom makefile or configure build commands on the Tools tab.

- If you use the SlickEdit build system, SlickEdit will use the project properties associated with the currently active configuration. It will direct the build output to a directory with the same name as the project configuration. For example, if Debug is active, SlickEdit will direct the build output to a directory, named "Debug", inside the project directory. You can specify an output directory for a configuration by selecting **Build → Configurations** from the main menu, and then enter a directory in the **Object directory** field.
- If you are using custom build commands on the Tools tab, you can use the **%b** (current configuration) or **%bd** (object directory) escape sequences to implement configuration-specific build behaviors.
- If you are using a custom makefile, you can define a macro, such as CFG, which represents the configuration you want to build. Add code to the makefile to check for this macro and perform different statements, like choosing different compile options or 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

Note

SlickEdit uses the vsbuild utility for all 3 build methods. Even when you build using a custom build command or makefile, SlickEdit uses vsbuild as a wrapper to set up the environment and to determine when the build has completed.

Using Build and Compile Operations (Pro only)

SlickEdit® provides the capability to build a project or compile single files.

- To build the active project, click **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, click **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. Click **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 language-specific **Single File Projects** options screen (see [Defining Language-Specific Projects](#)). 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 language-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 click **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, click **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 allow 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 click **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 unavailable 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**.

Escape Sequences for Build Commands

The following escape sequences may be used when creating build commands using the [Tools Tab](#) on the [Project Properties Dialog](#).

Sequence	Expands to
%B	Configuration
%BD	Configuration build directory. Escapes in the return value are expanded.
%BN	Configuration name. Same as %B option except for Visual C++ configuration names where configuration names are of the form CFG="[ConfigName]" or [ConfigName][[Platform]]

Using Build and Compile Operations (Pro only)

Sequence	Expands to
%C	Current word
%CP	Java class path including -classpath . Escapes in the return value are expanded.
%DEFD	Configuration defines with dashes. Escapes in the return value are expanded. Example: %DEFD, project def = 'test' produces "-Dtest"
%DEFS	Configuration defines with slashes. Escapes in the return value are expanded. Example: %DEFS, project def = 'test' produces "/Dtest"
%DM	The file name only of the current buffer
%E	File extension with dot
%EXE	On Windows, returns ".exe". Returns empty string for other platforms.
%F	Absolute filename
%H	(Java only) Builds a temp HTML file to run the compiled applet, %H is replaced by the temp HTML file name
%I	Absolute include directories (individually listed) including '-i'. Escapes in the return value are expanded. Example: '-ic:\folder1 -ic:\folder2'
%IR	Relative include directories (to the project) including '-I', separated by semicolons (colons on UNIX). Escapes in the return value are expanded. Example: '-Ic:\folder1;c:\folder2'
%IN	Absolute include directories (individually listed) including '-i'. Escapes in the return value are expanded. Example: '-i c:\folder1 -i c:\folder2'
%JBD	Java build directory including -d. Escapes in the return value are expanded.
%L	Command line. Currently not available to vsbuild.

Using Build and Compile Operations (Pro only)

Sequence	Expands to
%LF	Current buffer name
%LIBS	Libraries space delimited. Escapes in the return value are expanded.
%N	Filename without extension or path
%O	Output filename. Currently only GNU projects have an output filename on the Link Tab. Escapes in the return value are expanded.
%OBJS	Project objects (including libraries). Escapes in the return value are expanded.
%OE	Output extension with dot. Escapes in the return value are expanded.
%ON	Output filename with no extension or path. Escapes in the return value are expanded.
%OP	Output path. Escapes in the return value are expanded.
%P	Path of current file
%R	Absolute project name
%RE	Project extension
%RM	Project display name (for associated workspaces)
%RN	Project filename without extension or path
%RP	Project path
%RV	(Windows only) Project drive with :
%RW	Project working directory
%T	Project configuration target
%V	(Windows Only) Drive of current file with :
%W	Absolute workspace filename

Using Build and Compile Operations (Pro only)

Sequence	Expands to
%WE	Workspace extension with dot
%WN	Workspace filename with no extension or path
%WP	Workspace path
%WV or %WD	Workspace drive with :
%WX	The workspace folder name only. Example: %WX, workspace = 'c:\a\b\c\workspace.vpw' produces 'c'
%XUP	Translate all back slashes that follow to forward slashes (UNIX file separator)
%XWP	Translate all forward slashes to back slashes (Windows file separator)
%-#	Removes the previous # characters
%#	The # item in argline (items are separated by spaces)
%{ * }	A list of project files matching the pattern in braces
b%[regkey]	Value of Windows registry entry. Example: %[HKLM:\Software\Microsoft\Communicator@InstallationDirectory]
b%`shell command`	Execute a shell command and take the value from standard output. Example: %`readlink symlinkName`
%(envvar)	Value of environment variable envvar. Escapes in the return value are expanded.
%(env envvar)	This alternate syntax can be used to guarantee that the contents of an environment variable is returned. Right now, there aren't many conflicts. Note that expansion is done on ALL arguments to %(word args) syntax commands. This means you need to use %% if an environment variable name has a % in it. Escapes in the return value are expanded.
%(macro functionName arg1_args)	Not supported in vsbuild. Calls a macro function

Language-Specific Build Methods (Pro only)

Sequence	Expands to
	with one argument (<code>arg1_args</code>) if there are any. Any return value is included in the build command. <code>functionName</code> and <code>arg1_args</code> are expanded before parsed. Parenthesis must match. Example: %({macro my_function %(PATH)}) , where <code>_str my_function(_str path)</code> is a macro function
%({last-path-part count pathSpec})	Return one path part. Starts from end of <code>pathSpec</code> where <code>count=0</code> is the name without path, <code>count=1</code> is first path part before name, <code>count=2</code> is path before that, etc. Example: %({last-path-part 1 c:\a\b\c\d\test.txt}) produces 'd'. Example: %({last-path-part 2 c:\a\b\c\d\test.txt}) produces 'c'.
%({last-path count pathSpec})	Return number of path parts specified. Starts from end of <code>pathSpec</code> where <code>count=0</code> returns name without path, <code>count=1</code> returns first path part before name and name, <code>count=2</code> returns is first and second path parts before name and name, etc. Example: %({last-path 1 c:\a\b\c\d\test.txt}) produces 'd\test.txt'. Example: %({last-path 2 c:\a\b\c\d\test.txt}) produces 'c\d\test.txt'.
%({prompt prompt-text[: initial_value]})	Not supported by vsbuild. Prompts the user for a value. Returns user input. <code>prompt-text</code> and <code>initial_value</code> is expanded before being parsed. Parenthesis must match. Example: %({prompt Text:initial value}) , will prompt the user with the text 'Prompt text' with 'initial value' in the text box.
%({last-prompt-result})	Not supported by vsbuild. Returns result from last %({prompt Text:initial value}) .
%({open-paren})	Returns '(' . Intended for use inside a parenthesized expression which would otherwise have mismatched parenthesis.
%({close-paren})	Returns ')'. Intended for use inside a parenthesized expression which would otherwise have mismatched parenthesis.
%%	Percent character

Language-Specific Build Methods (Pro only)

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 use the 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 use the 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 use the 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-fMakefileCFG=ConfigName.

Cygwin: Using GNU C/C++ 'alternatives' system

On Cygwin, with version 4 of GNU C/C++, "gcc" and "g++" are symbolic links to one of the version-specific executables: "gcc-3", "gcc-4", "g++-3", and "g++-4". A proprietary system called "alternatives" is used to link the unversioned commands to the version-specific ones.

This example shows how to configure which version is used:

```
$ /usr/sbin/alternatives --config g++
```

```
There are 2 programs which provide 'g++'.
```

Selection	Command
*	/usr/bin/g++-4.exe
+	/usr/bin/g++-3.exe

```
Enter to keep the current selection[+], or type selection number: 1
```

SlickEdit uses the information from the alternatives system to run the specified version of the compiler. This allows you to use the same build command within SlickEdit as you do from the Cygwin shell. If you don't want to control this using the alternatives system, you can configure the build system to use "gcc-3", "gcc-4", etc. Select **Build → GNU C Options** then set the **Compiler** field on the **Compile** tab and the **Linker** field on the **Link** tab.

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, click **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, click **Project → Set Active Project**.
5. Select the project that you want to use.
6. Click **Build → Set Active Configuration**.
7. Select the style that you want.
8. Click **Build → Build**.
9. The project is then built, and you can work with your project.

Build Methods for Ant and NAnt

SlickEdit® supports Apache Ant XML build files and NAnt build files. Apache Ant is a popular make facility used to build Java components. NAnt is a .NET build tool that is similar to Ant.

Note

Ant build files must end with the `.xml` extension in order to be recognized as build files. NAnt build files must end with the `.build` extension.

You must use SlickEdit projects for your Ant/NAnt files in order to access the commands that invoke build file targets.

When you open an Ant XML or NAnt build file, SlickEdit automatically either opens the project if it already exists, or creates a new project and adds the file to it.

- **To open an Ant XML file** - From the main menu, click **Project** → **Open Other Workspace** → **Ant XML Build File**, or use the **workspace_open_ant** command.
- **To open an NAnt build file** - From the main menu, click **Project** → **Open Other Workspace** → **NAnt .build file**, or use the **workspace_open_nant** command.

Alternately, you can manually create a project and add the build files or add the files to an existing project.

When adding build files to a project, they are scanned for callable targets. If any targets are found in the file, the icon in the project tree is changed to the "bull's eye" icon.

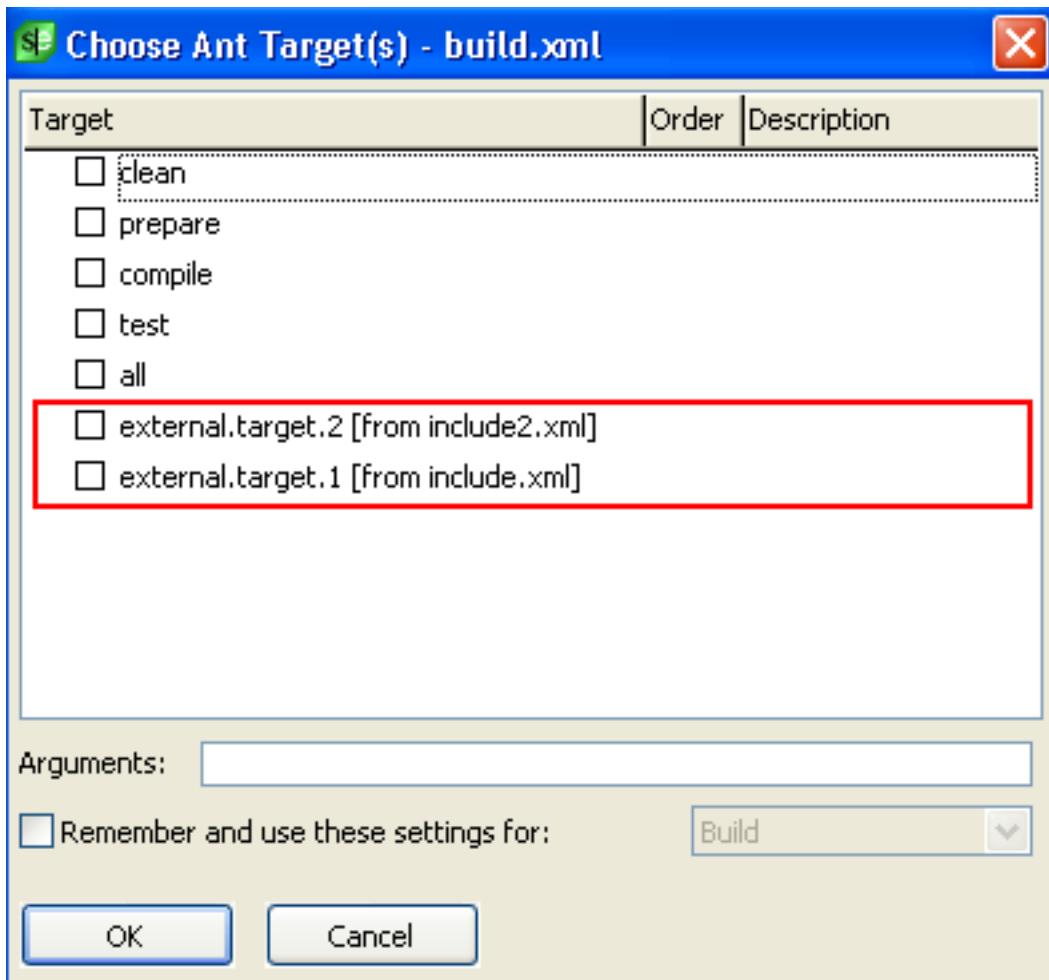
See [Creating Projects](#) and [Adding and Removing Files](#) for more information about creating projects and adding files to them.

Invoking Ant or NAnt Targets

Once you have a project that contains Ant or NAnt files, you can execute Ant or NAnt targets. The SlickEdit commands that invoke the build file targets are only available from the **Build** menu and the right-click context menu of the tree in the Projects tool window.

To execute a single target, pick the target menu item. For example, for Ant, from the main menu, click **Build** → **Execute Ant Target**, then navigate to the target. To specify arguments or execute multiple targets, use the Select Multiple Targets menu item. For example, for Ant, from the main menu, click **Build** → **Execute Ant Target** → **Select Multiple Targets**. Alternately, you can right-click on an Ant or NAnt project in the Projects tool window and execute one target or multiple targets.

The **Execute Single Ant Target** menu and the **Choose Ant Target(s)** dialog will display targets which are imported into the selected build file from other Ant files. This behavior can be turned off at **Tools** → **Options** → **Languages** → **XML/Text Languages** → **Ant** → **Options**.



Setting Shortcuts for Build and Rebuild

To set up the **Build** → **Build** menu items or **Build** → **Rebuild** menu items or both to invoke a specific set of targets, first select one of the target menu items:

- For Ant XML files, from the main menu, click **Build** → **Execute Ant Target** → **Select Multiple Targets**, and choose **Ant XML File**.
- For NAnt build files, click **Build** → **Execute NAnt target** → **Select Multiple Targets**, and choose **NAnt .build file**.

Then, complete the following steps:

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**.

Working with Build Errors (Pro only)

One key advantage of building within SlickEdit is the ability to jump from an error message to the location in the code associated with that error. This makes it much faster to find and fix problems after doing a build.

Viewing Errors

SlickEdit provides a variety of mechanisms to display errors from a build, including:

- Error markers are placed in the margin of the editor window.
- The Build tool window displays the output from the build process.
- The Message List tool window displays a list of errors and warnings parsed from the build output.
- **list_errors** will display the output from a build in a pop-up window. This is useful if you have configured a build command and configured it not to output to the Build window.

For information about how to jump from an error to the source code, see [Navigating from Build Errors to Source Locations](#).

Viewing Errors in the Editor Window

Error markers are displayed as red X bitmaps in the left margin of the editor window after a build or compile is completed. To clear these markers, fix the errors and rebuild. You can also clear the markers by selecting **Build → Clear All Error Markers**, from the main menu, or by using the **clear_all_error_markers** command.

You can move from one error to the next using **next-error** and **prev-error**. These commands determine the next and previous error based on their position in the build output and the current error, marked with a green triangle. See [Navigating from Build Errors to Source Locations](#) for details on using these commands.

Viewing Build Results in the Build Tool Window

The output from a build (**Build → Build**) or a compile (**Build → Compile**) are sent to the Build tool window, docked at the bottom of the SlickEdit window by default. This is the same text you would see if you ran the build in an external command shell. See [Navigating from Build Errors to Source Locations](#) for details on jumping from an error message to the corresponding source location and for navigating to the next or previous error.

SlickEdit is already configured to parse many common error formats. If yours is not recognized, you need to configure a new error regular expression. See [Parsing Errors with Regular Expressions](#) for more information.

Viewing Build Errors in the Message List Tool Window

The Message List tool window parses the errors from the Build tool window and displays a list of errors and warnings in a tabular form. Messages can be sorted and filtered. You can also jump from a message

to the corresponding location in the source code. For more information see [Message List](#).

Listing Errors with list-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.

Navigating from Build Errors to Source Locations

SlickEdit provides the means to jump from an error in the Build window to the corresponding location in the source code. You can do this by any of the following:

1. Double-clicking on an error in the Build window.
2. Running the **cursor-error** command, bound to **Alt +1** in CUA emulation.
3. Selecting **Build → Go to Error or Include**.

In each case you must select or position the cursor within a line that contains the filename and, optionally, the line number and column number.

You can move from one error in the Build window to the next using the **next-error** command, bound to **Ctrl +Shift +Down** in the CUA emulation. The same operation is available on the main menu, by selecting **Build → Next Error**.

Use **prev-error**, bound to **Ctrl +Shift +Up**, to move to the previous error. Again, you can select this operation from the main menu at **Build → Previous Error**.

Note

The key bindings for **next-error** and **prev-error** can be used in both the Build tool window and the editor window. In both cases, the result is driven by the order of errors in the Build tool window and the current error, marked by a green triangle.

If the error is within the same file, SlickEdit will move the cursor to that line. Otherwise, SlickEdit will open the corresponding file and move to the indicated line.

Tip

Navigating to a source code location from the Build tool window is not limited to build output. You can execute commands, like sgrep, in that window and use the same methods to jump to the indicated locations.

Parsing Errors with Regular Expressions

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 [SlickEdit® Regular Expressions](#)). 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 Options dialog to configure error parsing, creating new regular expressions or managing the list of existing ones. See the following sections for more information:

- [Configuring Error Parsing](#)
- [Enabling Expressions](#)
- [Setting Priority](#)
- [Adding New Categories](#)
- [Adding New Expressions](#)
- [Editing Expressions](#)
- [Error Expression Groups](#)
- [Sample: Creating a New Error Parsing Expression](#)
- [Testing Expressions](#)

Configuring Error Parsing

To configure error parsing, use the Configure Error Parsing option screen. It can be accessed from the main menu by clicking **Build** → **Configure Error Parsing**, or by opening the Options dialog (**Tools** → **Options**) and selecting **Configure Error Parsing** from the **Tools** category. You can also display the screen with the `configure_error_regex` command.

Configure Error Parsing

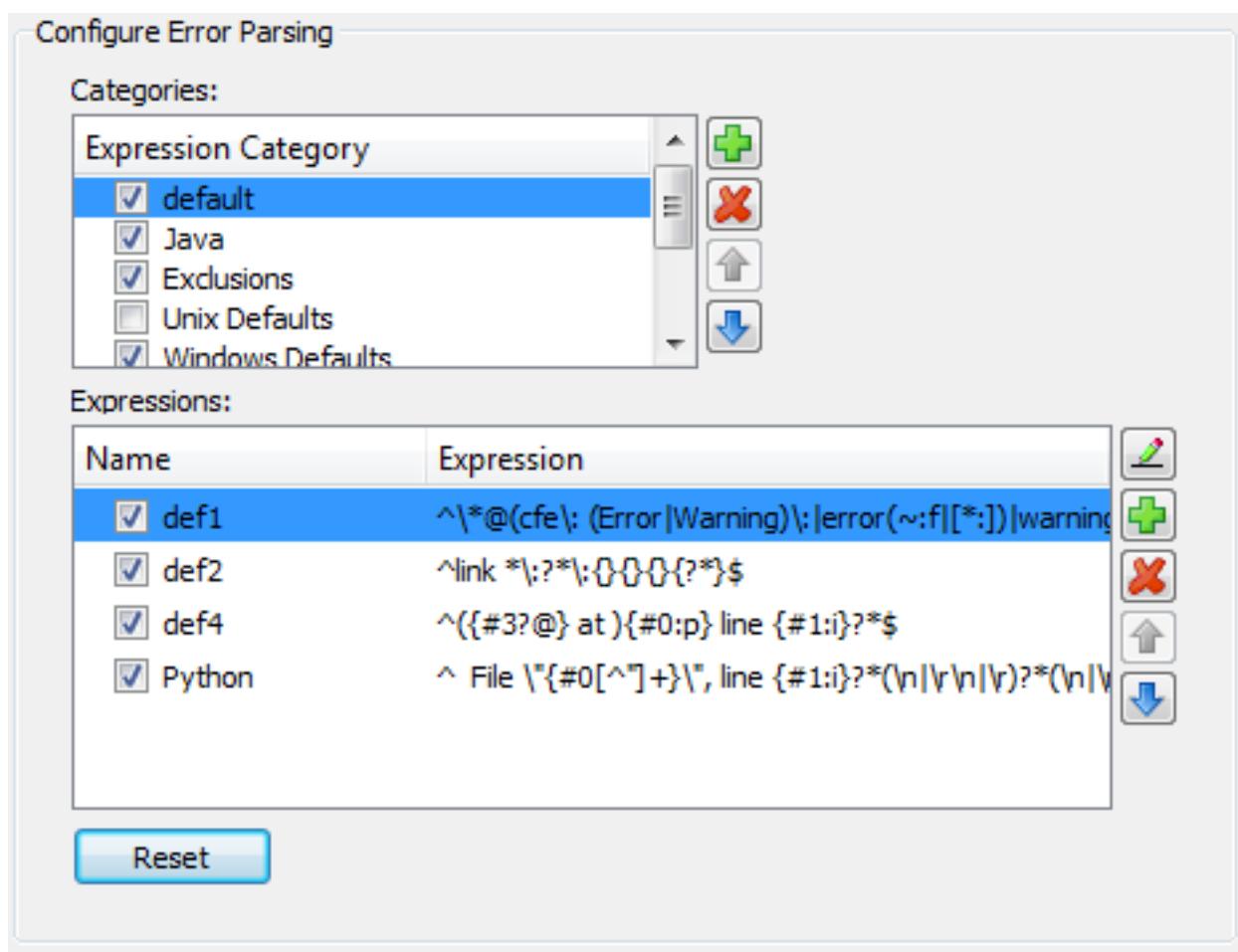
Categories:

Expression Category	
<input checked="" type="checkbox"/>	default
<input checked="" type="checkbox"/>	Java
<input checked="" type="checkbox"/>	Exclusions
<input type="checkbox"/>	Unix Defaults
<input checked="" type="checkbox"/>	Windows Defaults

Expressions:

Name	Expression
<input checked="" type="checkbox"/> def1	<code>^ *@(cfe\ : (Error Warning)\ ; error(~:f [*:]) warning(~:f [*:])\$</code>
<input checked="" type="checkbox"/> def2	<code>^link *\\?*\\{:}{}{}{?*}\$</code>
<input checked="" type="checkbox"/> def4	<code>^({#3?@} at){#0:p} line {#1:i}?*\$</code>
<input checked="" type="checkbox"/> Python	<code>^ File \"{#0[^"]+}\", line {#1:i}?*(\\n \\r\\n \\r)?*(\\n \\r\\n \\r)\$</code>

Reset



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 check box 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 blue **Up** and **Down** arrow buttons.

Resetting Configuration

To reset the configuration settings back to their installation defaults, click the **Reset** button.

Adding New Categories

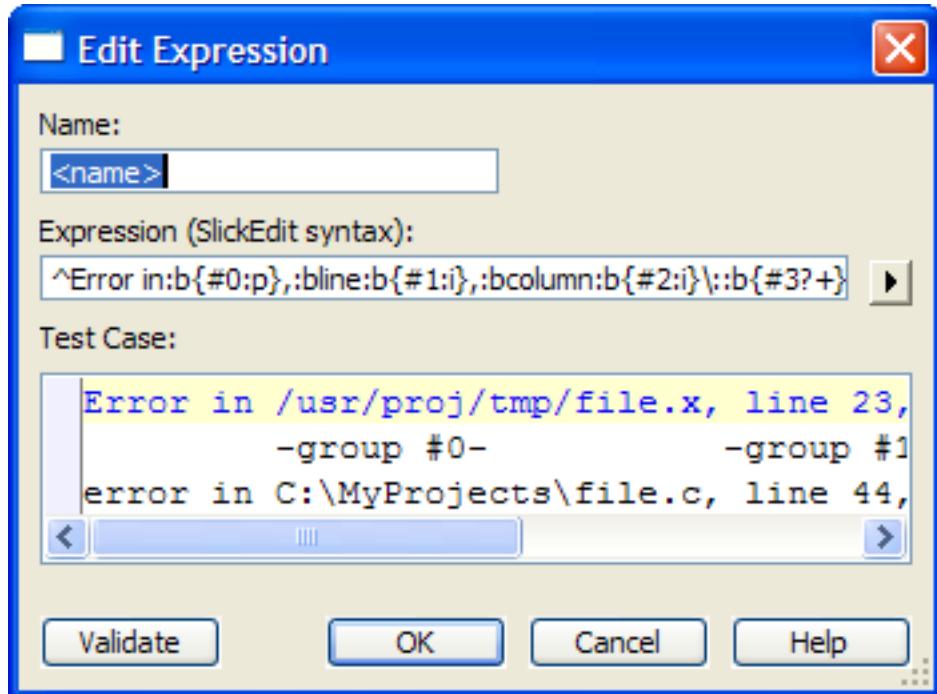
Click the green **Plus** button next to the category listing. The following prompt is shown.



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** button 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** button 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) 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\::b{#3?+} \$**
7. Now test the expression. Click the **Validate** button. You should see a pop-up 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 pop-up 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 click **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 (Pro only)

Running a Program(Pro only)

To run a program, complete the following steps:

1. From the main menu, click **Build** → **Execute**.
2. If there is more than one main program you are prompted to select the one to run.

Debugging (Pro only)

SlickEdit® provides integrated debuggers for the following. Other programs will result in SlickEdit launching an external debugger.

- GNU C/C++
- LLVM C/C++
- Microsoft Visual Studio C++ programs using WinDbg
- Java (Including Android projects. See [Working with Android Projects](#))
- Python
- PHP
- Ruby
- Perl
- Google Go
- Swift
- Groovy
- XCode projects (Open Project created by XCode)
- Scala
- C# using Mono
- Visual Basic using Mono

Use one of the following methods for debugging your code:

- **Debug** → **Start** executes the program and will stop when a breakpoint is reached.

- **Debug** → **Step Into** places you on the first executable line of the program.
- **Debug** → **Restart** stops the current debugger session if necessary and then places you on the first executable line of the program (like **Debug** → **Step Into**).
- **Debug** → **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**).

64-bit Versus 32-bit Programs

On Windows and Linux, SlickEdit is available in both 64-bit and 32-bit versions. You must use the version that matches the code you are debugging. To debug 64-bit programs, you need to use the 64-bit version of SlickEdit. To debug 32-bit programs, use the 32-bit version.

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+PadStar (* on the numeric keypad)	Show next statement
Ctrl+Alt+B or Alt+F9	Activate breakpoints window
Alt+3 or Ctrl+Alt+W	Activate watch window
Alt+4 or Ctrl+Alt+V	Activate variables window
Alt+7 or 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®. For example, you can have one session debugging using GDB, and one using Java at the same time.

To create an additional debugger session, use any of the menu items under the **Debug → Attach Debugger** menu.

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++ or Clang only)

To attach to a running process, complete the following steps:

1. Click **Debug → Attach Debugger → Attach to Running Process(GDB)** or **Debug → Attach Debugger → Attach to Running Process(LLDB)**, 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, click **Debug → Attach Debugger → Disconnect Remote(GDB)** or **Debug → Attach Debugger → Disconnect Remote(LLDB)**.

Attaching to a Remote Process (GNU C++ or Clang only)

To attach to a remote GDB server or GDB stub process, complete the following steps:

1. Click **Debug** → **Attach Debugger** → **Attach to Remote Process(GDB)** or **Debug** → **Attach Debugger** → **Attach to Remote Process(LLDB)**.
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, click **Debug** → **Attach Debugger** → **Disconnect Remote(GDB)** or **Debug** → **Attach Debugger** → **Disconnect Remote(LLDB)**.

Attaching to a Core File (GNU C++ or Clang, UNIX only)

To attach to a core file, complete the following steps:

1. Click **Debug** → **Attach Debugger** → **Analyze Core File(GDB)** or **Debug** → **Attach Debugger** → **Analyze Core File(LLDB)**.
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 JVM (Java only)

To attach to a remote JVM, complete the following steps:

1. Start the remote JVM 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, click **Debug** → **Attach Debugger** → **Attach to Java Virtual Machine**.

To detach from a remote debugging session, click **Debug** → **Disconnect Remote(JVM)**.

Attaching to a Remote VM (Mono only)

To attach to a remote Mono VM, complete the following steps:

1. Start the remote VM with command arguments similar to the following example:

```
mono  
--debugger-agent="transport=dt_socket,server=y,suspend=y,address=localhost:8000"  
Program.exe arg1 arg2
```

2. From the main menu, click **Debug** → **Attach Debugger** → **Attach to Mono Virtual Machine**.

To detach from a remote debugging session, click **Debug** → **Disconnect Remote(JVM)**.

Setting Breakpoints (Pro only)

A new breakpoint can be set using any of the following methods:

- Pressing **F9** - Toggles a breakpoint on the current line. This is the fastest way to set or clear a breakpoint. This runs the **debug_toggle_breakpoint** command. You can bind this command to another key if you like.
- Double-clicking in the left margin. This sets or clears a breakpoint on the associated line. Once a breakpoint is set, click once to disable it or double-click to remove it.
- Selecting **Debug** → **Toggle Breakpoint** from the main menu.
- Selecting **Set Breakpoint** from the context menu. This menu entry is only available when no selection has been made.
- Clicking on **Toggle Breakpoint** button in the **Debug Toolbar**(**View** → **Toolbars** → **Debug**).
- Executing the **debug_toggle_breakpoint** command from the SlickEdit command line.

Breakpoints can be disabled so that their location is preserved but they no longer stop execution in the debugger. To disable a breakpoint, you can click on the breakpoint icon in the window left margin or right-click in the editor on the associated line of code and select **Disable breakpoint**. Breakpoints can be reenabled in a similar manner.

A Breakpoints toolbar (**Debug** → **Windows** → **Breakpoints**) is also available that displays all of the breakpoints and lets you easily add, remove, and activate breakpoints.

Setting Conditional Breakpoints

To set a conditional breakpoint, complete the following steps:

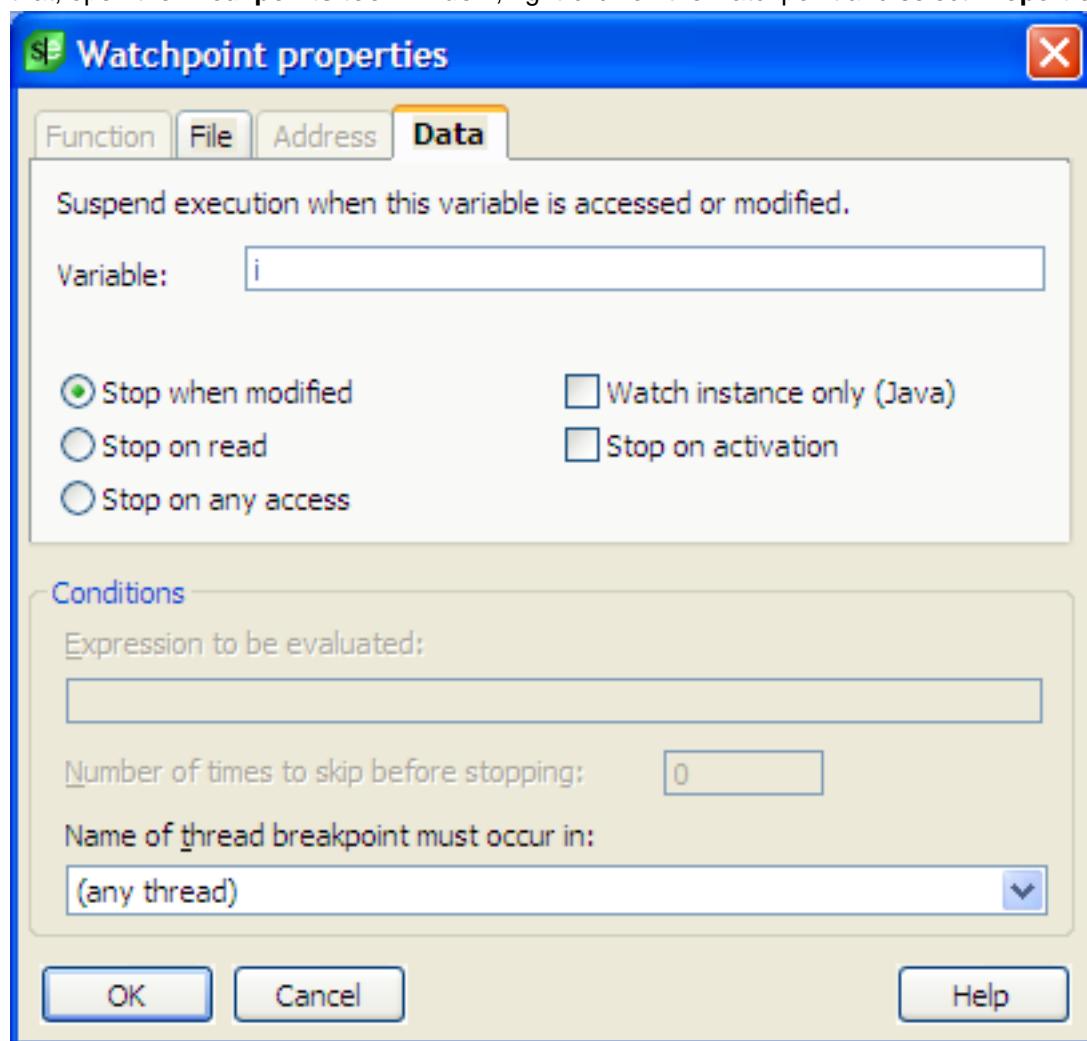
1. Set a breakpoint.
2. Select the **Breakpoints** tab on the Breakpoints toolbar.
3. Double-click on the breakpoint for which you want to set a conditional breakpoint.
4. Set the **Expression** to be evaluated or the **Number of times to skip before stopping**.
5. Click **OK**.

6. Click **Close**.

Watches and Watchpoints

Watchpoints interrupt the debugger when a variable is read, modified, or accessed. To add a watchpoint, select a variable, right-click and select **Set Watchpoint** from the context menu or select **Debug → Set Watchpoint** from the main menu. A green circle will be displayed in the editor left margin, indicating the line where the watchpoint is created. An entry will be created in the Breakpoints tool window listing the watchpoint. Watchpoints do not interrupt debugging at a particular line; they interrupt it when the variable is read, modified, or accessed. So, these markers are used just to manage the watchpoint.

Once you have created a watchpoint, you can control it using the **Watchpoint properties** dialog. To view that, open the **Breakpoints tool window**, right-click on the watchpoint and select **Properties**.



Setting a watch, adds an expression to the **Watch tool window** in the debugger view. These expressions are evaluated any time execution stops and the resulting value is displayed. They do not cause the debugger to stop running.

Setting Java or C# 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.

Relocatable Code Markers

Breakpoints use relocatable code markers to store their location within the source code. This allows SlickEdit to find the new location if someone makes changes to the file externally, like modifying 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 breakpoint. If the code has changed too much, SlickEdit may not be able to find the new location. Instead, the breakpoint will be placed at the line number where it was last known to be.

SlickEdit does not attempt to relocate breakpoints during debugging sessions while the external debugger is in control of placing and tracking breakpoints. Note that if multiple debug sessions each have breakpoints in a common file, this will cause the relocatable marker information for all sessions to be cleared when the debugger enters that file.

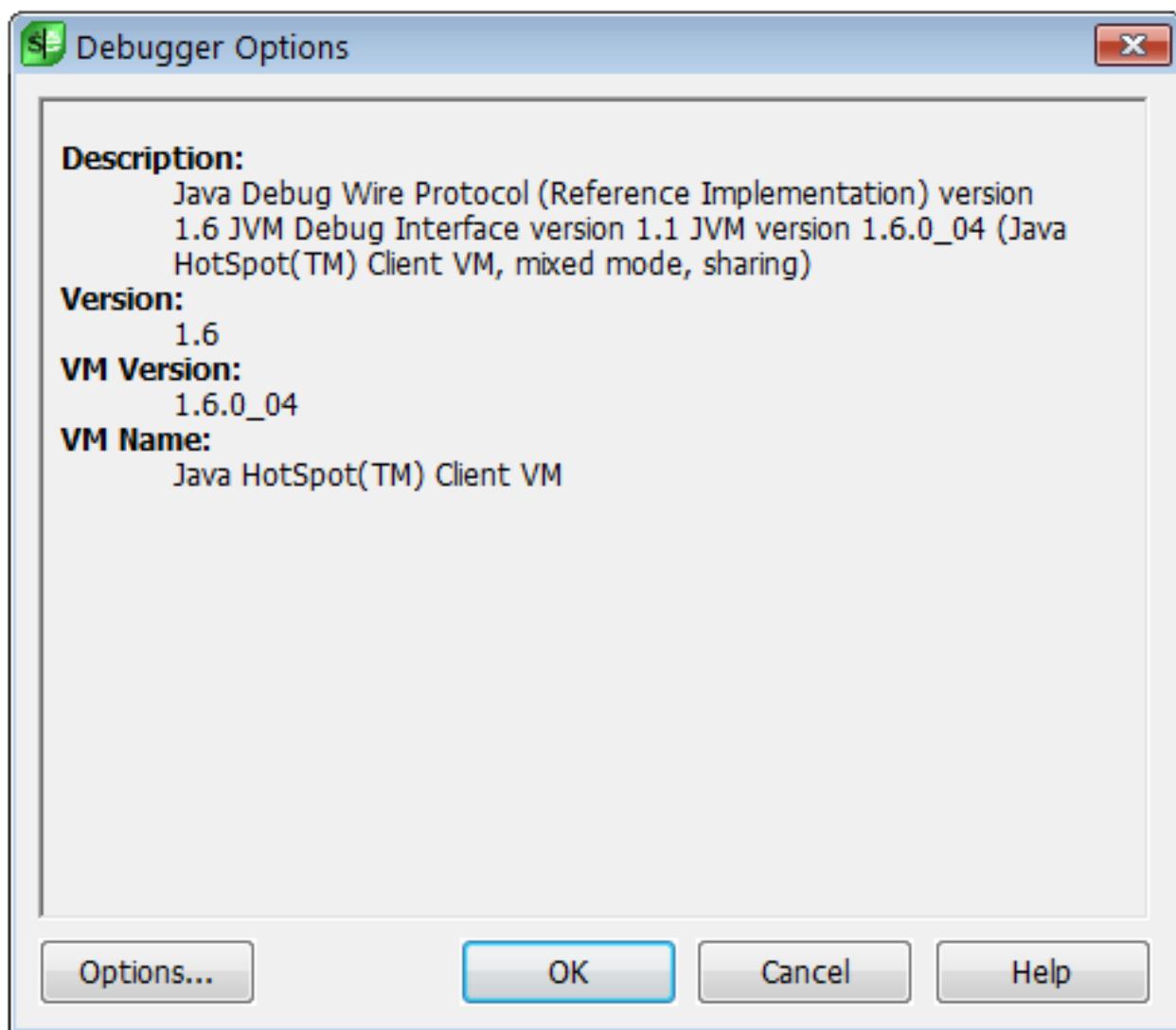
Generate Debug

This feature supports C#, C++, Java, and Slick-C®. Place the cursor on a function name, then click **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 click **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.

Viewing Debugger Info and Setting Options

To view the properties of the underlying debugger system, including a general description retrieved from the debugger, version number, run-time version, and debugger name, make sure you're in debug mode, then from the main menu, click **Debug** → **Debugger Information** (or use the **debug_props** command).



Click the **Options** button to tune the run-time performance of the integrated debugger, examine the properties of the underlying debugger system, set class filters, and/or control the directories searched for source files. See [Debugging Options](#) for more information.

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.

Debugging GNU C/C++ (Pro only)

You can debug a GNU C++ program in one of the following ways:

- Create SlickEdit project with the project type set to GNU C/C++, Clang++, or Cross Platform C++

Wizard.

- Attach to a running process. See [Attaching to a Running Process](#))
- Attach to a core file. See [Attaching to a Core File](#))
- Attach to a remote process file. See [Attaching to a Remote Process](#))

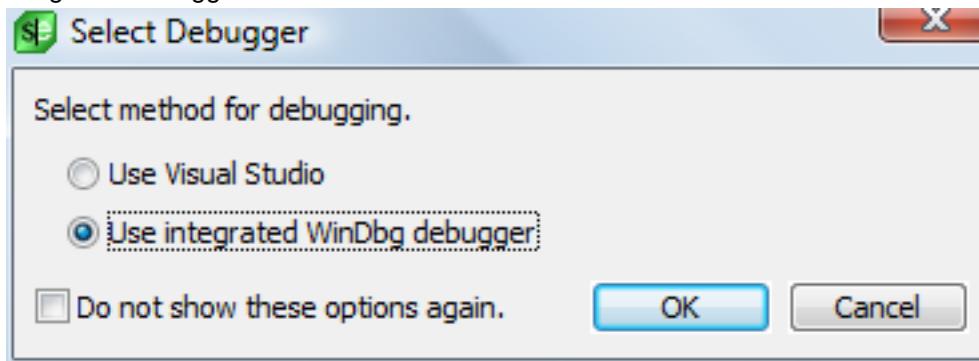
Debugging for GNU C/C++ programs uses a customized version of GDB. Please refer to the release notes for specific version information. You can download the customized source from www.slickedit.com/gdb [http://www.slickedit.com/gdb].

Note

The GDB shipped with SlickEdit on Windows is based on MinGW. If you prefer to use Cygwin for GDB, you can 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.

Debugging Microsoft Visual Studio C++ Programs Using WinDbg (Pro only)

When debugging C++ projects in a Visual Studio solution, you have the option to use the integrated SlickEdit debugger with WinDbg or to use Visual Studio for debugging. By default, you are prompted to select the method for debugging (see image, below). There is an option to never show the prompt again, in which case it will always use the selected method for debugging. This option is specified in configuration variable **def_vcproj_debug_prefs**. Setting this configuration macro variable to blank (the default) will prompt always, a value of 1 prefers using the Visual Studio and a value of 2 prefers using the integrated debugger.



WinDbg is available from Microsoft as part of their Debugging Tools for Windows. To download WinDbg and read more about it, visit Microsoft's website.

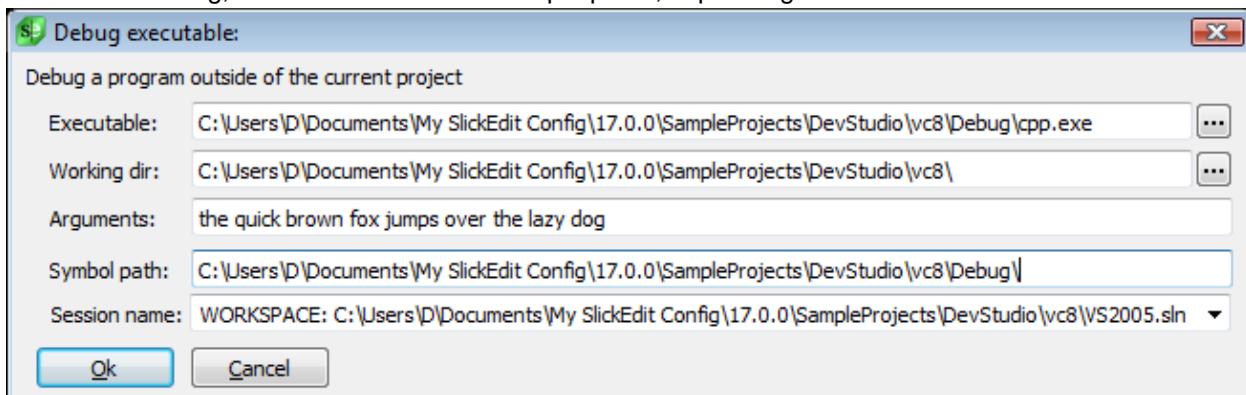
SlickEdit requires dbgeng.dll and dbghelp.dll for debugging. By default, it will search for the dlls in the default dynamic-link library search paths (SlickEdit directory, Windows directory, Windows system directory, directories under PATH environment variable). You can also specify a path with a configuration macro variable **def_windbg_path**. Set it using the SlickEdit command-line (set-var def_windbg_path) or

Macro > Set Macro Variable.

Note

SlickEdit 64-bit requires 64-bit (x64) dbgeng.dll and dbghelp.dll and can debug both 32 and 64-bit executables. SlickEdit 32-bit requires 32-bit (x86) dbgeng.dll and dbghelp.dll and can only debug 32-bit executables.

To debug, WinDbg needs the path to the executable image and the path to the symbol information. The path to the executable image specifies the location of the .exe and .dll files that are being debugged. The path to the symbol information specifies the location of symbol files (.pdb), which contain debugging information. They are generated by the compiler and linker. SlickEdit may be able to determine these paths. If it can't you can specify these paths in the dialogs that launch the debugger. See the **Debug executable** dialog, below. You can enter multiple paths, separating each with semicolons.

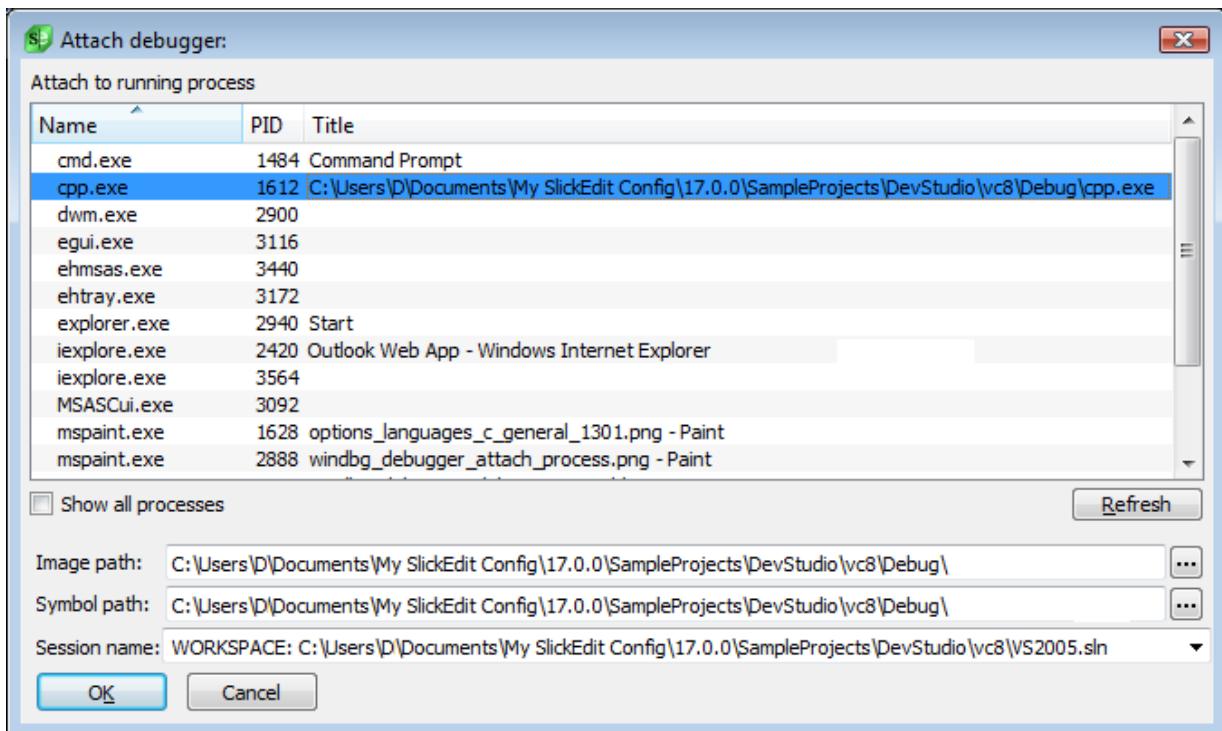


WinDbg integration supports controlling target processes, stepping through source code, setting breakpoints, and accessing memory and registers. Locals and member variables are automatically generated based on the current thread and stack scope. Watches can be set for any symbol name or a C++ expression, evaluated by the current thread and stack scope. A symbol name can be qualified by its module name using an exclamation mark (!) separating the module name from the symbol name. Specifying the module name in the expression will usually result in faster evaluations and resolves any symbol ambiguity, such as if the symbol name could be interpreted as a hexadecimal number. To restrict to local scope only, prefix a dollar sign and exclamation point (\$!) to the symbol name.

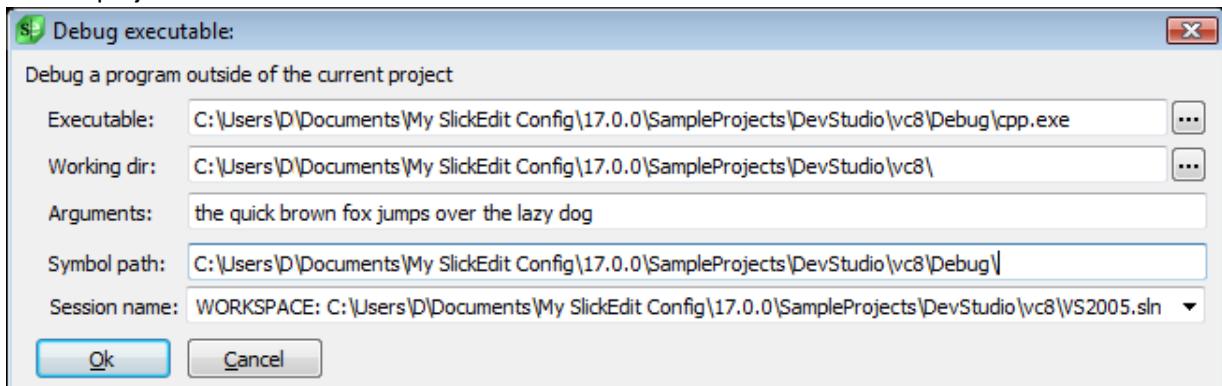
Use the WinDbg debugger by selecting any of the following from the main menu:

- **Debug → Start** - You will be prompted which debugger to use.
- **Debug → Attach Debugger → WinDbg → Attach Process** - Attaches the debugger to a running process by process ID.

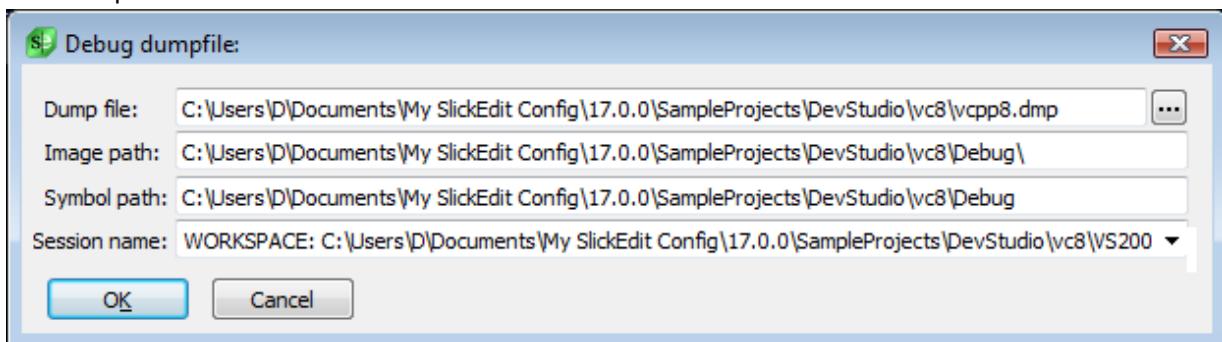
WinDbg (Pro only)



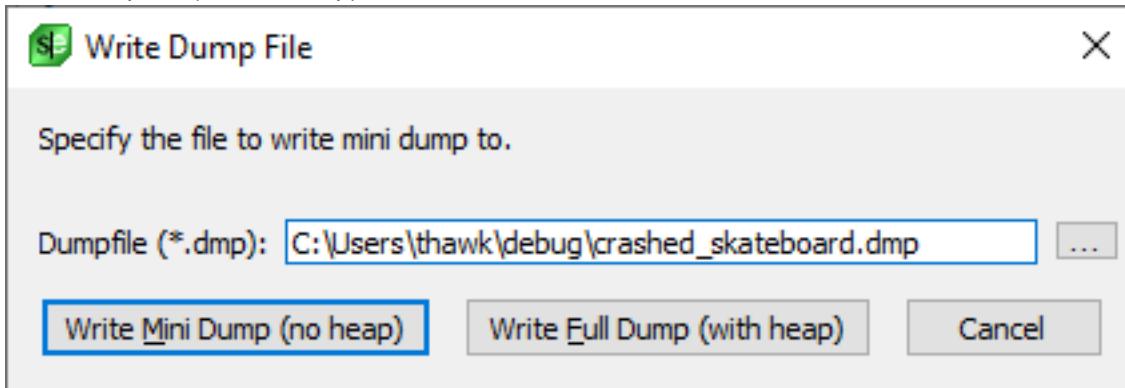
- **Debug → Attach Debugger → Debug Executable (WinDbg)** - Debugs an executable outside the current project.



- **Debug → Attach Debugger → Open Dump File (WinDbg)** - Opens the debugger on a Visual Studio mini dump file.



- **Debug → Attach Debugger → Write Dump File (WinDbg)** - Writes a Visual Studio mini dump file for the current active WinDBG debugging session. You can select to write a full dump file (with heap) or a small dump file (without heap).



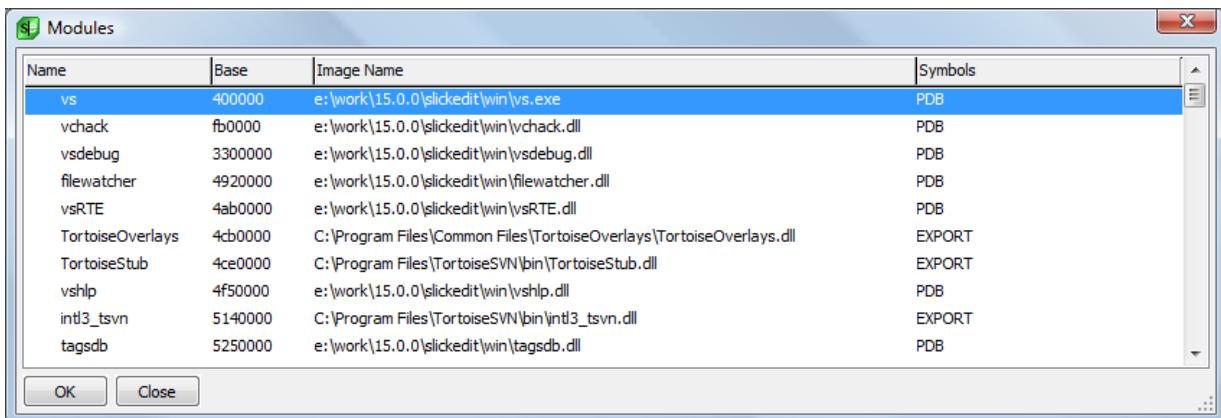
The following Debugging tool windows and operations are supported for WinDbg:

- Call Stack
- Threads
- Registers
- Breakpoints
- Members
- Locals
- Watch
- Memory
- Show Disassembly
- Step into, Step out, Step over, Continue, Break

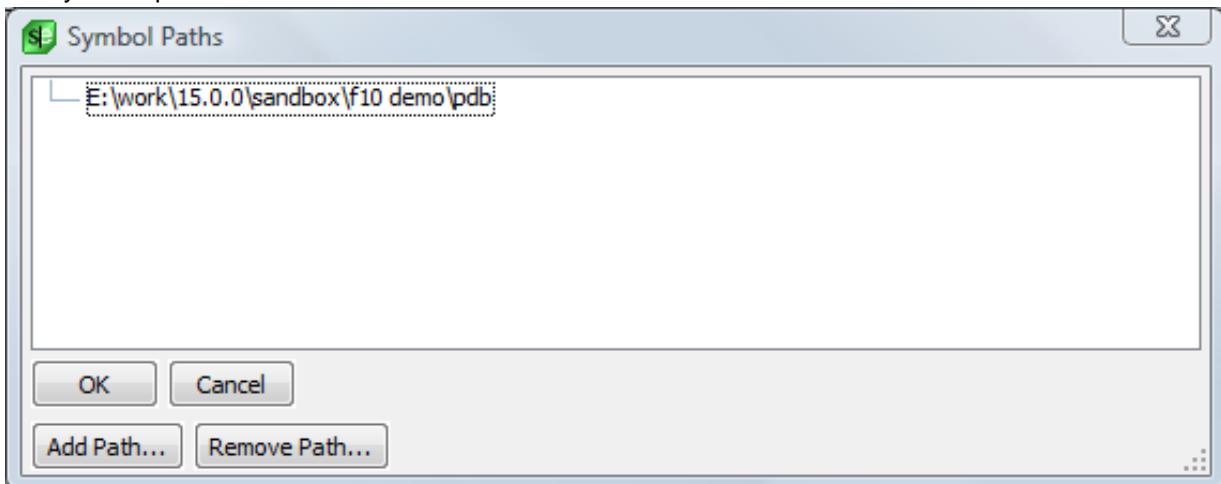
The following commands are available on the SlickEdit command line:

- **windbg_write_dumpfile** - Write current debugging session to a dump file.
- **windbg_list_modules** - List the currently loaded exe and DLL's. It also lists the base memory address of the loaded module, the image name (name and extension), and the symbol file type (if any).

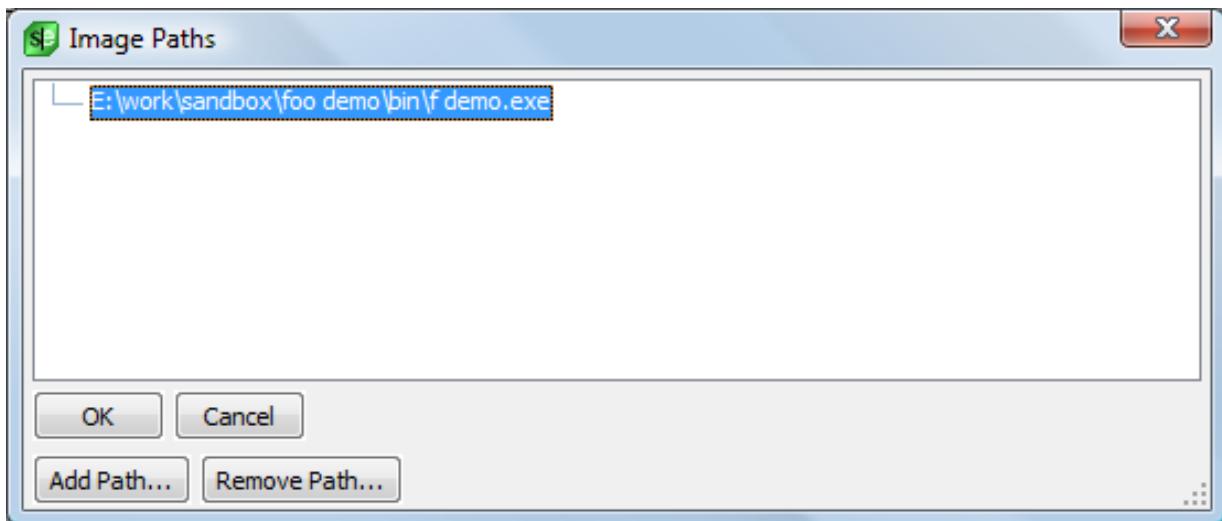
Debugging Microsoft Visual Studio C++ Programs Using



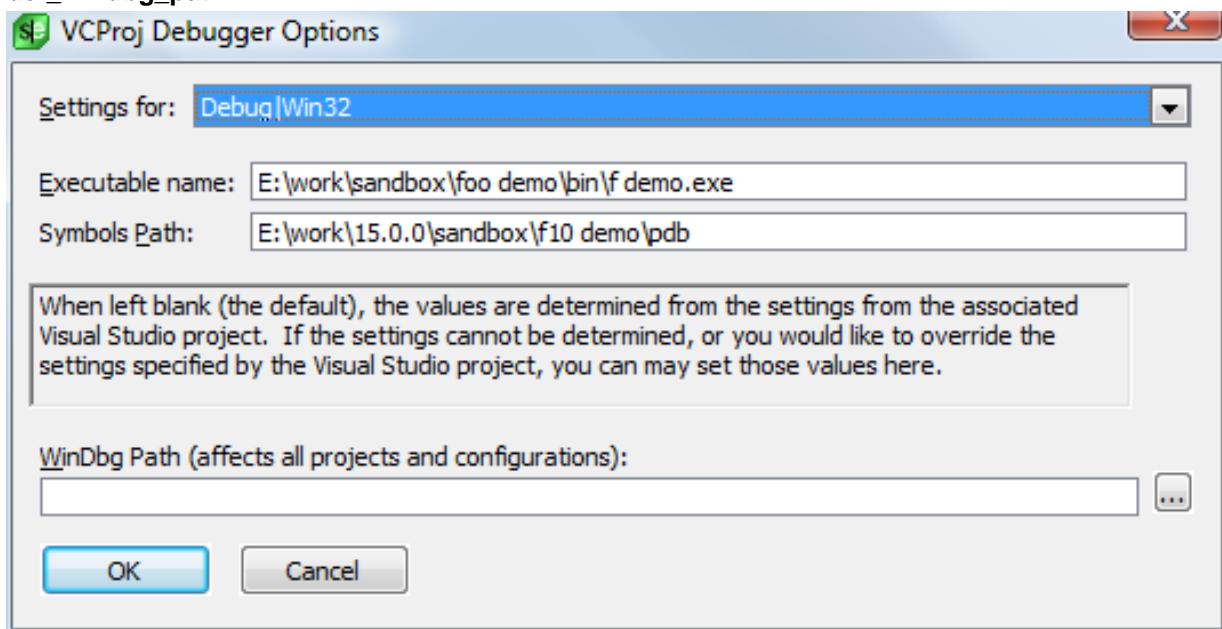
- **windbg_update_symbols_path** - List and update currently loaded Symbol paths for the current debug session. Use this command to update the symbols path during debugging in case you need to add a path to a PDB file after launching the debugger. Symbols are automatically reloaded when you update the symbols path.



- **windbg_update_image_path** - List and update currently loaded Image paths for the current debug session. Use this command to update the image path during debugging in case you need to add a path after launching the debugger. Images are automatically reloaded when you update the image path.



- **vcproj_debug_options** - Option to set executable name and path and Symbols paths for current Visual Studio project. By default, SlickEdit will try to determine the location of the output Executable file name and the location of the program database symbols file (PDB) directly from the Visual Studio C++ project file. If you need to specify a custom location for the output filename or SlickEdit cannot correctly evaluate the correct location from the Visual Studio project file, you can specify the path name here. Or if you need to specify multiple paths for symbol files, you can do that here. If left blank, then either field it will fall back to the Visual Studio project file settings. The WinDbg path is the global setting for **def_windbg_path**.



Running and Debugging PHP (Pro only)

To run a PHP script you need:

- PHP 5.x or later.
- A PHP project.

Additionally, to debug a PHP script you need:

- The Xdebug plugin 2.0.2 or later for PHP installed on your PHP server. You can obtain the Xdebug plugin from <http://xdebug.org>.

After installing the Xdebug plugin and creating a PHP project you can debug local or remote scripts and web pages.

Installing Xdebug

PHP projects support debugging with the Xdebug plugin for PHP. You can download the plugin from <http://xdebug.org>.

1. Extract the Xdebug dll/lib to your PHP extension directory. Windows users can use the Windows installer package provided on the xdebug.org site. Linux users may be able to install the Xdebug plugin from their package manager. *Make sure you are using Xdebug 2.0.2 or later.*

Note

Set up a test page that prints out results of `phpinfo()` to determine PHP settings, including where your extension directory and 'php.ini' config file resides. A test page looks like:

```
<!-- phpinfo.php -->
<?php
echo phpinfo();
?>
```

2. Add the following section to your 'php.ini' config file (see note above if you do not know the location of your 'php.ini' file):

```
; Xdebug debugger extension
[Xdebug]
; Xdebug plugin installed via Windows pre-built binaries: Use thread-safe
zend_extension_ts="..."
; Xdebug plugin installed via PECL (typically UNIX): Use non-thread-safe
zend_extension="..."
; Xdebug plugin built from source: Follow directions from xdebug.org site
zend_extension_ts="c:/php5/ext/php_xdebug-2.0.3-5.2.5.dll"
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
```

```
xdebug.idekey=slickedit
xdebug.remote_host=127.0.0.1 ; for remote debugging
xdebug.remote_port=9000
```

Warning

Comment out any preexisting Zend optimizer and debugger extensions. Zend optimizer and debugger extensions are not compatible with Xdebug.

Warning

Windows users can install pre-compiled modules downloaded from xdebug.org. These modules are thread-safe and should therefore be installed using `zend_extension_ts="..."` as outlined in the example above.

UNIX users that install from PECL or a package manager will typically be installing the non-thread-safe version of the Xdebug plugin and should therefore be using `zend_extension="..."` instead of `zend_extension_ts="..."`.

Note

You must change the `zend_extension[_ts]` line to match the path you extracted the dll/lib to in step #1.

For the `xdebug.remote_host` line: if your web server resides on your local machine, then no changes need to be made. If your web server is remote, then use the IP address that SlickEdit will be listening on for a connection from Xdebug.

3. Restart your web server.
4. Test that Xdebug is installed successfully by creating a test page that echoes `'phpinfo()'` (see example in step #1).

You should see a banner similar to the following indicating that the PHP server is using Xdebug:

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.2.0, Copyright (c) 1998-2008 Zend Technologies
with Xdebug v2.0.3, Copyright (c) 2002-2007, by Derick Rethans



Alternatively, if you are debugging standalone scripts, you can issue the following command from a console and look for the Xdebug line:

```
> php -v
```

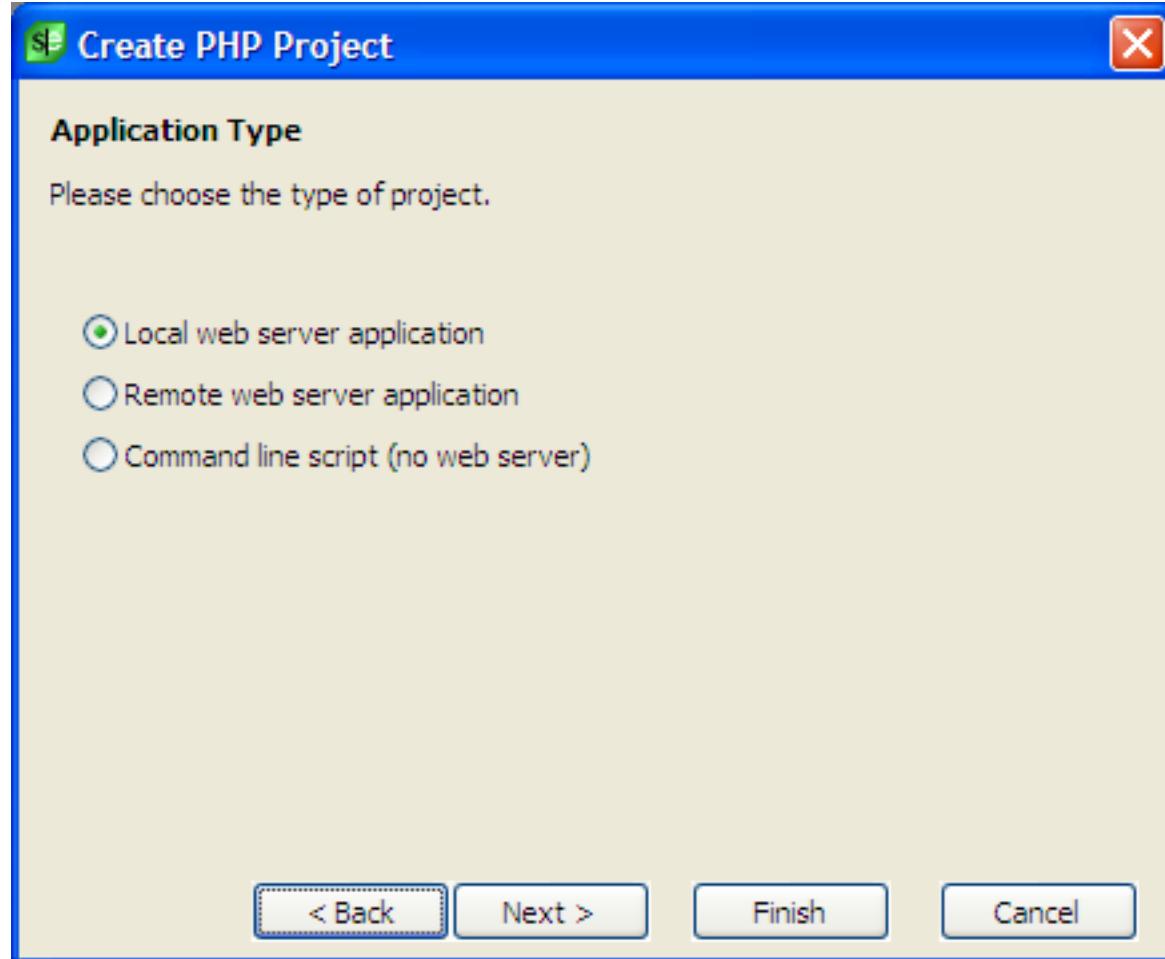
```
PHP 5.2.6 (cli) (built: May 2 2008 18:02:07)
Copyright (c) 1997-2008 The PHP Group
Zend Engine v2.2.0, Copyright (c) 1998-2008 Zend Technologies
with Xdebug v2.0.3, Copyright (c) 2002-2007, by Derick Rethans
```

- Once Xdebug is installed and working, verify that the remote_host and remote_port settings in your 'php.ini' config file match those set up for your PHP project (**Build → PHP**).

Setting Up a PHP Project

A PHP project lets you run and debug your PHP web pages and scripts.

To create a PHP project, run the Create PHP Project wizard by selecting **Project → New** from the main menu. For PHP, you only have one choice: "PHP". Fill in the project name and location for the new project, and click **OK**. For more information on projects, project types, and creating projects, see [Managing Projects](#).



SlickEdit will display the **Create PHP Project** wizard, which will walk you through steps to configure the PHP project, including:

1. Where your PHP files reside on your local file system.
2. How your local files map onto a web page URL (for the case of web projects).
3. How a PHP file on a remote server maps to a local PHP file (for the case of debugging remote web projects).

Note

Your PHP project must contain local copies of all files being debugged.

Select **Project → Project Properties** to add local files to your project. After you have created your PHP project, select **Build → PHP Options** to make changes to file mappings and debugger settings.

Executing and Debugging a Web Page

For web-based projects, local files in your project map to web page URLs on your web server. To launch a web page in your web browser, open a PHP file from your project and execute it (**Build → Execute**).

Debugging requires the Xdebug PHP plugin be installed on your web server. If you have not installed and verified your Xdebug installation, then please read [Installing Xdebug](#). Verify that your project's Xdebug server settings (**Build → PHP Options, Debug tab**) match the Xdebug host:port you configured for your web server.

There are two ways to start a debug session:

1. Debugging a local file - Open a local PHP file in your project and start the debugger (**Debug → Start**). The File-to-URL mapping you set up when you created the project will be used to map the local file onto a web page URL and launch a browser to start debugging.
2. Listening for Xdebug connection - You can start a debug session from your browser by appending an `XDEBUG_SESSION_START` argument to the URL:

```
http://localhost/index.php?XDEBUG_SESSION_START=slickedit
```

The web server will then attempt to connect back to your project and start a debug session. Make sure your project is listening for the connection by toggling **Debug → Xdebug Listen in Background**.

You can stop a debug session started from your browser by appending the `XDEBUG_SESSION_STOP` argument to the URL:

```
http://localhost/index.php?XDEBUG_SESSION_STOP
```

Note

If you use Firefox, then there is a great Firefox add-on called Xdebug Helper. You can get it from <http://addons.mozilla.org>. It allows you to toggle start/stop an Xdebug session from Firefox without messing with URL arguments. You toggle from Xdebug Helper icon in the Firefox tray (lower-right).

Executing and Debugging a Local Script

Local scripts are PHP scripts that you run from a console. If your project was set up to run as a local script (**Build** → **PHP Options**, Run tab, Run as), then execute your script by selecting Execute from the Build menu **Build** → **Execute**.

Debugging requires the Xdebug PHP plugin be installed. If you have not installed and verified your Xdebug installation yet, then please read [Installing Xdebug](#).

Verify that your project's Xdebug server settings (**Build** → **PHP Options**, Debug tab) match the Xdebug host:port you configured in your php.ini configuration file.

Debug your script by selecting Start from the Debug menu (**Debug** → **Start**).

PHP Options

You can set a number of options to control the execution and debugging of PHP scripts. You can access PHP options by selecting **Build** → **PHP Options** from the main menu. This menu entry is only available if the active project is a PHP project. Options are broken into two groups, each with its own tab:

- [Run Options](#)
- [Debug Options](#)

At the top of the PHP Options dialog, you can pick the configuration that these settings apply to. The default is **All Configurations**. However, you can define different settings for separate Run and Debug configurations if you choose.

At the bottom of the dialog you can set the **PHP interpreter**. This is the path to the PHP interpreter to use.

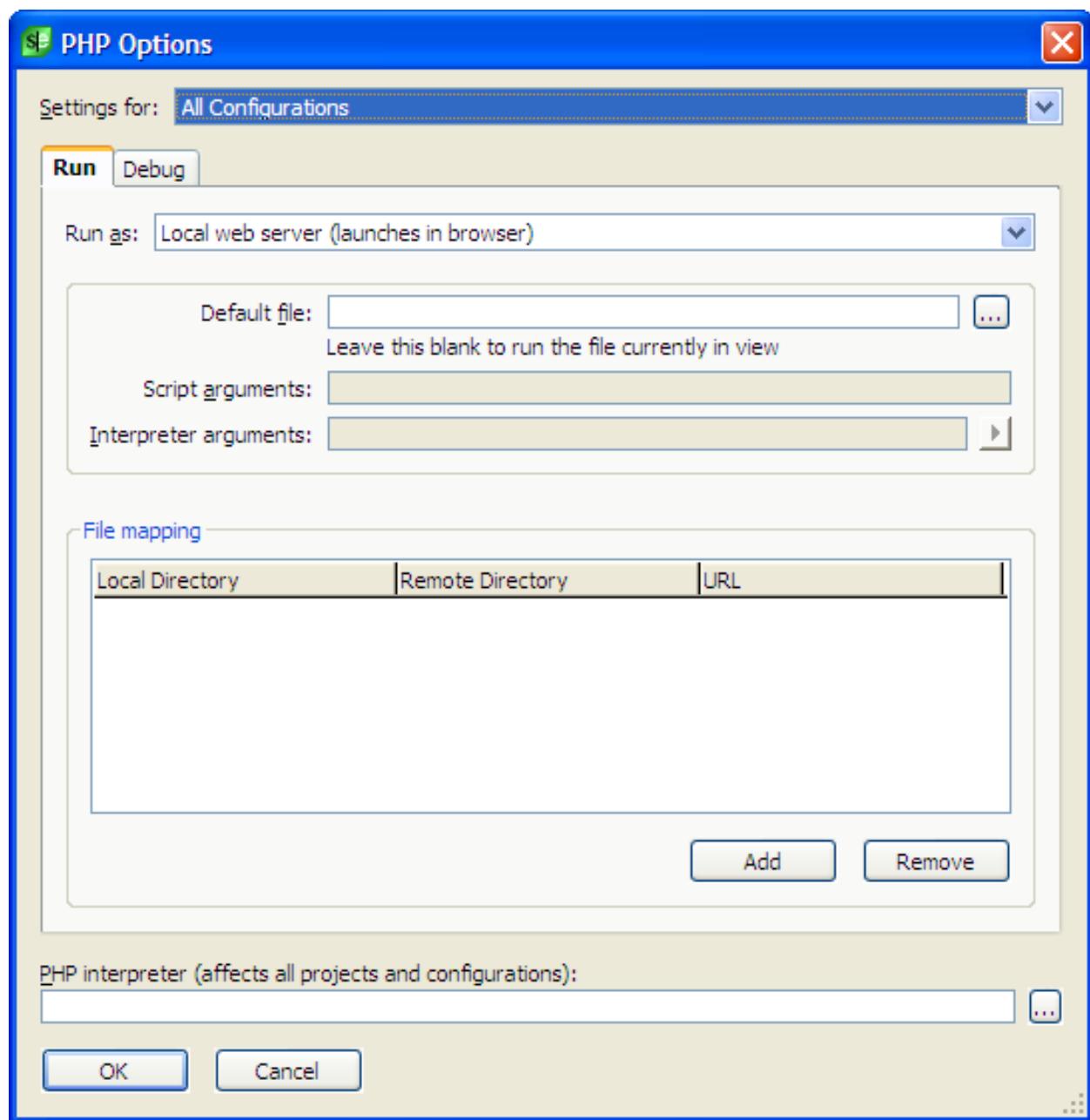
Note

The path entered for the PHP interpreter affects all projects and configurations. SlickEdit currently cannot use different interpreters for different projects.

Run Options

The Run options control the execution of PHP scripts both in and out of the debugger.

Running and Debugging PHP (Pro only)

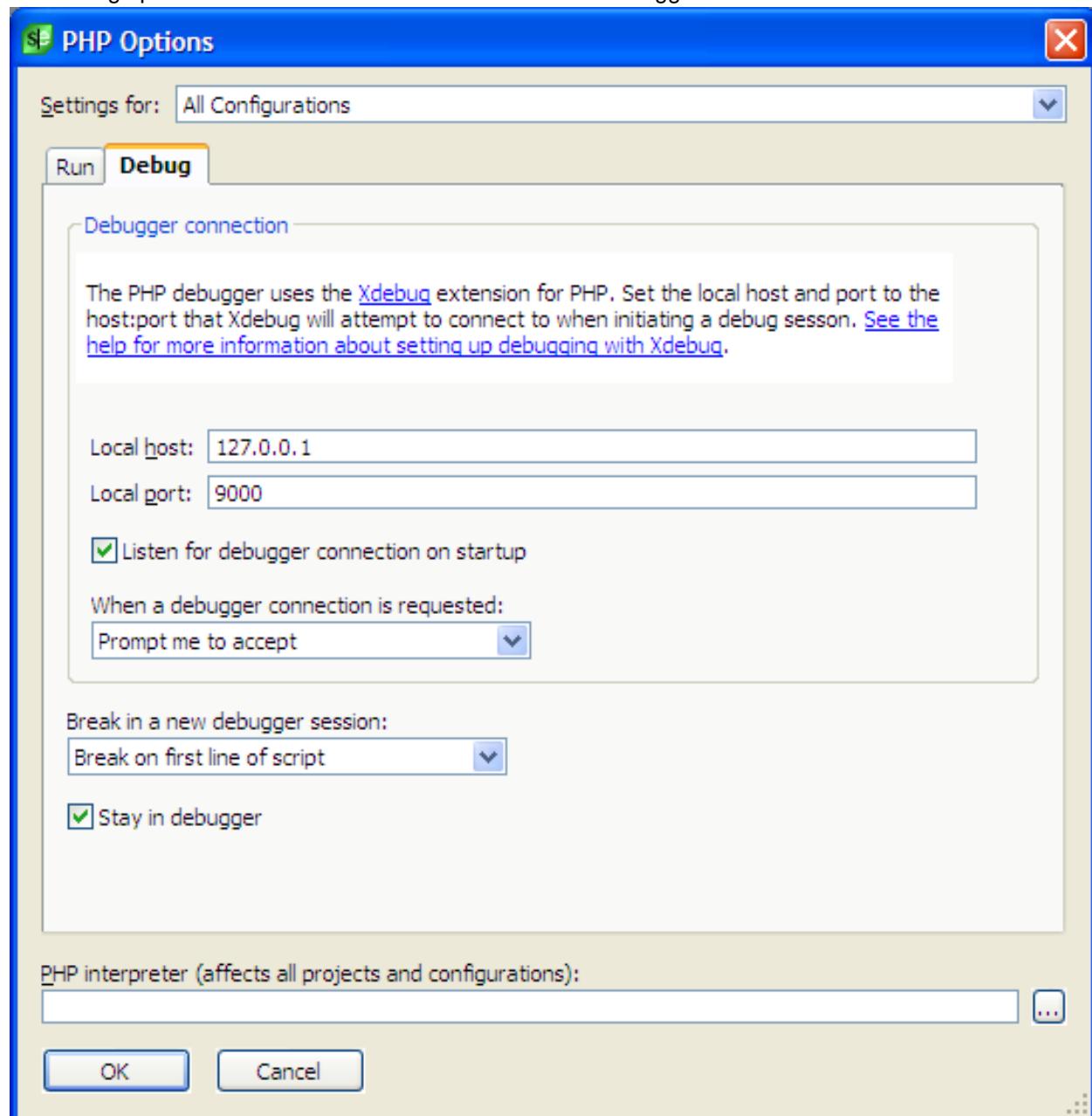


- **Run as** - determines how this script will be run. Pick one of the following:
 - **Local web server (launches in browser)** - runs the script using a web server on this machine.
 - **Local script (command line)** - runs the script from the command line.
 - **Remote web server (launches in browser)** - runs the script on a remote web server.
- **Default file** - identifies the file to use to start the execution.
- **Script arguments** - arguments to be passed to the script.
- **Interpreter arguments** - arguments to be passed to the PHP interpreter.

- **File mappings** - File mappings are very important when debugging remote scripts (usually web pages). They are used to:
 - Map a local file to a web page URL in order to execute a web page
 - Map a remote file to a local file when debugging.

Debug Options

The debug options set values that are used to control the debugger.



- **Local host** - the IP address on your local machine that Xdebug will connect to when initiating a

debugger session. This value needs to be the same as set in the `php.ini` file for Xdebug. See [Installing Xdebug](#) for more information.

- **Local port** - the port on your machine that Xdebug will connect to when initiating a debugger session. This goes with the Local host value, above.
- **Listen for debugger connection on startup** - when checked, SlickEdit will begin listening for a connection when you start the debugger.
- **When a debugger connection is requested** - describes how to handle a request for a debugger connection. Select one of the following:
 - **Prompt me to accept** - prompts each time a debugger connection is requested.
 - **Always accept** - silently accepts all debugger connections.
 - **Never accept** - silently refuses all debugger connections.
- **Break in a new debugger session** - defines when to break for a new debugger session. Select one of the following:
 - **Break on first line of script**
 - **Run to first breakpoint**
- **Stay in debugger** - Set this option when you do not want to exit the debugging session when a script has completed. This is especially useful when debugging a website and you will be jumping in and out of pages as you navigate the site.

Using an SSH Tunnel to Debug a Remote Web Page

If your web server resides on a host that supports ssh, then it is very convenient to set up an ssh tunnel to tunnel debugger connections from your remote server back to your local machine. As an example, if your remote web server is called 'myhost.com' and you are using the default debugger connection settings of 127.0.0.1 on port 9000 both locally and on the remote server, then start an ssh tunnel with the following command:

```
ssh username@myhost.com -R 9000:127.0.0.1:9000
```

This saves you the hassle of having to ensure you have picked the correct interface on which to listen for debugger connections from the remote server.

Running and Debugging Python (Pro only)

To run or debug a Python script you need:

- Python 2.6 and higher.

- A Python project.

Executing and Debugging a Local Script

Execute your script by selecting Execute from the Build menu **Build → Execute**.

Debug your script by selecting Start from the Debug menu (**Debug → Start**).

Debugging a Remote Script

There are a few requirements you must meet to debug Python on remote systems.

- A copy of the script you want to debug needs to be on the local system with the editor. You need to make sure you have the same version of that source that is on your remote system.
- You will need a copy of the PTVSD debugger on the remote system.
- Your network will need to allow TCP connections from your editor machine to the remote host, on a port of your choosing.

If your remote source code is on a different path on the remote machine than it is on your local machine, you'll need to configure the remote directory mapping so the debugger can display the correct files. See [Remote Mappings](#) for details on remote mapping.

The easiest way to get PTVSD onto your remote system is to copy the version that is shipped with the editor, under the application folder in resource/tools/ptvsd.

```
$ scp -r /opt/SlickEdit/resource/tools/ptvsd  
remotehost:/home/user/tools
```

Once PTVSD is on the remote host, you can start your program on the remote host via the PTVSD command line. The debugger will listen for connections from the editor on a port you pick.

You can run a script file under the debugger:

```
$ python -m tools/ptvsd --host localhost --port 5678 --wait  
yourprogram.py
```

You can also run a module under the debugger:

```
$ python -m tools/ptvsd --host localhost --port 5678 --wait -m  
/path/yourmodule
```

To connect to the running remote debugger in the editor, go to the Debug menu, select the Attach Debugger sub-menu, and select "Attach to Python (PTVSD)...". You will be prompted for the remote host name, and the port you told PTVSD to listen on. Once connected, the debugger will work just like a local

session.

Python Options

You can set a number of options to control the execution and debugging of PYTHON scripts. access Python options by selecting **Build → Python Options** from the main menu. This menu entry is only available if the active project is a Python project. Options are broken into three groups, each with its own tab:

- [Run Options](#)
- [Debug Options](#)
- [Remote Mappings](#)

At the top of the Python Options dialog, you can pick the configuration that these settings apply to. The default is **All Configurations**. However, you can define different settings for separate Run and Debug configurations if you choose.

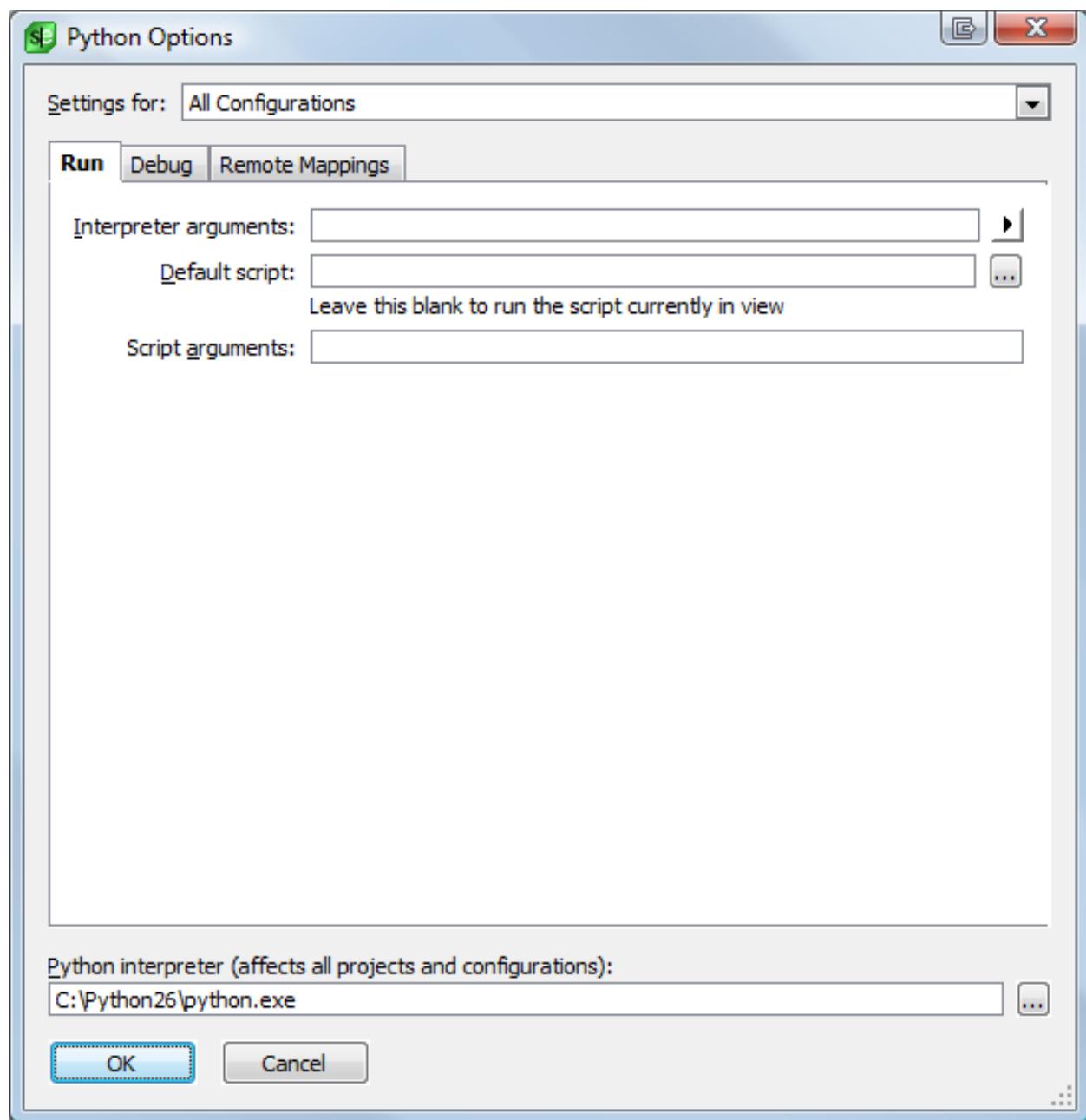
At the bottom of the dialog you can set the **Python interpreter**. This is the path to the Python interpreter to use.

Note

The path entered for the Python interpreter affects all projects and configurations. SlickEdit currently cannot use different interpreters for different projects.

Run Options

Running and Debugging Python (Pro only)

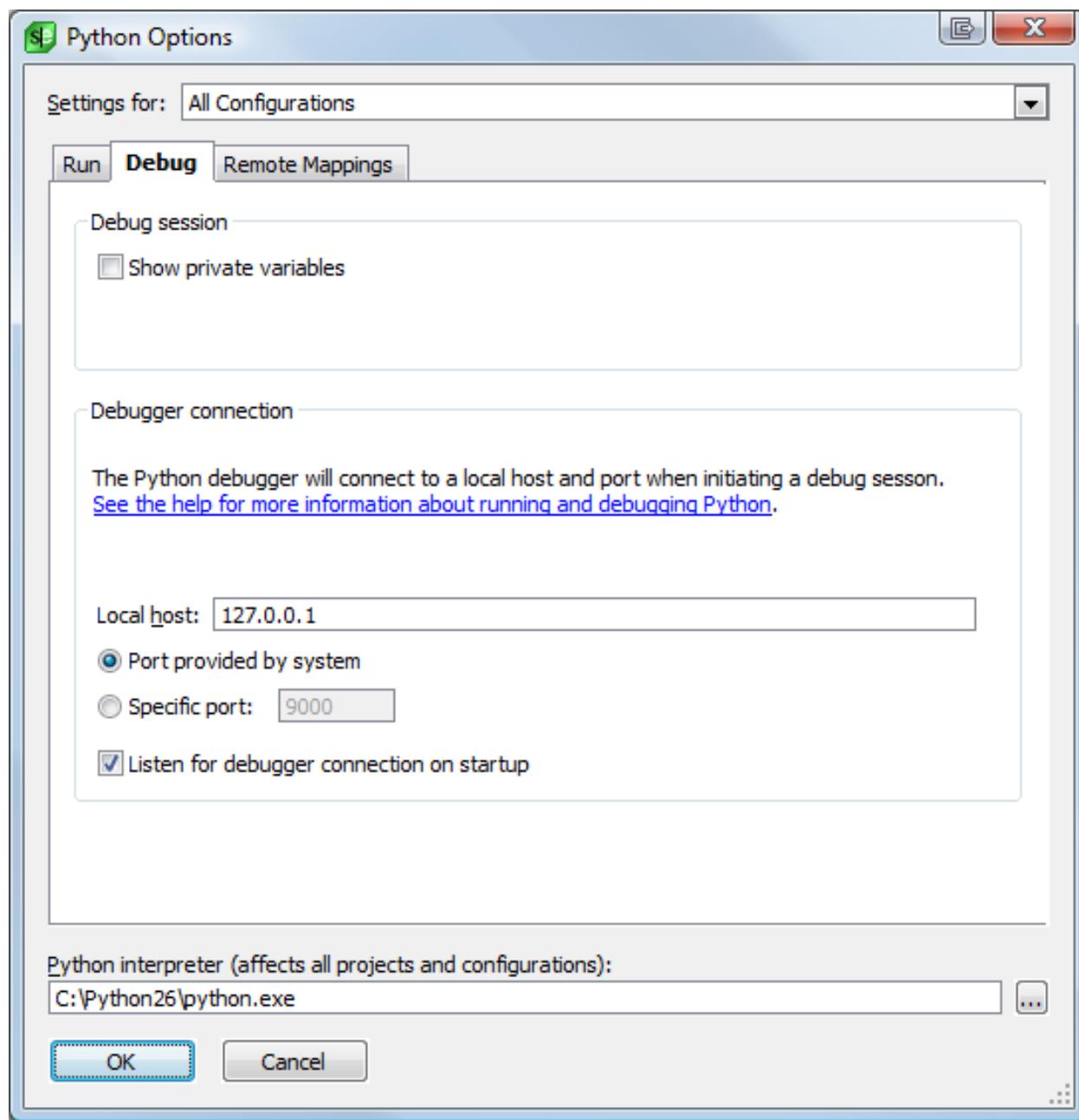


- **Interpreter arguments** - arguments to be passed to the Python interpreter.
- **Default script** - identifies the file to use to start the execution.
- **Script arguments** - arguments to be passed to the script.

Debug Options

The **Debug** tab contains options that pertain to debugging Python scripts.

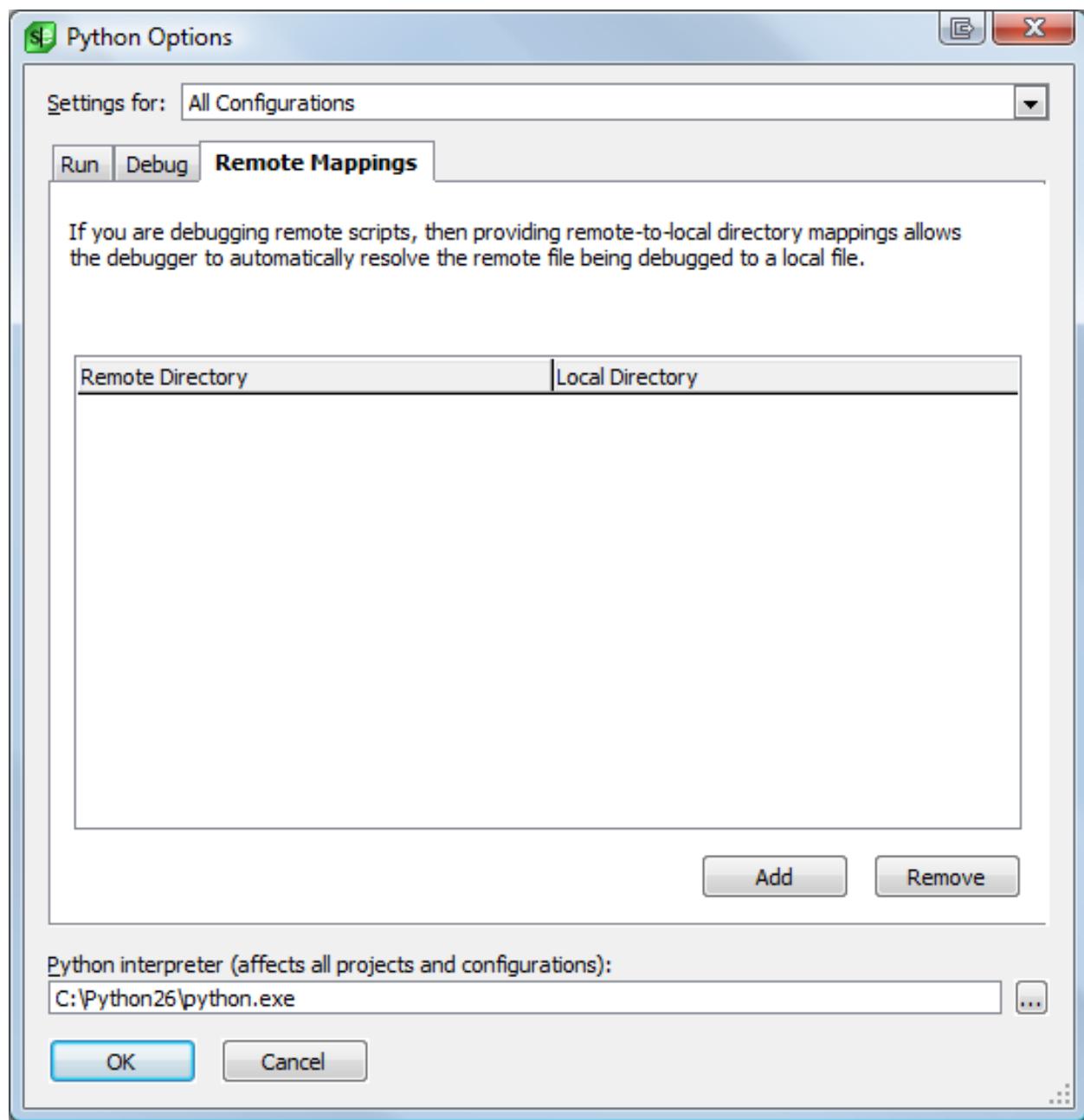
Running and Debugging Python (Pro only)



- **Local host** - Sets the local host interface PTVSD will listen on when starting a debugger session. The default is 127.0.0.1.
- **Specific port** - Sets the port PTVSD will listen on when starting a debugger session. Defaults to 5678.

Remote Mappings

Remote Mappings allow you to define remote-to-local directory mappings. This allows the debugger to automatically resolve the remote file being debugged to a local file.



Click **Add** or **Remove** to manage the list of mappings.

Running and Debugging Perl (Pro only)

To run or debug a Perl script you need:

- Perl 5
- A Perl project

Executing and Debugging a Local Script

Execute your script by selecting Execute from the Build menu **Build → Execute**.

Debug your script by selecting Start from the Debug menu (**Debug → Start**).

Debugging a Remote Script

If your script will run on a remote host, then you will need to copy the perl5db debugger to the remote host in order to make the debugger connection back to your local host possible. The perl5db debugger folder is located under the application folder in resource/tools/perl5db-x.x/. Copy the entire folder to your remote host.

Before attempting to initiate a debug session from the remote host, you must make sure you are listening for a debugger connection on a local interface that can accept connections from the remote host (**Build → Perl Options**, Debug tab). Verify that you are listening by setting Listen in Background (**Debug → perl5db Listen in Background**). Note the host:port that you are listening on by hovering over the listener icon in the lower, right-hand corner of the application window. You should see something like:

```
Listening for perl5db connection on 192.168.0.101:52030
```

Where the host is 192.168.0.101 and the port is 52030.

Note

If your remote host supports ssh, then see [Using an SSH Tunnel to Debug a Remote Script](#) for a convenient way to tunnel remote debugger connections back to your local host.

From the remote host, set up the environment and issue the perl5db command in order to initiate a debugger connection back to your local machine:

```
$ export PERL5DB=BEGIN { require 'perl5db.pl'; }
$ export PERL5LIB=/path/to/perl5db-0.30
$ export PERLDB_OPTS=RemotePort=192.168.0.101:52030
$ perl -d path/to/script-to-debug.pl
```

If everything was set up correctly, then you should get a connection request on your local machine to start a debugging session.

Using an SSH Tunnel to Debug a Remote Script

If your script will run on a remote host that supports ssh, then it is very convenient to set up an ssh tunnel to tunnel debugger connections from your remote server back to your local machine. As an example, if your remote server is called 'myhost.com' and you are listening for a debugger connection at 127.0.0.1 on port 52030, then start an ssh tunnel with the following command:

```
ssh username@myhost.com -R 52030:127.0.0.1:52030
```

This saves you the hassle of having to ensure you have picked the correct interface on which to listen for debugger connections from the remote server.

Follow directions for starting a debugger connection from the remote host as described in [Debugging a Remote Script](#).

Perl Options

You can set a number of options to control the execution and debugging of Perl scripts. Access Perl options by selecting **Build → Perl Options** from the main menu. This menu entry is only available if the active project is a Perl project. Options are broken into three groups, each with its own tab:

- [Run Options](#)
- [Debug Options](#)
- [Remote Mappings](#)

At the top of the Perl Options dialog, you can pick the configuration that these settings apply to. The default is **All Configurations**. However, you can define different settings for separate Run and Debug configurations if you choose.

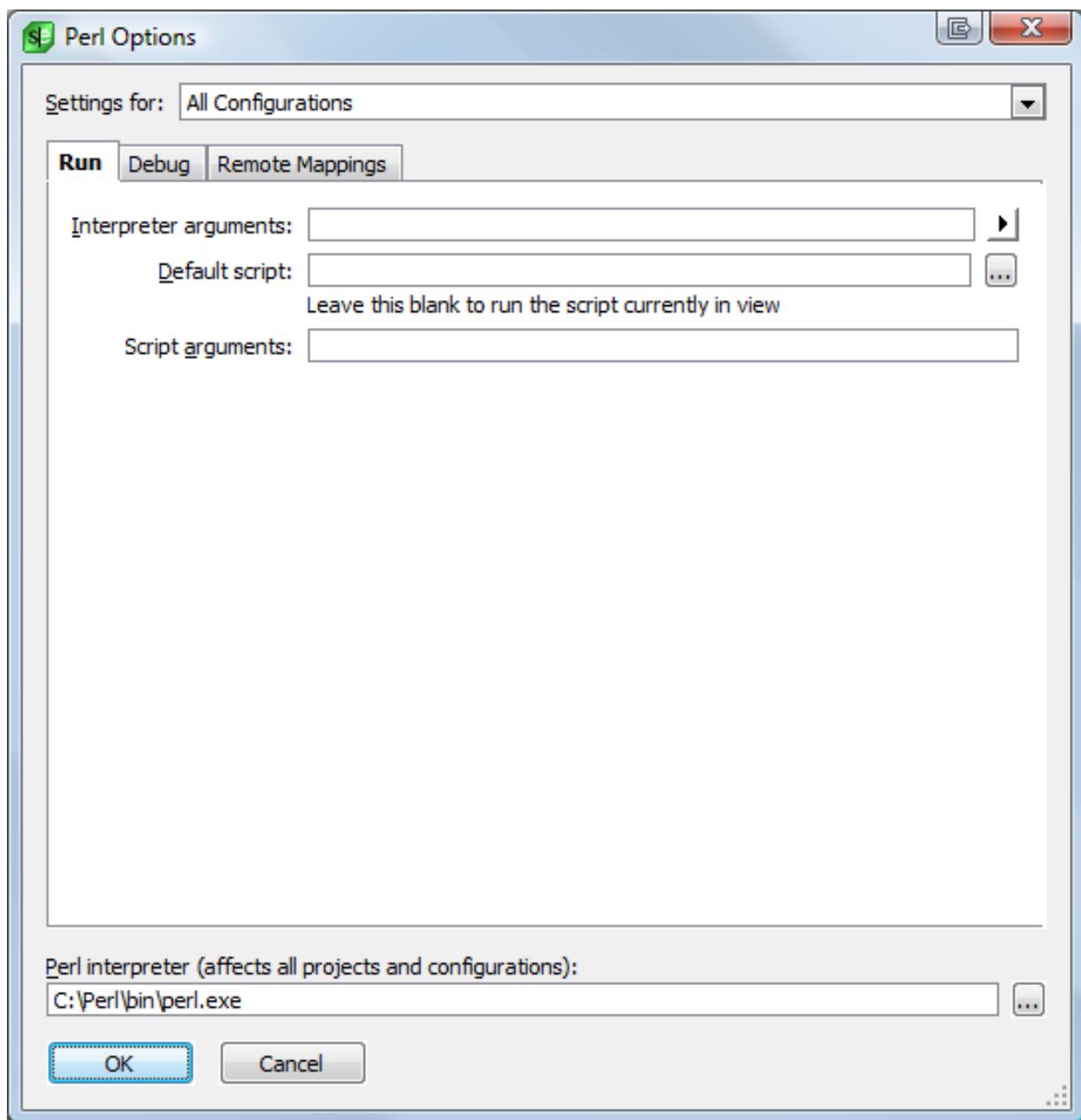
At the bottom of the dialog you can set the **Perl interpreter**. This is the path to the Perl interpreter to use.

Note

The path entered for the Perl interpreter affects all projects and configurations. SlickEdit currently cannot use different interpreters for different projects.

Run Options

Running and Debugging Perl (Pro only)

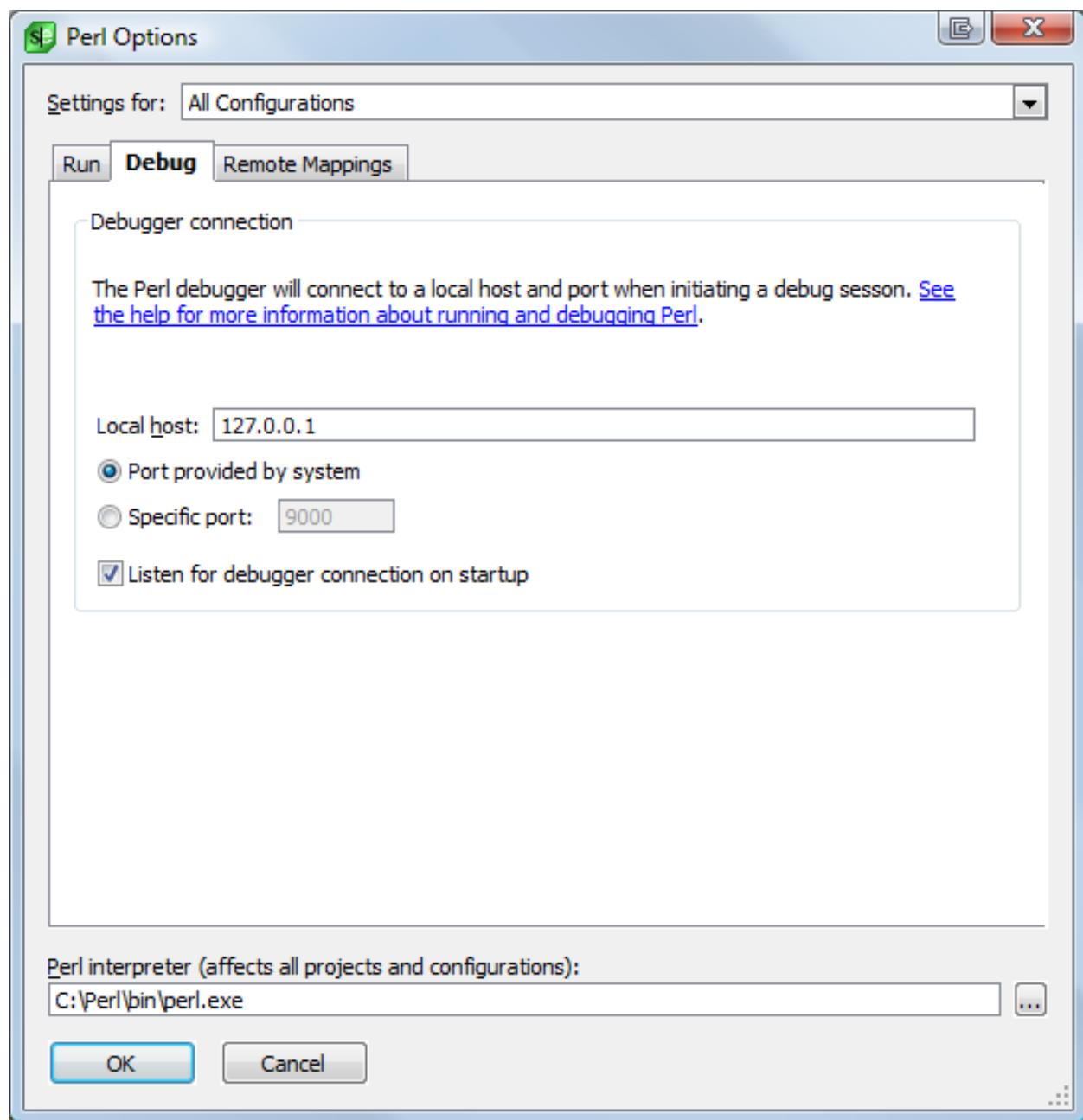


- **Interpreter arguments** - arguments to be passed to the Perl interpreter.
- **Default script** - identifies the file to use to start the execution.
- **Script arguments** - arguments to be passed to the script.

Debug Options

The **Debug** tab contains options that pertain to debugging Perl scripts.

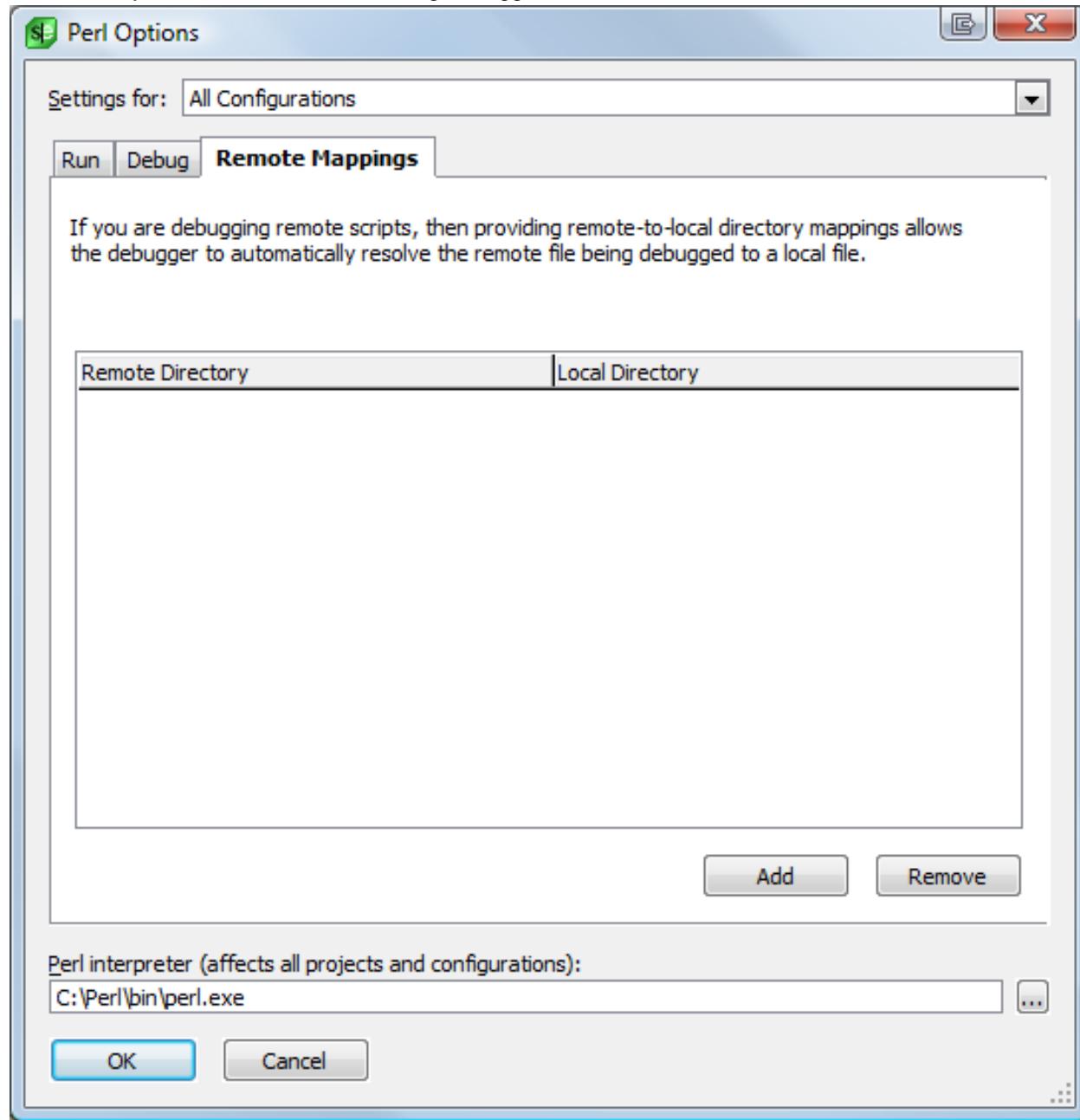
Running and Debugging Perl (Pro only)



- **Show private variable** - Set this option if you want to show private variables in the debugger.
- **Local host** - Set the local host and port that pydbgp will connect to when initiating a debugger session. The default is 127.0.0.1 on an automatically assigned port.
- **Port provided by system** - Use a port that is automatically assigned by the system.
- **Specific port** - use this to specify a port to connect to.
- **Listen for debugger connection on startup** - Set this option if you want to listen for a debugger connection in the background when the project is opened.

Remote Mappings

Remote Mappings allow you to define remote-to-local directory mappings. This allows the debugger to automatically resolve the remote file being debugged to a local file.



Click **Add** or **Remove** to manage the list of mappings.

Running and Debugging Ruby (Pro only)

To run or debug a Ruby script you need:

- A Ruby project
- Ruby 1.8 (1.8.4 or higher for debugging)
- ruby-debug-base 0.9.1 or higher (for debugging)

Note

Use the Ruby 'gem' package manager to install the 'ruby-debug-base' gem package:

```
gem install ruby-debug-base
```

Warning

If you used a MinGW RubyInstaller installer to install Ruby on Windows (very likely), then you will need to download and extract the Ruby-DevKit from rubyinstaller.org and perform the gem install from the msys console by running 'msys.bat':

```
$ cd /c/Ruby187/bin  
$ gem install ruby-debug-base
```

Executing and Debugging a Local Script

Execute your script by selecting Execute from the Build menu **Build → Execute**.

Debug your script by selecting Start from the Debug menu (**Debug → Start**).

Debugging a Remote Script

If your script will run on a remote host, then you will need to copy the rdbgp debugger to the remote host in order to make the debugger connection back to your local host possible. The rdbgp debugger folder is located under the application folder in resource/tools/rdbgp-x.x/. Copy the entire folder to your remote host.

Warning

From the remote host, make sure you have installed the ruby-debug-base gem package described at the beginning of this section. If you do not install the ruby-debug-base package, then you will see an error like the following in the **Build** window:

```
Error: no such file to load -- ruby-debug-base
```

Before attempting to initiate a debug session from the remote host, you must make sure you are listening for a debugger connection on a local interface that can accept connections from the remote host (**Build** → **Ruby Options**, Debug tab). Verify that you are listening by setting Listen in Background (**Debug** → **rdbgp Listen in Background**). Note the host:port that you are listening on by hovering over the listener icon in the lower, right-hand corner of the application window. You should see something like:

```
Listening for rdbgp connection on 192.168.0.101:52030
```

Where the host is 192.168.0.101 and the port is 52030.

Note

If your remote host supports ssh, then see [Using an SSH Tunnel to Debug a Remote Script](#) for a convenient way to tunnel remote debugger connections back to your local host.

From the remote host, set up the environment and issue the rdbgp command in order to initiate a debugger connection back to your local machine:

```
$ export RUBYDB_LIB=/path/to/rdbgp-2.0
$ export RUBYDB_OPTS=HOST=192.168.0.101 PORT=52030
$ ruby -I $RUBYDB_LIB -r $RUBYDB_LIB/rdbgp.rb
path/to/script-to-debug.rb
```

If everything was set up correctly, then you should get a connection request on your local machine to start a debugging session.

Using an SSH Tunnel to Debug a Remote Script

If your script will run on a remote host that supports ssh, then it is very convenient to set up an ssh tunnel to tunnel debugger connections from your remote server back to your local machine. As an example, if your remote server is called 'myhost.com' and you are listening for a debugger connection at 127.0.0.1 on port 52030, then start an ssh tunnel with the following command:

```
ssh username@myhost.com -R 52030:127.0.0.1:52030
```

This saves you the hassle of having to ensure you have picked the correct interface on which to listen for debugger connections from the remote server.

Follow directions for starting a debugger connection from the remote host as described in [Debugging a Remote Script](#).

Ruby Options

You can set a number of options to control the execution and debugging of Ruby scripts. Access Ruby options by selecting **Build → Ruby Options** from the main menu. This menu entry is only available if the active project is a Ruby project. Options are broken into three groups, each with its own tab:

- [Run Options](#)
- [Debug Options](#)
- [Remote Mappings](#)

At the top of the Ruby Options dialog, you can pick the configuration that these settings apply to. The default is **All Configurations**. However, you can define different settings for separate Run and Debug configurations if you choose.

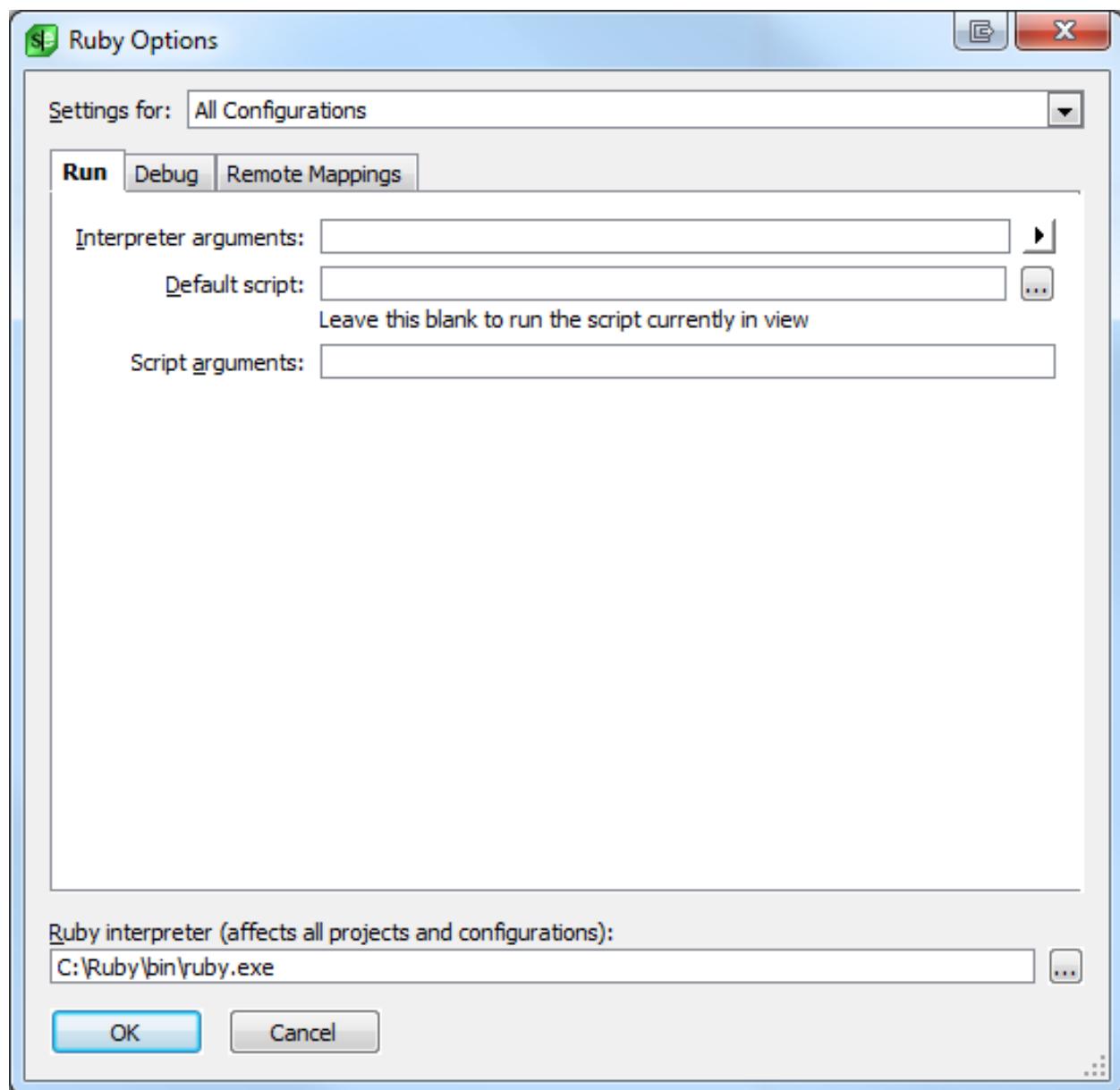
At the bottom of the dialog you can set the **Ruby interpreter**. This is the path to the Ruby interpreter to use.

Note

The path entered for the Ruby interpreter affects all projects and configurations. SlickEdit currently cannot use different interpreters for different projects.

Run Options

Running and Debugging Ruby (Pro only)

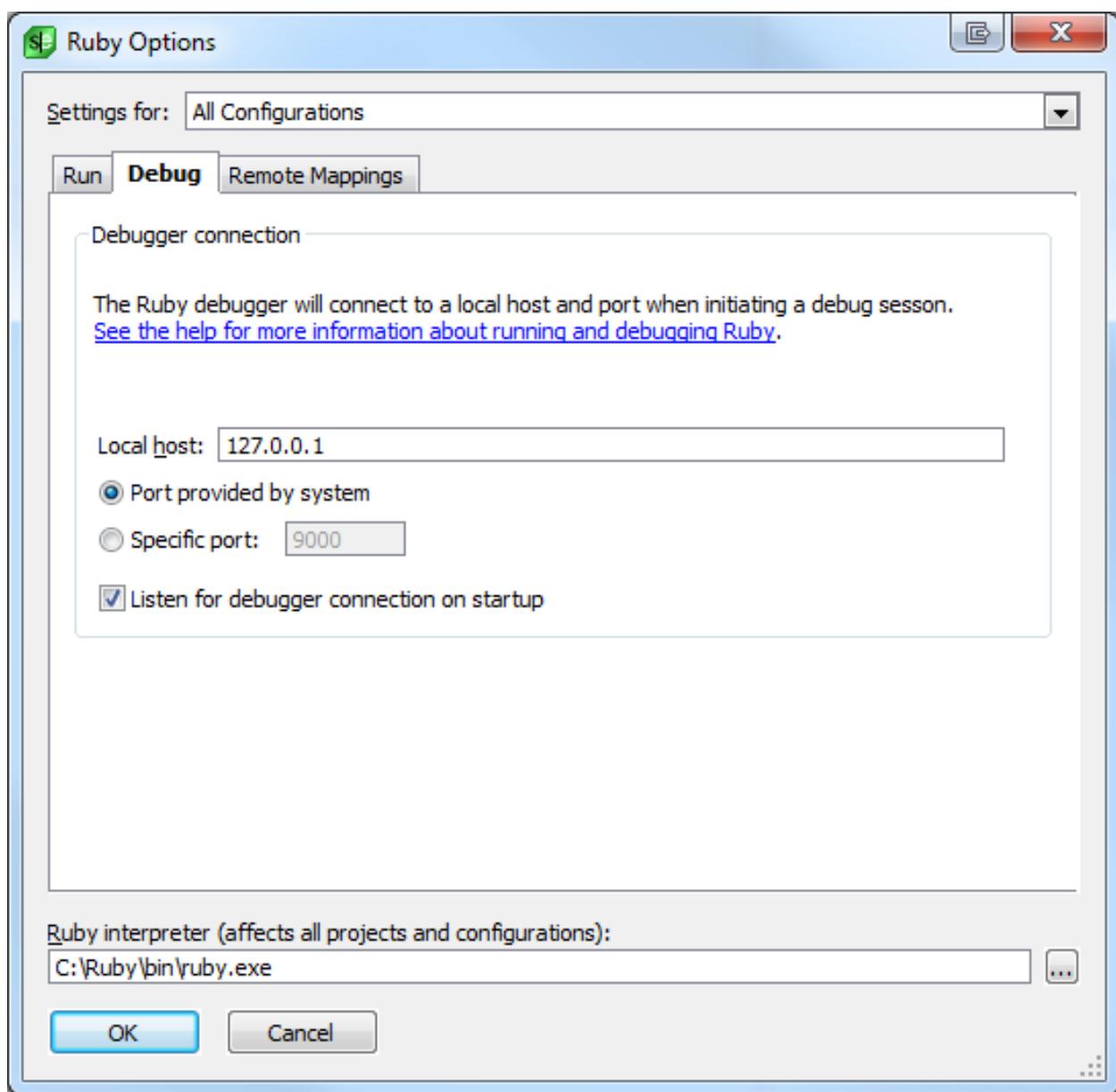


- **Interpreter arguments** - arguments to be passed to the Ruby interpreter.
- **Default script** - identifies the file to use to start the execution.
- **Script arguments** - arguments to be passed to the script.

Debug Options

The **Debug** tab contains options that pertain to debugging Ruby scripts.

Running and Debugging Ruby (Pro only)

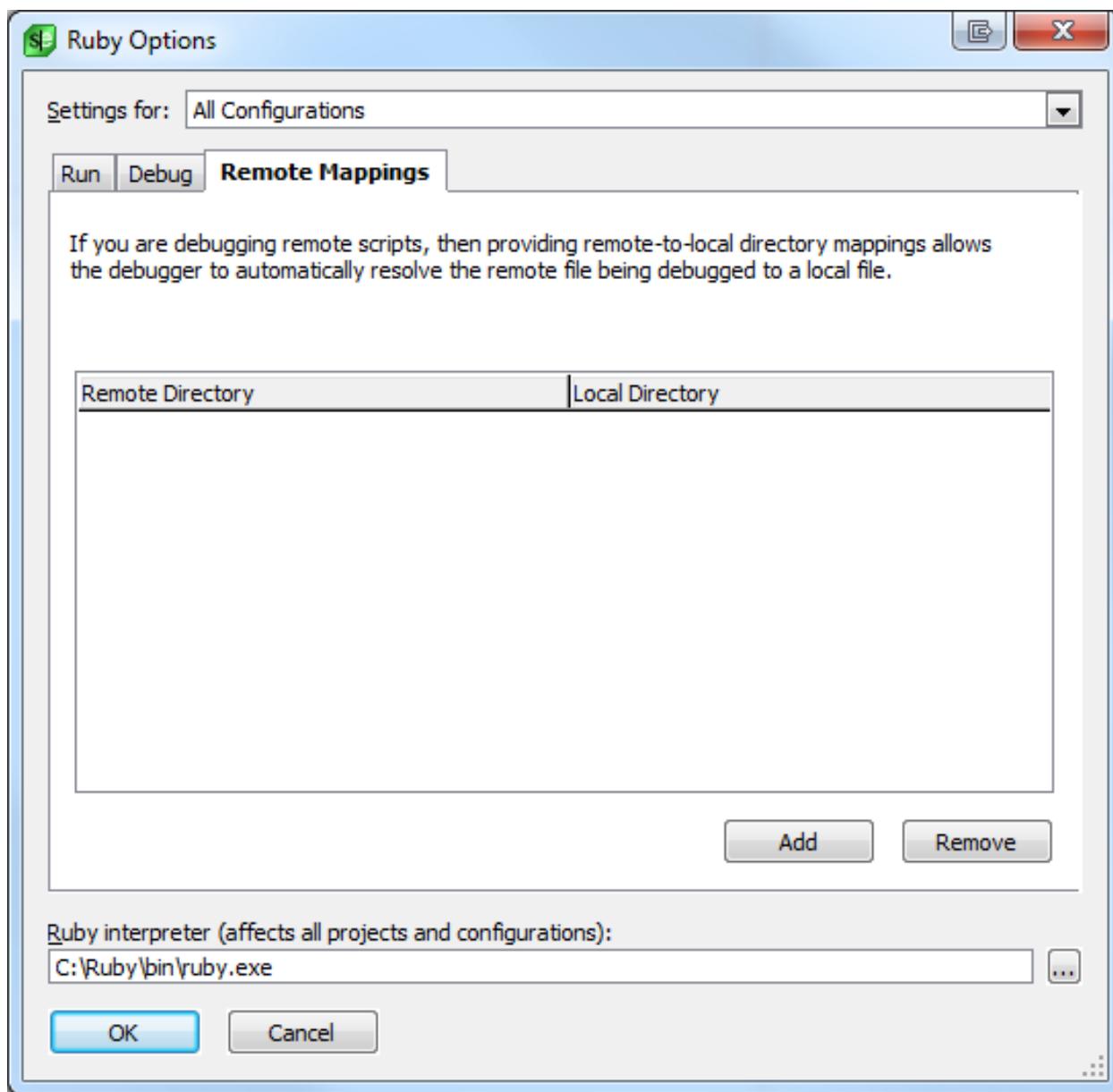


- **Local host** - Set the local host and port that rdbgp will connect to when initiating a debugger session. The default is 127.0.0.1 on an automatically assigned port.
- **Port provided by system** - Use a port that is automatically assigned by the system.
- **Specific port** - use this to specify a port to connect to.
- **Listen for debugger connection on startup** - Set this option if you want to listen for a debugger connection in the background when the project is opened.

Remote Mappings

Remote Mappings allow you to define remote-to-local directory mappings. This allows the debugger to automatically resolve the remote file being debugged to a local file.

Running and Debugging Google Go (Pro only)



Click **Add** or **Remove** to manage the list of mappings.

Running and Debugging Google Go (Pro only)

To run or debug a Google Go program you need:

- An installation of the Go Programming Language which includes go.exe (Unix: go)
- A Google Go project. Create a project and select Google Go for the project type if you have not already done so.

Debugging for Google Go programs uses a customized version of GDB. Please refer to the release notes for specific version information. You can download the customized source from www.slickedit.com/gdb

<http://www.slickedit.com/gdb>.

Note

The GDB shipped with SlickEdit on Windows is based on MinGW. If you prefer to use Cygwin for GDB, you can 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.

Working With Google Web Toolkit Projects (Pro only)

Getting Started

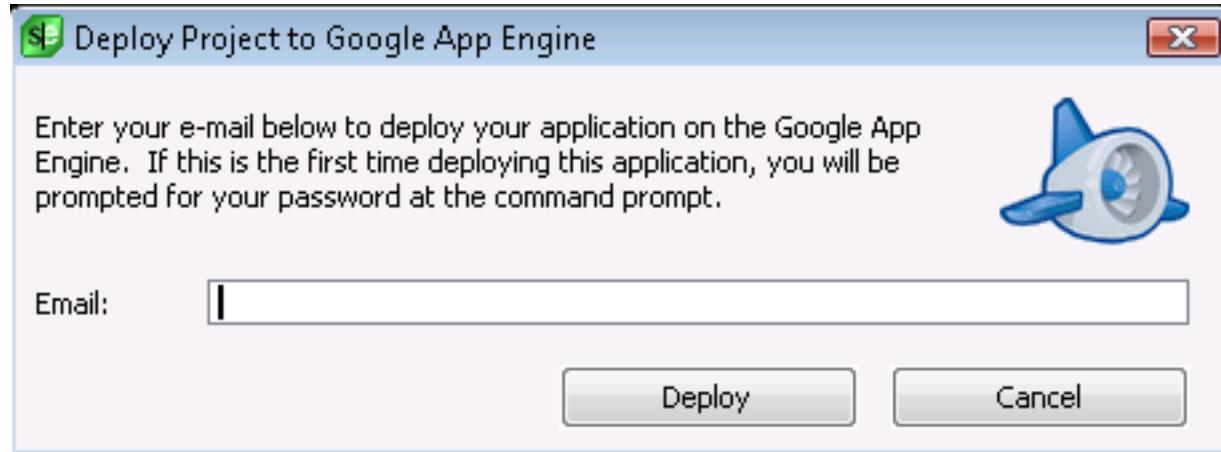
Create a new Google™ Web Toolkit (GWT) Application project at **Project → New → Java → Java - GWT Application** or **Project → New → Python → Python - GWT Application**. This creates an empty workspace and project to which you can add files. The project contains commands to build, debug, and deploy a GWT application. To create a SlickEdit project and workspace from an existing Java GWT Application source tree, select **Project → Open Other Workspace → Ant XML Build File**, and browse to the main Ant build file for your application.

Debugging (Java Only)

By default, the **Debug** command for Java GWT projects uses the Java Debug Wire Protocol (JDWP) with a transport address of 8000. See the **debug** Ant target in the build.xml file for the project in order to customize this command.

Deploying to the Google App Engine

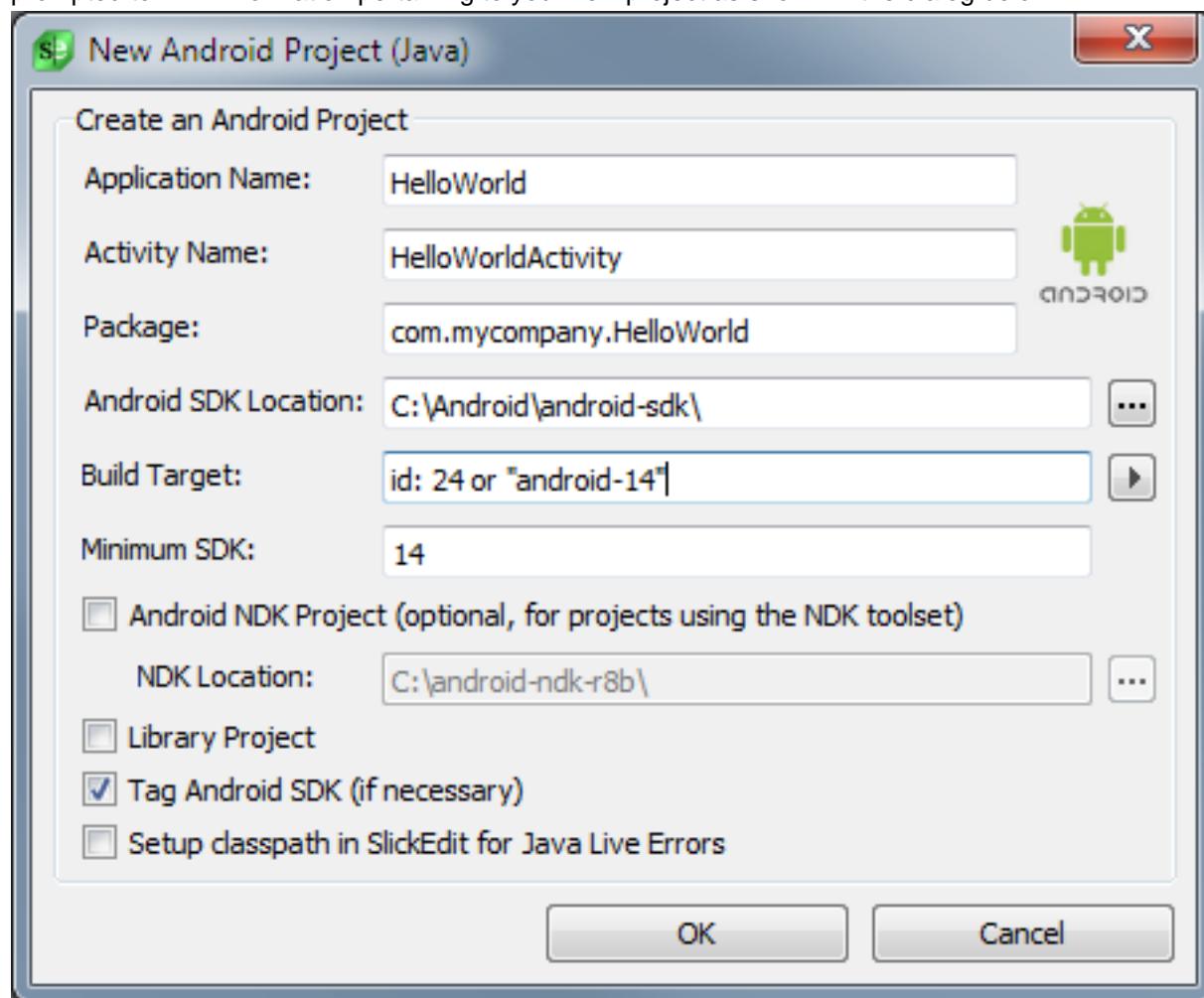
To deploy an application to the Google™ App Engine, use the **Deploy Project** command, found at **Build → Deploy Project**. After entering your e-mail address, a command prompt will be launched and your application will be uploaded. If this is the first time deploying this project, you will be prompted for your password in the command prompt.



Working With Android Projects (Pro only)

Getting Started

Create a new Android project at **Project → New → Java → Java - Android Application**. You will be prompted to fill in information pertaining to your new project as shown in the dialog below.



The **Build Target** field refers to the Android platform library that you would like to build your project against. The menu to the right of this field will allow you to choose a specific target from those available if you have already specified an Android SDK location. If you choose a target from this menu, the **Minimum SDK** field will be automatically generated.

If your Android project uses the Android NDK toolset, select **Android NDK Project** and specify the location of your Android NDK installation in the **NDK Location** field. SlickEdit will add your native-code source files and makefiles to your project, and run the NDK build tools as part of the normal build process for your application.

Note

Set the **ANDROID_NDK_ROOT** environment variable to the root of your Android NDK installation when working with NDK applications in SlickEdit.

Open an existing Android project at **Project → Open Other Workspace → Android Project...**, and browse to your existing `AndroidManifest.xml` file. You will then be prompted for some project information in case you want to update your Android project before opening in SlickEdit.

Note

We recommend that you update your existing project with the location of the local Android SDK (and NDK, if applicable) and your desired build target in order to ensure that SlickEdit is properly configured to work with your project.

Android Toolbar

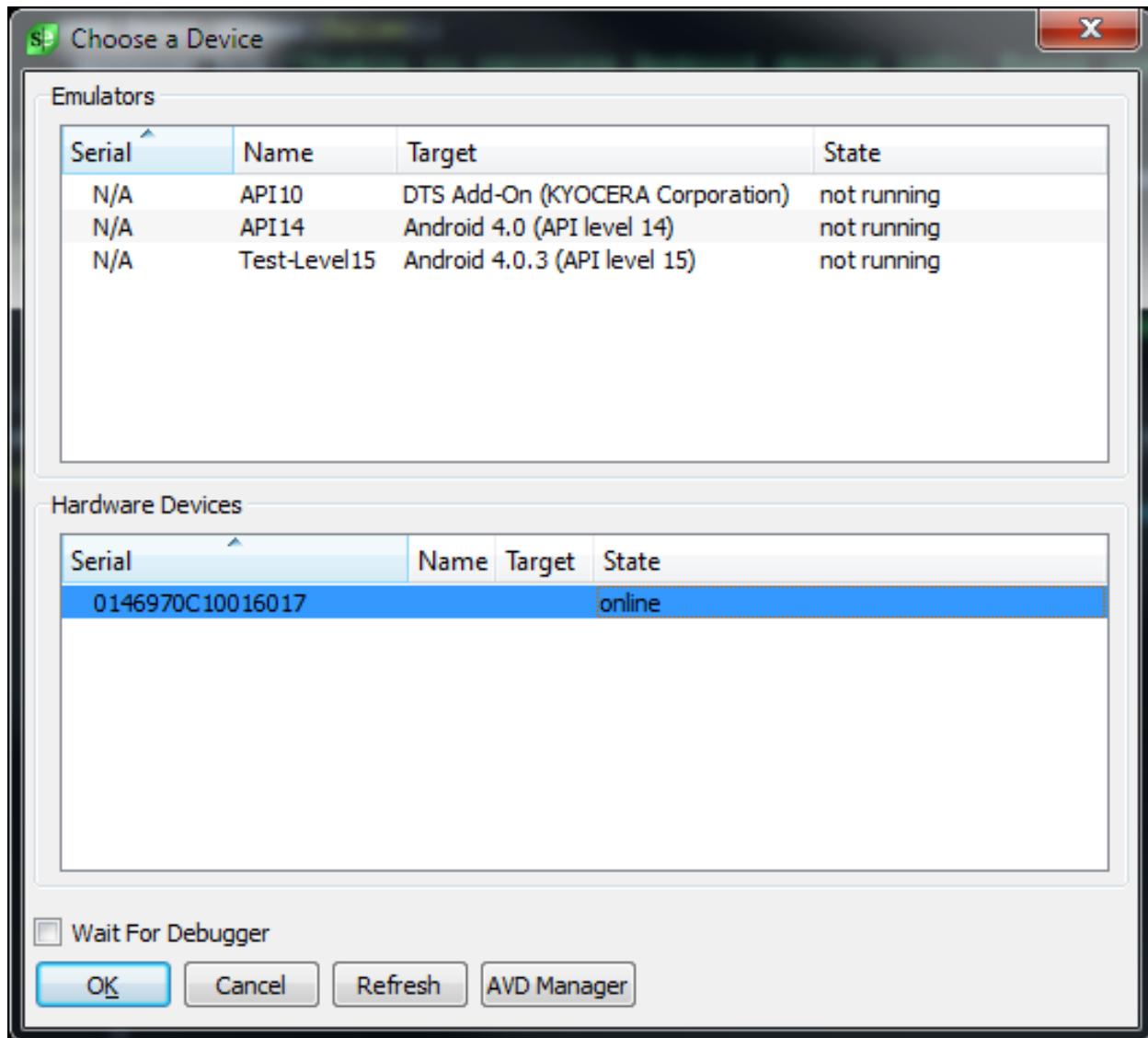
The Android Toolbar will be automatically shown when an Android project is opened in SlickEdit. You can manually show the toolbar at **View → Toolbars**. The toolbar has buttons for opening Android utility tools contained in the Android SDK (from left to right: Android AVD Manager, Android SDK Manager, and DDMS).



Building and Running

Once your Android project is open in SlickEdit, the following build tools will be set up to work with the Ant build files of your project: **Build → Build**, **Build → Rebuild**, and **Build → Clean**. SlickEdit will build the debug or release configuration for your project depending on which configuration is set as active (**Build → Set Active Configuration**).

Install and run your Android application on a device at **Build → Execute on Device...**. This will launch the device chooser shown below, where you can select any Android emulator or connected hardware device for testing.



- **Wait for Debugger** - When selected, this will prevent the application from running until a debugger attaches to the process.
- **AVD Manager** - Launches the Android Virtual Device Manager tool, located within the Android SDK. This tool allows you to create, edit, or delete Android emulators.

When you select a device, your Android application will be installed and run. If the device chosen is an emulator which is not currently running, then the emulator will be automatically started for you, and the application will be queued up to start when the emulator is determined to be ready.

Debugging

Java

You can use the SlickEdit debugger to debug an Android application on an emulator or hardware device

by attaching the debugger to the running process.

1. Once an application is running, activate the Android DDMS tool. If you have the Android Toolbar shown in SlickEdit, you can simply click the DDMS icon. If you need to manually open the utility it is located at `ANDROID_SDK_DIR/tools/ddms`.
2. In DDMS, locate the running application you wish to debug in the process list. If you select the process you will see two ports listed, one of which will be port 8700 (by default).
3. In SlickEdit select **Debug** → **Attach Debugger** → **Attach to Java Virtual Machine...**, and use either of the ports found in step 2 as the port, and use **localhost** as the host. Click **OK** to attach the debugger.
4. You can now use the SlickEdit debugger to enable/disable breakpoints and step through the execution of your Android application just like you would any other Java application.

C/C++

You can use the SlickEdit debugger to launch a native debugging session with GDB if you want to debug a native-code library in your Android application. The tools used for NDK debugging require that you have `bash` and `make` on your system and in your PATH.

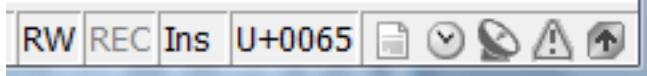
1. Create a GDB Debugger Configuration at **Tools** → **Options** → **Debugging** → **Configurations** for the GDB of your choice. The GDB builds for use with the Android NDK can be found under `ANDROID_NDK_DIR/toolchains/`. Make sure that you mark this configuration as the **Default native debugging configuration**.
2. Build the **Debug** configuration of your Android application and run it via **Build** → **Execute on Device....**
3. Select **Debug** → **Attach Debugger** → **GDB** → **Attach to Android Application Process (GDB)** (**Debug** → **Attach Debugger** → **Attach to Android Application Process (GDB for NDK)** on Mac and Linux) and the debugger will automatically use your default native debugging configuration to attach to the running Android process. If there are multiple online Android devices you will be prompted to select which device you would like to debug.

Editing Features

This chapter describes the editing features of SlickEdit that are not specific to a particular language.

Notifications

To help you better understand what's going on in the editor, SlickEdit displays notification icons in the status area (see image, below). A pop-up message describing the activity is briefly displayed.



Icons are displayed for the following activities:

- **Feature Notifications** - uses the document icon to inform you about automatic editing operations that were performed by the editor, including features like Syntax Expansion and Comment Wrapping. Icons in the pop-up provide access to the options screen and the help for that feature.
- **Background Processes** - uses the clock icon to notify you when SlickEdit is performing operations in the background. This helps you correlate high levels of system activity with otherwise invisible operations in the editor.
- **Debugger Listener(Pro only)** - displays the satellite icon when a debugger is listening for a connection, another icon is displayed.
- **Warning Notifications** - uses an exclamation point in a yellow triangle icon to notify you about important information. Where Feature Notifications provide helpful information about what the editor is doing, Warning Notifications are about the result of operations. Consequently, you can't disable the Warning Notifications.
- **Update Notifications** - displays the SlickEdit "shield" with an up-arrow indicating that a hot fix or new version is available.

For information on configuring **Notifications**, including a description of the notification levels, see [Notification Options](#).

Feature Notifications

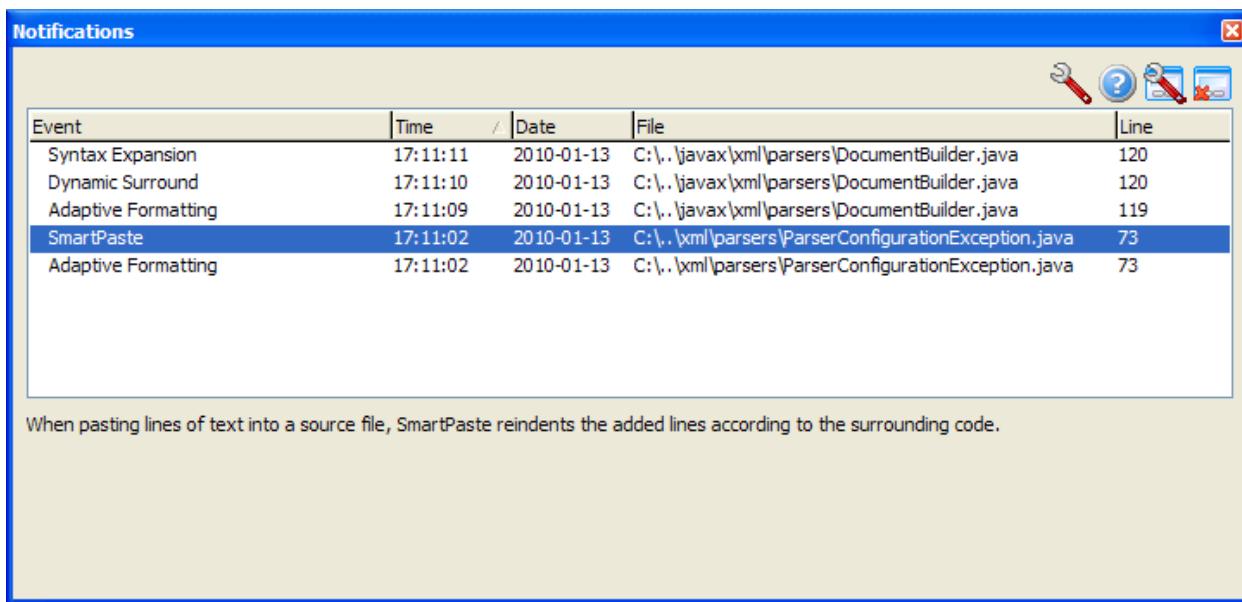
SlickEdit offers five different levels of notifications. Some notification mechanisms are more disruptive than others and are useful for features that have more surprising results. Once you are used to the feature, you can select a different, less disruptive notification or turn off notifications completely.

Notification Tool Window

Regardless of the notification level selected, all feature notifications are added to the **Notification** tool window. To view this tool window, select **View → Tool Windows → Notifications**.

Files, Buffers, and Editor

Windows



The screenshot shows the 'Notifications' tool window in the Eclipse IDE. The window has a title bar with the title 'Notifications' and standard window controls. Below the title bar is a toolbar with icons for search, help, and configuration. The main area is a table with the following data:

Event	Time	/	Date	File	Line
Syntax Expansion	17:11:11		2010-01-13	C:\..\javax\xml\parsers\DocumentBuilder.java	120
Dynamic Surround	17:11:10		2010-01-13	C:\..\javax\xml\parsers\DocumentBuilder.java	120
Adaptive Formatting	17:11:09		2010-01-13	C:\..\javax\xml\parsers\DocumentBuilder.java	119
SmartPaste	17:11:02		2010-01-13	C:\..\xml\parsers\ParserConfigurationException.java	73
Adaptive Formatting	17:11:02		2010-01-13	C:\..\xml\parsers\ParserConfigurationException.java	73

When pasting lines of text into a source file, SmartPaste reindents the added lines according to the surrounding code.

A list of all feature notifications for the current editing session in the current workspace is displayed. Each line includes additional information, such as when the feature was activated and the file and line number where changes were made. Selecting a notification in the list displays more information about the feature. You are also be provided with buttons to configure the feature or read more about it in the help documentation. There is also a button to configure notification settings for all features. If you wish to clear all notifications in the tool window, use the right-most button.

Files, Buffers, and Editor Windows

When you edit a file, it is loaded into a buffer. A 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. You can split windows, so that the same buffer is visible in multiple windows.

Note

When you split a window, both windows display the same buffer. Any changes made in one are visible in the other.

Each buffer has an associated tab in the [File Tabs](#) tool window, which displays the name of the file. If you open a file that is already open, the existing buffer is displayed. Therefore, the terms "file" and "buffer" are sometimes used interchangeably.

SlickEdit® provides two distinct approaches to managing buffers and editor windows, controlled through the **Files per window** option (**Tools** → **Options** → **Editing** → **Editor Windows**). 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** - This maintains a one-to-one correspondence between a buffer and an editor window. Each buffer is displayed in its own editor window. Closing a window closes the associated file, and closing a file closes the associated window. This is the default behavior for SlickEdit, preferred by many for its simplicity. In this mode, switching buffers and switching windows does the very same thing, since each buffer has its own window ([Switching Between Buffers or Windows](#) for more information.)
2. **Multiple files share 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. Auto-restore has better performance when restoring many buffers because each buffer doesn't need a window. Editing and closing many files (100 or more) is also faster. 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 Open Files...** (or the **list_buffers** command) to view a list of the buffers and select one.

This approach typically only appeals to users of emulations such as Brief, SlickEdit, Epsilon, GNU, Emacs, and Vim. You can switch between these locations using the `next_window` and `prev_window` commands (see [Switching Between Buffers or Windows](#) for more information).

Tip

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

When a buffer is modified (changed and not yet saved), an asterisk will be displayed to the right of the file name in the title bar and the Document tab.

Managing Windows

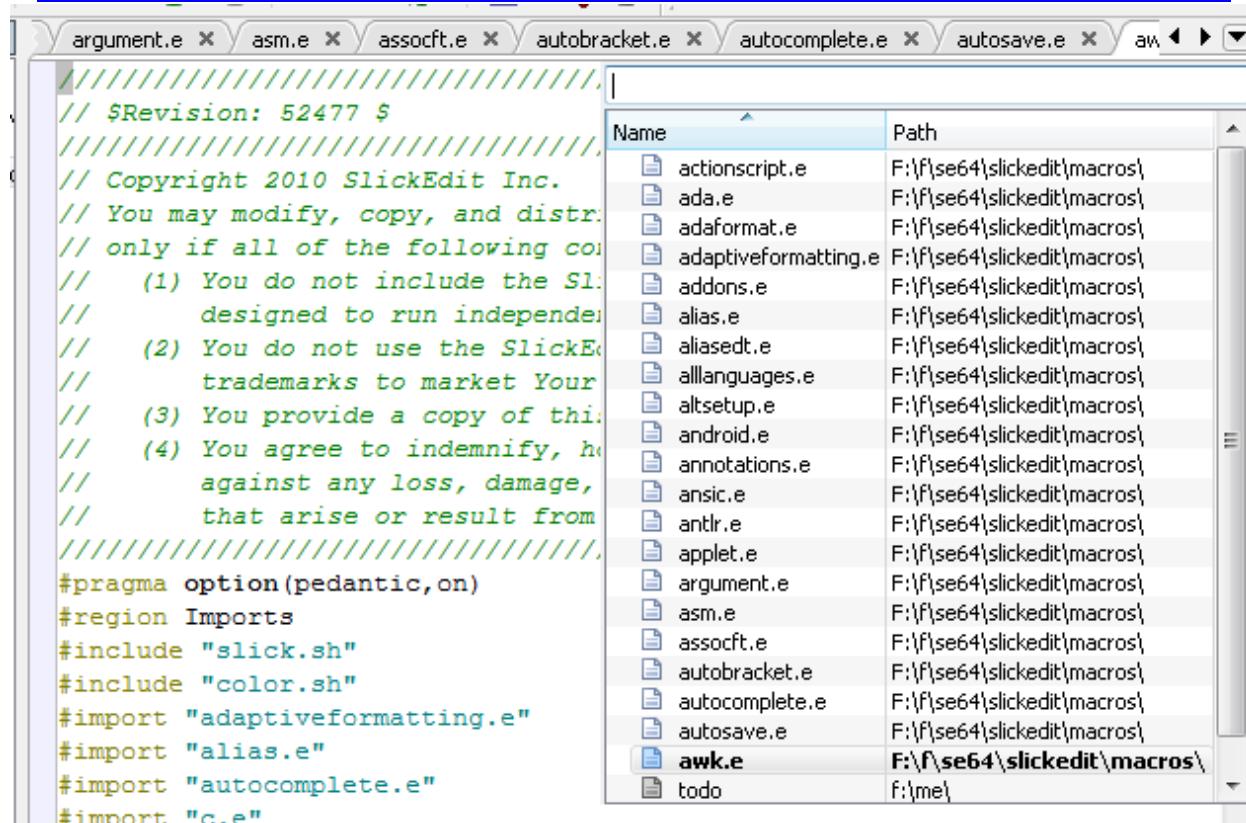
Document Tabs

When you open a file in SlickEdit®, by default, an associated Document tab is displayed above the editor window to indicate the name of the file. You can click on a document tab to select the buffer you want to edit.

Note

If you want to see a document tab per buffer (probably because you are using "Multiple files share window"), try using the **File Tabs** tool window. See [File Tabs](#) for more information.

You may want to hide the Document tabs when you only have one edit window. To do this, set the **Zoom (hide tabs) when one window** to **Always** at **Tools** → **Options** → **Editing** → **Editor Windows**.



For each window you create (not buffer), you will see a document tab. If there isn't enough space for all of the document tabs to be displayed, a left and right arrow icon is drawn on the right, allowing you to scroll the tabs. A down-arrow icon is always visible, allowing you to select a file from a list of the open files. This is a convenient way to select a file when you have a lot of files open and some tabs aren't visible.

Tip

You may also find that using the **Files** tool window provides a convenient way to view a list of open buffers and select one for editing (see [Document Dialogs and Tool Windows](#)).

When a buffer is modified (changed and not yet saved), an asterisk will be displayed on the right of the document tab.

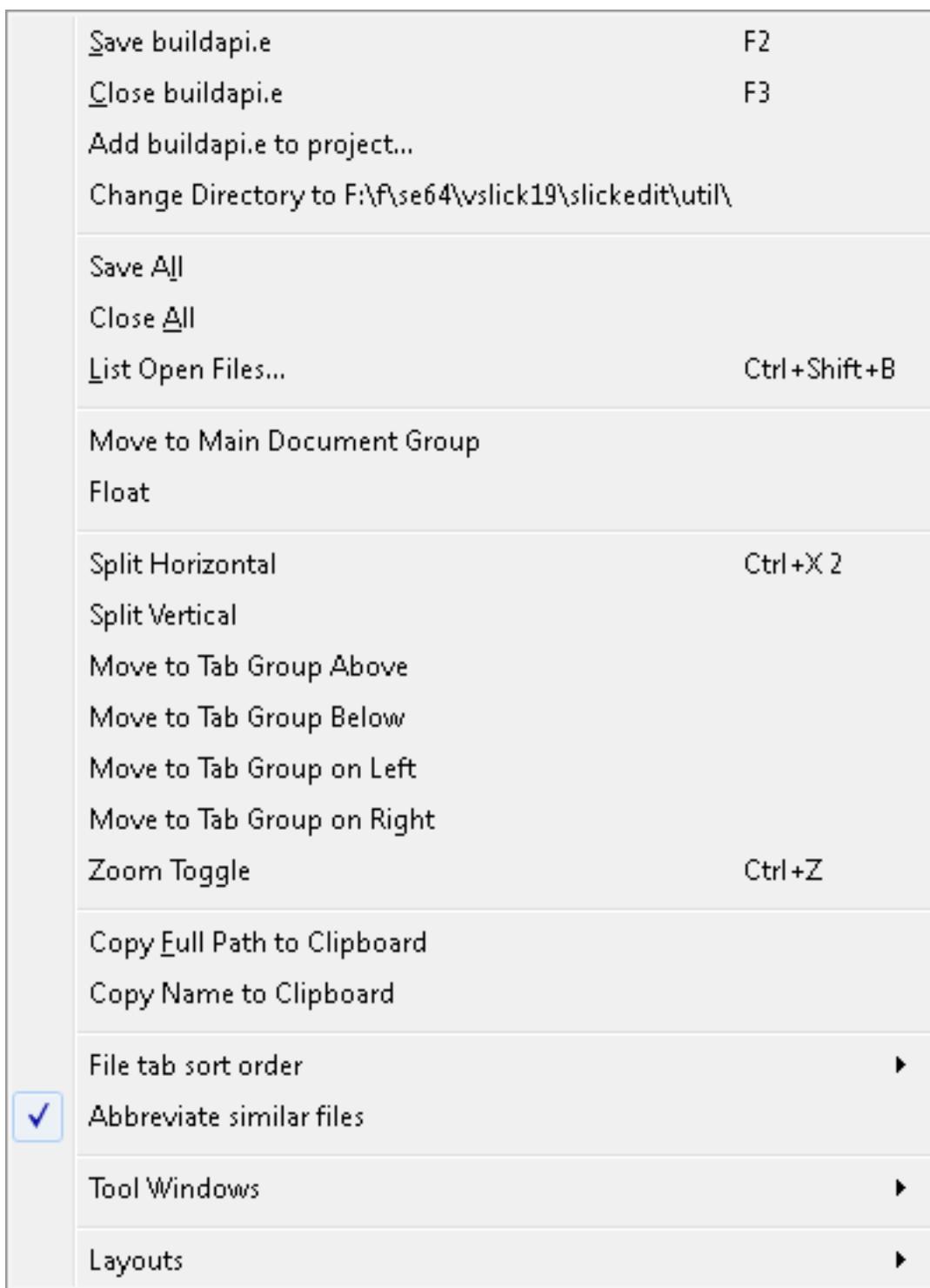
When you create a new, unnamed "scratchpad" buffer, it is indicated by the text "Untitled" in the document tab along with a number that indicates the internal ID. You can create a scratchpad buffer by using the menu item **File** → **New** and not naming the file.

If you prefer to keep your hands on the keyboard for buffer/file navigation, two commands are available: **next_doc_tab** and **prev_doc_tab**. Use these commands to navigate through the document tabs in the order they are displayed. Both circle around to the other end when you are on the last item. These commands are not bound to keys by default. To create key bindings for these commands, see [Creating Bindings](#).

Document Tab Context Menu

The right-click context menu in the **Document Tabs** tool window provides operations for saving files, closing files, splitting windows, and controlling the appearance of the Document tabs.

The right-click context menu for the document tabs is shown below.



The items available in the right-click context menu are outlined below. Some items apply to the file specified by the document tab underneath the mouse. Others apply to the document tabs as a whole.

- **Save <file>** - Saves the file specified by the document tab under the mouse.
- **Close <file>** - Closes the file specified by the document tab under the mouse.

- **Add <file> to project...** - Adds the file to a project in the workspace. You are prompted to choose which project.
- **Change Directory to <path>** - Changes directory to the path specified by the document tab under the mouse.
- **Browse Folder <path>** - Open the Folder corresponding to the path specified by the document tab under the mouse in the system File Manager (for example, Explorer on Windows or Finder on macOS).
- **List Files in Folder <path>** - Show a directory listing for the path specified by the document tab under the mouse using SlickEdit's File Manager.
- **Save All** - Saves all modified files.
- **Close All** - Closes all open files.
- **Close Others Document Tabs** - Closes all open files except for the one specified by the document tab under the mouse.
- **List Open Files...** - Shows the **Files** tool window See [Files Tool Window](#) for more information.
- **Move to Main Window Group** - Moves floating document tab from floating window group to main window group. This menu item is not present on context menu for main window group document tabs.
- **Float** - Floats (undocks) the document tab under the mouse.
- **Split Horizontal** - Creates another window viewing the document tab under the mouse. The new window is created below the current document window.
- **Split Vertical** - Creates another window viewing the document tab under the mouse. The new window is created to the right of the current document window.
- **Move to Tab Group Above, Below, on Left, on Right** - Moves the document tab under the mouse to the document tab group above, below, on left, or on right respectively.
- **Zoom Toggle (Show/Hide Document Tabs)** - Causes the document tab under the mouse to expand to take up all of the document area and the document tabs to disappear. You can also toggle the existing zoom state by using the **Window → Zoom Toggle (Document Tabs)** menu item.
- **Copy Full Path to Clipboard** - Copies the full path of the file on the document tab under the mouse to the clipboard.
- **Copy Name to Clipboard** - Copies the name without the path of the file on the document tab under the mouse to the clipboard.
- **File tab sort order** - The default order for the file or document tabs is alphabetic. This makes it easy to predict where a file or document tab will be based on the file name. You can also change this value by going to **Tools → Options → Editing → Editor Windows**. This option has several possible values: **Alphabetical**, **Most recently opened**, **Most recently viewed**, or **Manual**. For more information see [File Tab Sort Order Options](#).
-

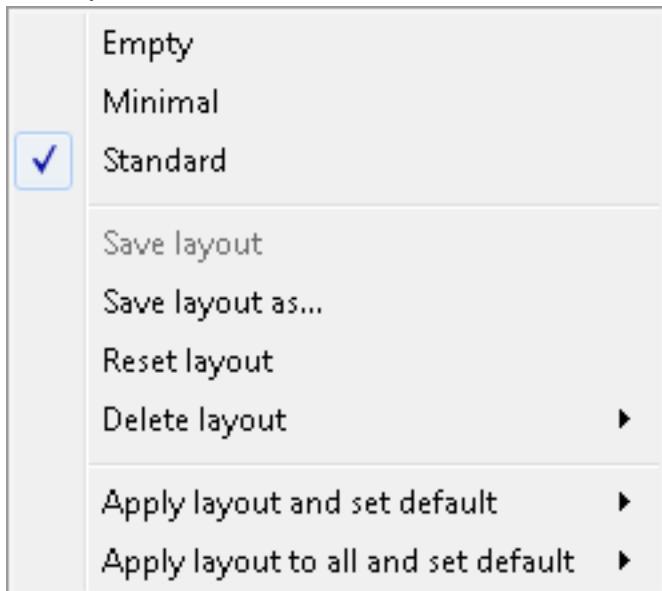
Abbreviate similar files - When the file or document tabs are sorted alphabetically, by default, the tabs do not show the complete name of the file when adjacent files differ only by file extension. This saves space and provides better visibility for associated files. For file names to be abbreviated in this style, their paths and base file name must match exactly. For example, C:\rectangles\BorderRectangle.cpp would not abbreviate with C:\src\include\BorderRectangle.h. You can also set this option by going to **Tools → Options → Editing → Editor Windows**.

- **Tool Windows** - Displays menu of tool windows. Selecting a tool window will either activate the tool window or create a new instance. Not all tool windows support multiple instances.
- **Layouts** - Displays menu for creating and customizing the default tool window layout for dragged out document tabs. This menu is not present on context menu for main window group document tabs.

Customizing the Default Layout Applied to Dragged out Document Tabs

When you drag out a document tab, a default tool window layout is applied. Use the Layouts menu found on the context menu of a document tab in a floating window group to customize the default layout applied. Note that the context menu of a document tab of the main window group does not have the Layouts menu.

The Layouts menu is shown below:



- **Save layout** - Saves the tool window layout for current layout.
- **Save layout as...** - Saves the tool window layout to a layout name of your choosing. Turn on the "Set as default layout" check box to also set the default layout applied to dragged out document tabs.
- **Reset layout** - Restores the current tool window layout to its original state before modifications were made.
- **Delete layout** - Deletes the current layout. System tool window layouts can't be deleted.

- **Apply layout and set default** - List all layouts and allows you to set the default layout and apply the layout to the current floating window group.
- **Apply layout to all and set default** - List all layouts and allows you to set the default layout and apply the layout to all current floating window group.

Docking Tool Windows to Floating Window Groups

Document tabs can be drag and dropped outside the main window group (also called the application main window). This is very useful when you have multiple monitors. There are couple of ways to dock tool windows to floating window groups.

Use the **View → Tool Windows** menu to create a new instance of a tool window or activate a tool window. Note that not all tool window support duplicates.

To duplicate a tool window, right click on the tool window title bar and select **Duplicate**. This will create a new floating instance of that tool window. Now you drag the new tool window by the tool window title bar to the floating window group and dock it.

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, click **Tools → Options → Appearance → General**, then set the value of the option **Window left margin** to 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, click **Window → Split Horizontally (Ctrl+H or hsplit_window command)** or **Window → Split Vertically (vsplit_window command)**. You can also right-click on the document or file tab of the current window to perform the same operations.

To view two different editor windows side-by-side, follow these steps:

- Open the two files. This will create two document tabs (unless you are using the **Multiple windows share window** option).
- Now start dragging the first document tab over the second documents edit window. You will see what we call docking guides displayed to help indicate where the document tab will be docked (on left, on right, above, below, or same document tab group).
- While the mouse is over the docking guide portion indicating either on left, on right, above, or below, let go of the mouse.

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.

Tiling Windows

To resize and rearrange the open editor windows so they don't overlap, click **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, click **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.

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 [Files, Buffers, and Editor Windows](#)). If you have selected **One file per window**, switching buffers and switching windows is the same thing. If you have **Multiple files share a window** selected, 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.

Note

Under the Gnome desktop environment, **Smart next window** may not work correctly when the mouse option 'Highlight the pointer when you press Ctrl' is enabled.

Smart Next Window is on by default (**Tools** → **Options** → **Editing** → **Editor Windows > Smart next window style**). 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 buffers in the order they are displayed in the File Tabs tool window. **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 using a circular list in the order they were last used. A new file is inserted after the current file. These commands are bound to the menu items **Document** → **Next Buffer** (**Ctrl+N**) and **Document** → **Previous Buffer** (**Ctrl+P**). If you are using **Multiple files share a window**, then these operations will cycle through the buffers using the currently active window. If you are using **One file per window**, then these operations will cycle through the windows.

Note

While using **push-tag** (**Ctrl+Dot**) to navigate from one file to another does alter the order of the buffer list, using **pop-bookmark** (**Ctrl+Comma**) to navigate back does not. So, if you have a

buffer list of A, B, C and you use **Ctrl+Dot** to open D the list will be A, B, C, D. When you press **Ctrl+Comma** to return to C, the buffer list will still be ordered: A, B, C, D. So you will need to use **next_buffer** to get to D.

- **forward/back** - Navigates through previously visited locations in the code as a linear list of locations. A new location is created by any navigation action except for simple cursor navigation, like up, down, left, right, page up, page down, etc. 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)**.

- **next_doc_tab/prev_doc_tab** - Navigates through the windows in the order they are displayed in the Document tabs. **next_doc_tab** moves to the right and **prev_doc_tab** moves to the left. When the end is reached, these commands switch to the next or previous Document tab group. These commands are not bound to keys by default.

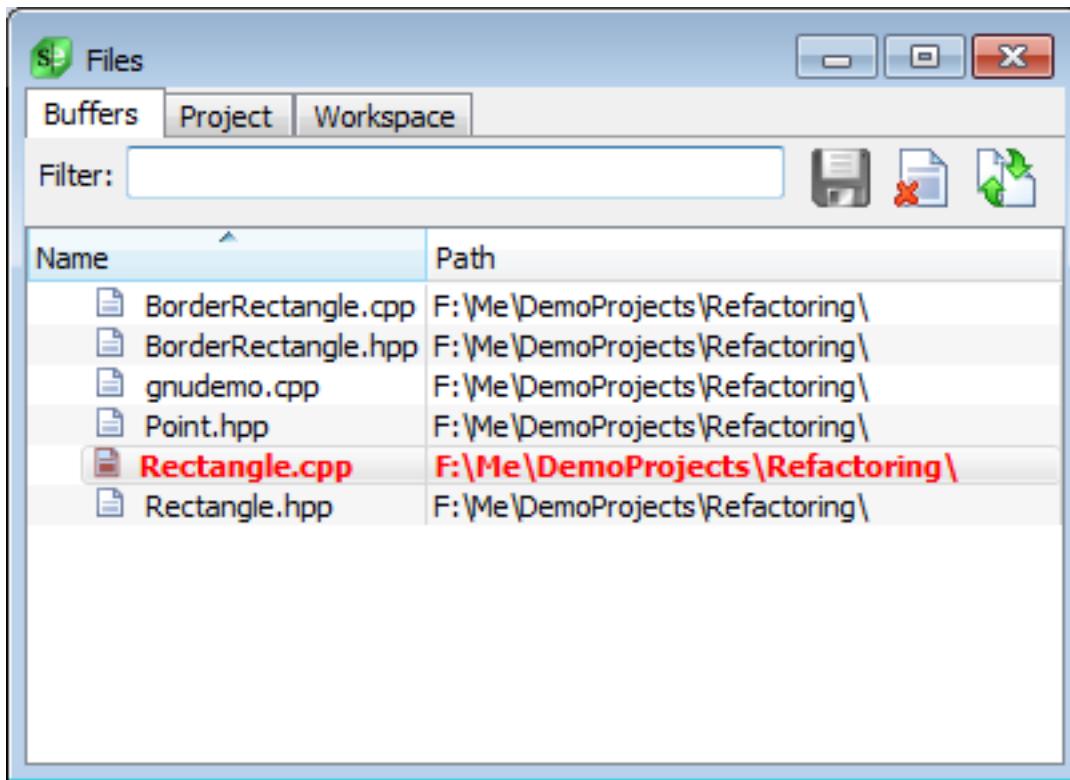
Listing Open Files

To view and work with a list of open files in SlickEdit®, click **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.

Note

For documentation purposes, the word "files" generally includes both files and buffers.

Switching Between Buffers or Windows



Tip

- 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** buttons or the view settings on the right-click context menu to change the display.

Using the buttons on the Files tool window (or the right-click context 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 are listed in a red. You can click the Disk bitmap 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** → **Editing** → **Editor Windows** → **Files per window**), 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 [Files Tool Window](#) for more details about filtering and searching the Files list.

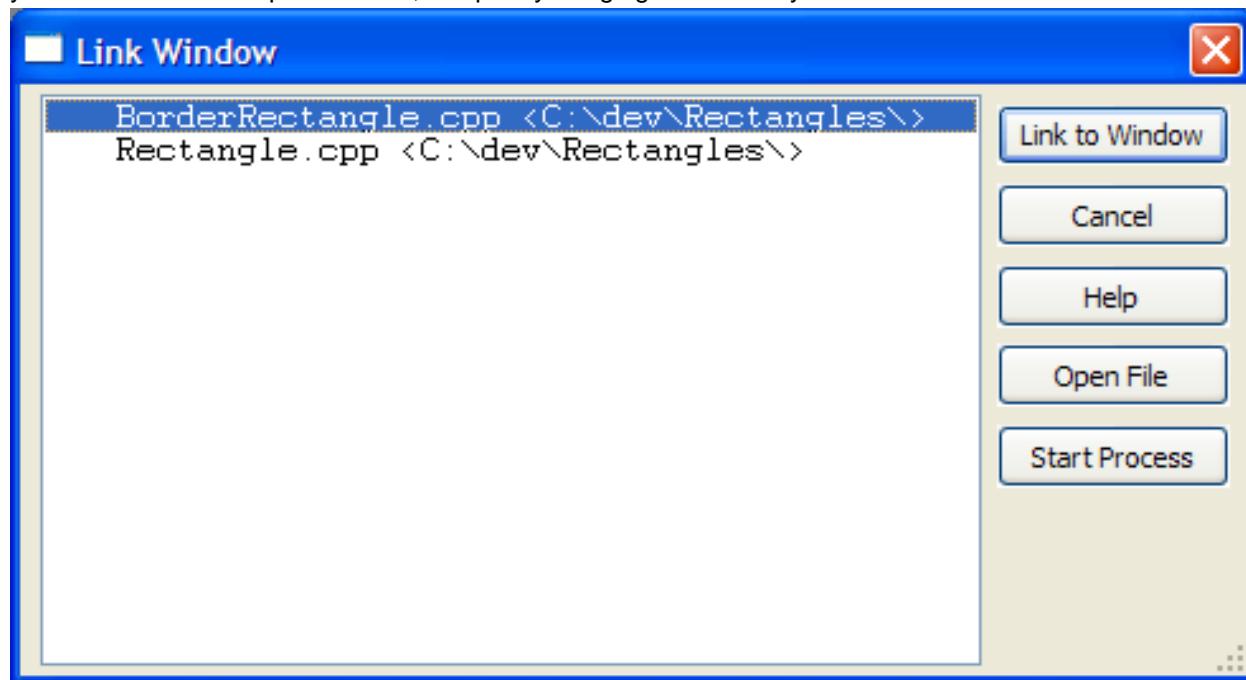
Tip

You can collapse display of the buttons and the Filter text box by clicking the **Minus** button 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 buttons and Filter text box again, click the **Plus** button. Collapsing the buttons 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 [Document Dialogs and Tool Windows](#).

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

If you have set **Files per window** to **One file per window**, closing a file is the same as closing a window. If you have set it to **Multiple files share window**, then windows and files are closed separately. When you close a modified buffer, you will be prompted to save the contents.

- To close an editor window, click **Window → Close** or use the **close_window** command.
- To close a single file, select **File → Close** or use the **close-buffer** command.
- To close all files, select **File → Close All** or use the **close_all** command.

You can also access these items from the right-click context menu on the File Tabs tool window, as well as an additional operation, **Close Others**, which closes all open files except for the one selected.

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

Basic Editing

Overview

SlickEdit® provides familiar operations for selecting, copying, moving, and operating on text, with enhanced capabilities to meet the needs of developers.

The available editing features depend on the current emulation. Different editors provide different capabilities, and SlickEdit attempts to match these features in each emulation. For example, the Brief emulation provides a **brief_iselect_char** command that will start an inclusive character selection. This is an operation familiar to Brief users that the other emulations don't necessarily provide. However, CUA is the default emulation mode for SlickEdit, so the operations described in this section are based on that mode. See [Emulations](#) for more information about emulation modes.

In addition to the basic and advanced editing features described in this section, SlickEdit provides many more text editing-related features that can be managed on a per-language basis. See topics in the [Editing Features](#) chapter for information.

SlickEdit and Selections

Many editing operations are performed on selected text, so you need to know how selections work in SlickEdit in order to gain the power of its editing features. There are several types of selections, and some are handled differently than others regarding operations and features. In particular, SlickEdit handles line selections differently than most other editors. See [Selections](#) for more information.

SlickEdit® Clipboards

Most text editing operations involve clipboards. Clipboards in SlickEdit are internal to the editor and separate from the system clipboard provided by the operating system. While most operating systems only allow one clipboard at a time, SlickEdit, by default, keeps a stack of the 50 most recently used clipboards. You can see a list of your clipboards by using the Clipboards tool window (**Ctrl+Shift+V**, **Edit → List Clipboards** or **list_clipboards** command). See [Clipboards](#) for more information.

Insert/Replace Editing Mode

By default, SlickEdit® starts in Insert mode, which means text that you type is inserted at the cursor. When the editor is in Replace mode, text is typed over the subsequent characters, essentially replacing text as you type. The Insert or Replace editing mode is indicated in the status line of the editor (**Ins** or **Rep**). To toggle the editing mode between Insert and Replace, click on the indicator, press the **Insert** key, or use the **insert_toggle** command. To change the default start mode, from the main menu, click **Tools → Options**, expand **Editing** and click **General**, then change the value of the **Start mode** option.

Improve Your Editing Efficiency

The subsequent sections describe many editing commands. If a command you like to use isn't bound to a key or key sequence already, it's a good practice to give it a key binding for quicker keyboard access. See [Creating Bindings](#) for more information.

If you frequently use multiple text editing operations in succession, record the steps as a macro and bind it to a key to save time in the future. See [Recording a Macro](#) for more information.

All editing features are not necessarily documented here, nor are all commands documented for each feature. For example, in a subsequent section, common Cut operations are described, but more commands are also available. Usually these are emulation-specific. A good way to discover related commands is to type a portion of the command into the **Search by command** box on the Key Bindings option screen (**Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**). For example, type "cut" in this box, and the list of Commands is filtered to show only those with that contain the text "cut". Now you can see that some additional Cut commands are **append_next_cut** and **cut_level**, two features of the GNU Emacs emulation.

Undoing Edit Operations

To undo an edit operation, use the **undo** command (**Edit** → **Undo** or **Ctrl+Z**). To redo the operation after using Undo, use the **redo** command (**Edit** → **Redo** or **Ctrl+Y**). To cancel a text selection, use the **deselect** command or press **Ctrl+U**.

Selections

Most applications let you select text and perform operations on the selected text, such as Cut, Copy, and Move. SlickEdit® offers three types of selections: character, line, and block. Each selection type provides different capabilities for different editing situations - and easy access.

Selected text is rendered with a shaded background. You can change the color of the shading by modifying the Background color of the **Selection** screen element (**Tools** → **Options** → **Appearance** → **Colors**). See [Setting Colors for Screen Elements](#) for more information.

Selection Types

There are three selection types in SlickEdit®: character, line, and block. The following table shows a summary of each type and some methods for creating the selection. Each type is explained in more detail below.

Selection Type	Description	Creation Methods
Character selection	This is created when one or more individual characters are selected. This is also known as a "char" selection.	Use the mouse to drag or use the select_char command (F8 or Edit → Select → Char). See also Character Selections .
Line selection	This is created when one or more whole lines are selected as lines. You can select all of the characters on a line and still have a character selection. The selection type depends on how	Triple-click within a line or use the select_line command (Ctrl+L or Edit → Select → Line). For multiple lines, drag in the left margin area of the edit window (when the mouse pointer changes

Selection Type	Description	Creation Methods
	the selection was created.	to point right). See also Line Selections .
Block selection	This is created when columns of text are selected, also known as a "column selection".	Right-click and drag or use the select_block command (Ctrl+B or Edit → Select → Block). See also Block Selections .

Character Selections

Character selections (also called "char" 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 any of the following methods:

- Use the mouse to click and drag.
- Use the **select_char** command (**F8** or **Edit → Select → Char**), then use the arrow keys to extend the selection, or use the mouse to click at the end of the selection.
- Press and hold the **Shift** key with any navigation key. See [Starting/Extending a Character Selection](#) below for examples.

You can also create character selections on words:

- To select the whole word under the cursor, double-click on the word or use the **select_whole_word** command (**Edit → Select → Word**).
- To select from the cursor to the end of the current word or the next word, use the **select_word** command.

When viewing a list of clipboards, character selection types are indicated with the text "CHAR" (see [Viewing and Inserting Clipboards](#) for more information).

[Starting/Extending a Character Selection](#)

You can start or extend a character selection with **Shift** key shortcuts, described in the table below. For example, press **Shift+Home** to create a character selection from the cursor to the beginning of the line. Press **Shift+End** to 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 to the top of the file, or **Ctrl+Shift+End** to create a selection from the cursor to the end of the file.

Note that these shortcuts are based on the default CUA emulation.

Shortcut for Extending a Char Selection	Description

Shortcut for Extending a Char Selection	Description
Shift+Right	Start or extend selection to right.
Shift+Left	Start or extend selection to left.
Shift+Up	Start or extend selection up one line.
Shift+Down	Start or extend selection down one line.
Shift+Home	Start or extend selection to beginning of line.
Shift+End	Start or extend selection to end of 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.

Line Selections

A line selection is created when one or more complete lines are selected (partially selected lines are treated as character selections).

SlickEdit® treats line selections very differently from character selections. A line selection can only be inserted before or after another line of code. That's because a line of code is a meaningful unit of functionality in most languages, and it would never be inserted inside another line of code. Handling line selections in this manner makes it faster to copy and paste lines of code.

Line selections are pasted before or after the current line, depending on your **Line insert style** setting (**Tools** → **Options** → **Editing** → **General**). Furthermore, line selections work with [SmartPaste®](#), which reindents inserted lines according to the surrounding code. See [Inserting Lines](#) for more information.

To select a line, use one of the following methods:

- Use the mouse to triple-click within a line, or to select multiple lines, drag in the left margin area of the edit window (when the mouse pointer changes to point right).
- Use the **select_line** command (**Ctrl+L** or **Edit** → **Select** → **Line**). This selects the current line, or, you can use the arrow keys to extend the selection to include more lines (or click with the mouse on the last line of the selection).

The following operations are also treated as line selections:

- To select the current code block (an entire block statement such as if, loop, switch, etc.), use the **select_code_block** command (**Edit** → **Select** → **Code Block**).
- To select the current procedure/function, use the **select_proc** command (**Edit** → **Select** → **Procedure**).
- To select the entire buffer, use the **select_all** command (**Ctrl+A** or **Edit** → **Select** → **All**).

When viewing a list of clipboards, line selection types are indicated with the text "LINE" (see [Viewing and Inserting Clipboards](#) for more information).

Block Selections

Block selections, also known as column selections, are used to process columns of text. To create a block selection, use any of the following methods:

- Use the mouse to right-click and drag.
- Use the **select_block** command (**Ctrl+B** or **Edit** → **Select** → **Block**), then use the arrow keys to extend the selection, or use the mouse to click at the end of the selection.
- Use **Alt+Shift+Left/Right/Up/Down/Home/End**, to start, switch to, or extend a block selection.

When viewing a list of clipboards, block selection types are indicated with the text "BLOCK" (see [Viewing and Inserting Clipboards](#) for more information).

Selection Styles

Selection styles determine key behaviors for selections, like whether to extend the selection as the cursor moves or to deselect after a copy or paste operation. Selection features in SlickEdit® depend on the current selection style, which is set to match your emulation mode by default. However, note that the key bindings described in this section are based on the default emulation mode, CUA. If you are using a different emulation, see the emulation charts (located in the `docs` subdirectory of your installation directory) for a listing of selection keys.

To change the selection style, use the Selections option screen (**Tools** → **Options** → **Editing** → **Selections**). See [Selection Options](#) for more information.

Selection Indicator

SlickEdit® provides a selection indicator, located in the status area of the editor, to indicate the type of selection and the number of characters or lines in a selection. This is useful to quickly determine the selection type you have made, and to measure the length of a word or string, or the number of lines in a function.

The selection indicator 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. For example, if three characters are selected, the indicator displays "3 Cols".

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 and number of columns (+N) to indicate if there are "extra" characters selected, or a minus sign and number of columns (-N) to indicate if there are fewer characters selected, depending on the start and end columns of the selection. For example, if the selection spans one entire line and part of the subsequent line, the indicator displays "1 Line+4".
- When the current selection is a **line selection**, the indicator displays the number of lines. For example, if two lines are selected, the indicator displays "2 Lines".
- When the current selection is a **block selection**, the indicator displays the size of the block in the format *Lines x Columns*. For example, if the selected block is two lines long and three columns wide, the indicator displays "2x3 Block".

Tip

The selection indicator can be used to count the number of characters in any text block. This can be useful for database work or any type of task that involves checking the number of characters. Simply paste the text into SlickEdit and select it with one of the character selection methods, then look at the selection indicator to see the number of characters.

Cycling Through Selections

You can quickly cycle through the three selection types (character, line, and block) with the mouse. To do this, press and hold the left button while clicking with the right button to change the selection type. For example, if you have a character selection, click once to start a block selection, or twice to make a line selection.

You can also cycle through successively larger selections by using the **select_toggle** command or by clicking on the [Selection Indicator](#) in the editor's status area. For example, if you have a character selection, you can 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.

Operating on Selected Text

SlickEdit® provides many methods for manipulating selected text. The table below describes some of the most common selection operations. See [Cut, Copy, Paste, and Move](#) for more.

Selection Operation	Description	Usage
Add Numbers in Selection	Adds selected numbers 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. Works with character, line, and block selections.	Use the add command.
Add Numbers to Selection (Enumeration)	Automatically adds incrementing numbers to a selection of code.	Use the enumerate command to auto-add numbers or use the gui_enumerate command (Edit → Other → Enumerate) to display the Enumerate Dialog , where you can specify options.
Align Block Selection Center	Centers text in a block selection within the selected area.	Use the align_selection_center command.
Align Block Selection Left	Aligns text in a block selection so that the first non-blank character of each line is flush against the left edge of the selection.	Use the align_selection_left command.
Align Block Selection Right	Aligns text in a block selection so that the last non-blank character of each line is flush against the right edge of the selection.	Use the align_selection_right command.

Selections

Selection Operation	Description	Usage
Append Selection to Clipboard	Appends selected text to the clipboard.	Use the append_to_clipboard command (Ctrl+Shift+C or Edit → Append to Clipboard)
Beautify Selection	(Pro only) Beautifies the selected text according to the beautification settings for the current language. See Beautifying Code for more information.	Use the beautify_selection command, or check the Restrict to selection box on the Beautifier dialog (Tools → Beautify).
Cancel Selection	Cancels the selection.	Use the deselect command or press Ctrl+U .
Casing: Lowercase Selection	Translates characters within a selection to lowercase letters. See Case and Capitalization of Text for more casing options.	Use the lowcase_selection command (Ctrl+Shift+L or Edit → Other → Lowcase).
Casing: Toggle Selection Casing	Toggles the characters within a selection between lowercase and uppercase.	Use the togglecase_selection command.
Casing: Uppercase Selection	Translates characters within a selection to uppercase letters. See Case and Capitalization of Text for more casing options.	Use the upcase_selection command (Ctrl+Shift+U or Edit → Other → Upcase).
Copy Selection by Dragging	Drag/copies selected text.	Press and hold Ctrl while clicking inside a selection and then dragging with the mouse to the desired location (Ctrl+LButtonDn).
Copy Selection to Clipboard	Copies selected text (or the entire line, if no selection) to the clipboard. You can also create a named clipboard with this operation (see Named Clipboards).	Use the copy_to_clipboard command (Ctrl+C or Edit → Copy).
Copy Selection to Cursor	Copies selected text to the cursor location. Char and block selections are inserted before the character at the cursor, while lines	Use the copy_to_cursor command or press Ctrl+Shift while holding the right mouse button (Ctrl+Shift+RbuttonDn).

Selections

Selection Operation	Description	Usage
	are inserted at the location specified by the Line insert style setting (Tools → Options → Editing → General).	
Cut Selection	Deletes a selection and copies it to the clipboard.	Use the cut command (Ctrl+X or Edit → Cut).
Delete and Append to Clipboard	Deletes a selection and appends it to the clipboard.	Use the append_cut command (Ctrl+Shift+X or Edit → Append Cut).
Execute Selection	Executes each line or sub-line of a selection as if entered on the command line.	Use the execute_selection command or press Alt+= .
Files: Append Selection to File	Appends selected text to the specified file.	Specify a file name with the append command, or use the gui_append_selection command to display a dialog where you can browse to pick the file.
Files: Write Selection to File	Writes selected text to the specified file.	Specify a file name with the put command, or use the gui_write_selection command (File → Write Selection) to display a dialog where you can browse to pick the file.
Fill Selection	Fills a selection with the specified key character.	Use the fill_selection command to be prompted on the command line for the key, or use the gui_fill_selection command (Edit → Fill) to display a dialog prompt.
Hide Selection	Hides all lines in a selection by collapsing as a Selective Display unit.	Use the hide_selection command (View → Hide Selection). Use the show_all command (View → Show All) to redisplay lines.
Indenting: Indent Selection	Indents the selected text according to the Syntax Indent settings or by one tab stop,	Use the indent_selection command (Tab or Edit → Indent).

Selections

Selection Operation	Description	Usage
	<p>depending on the Indent with tabs setting on the Language-Specific Formatting Options screen. One indent level is added for char and line selections, while one indent level starting from the left edge of the selection is used for block selections.</p>	
Indenting: Unindent Selection	<p>Unindents the selected text according to the Syntax Indent settings or by one tab stop, depending on the Indent with tabs setting on the Language-Specific Formatting Options screen. One indent level is removed from each line of char and line selections, while one indent level starting from the left edge of the selection is removed for block selections.</p>	<p>Use the unindent_selection command (Shift+Tab or Edit → Unindent).</p>
List Clipboards	<p>Allows you to view and insert a clipboard. See Clipboards for more information.</p>	<p>Use the list_clipboards command (Ctrl+Shift+V or Edit → List Clipboards).</p>
Move Selection by Dragging	<p>Drag/moves selected text.</p>	<p>Press and hold the left mouse button while clicking on a selection and then dragging with the mouse to the new location.</p>
Overlay Block Selection	<p>Overlays a block selection at the current cursor location.</p>	<p>Use the overlay_block_selection command (Edit → Other → Overlay Block).</p>
Overlay/Adjust Block Selection	<p>Overlays a block selection at the current cursor location, and fills the source selection with blanks.</p>	<p>Use the adjust_block_selection command (Edit → Other → Adjust Block).</p>
Paste	<p>Inserts the most recent clipboard at the current cursor location. To insert another clipboard, see the List Clipboards operation or Clipboards.</p>	<p>Use the paste command (Ctrl+V or Edit → Paste).</p>

Selection Operation	Description	Usage
Reflow Selection	<p>Reflows text within a selection according to the margin settings specified on the Word Wrap option screen (see Language-Specific Word Wrap Options).</p> <p>Block selections are wrapped within the columns of the block.</p> <p>Char selections are not supported for this operation.</p>	Use the reflow_selection command (Document → Reflow Selection).
Reverse Selection	Reverses the characters in a selection.	Use the reverse_selection command.
Shift Text in Selection Left	Shifts text within a selection to the left by one column, maintaining relative indentation. This operation supports line and block selections. If a character selection is used, it is converted to a line selection.	Use the shift_selection_left command (Shift+F7 or Edit → Other → Shift Left).
Shift Text in Selection Right	Shifts text within a selection to the right by one column, maintaining relative indentation. This operation supports line and block selections. If a character selection is used, it is converted to a line selection.	Use the shift_selection_right command (Shift+F8 or Edit → Other → Shift Right).
Sort Lines Within Selection	Sorts lines in a selected area in ascending order.	Use the sort_within_selection command or select the Sort within selection option on the Sort dialog (Tools → Sort).
Sort Selected Lines	Sorts lines in a selected area in ascending order, by comparing only the first columns.	Use the sort_on_selection command or select the Sort on selection option on the Sort dialog (Tools → Sort).
Spell Check Selection	Checks the spelling of selected text according to the Spell Option settings. See Spell Checking for more information.	Use the spell_check_selection command (Tools → Spell Check → Check Selection).

Cut, Copy, Paste, and Move

The main menu item **Edit** provides access to commonly used editing features. Each menu item and its associated command are described in the [Edit Menu](#) section. Keyboard shortcuts for each menu item (if available) are displayed by default on the menu itself, based on the current emulation.

Tip

Several editing operations affect words. You can change the characters that SlickEdit® uses to recognize words, on a per-language basis. To do this, use the **Word chars** option on the [Language-Specific General Options](#) screen (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **General**).

Cutting and Deleting

You can cut or delete any selected text, or individual words, lines, or entire code blocks. Cut operations copy the text to the clipboard before deleting. To remove selected text without copying it to the clipboard, press the **Delete** key.

The table below shows some common Cut operations.

Cut Operation	Description	Usage
Cut	Deletes the selection and copies it to the clipboard.	Use the cut command (Ctrl+X or Edit → Cut).
Append Cut	Deletes the selection and appends it to the clipboard.	Use the append_cut command (Ctrl+Shift+X or Edit → Append Cut).
Cut Word	Deletes text starting from the cursor to the end of the current word or next word, and copies it to the clipboard. Invoking this operation from the keyboard multiple times in succession creates one clipboard. See the Tip at the beginning of this section to change word recognition characters.	Use the cut_word command (Ctrl+Shift+k or Edit → Delete → Word).
Cut Line	Deletes the current line and copies it to the clipboard. Invoking this operation from the keyboard multiple times in succession creates one clipboard.	Use the cut_line command (Ctrl+Backspace or Edit → Delete → Line).

Cut Operation	Description	Usage
Cut to End of Line	Deletes text starting from the cursor to the end of the line, and copies it to the clipboard. Invoking this operation from the keyboard multiple times in succession creates one clipboard.	Use the cut_end_line command (Ctrl+E or Edit → Delete → To End of Line).
Cut Code Block	Prompts to delete the current code block statement and copies the lines to the clipboard.	Use the cut_code_block command or press Ctrl+Del .

Copying Text

The table below shows some common Copy operations.

Copy Operation	Description	Usage
Copy to Clipboard	Copies the selected text (or the entire line, if no selection) to the clipboard. You can also create a named clipboard with this operation (see Named Clipboards).	Use the copy_to_clipboard command (Ctrl+C or Edit → Copy).
Copy Word	Copies the word at the cursor to the clipboard. Invoking this operation from the keyboard multiple times in succession creates one clipboard. See the Tip at the beginning of this section to change word recognition characters.	Use the copy_word command (Ctrl+K or Edit → Copy Word).
Copy to Cursor	Copies the selected text to the cursor location. Char and block selections are inserted before the character at the cursor, while lines are inserted at the location specified by the Line insert style setting (Tools → Options → Editing → General).	Use the copy_to_cursor command or press Ctrl+Shift while holding the right mouse button (Ctrl+Shift+RbuttonDn).

Copy Operation	Description	Usage
Copy to System Clipboard	Passes the selected text to the operating systems clipboard.	Use the copy command.
Copy Visible	Copies the currently visible lines to the clipboard. Ignores lines hidden by Selective Display .	Use the copy_selective_display command (View → Copy Visible).

Pasting Text

When pasting text created from a character or block selection, the text is inserted before the character at the cursor. Line selections are inserted at the location specified by the **Line insert style** setting (**Tools** → **Options** → **Editing** → **General**), and by default, indented according to your indent level settings. See [Inserting Lines](#) for more information.

The most recent clipboard item can be inserted at the cursor location with the **paste** command (**Ctrl+V** or **Edit** → **Paste**).

To insert another clipboard, use the Clipboards tool window (**Edit** → **List Clipboards**), or use the **list_clipboards** command to display the Select Text to Paste dialog. Both the tool window and dialog show a list of your clipboards and let you select the clipboard to insert. The only difference is that the tool window can be docked and contains a Preview area that shows the entire color-coded contents of the clipboard. See [Clipboards](#) for more information about these features.

You can also cycle through and paste clipboards with the **paste_next_clipboard** and **paste_prev_clipboard** commands. These commands cycle through the clipboard ring and paste the top item, while leaving the pasted text selected, so you can use the command again to see the next (or previous) clipboard text, if that wasn't the clipboard you wanted. For example, if you have three clipboards named 1, 2, and 3, invoking the **paste_next_clipboard** command inserts (yet leaves selected) clipboard 2 and moves it to the top of the ring. Invoke the command again to see/paste the next clipboard instead, and so on.

Moving Text

To move a text selection from one location to another, use the mouse to drag and drop it where you want. SlickEdit® allows this capability by default. To disable it, from the main menu, click **Tools** → **Options**, expand **Editing** and click **General**, then set the **Allow drag drop text** option to **False**.

Clipboards

Use SlickEdit® clipboards to copy and move text in files, the SlickEdit command line, dialog text boxes, or any other application that supports text clipboards, such as a word processor. Clipboards in SlickEdit are internal to the editor and separate from the system clipboard provided by the operating system.

Common clipboard-related operations (cut, copy, paste, etc.) are available on the main **Edit** drop-down menu. The corresponding key binding for each item is also shown by default. See [Cut, Copy, Paste, and Move](#) for more information about basic editing operations.

When using a cut or copy operation, a clipboard is created. Pressing the same cut key multiple times in succession creates one clipboard. For example, the shortcut **Ctrl+Shift+K** is used to cut words (the binding for the **cut_word** command). If you press **Ctrl+Shift+K** three times to cut three words, one clipboard is created that you can insert with **Ctrl+V** (the **paste** command). This is true for **Ctrl+Backspace** (**cut_line** command) and **Ctrl+E** (**cut_end_line** command) as well.

Tip

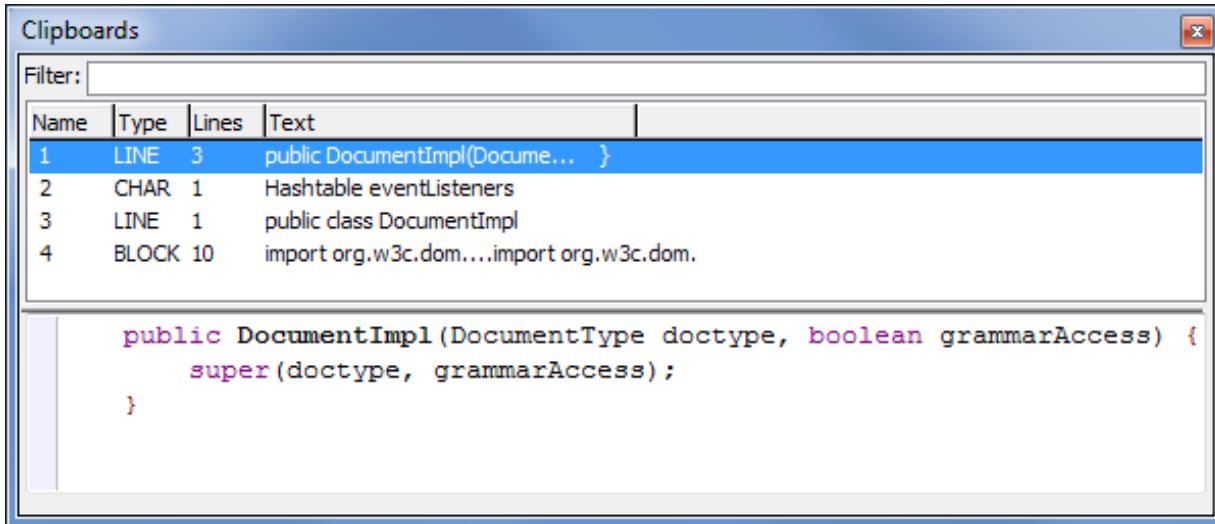
If you are using the Brief emulation and want to place cut text on a clipboard, bind the commands **cut_word**, **cut_end_line**, and **cut_line** to the appropriate keys.

Viewing and Inserting Clipboards

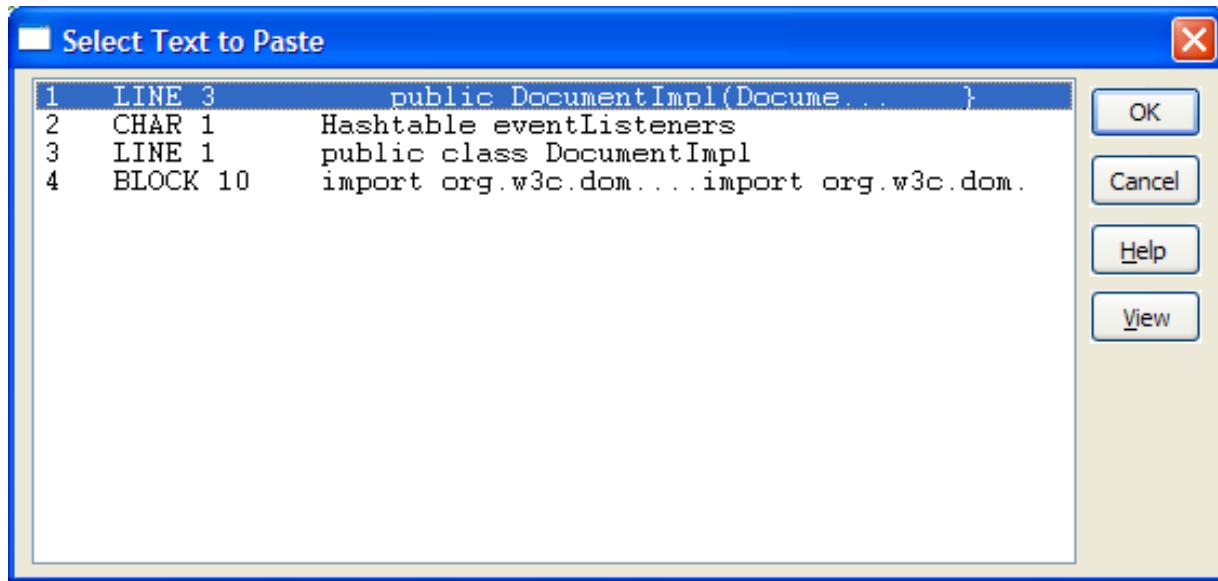
To insert the current clipboard into the buffer, from the main menu, select **Edit → Paste**, press **Ctrl+V**, or use the **paste** command.

In the case of multiple clipboards, there are two ways to view and insert: by using the Clipboards tool window, or by using the modal Select Text to Paste dialog. Both provide the same information, except the Clipboards tool window is dockable and contains a color-coded Preview area for previewing clipboard contents.

- To display the Clipboards tool window, from the main menu, select **Edit → List Clipboards**, press **Ctrl+Shift+V**, or use the **list_clipboards** command.



- To display the Select Text to Paste dialog, you can use either the **old_list_clipboards** command or the **list_clipboards_modal** command. These commands are identical.



With either method, double-click on a clipboard to insert it at the cursor location, or, select the clipboard to insert and press **Enter** or click **OK**.

Both the dialog and the tool window provide the same information:

- **Clipboard name/number** - This is the number of the clipboard or the name, if using [Named Clipboards](#). Clipboards are numbered with the most recent clipboard first, which always appears at the top of the list. You can use this value with the **paste** command to insert the specified clipboard. For example, type **paste 2** on the command line to insert clipboard 2 at the cursor location.
- **Clipboard type** - The clipboard type can be CHAR, LINE, or BLOCK. A CHAR type clipboard is inserted before the current character. 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** → **Editing** → **General**). A BLOCK type clipboard is inserted before the current character and pushes over all text intersecting with the block. No lines are inserted.
- **Line count** - The number following the clipboard type indicates the number of complete or partial lines of text in the clipboard.
- **Clipboard contents/summary** - This area shows all or a portion of the clipboard contents. If the contents exceed the viewing area, they are condensed.

The Clipboards tool window contains a Preview area that shows the selected clipboard's color-coded contents. You can copy text in the Preview to create a new clipboard. To see the entire contents of a condensed clipboard using the Select Text to Paste dialog, click the **View** button. The View Clipboard dialog opens showing the color-coded contents in an edit window. From here, you can copy all or part of the contents to the operating system clipboard.

The Clipboards tool window contains additional functionality:

- You can filter the list of clipboards by text. By typing a string into the **Filter** text box, only clipboards whose contents contain that entered string will be shown. Clearing the filter will restore all clipboards.

- To delete the selected clipboard item in the tool window, press **Delete**, or, right-click and select **Delete** from the context-menu. To delete all clipboards, select **Clear All** from the right-click context menu.
- To make the selected clipboard active, select **Set as Current Clipboard** from the right-click context menu.
- To save the clipboard to a file, select **Save clipboard to file** from the right-click context menu.
- To change the view of the tool window, the **View** menu item on the right-click context menu:
 - **Auto** - When this is selected, the Clipboards tool window switches between Horizontal and Vertical views automatically as you resize it.
 - **Horizontal** - When this is selected, the clipboard list is displayed above the Preview area.
 - **Vertical** - When this is selected, the clipboard list and Preview area are displayed side-by-side.

Named Clipboards

You can create a named clipboard by simply typing the name after the **copy_to_clipboard** command. For example, create a selection, then, on the SlickEdit® command line, type: **copy_to_clipboard a**. A clipboard named "a" is created. Now, you can use the name with the **paste** command to insert the named clipboard without using the Select Text to Paste dialog or Clipboards tool window (for example, **paste a**). Note that named clipboards are limited to two characters, and that the **cut** command is not supported for this feature.

Clipboards in the Command Line and Text Boxes

Only clipboards of one line can be inserted into the SlickEdit® command line or a text box. Both **Ctrl+V** and **Ctrl+Shift+V** key sequences can be used to insert clipboard text into these fields. The result of inserting a clipboard into a text box varies depending on the clipboard type.

Setting the Max Number of Clipboards

By default, a stack of the 50 most recently used clipboards is kept. To change the maximum number of clipboards saved, from the main menu, click **Tools** → **Options**, expand **Editing** and click **General**, then enter a value in the **Maximum clipboards** box.

Other Operations

Inserting Lines

SlickEdit® provides several ways to start a new line or split a line, as described in the table below.

Line Operation	Description	Usage
Split Line at Cursor	Splits the line at the cursor and appends enough blanks at the	Use the split_insert_line command or press Enter .

Line Operation	Description	Usage
	beginning of the new line to align it with the first non-blank character of the original line.	
No Split Insert Line (After)	Inserts a blank line after the current line, aligning the cursor with the first non-blank character of the current line. The current line is not split.	Use the nosplit_insert_line command or press Ctrl+Enter .
No Split Insert Line (Before)	Inserts a blank line before the current line, aligning the cursor with the first non-blank character of the current line. The current line is not split.	Use the nosplit_insert_line_above command or press Ctrl+Shift+Enter .

Case and Capitalization of Text

The table below shows some of the operations you can use to change the case and capitalization of characters and words.

Casing Operation	Description	Usage
Lowercase Selection	Translates characters within a selection to lowercase letters.	Use the lowcase_selection command (Ctrl+Shift+L or Edit → Other → Lowcase).
Lowercase Word	Translates the current word to lowercase letters and places the cursor after it. See the Tip at the beginning of this section to change word recognition characters.	Use the lowcase_word command.
Uppercase Selection	Translates characters within a selection to uppercase letters.	Use the upcase_selection command (Ctrl+Shift+U or Edit → Other → Upcase).
Uppercase Word	Translates the current word to uppercase letters and places the cursor after it. See the Tip at the beginning of this section to change word recognition	Use the upcase_word command.

Casing Operation	Description	Usage
	characters.	
Toggle Casing	Toggles the case of letters within a selection.	Use the togglecase_selection command.
Uppercase Mode	Toggles Uppercase mode on and off. Uppercase mode means that letters you type are automatically uppercased, so you don't need to press and hold Shift. Note that you can enable auto-caps on a language-specific basis with the Auto CAPS option (see Language-Specific General Options).	Use the caps command.
Capitalize Selection	Capitalizes the first letter of each word in a selection.	Use the cap_selection command (Ctrl+Shift+A or Edit → Other → Capitalize).
Capitalize Word	Capitalizes the first letter of the current word and places the cursor after the word. See the Tip at the beginning of this section to change word recognition characters.	Use the cap_word command.

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, click **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.

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);
```

Hex Mode Editing

You can enable Hex view/edit mode on a per-document or language-specific basis:

- To view the current binary or text file in a Hex mode, click **View → Hex** or **View → Line Hex** (or use the commands **hex** or **linehex**, respectively).
- To enable Hex or Line Hex view on a language-specific basis, so that each file opened in that language is displayed in Hex mode, use the [Language-Specific View Options](#).
- To specify the number of columns and number of bytes per column for Hex (not Line Hex) view, use the

[Language-Specific View Options](#)

If you close a file in Hex mode, the file will be displayed in Hex mode the next time it is opened. When the cursor is in hex data, the data 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 SlickEdit® is in Hex mode. Only character selections are displayed when in Hex mode.

See also [Hex/Line Hex View](#) for more information.

Hex/Text View 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 following commands perform hex cursor motion: **top_of_buffer**, **bottom_of_buffer**, **page_up**, **page_down**, **begin_line**, **end_line**, **begin_line_text_toggle**, **cursor_left**, and **cursor_right**.

Hex/Text View Operation	Key Shortcut
Delete Byte to Left of Cursor and Move Cursor Left	Backspace
Delete Byte Under Cursor	Delete
Move Cursor to Beginning of Hex Line	Home
Move Cursor to Last Character of Hex Line	End
Toggle Cursor Between Hex Data on Left, Text Data on Right	Tab and Shift+Tab

Multiple Cursors and Selections

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.

Adding a Cursor or Selection

To add a cursor, use **Ctrl-LButtonDown**. To add a selection, use **Ctrl-LButtonDown** and drag the mouse. **Ctrl-DoubleClick** will also add a selection. Use **Shift-RButtonDown** and drag to create multiple character (stream) selections.

When you use **Ctrl-LButtonDown** and drag, you will see what looks like a typical column selection being created. However, after you release the mouse a character selection will be created for each partial line selected by the column selection. Virtual space past the end of the line is not selected.

If you make a mistake (really easy to do) while adding a cursor/selection, you can use undo to remove it!

To add a cursor without using the mouse, use **Ctrl-| (add_multiple_cursors)**. Do this to initiate multiple-cursor mode, then move the cursor to another location you want a cursor at and hit **Ctrl-|** again. Repeat to create a set of cursors. If you have a multiple-line selection, **Ctrl-|** can be used to convert the selection to multiple cursors by creating a cursor on each line of the selection.

To add multiple-cursors above or below the current cursor location, you can also use **Ctrl-Shift-Alt-Up (add_multiple_cursor_up)** or **Ctrl-Shift-Alt-Down (add_multiple_cursor_down)**, respectively.

When Should I use Multiple Cursors and Selections

The best use of multiple cursors is for creating source code from a list of identifiers. If the identifiers are on separate lines, you can create the multiple selections very quickly using Shift+RbuttonDown. Once you have the multiple selections, you can make simultaneous edits possibly to create source code for case statements (case <CONSTANT>:) for a switch. Alternately, you can quickly create a quoted list of identifiers for a table. Many of us have been using macro recording to get repetitive editing tasks done. It works well and has the advantage that you can save and reuse macro recordings. Use the mechanism you are most comfortable with. In general, when you can use Shift+RButtonDown to create multiple selections, using multiple cursors and selections will take fewer key strokes.

Cut/Paste and Multiple Cursors

When SlickEdit creates a clipboard, it stores a count of the number of cursors there were when the clipboard was created. Then when you paste into a file with the same number of cursors as the clipboard, SlickEdit will attempt to paste segments of the clipboard at each cursor location.

If the number of lines in the clipboard is not divisible by the number of cursors or the number of cursors don't match, the entire clipboard is pasted at each cursor location.

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, you aren't getting the most out of SlickEdit®.

Symbol Navigation (Pro only)

Symbol Navigation allows you to jump from a symbol to its definition, declaration, 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, click **Search → Go to Definition** or use the **push_tag** command.
- **Go to Declaration** - To quickly move the cursor from a symbol to its declaration, pushing a bookmark in the process, press **Ctrl+Alt+Dot**. Alternatively, click **Search → Go to Declaration** or use the **push_alttag** command.

Note

This feature depends on the language-specific settings for symbol navigation priority with Context Tagging (see [Context Tagging Features](#)).

If these preferences are set to prioritize navigation to **Symbol definition (proc)**, then the **push_tag** command (**Go to Definition**) will attempt to navigate directly to symbol definitions, and the **push_alttag** command (**Go to Declaration**) will attempt to navigate directly to symbol declarations.

Conversely, if these preferences are set to prioritize navigation to **Symbol declaration (proto)**, then the **push_tag** command will attempt to navigate directly to symbol declarations, and the **push_alttag** command will attempt to navigate directly to symbol definitions. Note also that in menus, the usual order of the **Go to Definition** and **Go to Declaration** commands will be reversed.

Finally, if navigation is set to **Prompt with all choices**, then both commands will prompt you with

both definitions and declarations. However, **Go to Declaration** will also include forward declarations, even if you had specified to ignore them, and the [Select Symbol Dialog](#) dialog will always show the symbol navigation options. **Go to Declaration** will not appear on menus if navigation is set to **Prompt with all choices**.

Note

When a symbol has more than one definition or declaration, if you navigate to it using the **Go to Definition** or **Go to Declaration** commands, you may immediately repeat the same command again to cycle forward to additional definitions and declarations. For example, if you press **Ctrl+Dot** to navigate directly to a function's definition, you can quickly jump to its prototype simply by pressing **Ctrl+Dot** again.

- **Go to Reference** - To create a list of references and optionally jump to the first one, pushing a bookmark in the process, press **Ctrl+/.** Alternatively, click **Search → Go to Reference** or use the **push_ref** command.

The References tool window is stackable, meaning that you can create a stack of references searches and navigate between the different results sets. By default, whenever you do a references search, the search results are pushed onto the stack.

When you navigate through all the search results and have no more references, SlickEdit® will automatically remove the current references search from the stack and return to the previous set of references.

In a similar fashion, if you use Pop Bookmark to return to the location where a Go to Reference command was invoked, SlickEdit® will automatically remove the current references from the references stack, as returning to the original location was an indication that you were done looking at those references.

You can manually push and pop searches onto the references stack by pressing the 'Add' button (the **push_references_stack**) or 'Delete' button (the **pop_references_stack**) on the references tool window. You can also jump to a specific set of references deeper in the stack by clicking on its node in the references stack indicator bar. Hovering the mouse over one of the enabled nodes in the references stack indicator bar will give you a tooltip indicating the symbol and search criteria that the references search was for.

- **Pop Bookmark** - To pop the bookmark and return to the previous location, press **Ctrl+Comma**. Alternatively, click **Search → Bookmarks → 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](#)).

Tip

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 definition. You can tell SlickEdit® to always go to one or the other by setting one of the options **Prioritize navigation to symbol definition (proc)** or **Prioritize navigation to symbol declaration (proto)**. To set these options, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Context Tagging**®. If you configure SlickEdit® to navigate directly to definitions or to declarations, you can also navigate directly to the other by pressing **Ctrl+Alt+Dot**. 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 Symbol Dialog](#) the first time you navigate from a symbol to its definition.

Automatically Closing Visited Files

Some features and operations in SlickEdit® automatically open files for "visiting", such as Go to Definition, Go to Declaration, Go to References, Find in Files, and Pop Bookmark (see [Symbol Navigation](#)). A file is considered visited if it is opened as a result of a symbol navigation or search operation, not modified, and subsequently navigated away from. An option is available to automatically close these visited files. To access the **Automatically close visited files** option, from the main menu, click **Tools** → **Options**, then expand **Editing** and select **Bookmarks**. You can enable the option or you can choose to be prompted to close each time you navigate away from a visited file.

Navigating Between Multiple Instances

If more than one instance of the definition or reference is found, the [Select Symbol 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, pattern, 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 (**Space** 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.

Navigating Between Words

To navigate between words, use the **next_word** (**Ctrl+Right**) and **prev_word** (**Ctrl+Left**) 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.

A word is determined by the **Word chars** value you set for the programming language (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **General**). For C, C++, and Java this is set to **A-Za-z0-9_** by default. The **next_word** command, for example, will skip over any contiguous characters from that set.

You can specify whether the cursor moves to the beginning or the end of the next/previous word. Click **Tools** → **Options** → **Editing** → **General**, then set the **Next word style** to **Begin** or **End**. This affects both **next_word** and **prev_word** commands.

If you have enabled subword navigation (see [Subword Navigation](#)), the word navigation commands will behave like their subword navigation counterparts. You can still perform regular word navigation using the "full" word commands: **next_full_word**, **prev_full_word**, **select_full_word**, **copy_full_word**, **cut_full_word**, **delete_full_word**, and **delete_prev_full_word**.

Subword Navigation

Subword navigation provides the capability to navigate within a word, stopping at capitalized letters or letters following common dividers like underscore or dash. If the target word does not contain any subwords, then the subword commands behave like their word navigation counterparts.

You can configure SlickEdit to use subword navigation instead of the regular word navigation by selecting **Tools** → **Options** → **Editing** → **Cursor Movement** and setting **Subword Navigation** to **True**. When this option is on, you can still perform "full" word navigation using the **_full_word** commands. See [Navigating Between Words](#) for more information.

The following subword navigation commands are provided. For convenience, you can bind them to a key sequence using **Tools** → **Options** → **Key Bindings**. You can also use the **Key Bindings** screen to search for subword commands by entering "subword" in the **Search by command** field. Then you can view further documentation on each command.

- **next_subword** - Moves the cursor to the next subword.
- **prev_subword** - Moves the cursor to the previous subword.
- **select_subword** - Selects the next subword.
- **copy_subword** - Copies the next subword to the clipboard.
- **cut_subword** - Cuts the next subword, putting it in the clipboard.

- **delete_subword** - Deletes the next subword without putting it in the clipboard.
- **delete_prev_subword** - Deletes the previous subword without putting it in the clipboard.

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 click **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 language-specific **General** options screen to view or define the begin/end pairs for any language. To access this dialog, from the main menu, click **Tools → Options → Languages**, expand your language category and language, then select **General**.

In the **Begin/end pairs** text field, specify the pairs in a format similar to a regular expression.

Note

This text box is unavailable (dimmed) 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" (a semicolon followed by an upper-case 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 , (comma) is used to specify multiple begins or multiple ends. The | 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 **[Language] Formatting Options** screen (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]**). See [Language-Specific Formatting Options](#) for more information.

Setting the Paren Match Style

As you type a closing parenthesis, highlight and matching options are available. To specify these options, from the main menu, click **Tools** → **Options**, expand **Editing**, then click **General** and set the value of the **Parenthesis matching style** option.

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, from the main menu, click **Tools** → **Options** → **Appearance** → **Colors**, 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/Changing Configuration Variables](#) for more information.

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 Symbol 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 with S-expressions

S-expressions are symbolic expressions. They can be a single symbol or a set of symbols contained in a structure. First popularized in Lisp and Emacs, SlickEdit provides several navigation commands using S-expressions.

These commands are particularly useful in XML and HTML, where the structures created by begin and end tags are treated as S-expressions. These commands allow you to skip over or drill down into text bounded by tags.

The following commands are available, with their default keybindings in CUA emulation:

- **prev_sexp** - Moves to the previous S-expression (**Ctrl +Alt +Left**).
- **next_sexp** - Moves to the next S-expression (**Ctrl +Alt +Right**).
- **backward_up_sexp** - Navigates to the start of the immediately enclosing block (**Ctrl +Alt +Up**).
- **forward_down_sexp** - Drills down into the next block (**Ctrl +Alt +Down**).
- **select_prev_sexp** - Extends a character selection from the cursor to the start of the previous S-expression (**Ctrl +Alt +Shift +Left**).
- **select_next_sexp** - Extends a character selection from the cursor to the start of the next S-expression (**Ctrl +Alt +Shift +Right**).
- **cut_prev_sexp** - Deletes the S-expression to the left of the cursor and copies it to the clipboard (**Ctrl +Alt +Backspace**).

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:

- **cursor_right (Right Arrow)** - Moves the cursor one column to the right. If the cursor is at the end of the line, this command will move the cursor to the next line depending on the value for **Cursor right/left wraps to next/previous line** (**Tools → Options → Editing → Cursor Movement**).

- **cursor_left (Left Arrow)** - Moves the cursor one column to the left. If the cursor is at the beginning of the line, this command will move the cursor to the previous line depending on the value for **Cursor right/left wraps to next/previous line (Tools → Options → Editing → Cursor Movement)**.
- **cursor_up (Up Arrow)** - Moves the cursor to the previous line. If the cursor is located in a column that is beyond the last column of the previous line, the cursor position is controlled by **Cursor up/down places cursor in virtual space (Tools → Options → Editing → Cursor Movement)**.
- **cursor_down (Down Arrow)** - Moves the cursor to the next line. If the cursor is located in a column that is beyond the last column of the next line, the cursor position is controlled by **Cursor up/down places cursor in virtual space (Tools → Options → Editing → Cursor Movement)**.
- **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.
- **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 enabled (**Tools → Options → Editing → General**), 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 [Pushed 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.

Navigating to a Specific Line

To view and place the cursor on a specific line number, from the main menu, click **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 click **Search → Go to Offset**, or use the **gui_seek** command. This function is the same as the C **Iseek** 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.

Navigating to URLs

SlickEdit® treats URLs in editor windows as hyperlinks, making them easy to identify and open in a Web browser from within your code. By default, a string is interpreted as a URL if it begins with one of the following URI schemes, or, URL types (including the colon and slashes):

- file://
- ftp://
- http://
- https://

URLs are underlined. You can navigate to a link by hovering over it with the mouse and using **Ctrl+Click** (or **Command+Click** on the Mac). The link opens in a new Web browser window, or the current browser window if one is already open. The `file://` URI scheme is handled differently (see [Handling File URLs](#) below).

When using the mouse to hover over an `http://` link, click the green arrow to open the source code in SlickEdit.



The URI Schemes node of the Options dialog lets you specify the recognized URI schemes, and makes it easy to extend this feature. For example, you may want to add a `mailto` URI scheme so that e-mail URLs are recognized. To access these options, from the main menu, click **Tools** → **Options**, expand **Network & Internet Options**, then select **URI Schemes**. See [URI Scheme Options](#) for more information.

Handling File URLs

Files can be designated using the `file://` URI scheme. Depending on the file type, a file can be opened in a browser, passed to an application for opening, or executed. How the file is handled depends on the operating system and the settings in **Tools → Options → Languages → File Extension Manager**.

The File Extension Manager provides two settings to control this behavior:

- **Open Application** - Specifies an application to open files with the selected extension.
- **Use file association** - Overrides the application specified in **Open Application** and uses the operating system to determine what application to use. This is only applicable to Microsoft Windows operating systems.

If an application is specified in the **Open Application** field, the file will be passed to that application for opening.

If **Use file association** is checked, the operating system is used to determine what application to use. This is only applicable on Windows.

If both fields are left blank, SlickEdit® will use the operating system to determine what application to use. This is the same as if you checked **Use file association** and is only applicable on Windows.

Runnable Files

A `file://` URI scheme can be used to specify a runnable file, like a batch file, script file, or executable. On Windows, the operating system is used to automatically identify runnable files and run them, unless you have specified a value for **Open Application**.

On Linux, UNIX, or Mac you have to specify how to run a runnable file by specifying an application or system command in the **Open Application** field. For example, on Linux you can run a Perl file by specifying the path to the Perl interpreter in **Open Application**. You also need to include the escape sequence denoting the file name, for example, `/usr/bin/perl %f`. The `%f` inserts the full path for the file portion of the URL. If you want to run a binary file you would just specify put `%f` in the **Open Application** field.

Other URI Schemes

You can add additional URI schemes to be treated as links (see [URI Scheme Options](#)). On Windows, the operating system will determine how to handle the URL. For example, using `ms-help://` will open the associated link in MSDN Help. On all other platforms, the link will be sent to the browser.

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:

- [Symbols Tool Window](#)
- [Current Context Toolbar](#)
- [Defs Tool Window](#)
- [Class Tool Window](#)
- [Find Symbol Tool Window](#)
- [Preview Tool Window](#)
- [References Tool Window](#)
- [Viewing Symbol Uses with the Calling Tree](#)
- [Viewing Symbol Callers Tree](#)
- [Symbol Properties Tool Window](#)
- [Symbol Arguments Tool Window](#)

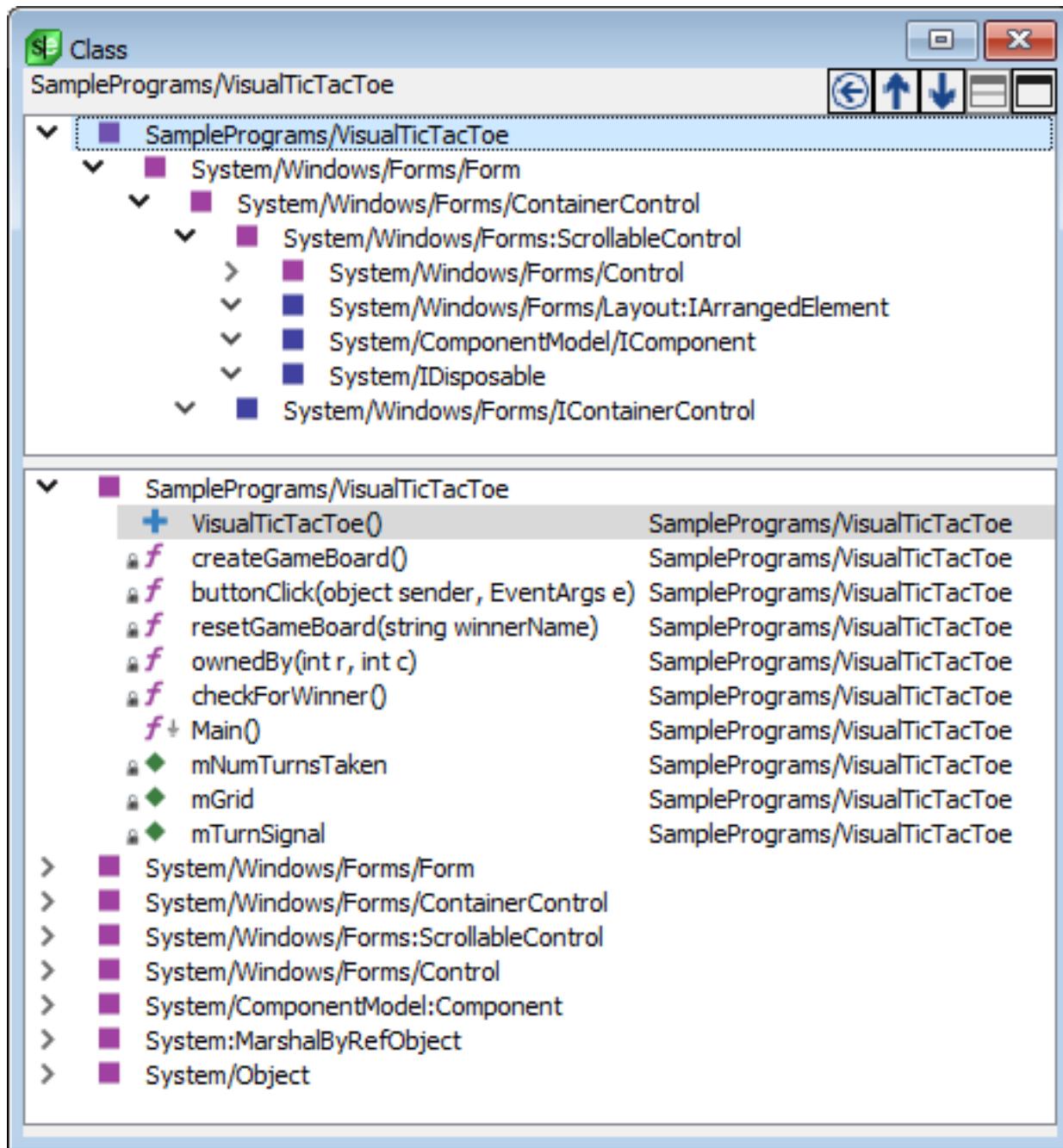
See also [Symbol Navigation](#) for information about how to navigate between symbols in files.

Class Tool Window (Pro only)

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 clicking **View → Tool Windows → Class** or by using the **toggle_tbclass** command. To display the tool window on demand, use the **activate_tbclass** command.

Class Tool Window (Pro only)



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 tool tip 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

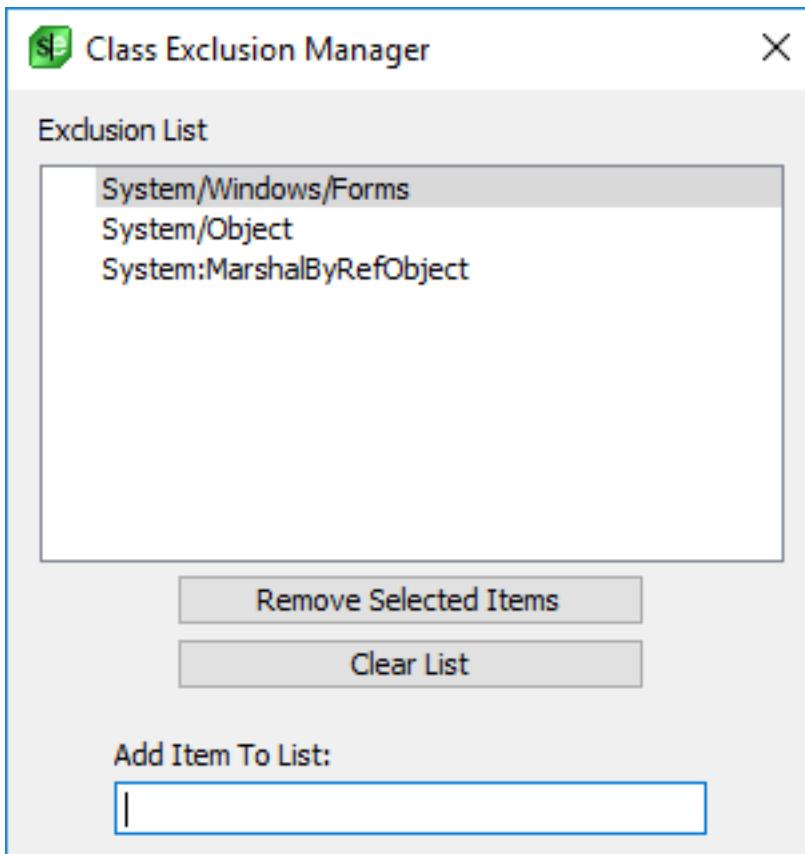
Right-click on a class in the hierarchy pane to display a list of filtering options. You can exclude entire namespaces or packages, anything above a certain level in the hierarchy, and anything outside of the current workspace. You can always include any class(es) you have excluded via the "Include" 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:

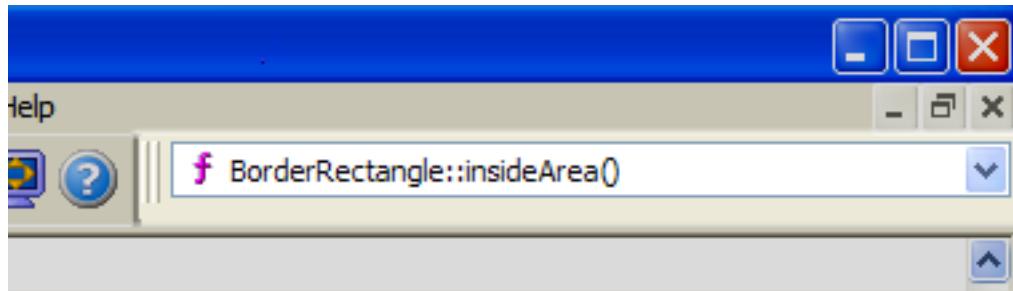
- **Quick Refactoring** - Offers two Quick Refactorings: Rename and Modify Parameter List. See [Quick Refactoring](#) for more information.
- **Add Member Function**, **Add Member Variable**, and **Add Virtual Function** - (C/C++ only) When these options are selected for a class, you are prompted with 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 and C# only) Organizes import statements in Java or C# source files. See [Organize Java Imports](#) and [Organize C# Imports](#) for more information. (C/C++, Objective-C, and Slick-C) Organized #include statements. See [Adding #includes](#) for more information.
- **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.

- **References** - Brings the References tool window into focus, displaying the references for the symbol. See [References Tool Window](#) for more information.
- **Calls or uses** - 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.
- **Callers** - Displays a tree of symbols which use by the selected symbol, for example, other functions that call the current function. See [Viewing Symbol Callers Tree](#) 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.
- **Jump on Single Click** - Toggle option to jump to the selected symbol on a single click (rather than requiring an double-click).
- **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 Toolbar

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.

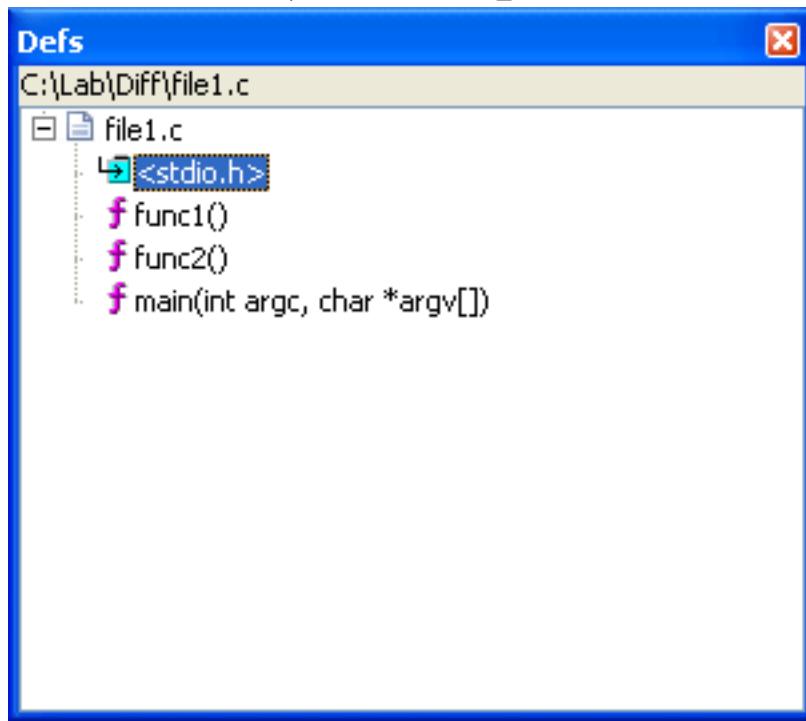
By default, the toolbar is docked in the top upper-right section of the editor. Display can be toggled on/off by clicking **View → Toolbars → Current Context** or by using the **toggle_context** command. To display the tool window on demand, use the **activate_context** command.



Defs Tool Window

The Defs 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 clicking **View → Tool Windows → 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 tool tip 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:

- **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 Statements** - 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. Note that statement-level tagging is not supported for all languages. Also note that when **Show Statements** is checked, the option to sort the Defs tool window by Function name is disabled.
- **Show Statements in All {lang} Files** - 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 and all other files in the current language mode. This allows you to see a primitive function flowchart or to navigate to a specific statement within a function. Note that statement-level tagging is not supported for all languages.
- **Show Nesting** - Organizes symbols and statements by their scope within the current file. Clear this option to display everything in one flat list.
- **Auto Expand** - Automatically expand Defs tree as you navigate within the current file in order to bring the current symbol or statement into view in the Defs tool window. If this option is cleared, the tree will only be initially expanded according to the **Expand All**, **Expand 1 Level**, **Expand 2 Levels**, or **Expand To Statements** option selected, and any further expansion will have to be done manually. When using the **Expand All** mode, this option will only have an effect on items in the Defs tool window that were manually collapsed.
- **Auto Collapse** - Automatically collapse items in the Defs tool window back to their original state when you move to a different symbol or statement in the current file. If this option is cleared, but **Auto Expand** is left enabled, the tree will be gradually expanded as you move around the file. This option has no effect when using the **Expand All** mode.
- **Expand All** - Expands all symbols and statements at all 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.
- **Expand To Statements** - Expands all symbols in the current file, but does not expand functions and statements. Note that statement-level tagging is not supported for all languages.
- **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.

Find Symbol Tool Window (Pro only)

- **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+I** in the editor window. See [Symbol Navigation](#) for more information.
- **Calls or uses** - 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.
- **Callers** - Displays a tree of symbols which use by the selected symbol, for example, other functions that call the current function. See [Viewing Symbol Callers 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.

Note

For XML, the Defs tool window can be customized to control how different elements are displayed. For more information see [Outline View for XML](#).

Find Symbol Tool Window (Pro only)

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, fast prefix match, or a symbol pattern.

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 [Search Dialogs and Tool Windows](#) provides this same capability, it cannot match the speed of Find Symbol, nor restrict the results just to symbol definitions and declarations.

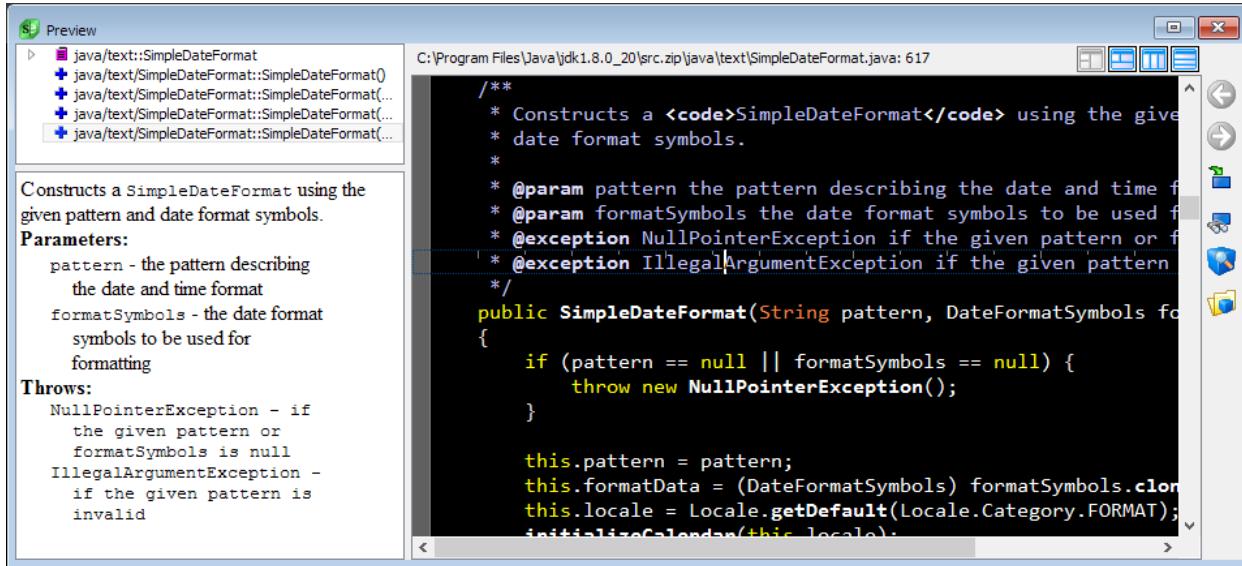
See [Find Symbol Tool Window](#) for information about the options that are available on the tool window.

Preview Tool Window (Pro only)

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 in 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 clicking **View** → **Tool Windows** → **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 tool tip 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.
- **Layout options** - Use the layout option buttons to select the preferred layout for the preview tool window.
 - **Automatic** - This is the default layout. If the tool window is wide enough, it will automatically switch to horizontal layout. If the tool window is significantly taller than it is wide, then it will automatically switch to vertical layout.
 - **Standard** - This is the standard layout used in previous versions of SlickEdit.
 - **Horizontal** - Use a horizontal layout scheme to maximize the number of lines of text visible.
 - **Vertical** - Use a vertical layout scheme to maximize the number of columns of text visible.
- **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](#).
 - **Manage Tag Files** - Opens the [Context Tagging - Tag Files Dialog](#) for building and maintaining tag files for indexing symbol information.

Information Displayed in 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 Defs Tool Window , Symbols Tool Window , Class Tool Window , Current Context Toolbar , and Find Symbol Tool Window for more information.

References Tool Window (Pro only)

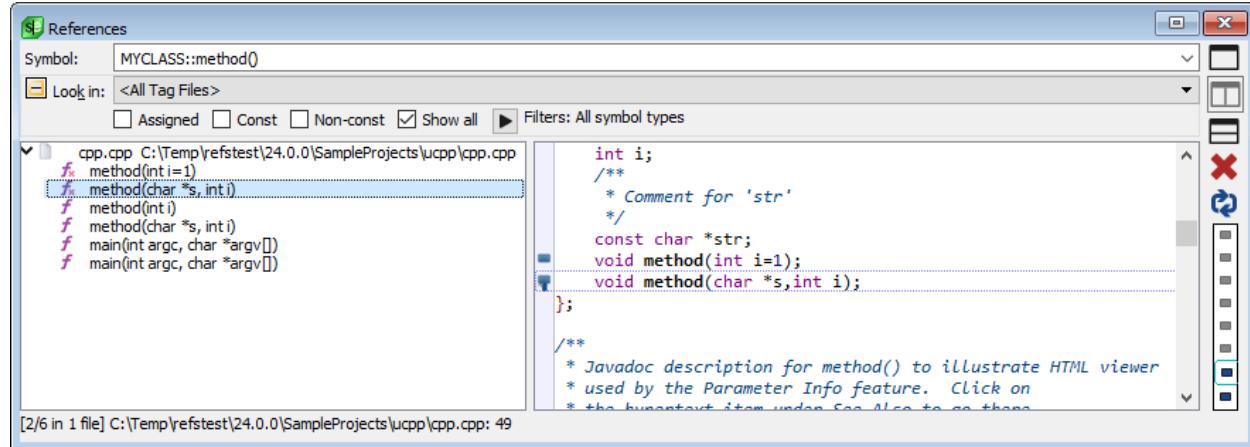
Editor Element in Use	Preview Window Display
Symbol Uses/Call Tree and References tool windows	The Preview window shows the location of the symbol reference or use. See Viewing Symbol Uses with the Calling Tree and References Tool Window for more information.
Symbol Refs/Callers Tree tool window	Single-click on a symbol and the Preview window displays the selected symbol and its documentation comments, if any exist. See Viewing Symbol Callers Tree for more information.
The Base Classes and Derived Classes tool windows	Single-click on a symbol and the Preview window displays the selected symbol and its documentation comments, if any exist. See Viewing Base and Derived Classes for more information.
The Bookmarks tool window	Single-click on a bookmark and the Preview window displays the location of the bookmark. See Bookmarks Tool Window for more information.
The Breakpoints tool window	Single-click on a breakpoint and the Preview window displays the location of the breakpoint. See Setting Breakpoints for more information.
The Message List tool window	Single-click on a message and the Preview window displays the message type, and the location of the message in the source code. See Message List Tool Window 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 Search Results Output 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 List Members and Auto-Complete for more information.

References Tool Window (Pro only)

References Tool Window (Pro only)

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 clicking **View → Tool Windows → 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 context 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 [Configuring Other Languages](#) for more information.

The References tool window supports filtering of the symbol references results using the right-click menu to set filtering options. This allows you to restrict the set of references to those that occur within certain types of symbols. Another important feature of this filtering is the ability to filter out unrecognized symbols (occurrences of a symbol name that tagging was not able to find) by unchecking the **Others+Unrecognized** filtering option. Normally, these would be displayed, because the system was unable to prove that the symbol was or was not an instance of the symbol we are searching for.

The **Symbol** combo box displays the symbol the references search is for. Pull down the combo box to select past references searches.

References search options can be fine-tuned by expanding the set of **Look in** options. The following options can be used to narrow down the search results in order to locate specific kinds of references to a symbol.

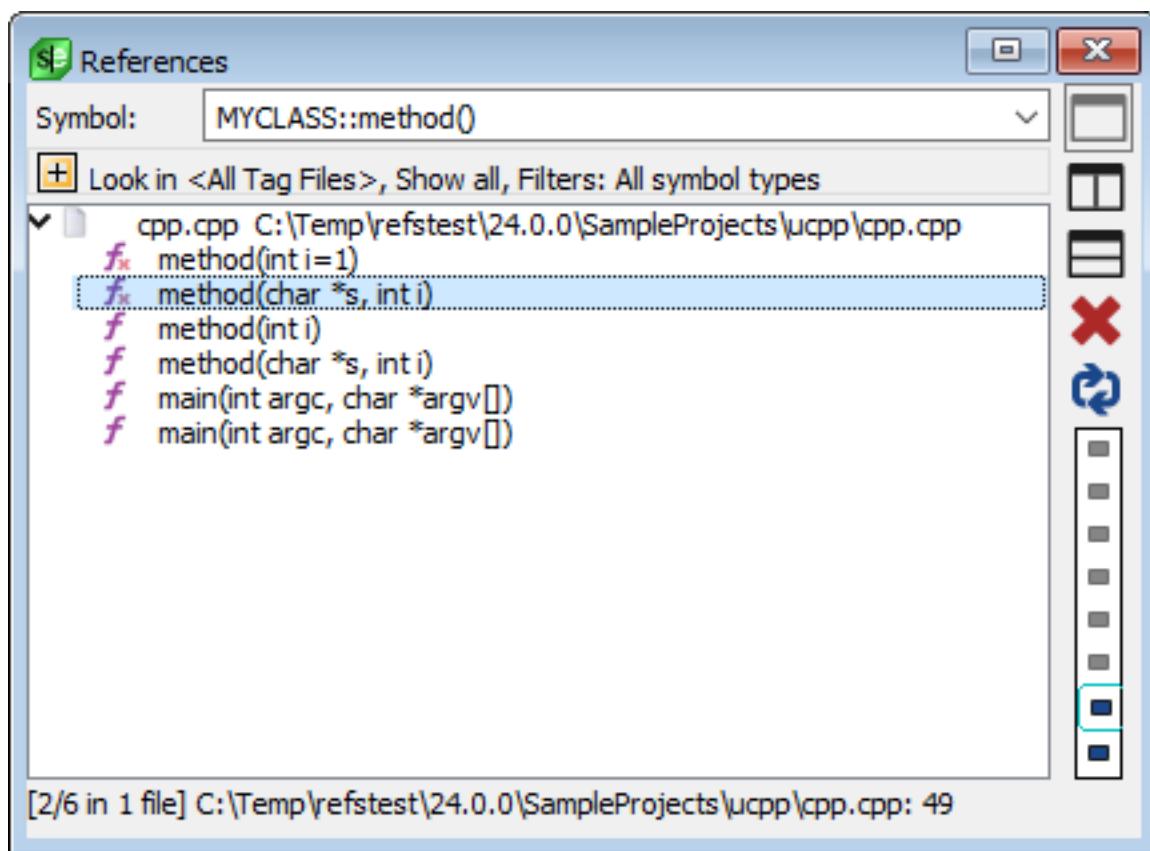
- **Look in** - The **Look in** combo box displays the scope of the references search. The default is to search the **Current Workspace**, but you can fine-tune the search to restrict its scope to the **Current Project**, **All Open Files** or just the **Current File**. In addition, you can restrict the scope to just projects in the workspace which contain the current file by selecting **Projects Containing Current File**. Likewise, you can expand the references search to include **All Tag Files** - this will also include language-specific tag files that are built with support for symbol cross-referencing. In any case, in addition to the **Look in** restrictions specified, the file containing the declaration and/or definition of the symbol will also be included.
- **Assigned** - For variables, the **Assigned** check box can be used to filter the set of references down to only the statements where the variable appeared to be used in an assignment statement. Note that this is less general than looking for strictly *write* references to a variable, as the references engine is not able to detect cases where a variable is passed to a function by reference, or cases where an object calls a non-const method.
- **Const** - The **Const** check box can be used to filter the set of references down to only the statements where the symbol is references in a const or read-only manner. This includes unqualified references in const methods, as well as qualified references where the expression before the symbol name evaluates to a const type.
- **Non-const** - The **Non-const** check box can be used to filter the set of references down to only the statements where the symbol is references in a non-const or other manner potentially allowing a write. Note that generally assignment statements are a subset of non-const references.
- **Show all** - The **Show all** check box will appear checked when all the filtering options are cleared and you are assured that all symbol references that can be found are being shown in the list of references. If **Show all** is not checked, clicking on it again will clear all filtering options other than the **Look in** scope option.
- **Filters** - Use filters to restrict the search to certain types of symbol contexts. The filters are the same the ones available on the Definitions tool window. See [Defs Tool Window](#) 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 tool tip 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.

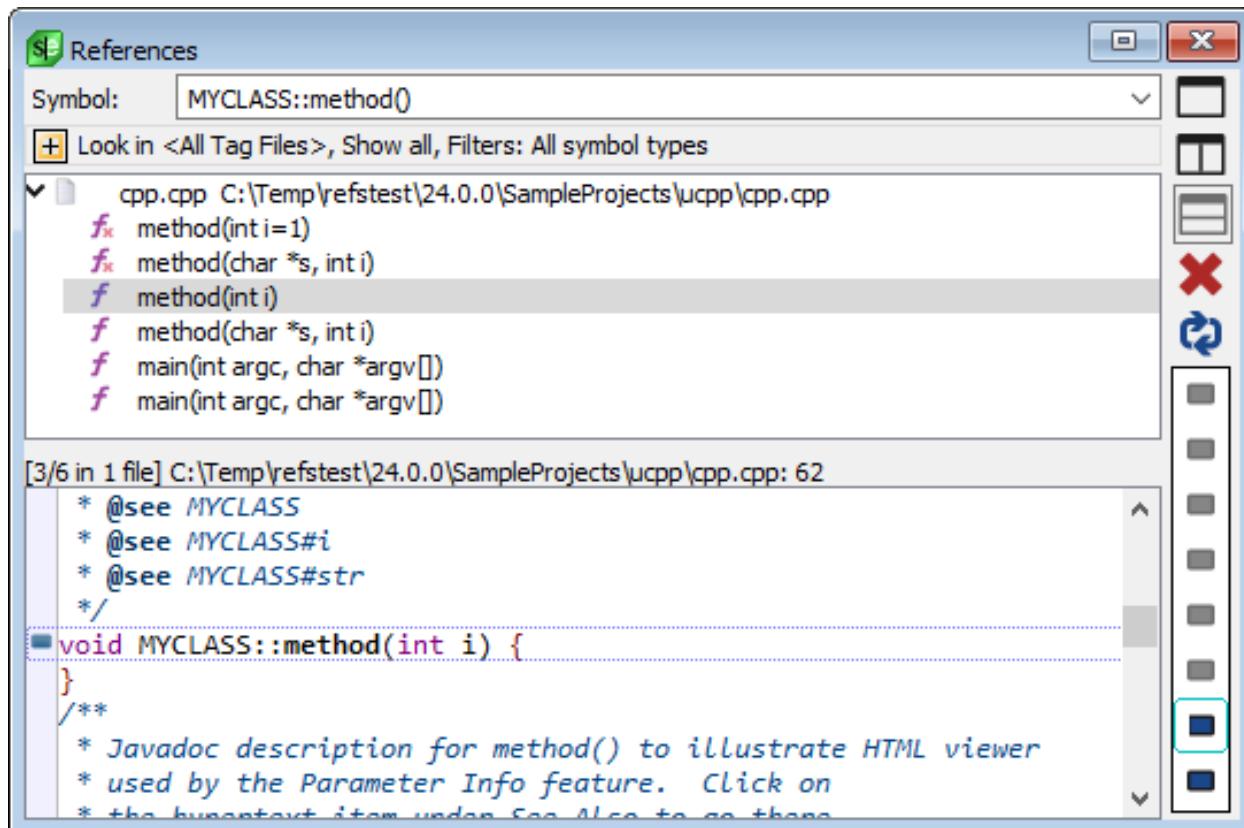
Use the buttons located at the top right corner of the tool window to toggle the preview pane on and off. Because source can also be previewed in the [Preview Tool Window](#), you may find it more efficient to use the References window with the preview pane off.

References Tool Window (Pro only)



You can also configure the References tool window to place the preview pane below the list of references if you prefer a vertical arrangement.

References Tool Window (Pro only)



Tip

When working with a large display, it can be beneficial to relocate the References tool window to the lower-left corner of the editor, docked below the tool window group containing Files, Projects, Defs, and Symbols, and to the left of the tool window group containing the Preview window. In this configuration, you can switch the References tool window to use the single-pane view.

Use the 'Delete' button to pop the top-most references stack item from the list.

Use the 'Add' button to add or refresh the current references stack item.

Use the references stack indicator bar to see the number of searches on the references stack, and to jump to a specific set of search results. If you hover the mouse over one of the enabled buttons, a tooltip will show you the symbol which was searched for. You can have up to 10 items on the references search stack. By default an item is added to the stack whenever you do a new references search.

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.

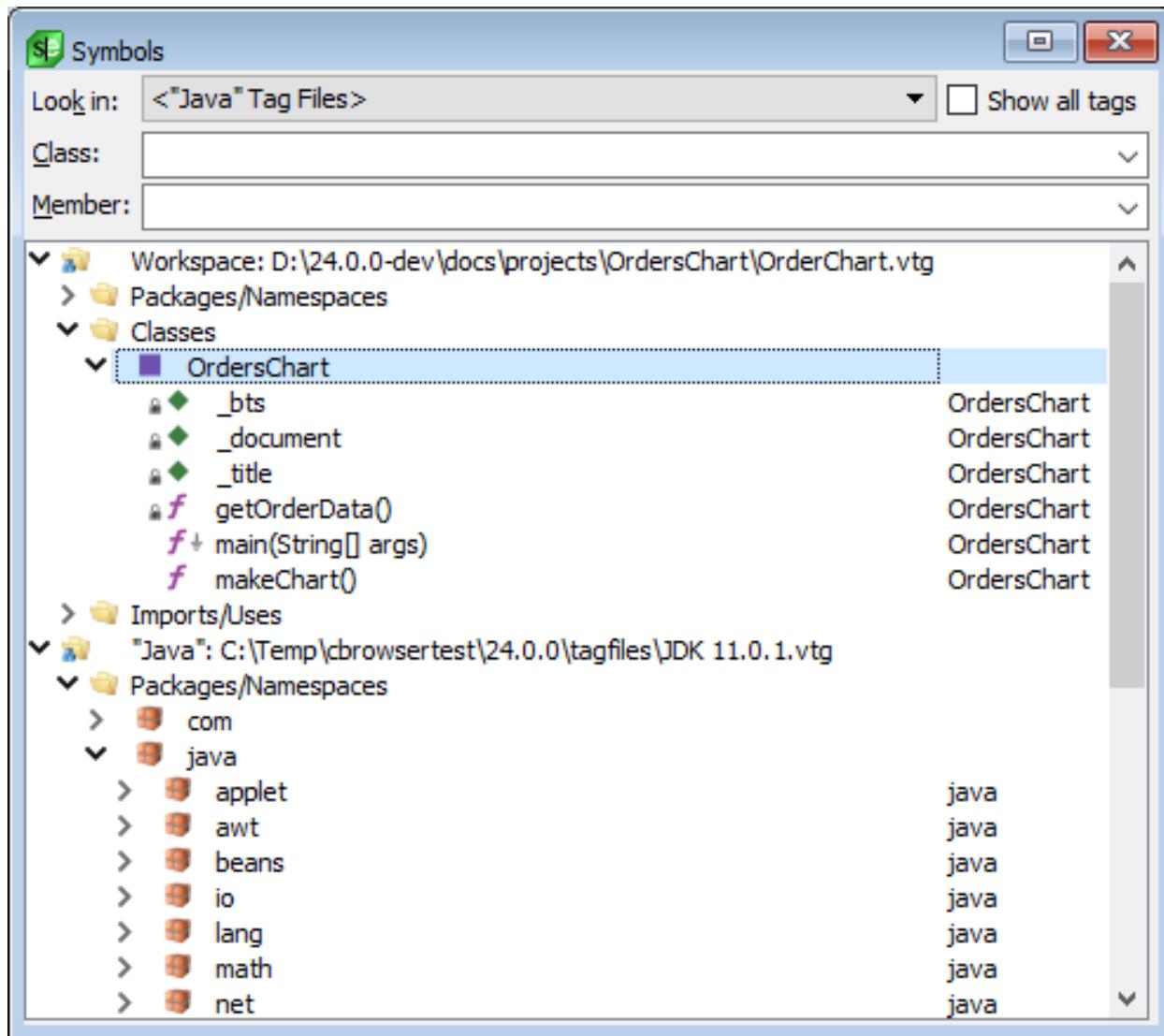
- **Print** - Displays the Print dialog, where you can configure options for printing the tree.
- **Save Subtree** and **Print Subtree** - These options function similarly to the above except they apply to the selected subtree.
- **Send to Search Results** - Send list of References to the Search Results tool window.
- **Quick filters, Scope, Functions, Variables, Data Types, Statements, and Others** - All of these items are for filtering the data displayed in the References tool window.

Symbols Tool Window (Pro only)

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 clicking **View** → **Tool Windows** → **Symbols** or by using the **toggle_symbols** command. To display the tool window on demand, use the **activate_symbols** command.

Symbols Tool Window (Pro only)



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.

The list of tag files can be controlled by selecting one of the following options from the **Look in:** combo box.

- **<All Tag Files>** - This is the default setting. Select this setting to browse all tag files for all languages.
- **<Use Context Tagging®>** - Uses Context Tagging to intelligently determine which tag files to browse, based on your current workspace and current language mode.
- **<Current Workspace>** - Select this setting to only browse tag files that are in the current workspace.
- **<Current Project>** - Select this setting to only browse symbols in the tag file associated with the current project. Note that this is essentially the same as **<Current Workspace>** unless you have a project-specific tag file for the current project.

- <Projects Containing Current File> - Select this setting to only browse symbols in the tag files associated with all projects that contain the current file. Note that this is essentially the same as <Current Workspace> unless you have a project-specific tag files.
- <Language Tag Files> - Select this setting to browse all language-specific tag files for the indicated extension. This may also include your workspace and project tag files.

Hover the mouse over the bitmap of a symbol to see a tool tip 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. You can also use a regular expression (see below).

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".

Note

- The filters can also use regular expressions, using the regular expression syntax defined in the default search options.
- The filters use the case-sensitivity options defined in the default search options.
- 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.
- **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.
- **Rebuild Tag File** - Rebuild the tag file currently being explored in the Symbols tool window.
- **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+I** in the editor window. See [Symbol Navigation](#) for more information.

- **Calls or 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.
- **Callers** - Displays a tree of symbols that call or use this function. See [Viewing Symbol Callers Tree](#) for more information.

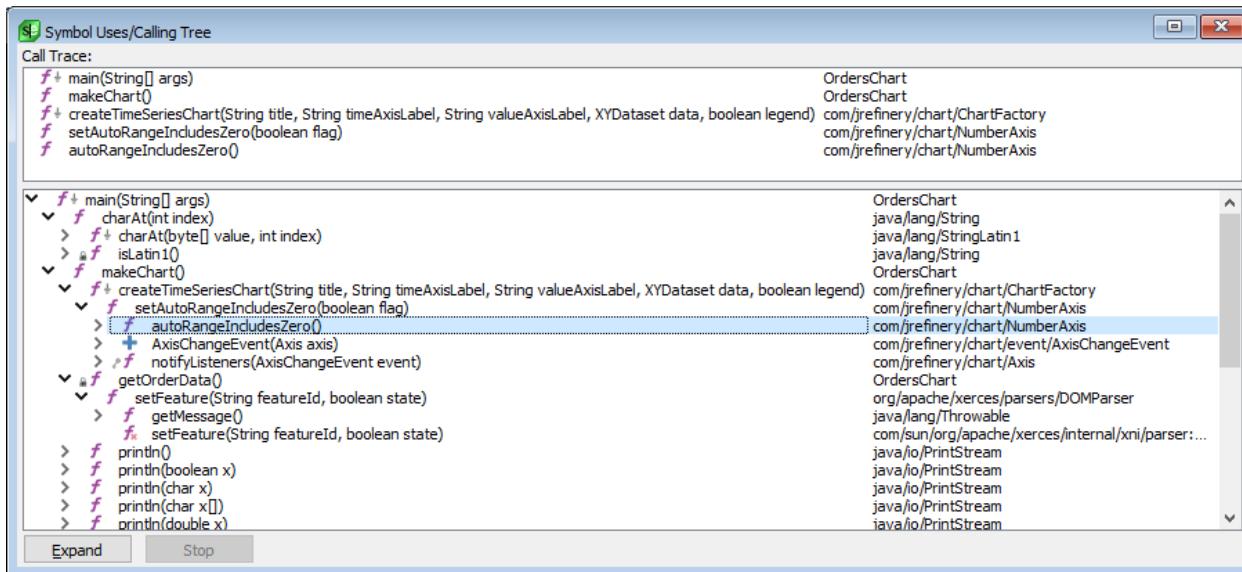
Viewing Symbol Uses with the Calling Tree (Pro only)

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](#) or [Class Tool Window](#) by right-clicking on a symbol and selecting **Calls or uses**, or by right-clicking on a symbol in the editor and selecting **Show Symbol+Calls or uses**.



The **Call Trace** list on the top shows a trace from the top-most item in the call tree to the currently selected item. This provides a simple overview of where you are when navigating the 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 Symbol Callers Tree (Pro only)

View symbol references to see what symbols (variables, functions, methods, classes, etc.) call or use a specific function or method.

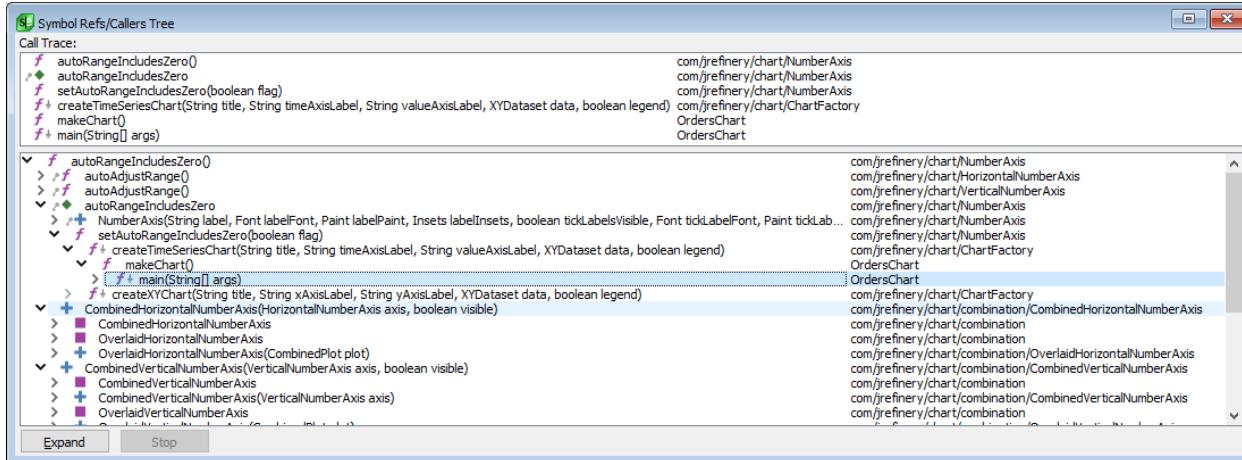
To view the symbols that call or use a particular function or method, first create a project or open an existing project. Then from the Symbols tool window, right-click on the desired function or method and select **Calls or uses**. The Symbol Refs/Callers Tree dialog will be displayed.

Note

The Symbol Refs/Callers Tree is similar in some ways to the [References Tool Window](#), because it finds what other symbols reference the originating symbol. The distinction is that it displays a multi-level caller relationship hierarchy rather than simply focusing in finding all the specific locations where a symbol is referenced.

Tip

You can also access the Symbol Refs/Callers Tree from within the [Defs Tool Window](#) or [Class Tool Window](#) by right-clicking on a symbol and selecting **Callers**, or by right-clicking on a symbol in the editor and selecting **Show Symbol+Callers**.



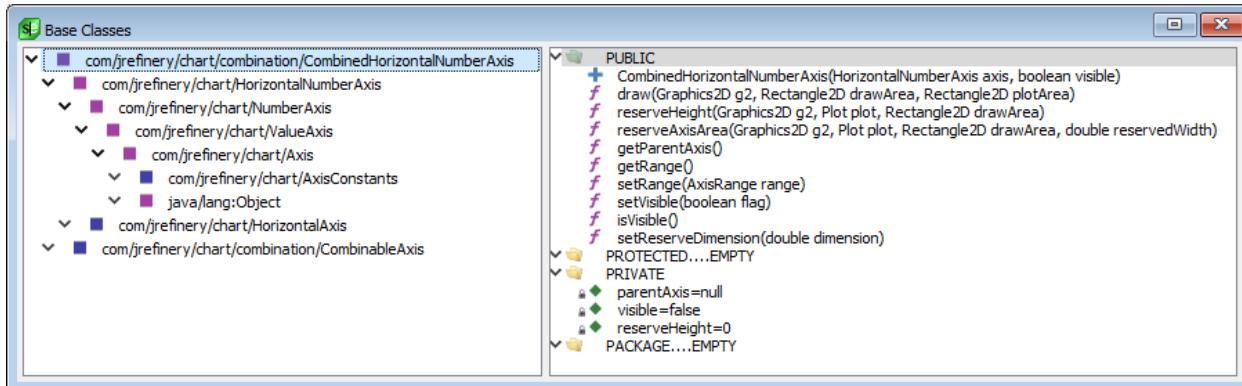
The **Call Trace** list on the top shows a trace from the top-most item in the callers tree to the currently selected item. This provides a simple overview of where you are when navigating the callers tree.

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.

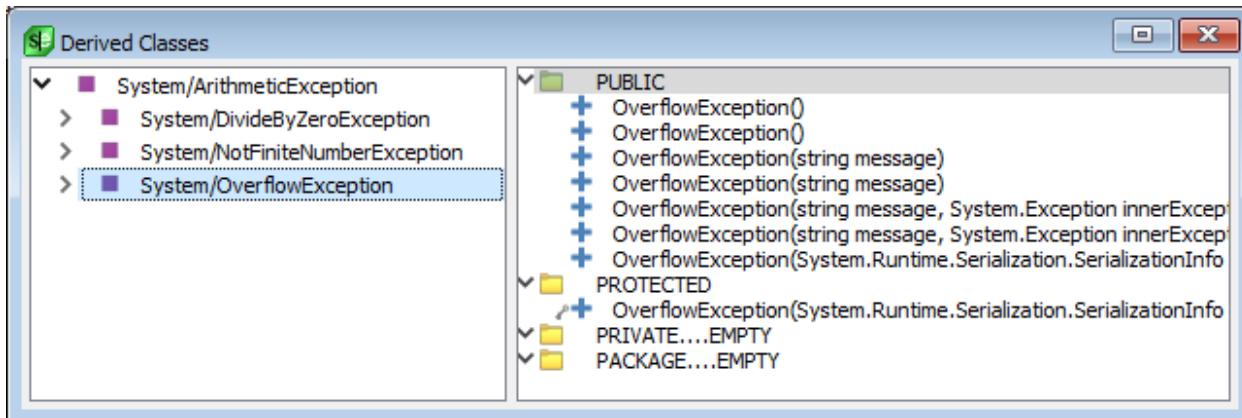
If the focus is in the Symbol Refs/Callers Tree dialog, the selected item will be shown in the [Preview Tool Window](#) tool window, just as it is in the [Symbols Tool Window](#).

Viewing Base and Derived Classes (Pro only)

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.

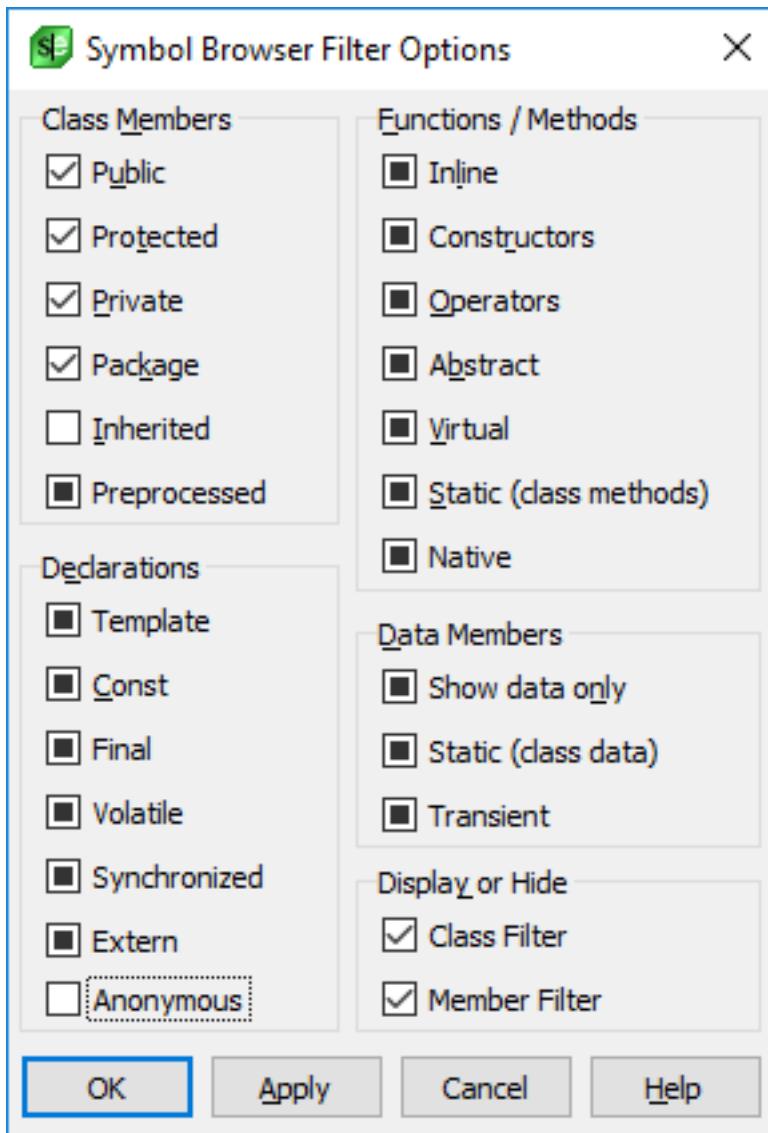
The left pane of each dialog contains a tree showing the class inheritance hierarchy (the class list). The right pane shows a list of the members of the selected class (the member list).

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 (Pro only)

To access symbol browser filter options, right-click in the Symbols tool window and click **Filters** → **Filtering Options**.



Each option has three states: If the option is selected, only the specified items will be displayed. If the option is cleared, 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 cleared, 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 cleared, only non-preprocess members displayed.
- **Declarations**
 - **Template** - (C++ only) When selected, only template classes are displayed. When cleared, only non-template classes are displayed.
 - **Const** - (C++ only) When selected, only methods which do not modify members (**method1() const**) are displayed. When cleared, 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 click **View** → **Tool Windows** → **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 cleared, only non-final members are displayed.
- **Volatile** - (C++ only) When selected, only volatile method members (**method1() volatile**) are displayed. When cleared, only non-volatile members are displayed.
- **Synchronized** - (Java only) When selected, only synchronized members are displayed. When cleared, only non-synchronized members are displayed.
- **Extern** - When selected, only identifiers defined explicitly using the **extern** keyword are displayed. When cleared, 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 cleared, 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 cleared, only non-abstract methods are displayed.
 - **Virtual** - When selected, only virtual methods are displayed. When cleared, 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 cleared, only non-static methods are displayed.
 - **Native** - When selected, only methods explicitly defined with the native keyword are displayed. When cleared, only non-native methods are displayed.

Symbol Properties Tool Window (Pro only)

- **Data Members**

- **Show data only** - When selected, only data members are displayed. When cleared, only methods are displayed.
- **Static (class data)** - When selected, only static data members are displayed. When cleared, only non-static data members are displayed.
- **Transient** - (Java only) When selected, only transient data members are displayed. When cleared, only non-transient data members are displayed.

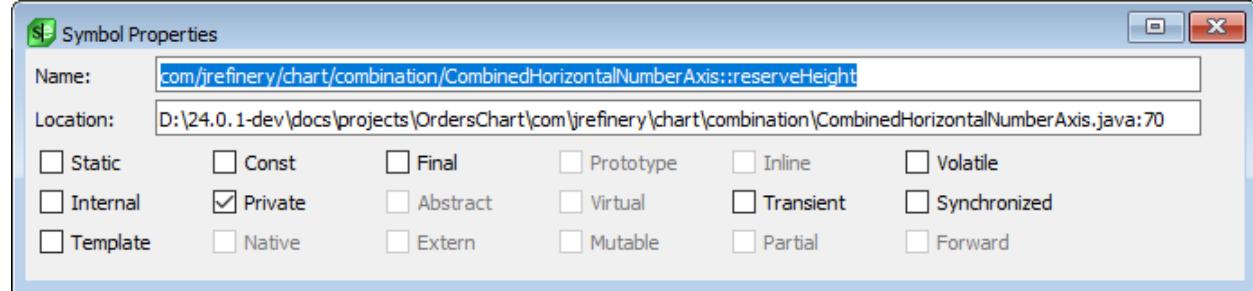
- **Display or Hide**

- **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 (Pro only)

The Symbol Properties tool window displays detailed symbol property information for the symbol at the cursor location. Note that this window is read-only, so you can't use it to change properties.

Display can be toggled on/off by clicking **View → Tool Windows → Symbol Properties**. To display the tool window on demand, right-click on a symbol in the Symbols tool window and select **Properties** or use the **activate_tag_properties_toolbar** command.

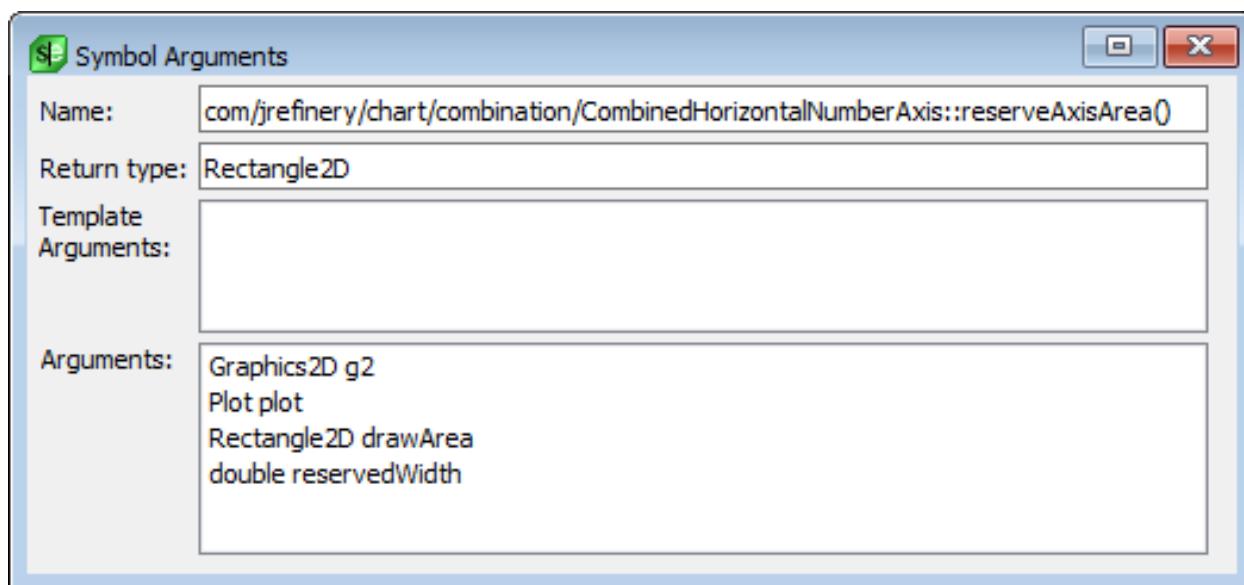


Symbol Arguments Tool Window (Pro only)

The Symbol Arguments tool window displays the return type and both template and function arguments for the symbol at the cursor location. Note that this window is read-only, so you can't use it to change the symbol.

Display can be toggled on/off by clicking **View → Tool Windows → Symbol Arguments**. To display the tool window on demand, right-click on a symbol in the Symbols tool window and select **Arguments** or use the **activate_tag_arguments_toolbar** command.

Viewing and Displaying



Viewing and Displaying

SlickEdit® offers several features and options regarding viewing and displaying. See the following topics for more information:

- [Colors and Color Coding](#)
- [Current Line](#)
- [Modified Lines](#)
- [Viewing Special Characters](#)
- [Viewing Line Numbers](#)
- [Soft Wrap](#)
- [Selective Display](#)
- [Hex/Line Hex View](#)
- [Other Display Options](#)

Colors and Color Coding

SlickEdit provides comprehensive capabilities to color the text in the editor window. For more information, see [Colors, Color Coding, and Symbol Colors](#).

Current Line

SlickEdit provides two ways to highlight the current line:

- **Draw a box around the current line** - You can enable the **Current line highlight** for all languages. This draws a box around the current line. You can specify what type of box to use: a plain box, a tabs ruler, a syntax indent ruler, or a decimal ruler. You can also control the color of the box and column markers when using a ruler. To enable the current line highlight and specify options, open the Options dialog (**Tools** → **Options**), expand **Appearance** and select **General**, then modify the options under the **Current line highlight** category.
- **Change the background and foreground color** - On a language-specific basis, you can enable a different background and foreground color for the current line. From the main menu, select **Tools** → **Options** → **Languages**, expand your language category and language, then select **View**. Put a check in **Current line**. To select the colors for the foreground and background, select **Tools** → **Options** → **Appearance** → **Colors**, then select **Current Line** under the **Selections** node. For more information on setting colors. see [Colors](#).

Note

Depending on the background color you select, SlickEdit will still use the foreground colors you have selected for other color coding elements, like strings, comments, etc. If the color you select for the background would make those colors too hard to read, SlickEdit will apply the foreground color selected for the **Current line** element.

Modified Lines

You can mark lines that have been modified or inserted during the current editing session. This will display a color indicator in the left margin of editor windows for each changed line. To enable this feature, select **Tools** → **Options** → **Languages** then expand your language category and select the language you are configuring. Select the **View** options and put a check in **Modified lines**. To select the colors to use, select **Tools** → **Options** → **Appearance** → **Colors**, then select **Modified Line** or **Inserted Line** under the **Modifications** node. For more information about setting colors, see [Colors](#).

SlickEdit can clear the modified and inserted line color when you save a file. To activate this feature, from the main menu, click **Tools** → **Options**, expand **File Options** and select the **Save** node. Then set the **Reset modified lines** option to **True**.

Tip

To show the modified lines on demand, 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.

Viewing Special Characters

By default, many important characters are not visible in the editor, like tabs, spaces, and newline characters. When you enable view of these special characters, SlickEdit® displays a visible character to represent the invisible characters.

You can enable view of special characters on a per-document or language-specific basis:

- **For the current document** - From the main menu, click **View** → **Special Chars**, or use the **view_specialchars_toggle** command. This toggles the display of all special characters (tabs, spaces, and newline characters) on and off. The menu also provides options to toggle display of these characters individually. See [View Menu](#) for more information.
- **For a specific language** - Using the Options dialog (**Tools** → **Options**), expand the **Languages** node and your language, then select **View**. Select the option **Special Characters**. This enables the display of all special characters for the chosen language. Alternately, you can select to enable display of the characters individually.

To define the characters that are displayed to represent the special characters, see [Defining Special](#)

[Characters](#). To define the colors that are used for special characters, see [Changing the Color of Special Characters](#).

Note

- Viewing special characters is only available for ASCII files.
- When the display of special characters is enabled along 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).

Defining Special Characters

To define the characters that are displayed to represent the special characters, from the main menu, click **Tools → Options**, expand **Appearance** and select **Special Characters**. Enter the character codes that you wish to use.

The CR and LF characters are only shown for end of line when End-Of-Line is set to a zero length string for Unicode Editor Windows or 13 for Non-Unicode Editor Windows.

Changing the Color of Special Characters

To change the colors and styles of special characters, use the **Color** options screen (**Tools → Options → Appearance → Colors**). Select **Special Characters** from the screen element drop-down list. For more information on color settings, see [Colors, Color Coding, and Symbol Colors](#).

Viewing Line Numbers

The current line number is always displayed in the editor's status line (see [The SlickEdit Interface](#)). Click on the line number indicator to display the **Go to Line** dialog. This shows the total number of lines and allows you to navigate to a specific line.

Line numbers can be displayed in the left margin area. You can enable them for a single document, a single language, or for all languages:

- **For the current document** - From the main menu, click **View → Line Numbers**, or use the `view_line_numbers_toggle` command. This toggles the display of line numbers on and off.
- **For a specific language** - Using the Options dialog (**Tools → Options**), expand the **Languages** node and your language, then select **View**. Select the option **Line numbers**.
- **For all languages** - From the main menu, select **Tools → Quick Start Configuration**. The Coding screen allows you to turn on line numbers for all languages.

You can select options that control the width of the line numbers. For more information see [Language-Specific View Options](#).

Tip

- To control whether a colon is displayed with line numbers, use the **line_numbers_show_colon** command. At the command line prompt, type **Y** (for yes) or **N** (for no).
- To change the amount of space used in the left margin of editor windows for line numbers, use the **line_numbers_set_width** command. At the command line prompt, enter the number of the desired width, in pixels.
- To change the color of line numbers, select **Tools → Options → Appearance → Colors** node in the Options dialog, and select the **Line Number** screen element.

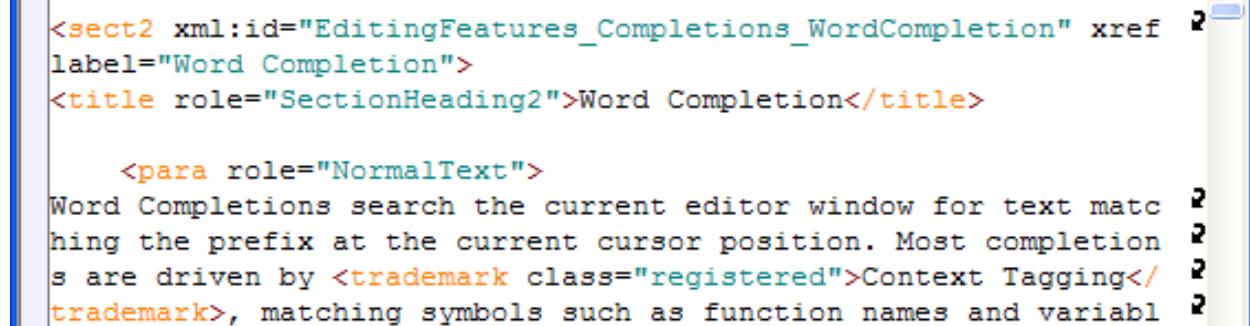
Soft Wrap

Soft Wrap makes it easy to view long lines of code without scrolling. When Soft Wrap is enabled, each line is wrapped as though a carriage return was inserted; however, unlike Word Wrap, the file itself is not modified.

You can enable Soft Wrap on a per-document or language-specific basis:

- **For the current document** - From the main menu, click **View → Soft Wrap**, or use the **softwrap_toggle** command. This toggles Soft Wrap on and off.
- **For a specific language** - Using the Options dialog (**Tools → Options**), expand the **Languages** node and your language, then select **Word Wrap**. Select the option **Wrap long lines to window width**. Options are also available here to break the text at the end of a word so that words are kept whole (**Break on word boundary**), and to enable Soft Wrap for all languages so you don't have to configure each one (**Enable soft wrap**).

When Soft Wrap is on, arrows in the right margin indicate lines that are wrapped. The following screen shows Soft Wrap enabled in an XML document:



```
<sect2 xml:id="EditingFeatures_Completions_WordCompletion" xref="2"
      label="Word Completion">
<title role="SectionHeading2">Word Completion</title>

      <para role="NormalText">
Word Completions search the current editor window for text matching the prefix at the current cursor position. Most completions are driven by <trademark class="registered">Context Tagging</trademark>, matching symbols such as function names and variabl
```

For related information, see *Word Wrap*, *Comment Wrap*, and *Reflowing Text*.

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 those regions that are relevant to your current editing session.

Use the Selective Display dialog to activate this feature and to specify the type of regions to display or hide. This dialog is displayed by clicking **View** → **Selective Display**, or by using the **selective_display** command. For more information, see [View Dialogs and Tool Windows](#). For a description of additional menu entries for Selective Display, see [View Menu](#).

When Selective Display is active, a **Plus** (+) or **Minus** (-) bitmap is placed before hidden or expanded lines in the editor window margin. The following screen shot shows a sample file with two function definitions expanded and the rest collapsed. If the mouse is hovering over the **Plus** (+) of a collapsed code region, a tooltip will be displayed showing the code which is hidden. If the hidden region is a documentation comment, the tooltip will display the formatted documentation comment.

The screenshot shows a Java code editor window titled "Rectangle.java". The code defines a class Rectangle with several methods. Two methods, `public Rectangle (float x, float y, float width, float height)` and `public Rectangle (Rectangle rect)`, are currently expanded, showing their internal code. Other methods like `getY()`, `setWidth (float width)`, and `setSize (float width, float height)` are collapsed, indicated by a minus sign (-) in the margin. A tooltip is visible above the collapsed block of code for `setWidth (float width)`, displaying the entire method definition. The status bar at the bottom right shows "Line 115 Col 1 No Selection RW REC Ins 09".

```

public Rectangle (float x, float y, float width, float height) {
    /* Constructs a rectangle based on the given rectangle
    Expand code block
    public Rectangle (Rectangle rect) {
        x = rect.x;
        y = rect.y;
        width = rect.width;
        height = rect.height;
    }
    the bottom left corner */
    the bottom left corner
}

public float getY () {
    /* Sets the y-coordinate of the bottom left corner
    public Rectangle setY (float y) {
        /* return the width */
    public float getWidth () {
        return width;
    }

    /* Sets the width of this rectangle
    public Rectangle setWidth (float width) {
        /* return the height */
    public float getHeight () {
        return height;
    }

    /* Sets the height of this rectangle
    public Rectangle setHeight (float height) {
        /* return the Vector2 with coordinates of this rectangle
    public Vector2 getPosition (Vector2 position) {
        /* Sets the x and y-coordinates of the bottom left corner from vector
    public Rectangle setPosition (Vector2 position) {
        /* Sets the x and y-coordinates of the bottom left corner
    public Rectangle setPosition (float x, float y) {
        /* Sets the width and height of this rectangle
    public Rectangle setSize (float width, float height) {
}

```

When Selective Display is active, you can perform the following operations:

- **Display or hide lines** - Double-click on the **Plus** (+) or **Minus** (-) bitmaps. Alternatively, click **View** → **Expand/Collapse Block**, press **Ctrl+I**, or use the **plusminus** command. See [Expanding/Collapsing](#)

[Code Blocks](#) for more details.

- **Copy visible text to the clipboard** - Click **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.
- **Redisplay all lines and remove the bitmaps** - From the main menu click **View → Show All(show_all)** command).

To define the type of information to show/hide, see [Selective Display Regions](#).

Expanding/Collapsing Code Blocks

SlickEdit® provides a more keyboard-centric way to expand and collapse code blocks. 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.

Note

- The definition of a "code block" is based on your language.
- Selective Display bitmaps 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, click **Tools → Options → Appearance → Advanced**, then set the value of **Expand/collapse** to **Expand on single click**.
- 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.

- [Selective Display Tool Window](#) - Displays lines that contain the specified search string or lines that do not contain the specified string.
- [Function Headers](#) - 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 outline](#) - Select this option to set multiple levels of Selective Display based on symbols, statements, braces or indent.
- [Comments](#) - Select this option to set collapse multi-line comments.
- [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 [View Dialogs and Tool Windows](#) for more information and details about the available settings.

Hex/Line Hex View

SlickEdit® supports hex/line hex viewing and editing on a per-document or per-language basis.

Hex view displays your code using hexadecimal values to represent each character. The ASCII representation is also shown on the right side of the editor window. When you position the cursor in one representation, the corresponding location in the other is highlighted. You can edit by changing the hex values or by changing the ASCII characters.

Line hex preserves your code layout, formatting, and color coding. It shows the hexadecimal value for each character below the corresponding character. Since two hex digits are needed, the value is displayed in a column below the corresponding character with the most significant digit at the top. This makes it easier to read your code and still see the hex values for each character. As with regular hex view, you can edit either the ASCII representation or the hexadecimal values.

See also [Hex Mode Editing](#) for more information.

Other Display Options

This section describes other general display options that you might find useful.

Displaying 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** (+) bitmap can be

displayed on line 0.

To enable this option, from the main menu, click **Tools** → **Options**, expand **Appearance** and select **General**, then set the **Top of file line** option to **True**.

Rather than using this option, you can use **Ctrl+Shift+Enter** (**Ctrl+Enter** in the Visual C++ and Visual Studio default emulations) 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, click **Tools** → **Options**, expand **Appearance** and select **General**, then 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 **Vertical line color** option to change the color of the vertical line. Note that the vertical line will only be displayed for fixed-width fonts. It will not be displayed for proportional fonts, as are used for Unicode files.

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. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Editing**.
2. 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 activate tab indenting, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then click **Editing**. Select the **Indent with tabs** option.

Setting Tab Spacing

Typical default values for the **Tab** key are four or 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.

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.

Note

- 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 [Redefining Common Keys](#) for information on other **Tab** and **Shift+Tab** key bindings. Regardless of the **Tab** key binding, if the language-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, from the main menu, click **Tools** → **Options**, expand **Keyboard** and select **Redefine Common Keys**, then set the **Backspace over tab** 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.

SmartPaste is enabled by default, and can be turned on and off from the language-specific Indent option screen. To access these options, from the main menu, click **Tools** → **Options**. Expand **Languages** in the tree, select the language category and language, then click **Editing**. Select or clear the **Use SmartPaste®** option.

Note

SmartPaste only works with line selections. For information about creating a line selection, see [Line Selections](#).

Adaptive Formatting

Many development teams set standards for code formatting styles. These standards often vary from project to project or between languages. In this environment, you can lose valuable time in having to change configurations, set/unset options, or run beautifiers from file to file just so you can meet the team's requirements.

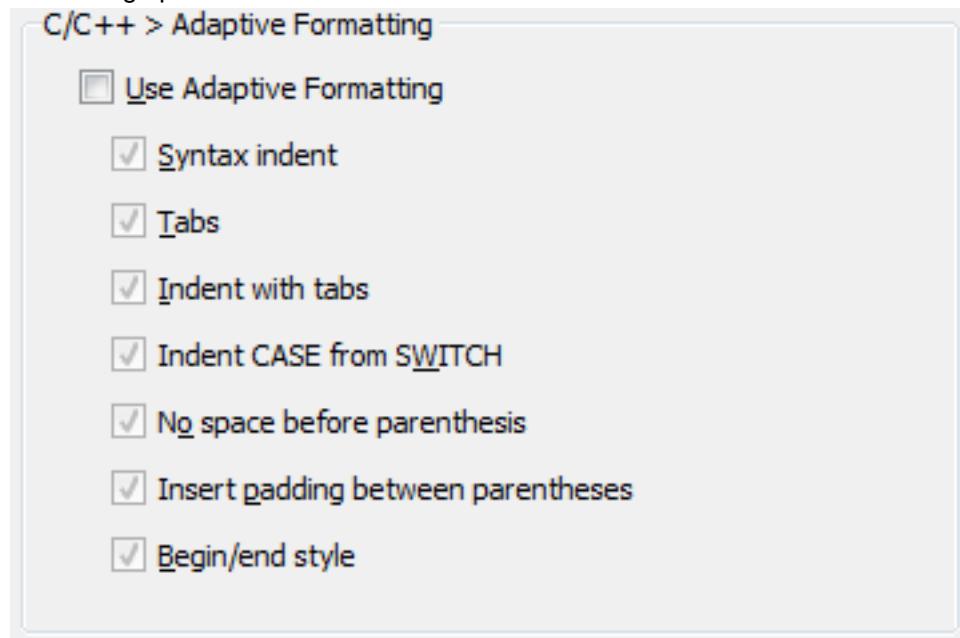
Adaptive Formatting addresses these situations by scanning a file for the formatting styles in use, and automatically matching those settings for the current editing session. This provides seamless integration of new code with existing code, making it easier to read, not only for you but for the next person who needs to edit the file.

Adaptive Formatting recognizes indentation and tab style settings, parentheses padding, and begin/end style settings. It also recognizes case settings, such as keyword casing for case-insensitive languages, and tag, attribute, and value casing for HTML-based languages.

[Feature Notifications](#) are used when Adaptive Formatting identifies formatting that conflicts with your settings. With Feature Notifications, you can control whether you get a dialog, a pop-up message, or no notice at all. By default, Adaptive Formatting is set to display a pop-up message. You can change the notification level by selecting **Tools** → **Options** → **Application Options** → **Notifications**.

Enabling/Disabling Adaptive Formatting

Adaptive Formatting is disabled by default. You can toggle it on and off on a language-specific basis, and you can also enable/disable each of the individual formatting settings on a language-specific basis as well. To access these options, from the main menu, select **Tools** → **Options** → **Languages** → **Application Languages** → **[Language]** → **Adaptive Formatting**. As an example, the C/C++ Adaptive Formatting options are shown below.



The specific Adaptive Formatting settings available for that language are shown on the options screen. Select or clear the **Use Adaptive Formatting** check box to enable or disable the feature. When Adaptive Formatting is enabled, use the subsequent check boxes to select the settings for which SlickEdit® should scan.

You can also toggle Adaptive Formatting on and off for a language by using the menu item **Document** → **Adaptive Formatting** (or the **adaptive_format_toggle** command). This turns the feature on and off for the current language without affecting the individual style settings.

Recognized Settings

The table below shows each option recognized by Adaptive Formatting, the path to the option in SlickEdit®, and a description, along with a link for more information. Because these options are language-specific, the option paths are relative to the language-specific portion of the Options dialog: click **Tools** → **Options** → **Languages**, expand your language category and then select your language.

Formatting Option	Path to Option in SlickEdit	Description
Syntax indent	Formatting (Language-Specific Formatting Options)	When enabled, the Enter key indents according to the language syntax, and you can specify the amount to indent for each level. See Syntax Indent for information.
Tabs	Formatting (Language-Specific Formatting Options)	Specifies tab stops, which can be in increments of a specific value or at specific column positions. See Indenting with Tabs for information.
Indent with tabs	Formatting (Language-Specific Formatting Options)	Determines whether the Tab key, Enter key, and paragraph reformat commands indent with spaces or tabs. See Indenting with Tabs for information.
Begin/End style	Formatting Options (Language-Specific Formatting Options)	The is the brace style used for Syntax Expansion and smart indenting. See the section for your language in the Language-Specific Editing chapter. For example, Language-Specific Formatting Options .
Indent CASE from SWITCH and Indent constant from case	Formatting Options (Language-Specific Formatting Options)	Specifies Syntax Expansion indentation. See the section for

Recognized Settings

Formatting Option	Path to Option in SlickEdit	Description
		your language in the Language-Specific Editing chapter. For example, Language-Specific Formatting Options .
No space before parenthesis	Formatting Options (Language-Specific Formatting Options)	When enabled, no space is placed between keywords (such as if , for , or while) and the open paren when Syntax Expansion occurs. See the section for your language in the Language-Specific Editing chapter. For example, Language-Specific Formatting Options .
Insert padding between parenthesis	Formatting Options (Language-Specific Formatting Options)	When enabled, a space is placed after the open parenthesis and before the close parenthesis, providing padding for the enclosed text. See the section for your language in the Language-Specific Editing chapter. For example, Language-Specific Formatting Options .
Keyword case	Formatting Options (Language-Specific Formatting Options)	Specifies the case of keywords used by Syntax Expansion. See the section for your language in the Language-Specific Editing chapter. For example, Ada Formatting Options .
Case for inserted tags	Formatting Options (Language-Specific Formatting Options)	Specifies the case for tags that are automatically inserted. See HTML Formatting Options for more information.
Case for inserted attributes	Formatting Options (Language-Specific Formatting Options)	Specifies the case for tag attributes that are automatically inserted. See HTML Formatting Options for more information.
Case for inserted single word values	Formatting Options (Language-Specific Formatting Options)	Specifies the case of word values that are automatically inserted inside a tag. See HTML

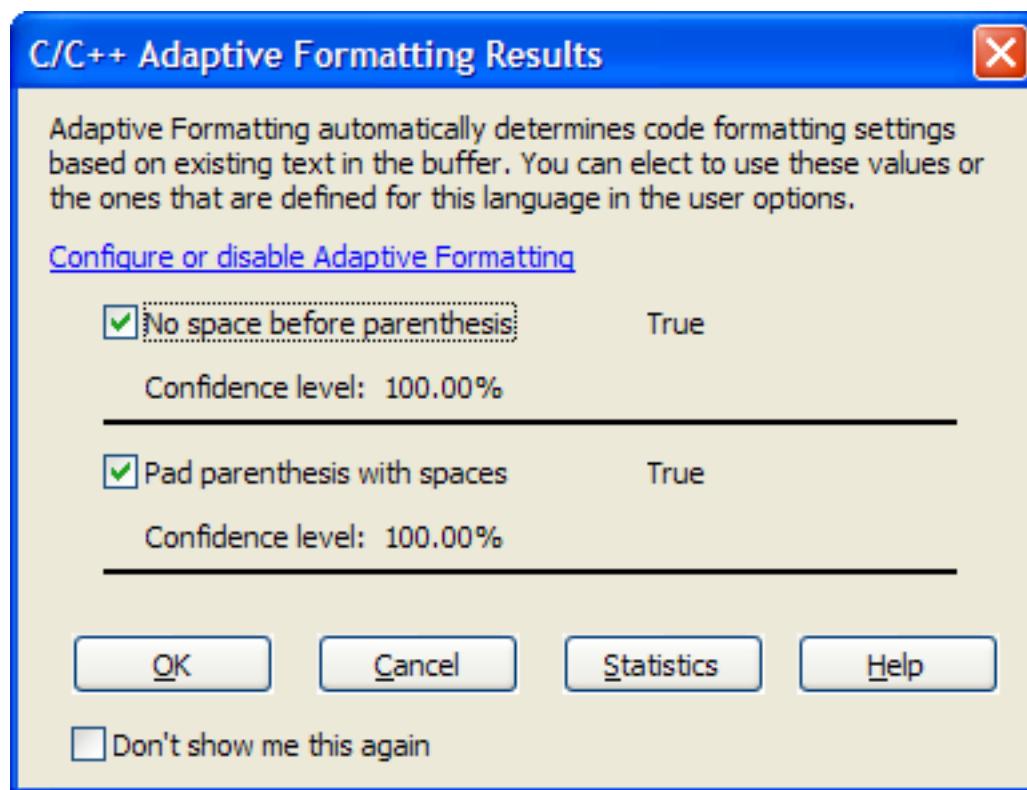
Formatting Option	Path to Option in SlickEdit	Description
		Formatting Options for more information.
Case for hex values	Formatting Options (Language-Specific Formatting Options)	Specifies the case for hex values that are automatically inserted inside a tag. See HTML Formatting Options for more information.

Scanning for Styles in Use

Adaptive Formatting has two modes of scanning: automatic, which is the default behavior, and manual, which lets you run a scan at any time to determine a file's formatting styles in order to quickly save them as the default settings.

When you open a file for editing, Adaptive Formatting initially scans for the indent-related settings that are in effect (**Syntax indent**, **Indent with tabs**, and **Tabs**), starting at the beginning of the file. Thereafter, Adaptive Formatting scans as you type to determine brace settings, parentheses padding, and other formatting styles. If Adaptive Formatting determines that the file is using a different formatting style than the current language-specific setting, a dialog is displayed that shows the settings in effect. You can then choose to use the settings or use the settings set for the language. Once the formatting style is determined, the options for those styles are used as long as the buffer is open for editing.

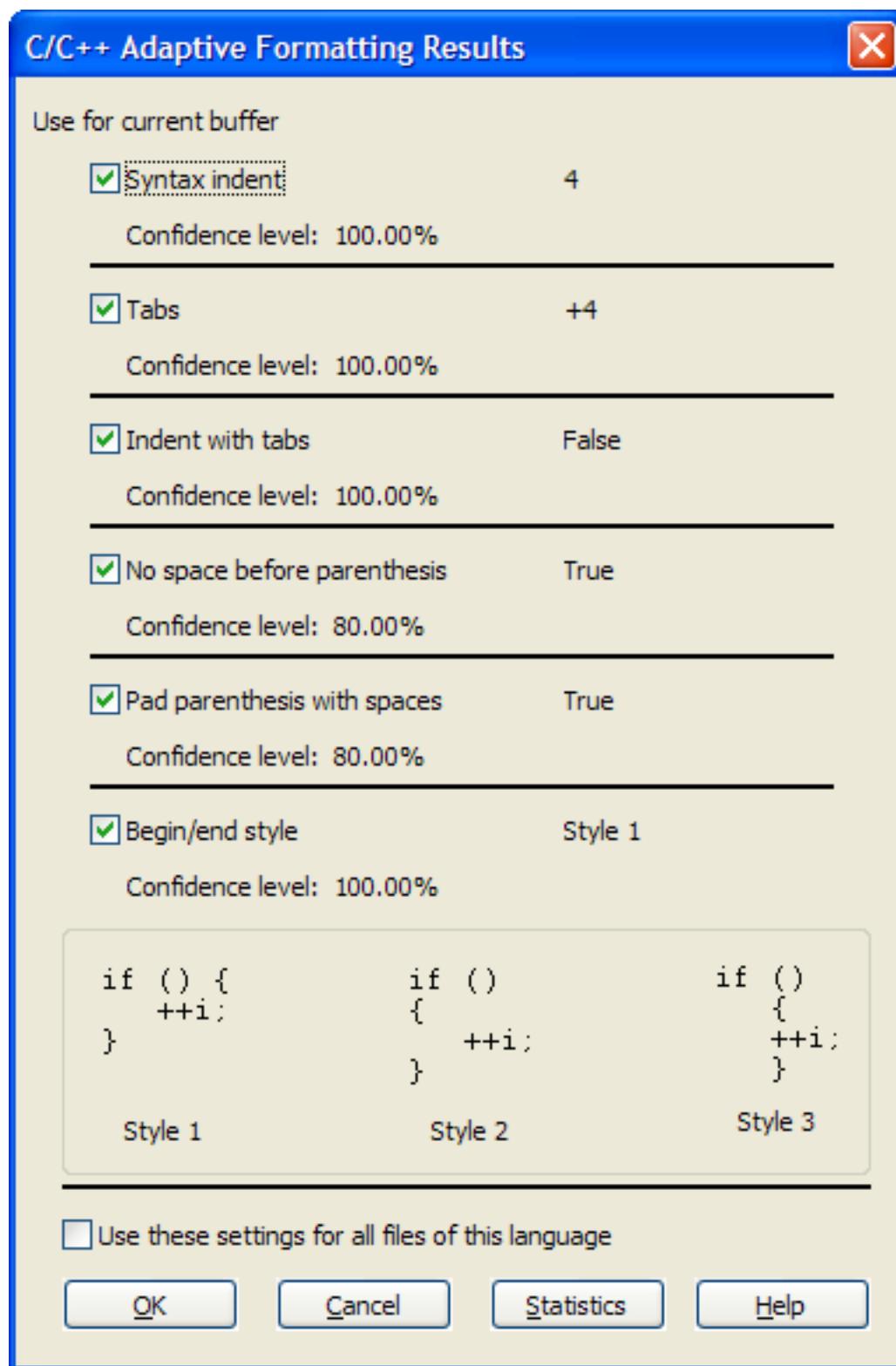
For example, when you type a keyword such as **if** in a C/C++ file that uses formatting styles different from the current settings, the C/C++ Adaptive Formatting Results dialog is displayed (depending on your settings for Adaptive Formatting under Feature Notifications).



On this dialog, the selected settings are used only for the current buffer. You can access the Adaptive Formatting options screen to enable/disable Adaptive Formatting for this language by clicking the hyperlink to **Configure or disable Adaptive Formatting**.

If you always want to accept the Adaptive Formatting results, select **Don't show me this again** to suppress the dialog from appearing in the future.

A similar dialog is shown when you run Adaptive Formatting manually. This mode lets you run Adaptive Formatting on the entire file at once, in order to quickly save the settings as the defaults for that language. To run a manual scan, use the **adaptive_format_stats** command. The Adaptive Formatting Results dialog for that language is displayed, showing all of the recognized style settings that are in use in the current file.



This dialog lets you enable/disable the settings just for the current buffer, or select **Use these settings for all files of this language** to quickly set these values as the default.

Confidence Level and Statistics

Both Adaptive Formatting Results dialogs show a confidence level and provide a link to a statistics screen. The Confidence Level is a statistical percentage, based on the frequency of use of that style in the file, that indicates the editor's confidence in this being a "correct" setting. To be considered the "correct" setting, the option must meet a confidence level of 66%. SlickEdit® ignores setting results with a confidence level of less than 66% and they do not appear on the results dialogs. For these options, the current language settings are used.

For example, if the dialog shows **Indent with tabs** enabled and a Confidence Level of 85%, this means that out of all of the examined instances of this style in the file, 85% of them were indented with tabs, and 15% were indented with spaces.

Click the **Statistics** button, and the Adaptive Formatting Statistics dialog shows each style that was found and the total number of times it occurred in the examined instances. Each individual setting is categorized in the tree according to type. Click on the plus/minus bitmaps to expand/collapse the tree categories.

Style	Total	Percentage
Pad Parenthesis		
Pad parens	4	100.00%
Non-padded parens	0	0.00%
Total	4	100.00%
Space Before Parenthesis		
Space before paren	0	0.00%
No space before paren	4	100.00%
Total	4	100.00%

Done

Rescanning

Certain events require the file to be rescanned in case any formatting changes have occurred. These events include beautification, auto-reload, and version control update. After one of these processes, Adaptive Formatting is placed again in automatic mode, just as when you first opened the file, scanning the entire file once for indent-type settings, then as you type for all of the other settings.

You can also rescan a file manually with the **adaptive_format_update** command. This clears the

Completions

Adaptive Formatting results from memory, then scans the entire file for all Adaptive Formatting settings available for that language. This is useful if you want to create a new file and use different settings for it than what you currently have configured.

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.
- [Auto-Close](#) - Automatically insert closing characters for bracketed and quotation punctuation pairs.
- [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.

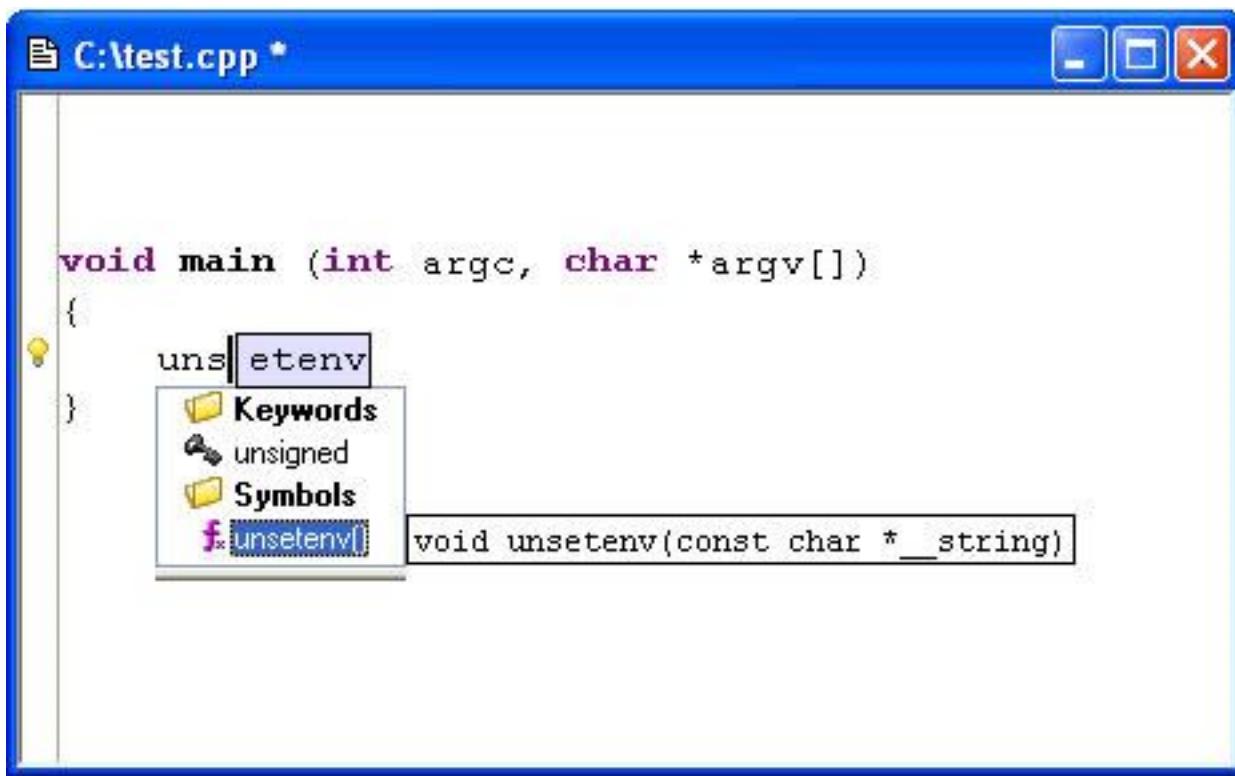
Auto-Complete

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 turned on or off, 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 turned on or off through the Auto-Complete options. See [Language-Specific Auto-Complete Options](#).

Tip

Auto-Complete can be activated manually by using the **autocomplete** command. Bind this command to a key sequence if you use it frequently. See [Creating Bindings](#) for more information.

To cancel out of Auto-Complete mode, use the **Escape** key.

To scroll through the items in the completion list, use the **Up**, **Down**, **PgUp**, and **PgDn** 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 on a language-specific basis. See [Language-Specific Auto-](#)

[Complete Options](#) for information.

Auto-Close

Auto-Close will automatically insert closing characters for bracketed and quotation punctuation pairs. The following list shows the available pairings.

- Parenthesis ()
- Bracket []
- Angle Bracket < >
- Double Quote " "
- Single Quote ' '
- Braces { }

Note

SlickEdit automatically closes block comments. For example, in C++ when you type "/*" SlickEdit automatically inserts "*/". This is not part of the Auto-Close feature. To configure this, go to **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Comments** and check or uncheck **Automatically close block comments**. See [Language-Specific Comment Options](#).

Auto-Close can be configured, as well as enabled/disabled, on a language specific basis. Specific pairs can also be enabled/disabled per language, as well as automatically inserting padding for Parenthesis, Brackets and Angle Brackets. To do this, select **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Auto-Close**. For more information on Auto-Close options, see [Language-Specific Auto-Close Options](#).

When an opening character in a pair is typed, the closing character will automatically be inserted. The closing character is automatically overtyped if you key in the matching close character as you are typing, helping to avoid any syntax errors. There are also navigation helpers when the close character is inserted. A hotspot marker is inserted on the right edge of the closing character. When this marker is visible, TAB or ENTER key will jump to the next column past the close bracket, and ESC will dismiss the marker as well as dismiss overtyping of the close character. Editing outside of the punctuation pair will also automatically dismiss the marker. You can disable the TAB or ENTER navigation key (or both) in **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Auto-Close**. If both TAB and ENTER are both disabled, the hotspot marker is not inserted, though overtyping is still available.

When enabled, Auto-Close will only insert the matching closing character for the specific punctuation where appropriate for the current language. This is determined by scanning the current location and line. For example, it does not insert any closing characters in comments or strings. In the C++ language, the angle brackets are only auto-closed when following the template or cast keywords (ex: static_cast).

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 file names 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. Some dialog such as the Find tool window, support dialog box retrieval. Dialog box retrieval enables previous responses for all check boxes, radio buttons, spin boxes, text boxes, and combo boxes to be retrieved. Press **F7** to retrieve the previous response, and **F8** to retrieve the next response

When this list is displayed, the following keys have different behaviors:

Key	Behavior
Esc	Closes the list.
Enter	Uses the current value selected.
Tab , Down , or Ctrl+K	Moves to the next item in the list.
Shift+Tab , Up , 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.
PgUp or PgDn	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:

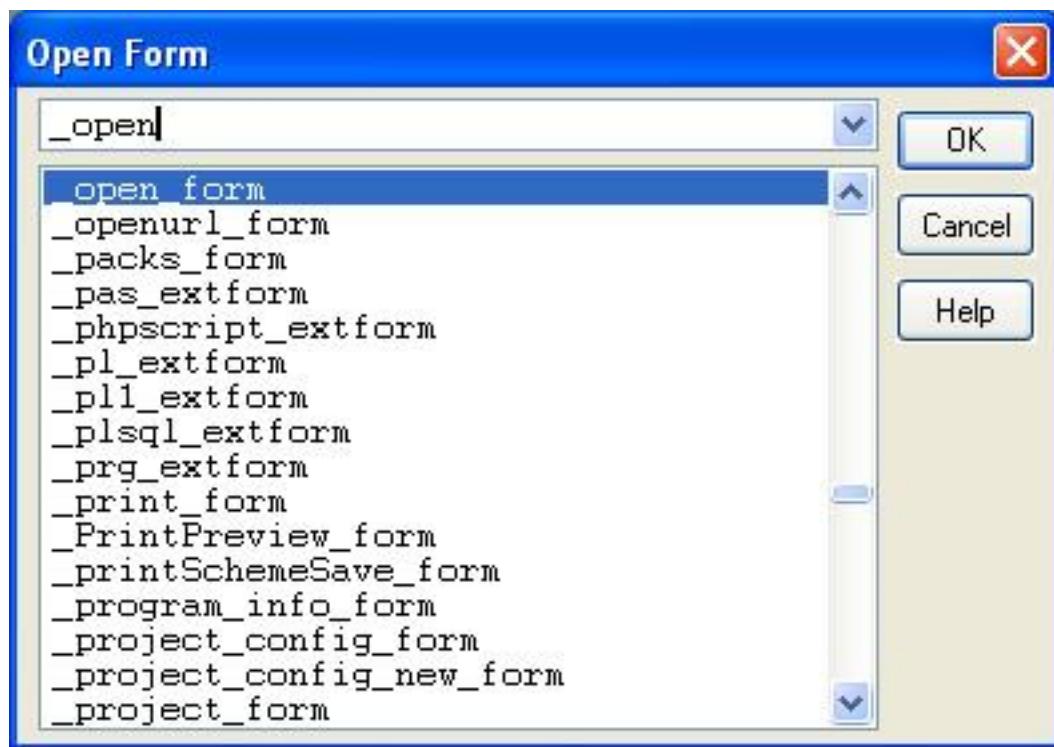
1. From the main menu, click **Project** → **New**.
2. Press **Tab** to jump to the **Location** field.
3. Start typing a directory path. Notice the completion options.
4. Press **Esc** to cancel.

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, click **Macro** → **Open Form**.

**Configuring Completion
Settings**



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, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Auto-Complete**. See [Language-Specific Auto-Complete Options](#) for more information.

Aliases

Aliases are identifiers that you can quickly type which are then expanded into snippets of text. You can use aliases for any text that you frequently type, including directory paths, function names, statements, and comment headers.

There are two types of aliases in SlickEdit®:

- [Global Aliases](#) - These aliases can be used across multiple languages. They are also very useful as directory aliases, because they save you from having to type long paths in file name or directory fields within the editor.
- [Language-Specific Aliases](#) - These aliases are set up on a per-language basis. For example, if you work in multiple languages, you could have one alias identifier for the same function but with different expansions applicable to each language.

Expanding Aliases

After typing the alias identifier, aliases can be expanded using any of these methods:

- Pressing **Ctrl+Shift+O** for the **expand_alias** command. This command always expands the alias, regardless of available completions.
- Pressing **Ctrl+Space** for the **codehelp_complete(Pro only)** command. This command will expand the alias only if there are no matching symbol completions. Otherwise, it will show a list of symbol completions. See [List Members](#) for more information.
- If Syntax Expansion is enabled (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Indent**), aliases will automatically be expanded by typing a space.
- Pause while typing and the alias will be displayed in the Auto-Complete list.

Tip

By default, alias expansion is not case-sensitive. However, if you wish alias identifier matching to be case-sensitive, you can get this behavior by setting the macro variable **def_alias_case** to **e**. To turn off case-sensitivity, set this variable to **i**. To set a configuration variable, go to **Macro** → **Set Macro Variable** or use the command **set-var**. For more information about configuration variables, see [Configuration Variables](#).

Tip

An option is available to show a tool tip of the matching alias for the word under the cursor. Click **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Auto-Complete**

and check the option **Alias expansion**. See [Completions](#) for more information.

Global Aliases

Global aliases work across all languages. One way to use global aliases is to save time in entering long directory paths. See [Directory Aliases](#) below.

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 directory and file name fields within the editor.

After typing the alias identifier, directory aliases can be expanded by pressing **Ctrl+Space**. Global aliases are stored in the file `user.cfg.xml`, located in the SlickEdit® root installation directory.

Note

SlickEdit Core doesn't modify Eclipse's file management-related dialogs such as **File → Open** or **File → Save As**. Therefore, directory aliases are not available in these dialogs.

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, click **Tools → Options → Editing → Global Aliases**.
2. Click **New**, then type the characters you wish to use for an identifier in the **Alias Name** text box.
3. Click **OK**. The identifier you entered is now displayed in the alias list box on the options page.
4. 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.
5. 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. On the SlickEdit command line, type **e** (for "edit").
2. Type the alias name (identifier) for the directory where the file resides.
3. Press **Ctrl+Space** to expand the alias.
4. Type the name of the file to open.
5. Press **Enter**.

When using the system native Open panel on macOS, use **Option+Escape** to trigger directory alias expansion.

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:

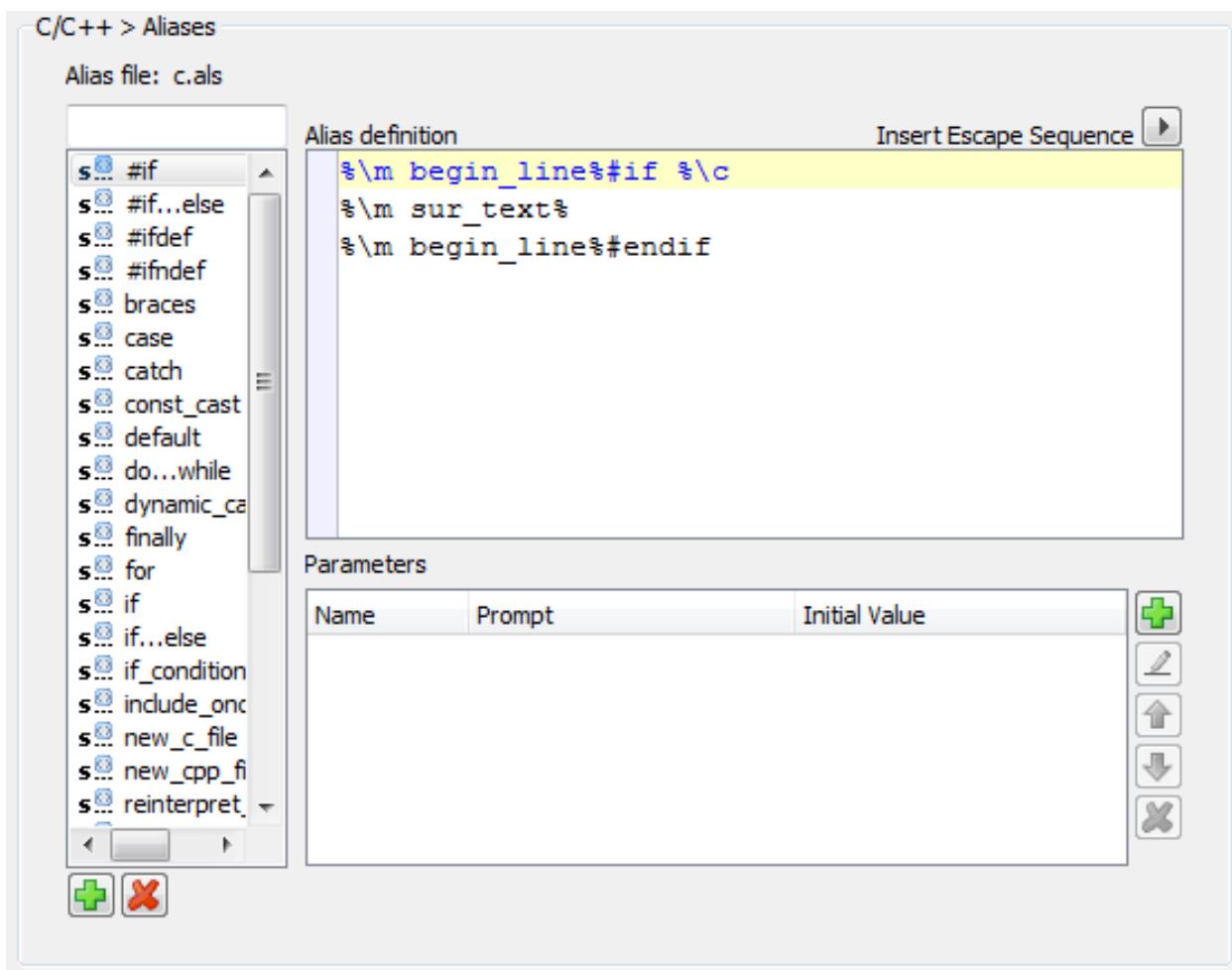
```
set VERSION=c:\version20
```

For more information about setting environment variables, see [Environment Variables](#).

Language-Specific Aliases

You can set up language-specific aliases for any frequently used text, such as comment headers. Each language can have one alias file, allowing aliases to be defined that do not affect other languages. Language-specific aliases are stored in files with the extension `.als` located in the user configuration directory.

The aliases that you create in a language are made available each time you open or create a file in that language. To manage language-specific aliases, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Aliases**. As an example, the C/C++ Aliases screen is shown below.



Alias names are displayed in the list box on the left. The value for the selected alias name is displayed in the large text box to the right. Click **Delete** to remove a selected alias and its value.

Creating a Language-Specific Alias

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. If you wish to create a Surround With alias, check the **Surround With** checkbox. For more information about Surround With aliases, see [Surround With](#).
2. Click **OK**. The identifier you entered is now displayed in the list box in the Alias options page.
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.
4. Click **OK**.

Tip

- 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.

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.

Note

If you leave a blank line in an alias, SlickEdit will automatically insert the sequence to preserve leading spaces, %\l, when you save the alias. This ensures that the blank line is preserved.

Escape Sequence	Description
%\a	Inserts the authors name.
%\c	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 Ctrl+] (next_hotspot command) to jump to the next cursor stop.
%\	Creates multiple cursors. This sequence can be used multiple times in the same alias value in order to create multiple cursors for initially entering the same text into multiple locations. After the alias is expanded, the multiple cursors will be active. Press Esc to cancel multiple-cursor mode.
%\ c	Creates multiple cursors. This sequence can be used multiple times in the same alias value in order to create multiple cursors for initially entering the same text into multiple locations. After the alias is expanded, the multiple cursors will be active. Press Esc to cancel multiple-cursor mode. In addition, each cursor location will be added to the series of "hot spots" within the alias. Press Ctrl+] (next_hotspot command) to jump to the next cursor stop.

Escape Sequence	Description
<code>%\d</code>	Inserts the date (locale-dependent).
<code>%\e</code>	Inserts the date in MMDDYY format.
<code>%\t</code>	Inserts the time (locale-dependent).
<code>%%</code>	Inserts a percent character.
<code>%\f</code>	Inserts the current file name without path.
<code>%\fn</code>	Inserts the current file name without path or extension.
<code>%\g</code>	Inserts a file separator character. This is the backslash on Windows (\) platforms and slash (/) on UNIX/Mac.
<code>%\w</code>	Outputs the line number.
<code>%\n</code>	Inserts the current function name.
<code>%\n-</code>	Inserts the current function name, qualified with the current class name if there is one
<code>%\n+</code>	Inserts the current function name, prefixed with the current qualified class name if there is one
<code>%\o</code>	Inserts the current function name with signature.
<code>%\j</code>	Inserts the current class name.
<code>%\j+</code>	Inserts the current class name, fully-qualified.
<code>%\i</code>	Indents.
<code>%\b</code>	Unindents.
<code>%\s</code>	Preserves trailing spaces. This should be placed at the end of a line.
<code>%\l</code>	Preserves leading spaces.
<code>%\x ColumnNumber</code>	Moves the cursor to the specified column number.

Escape Sequence	Description
%lx+ <i>ddd</i>	Increments column by <i>ddd</i> .
%lx- <i>ddd</i>	Decrements column by <i>ddd</i> .
%lm <i>MacroName ArgumentList %</i>	Calls the specified Slick-C® macro with a specified optional argument. This can be used for many purposes including surrounding text (see below) and inserting formatted dates (see Escape Sequence Examples).
%lm _maybe_prefix_quote_func <i>MacroName ArgumentList %</i>	Calls the specified Slick-C® macro with a specified optional argument and inserts the result with a leading double-quote if the value contains spaces.
%lm sur_text%	Uses the escape sequence to call a macro to surround the selected text. Indicates where the text to be surrounded is placed. See Surrounding and Unsurrounding for more information.
%lh <i>alias_name %</i>	Embeds another alias within the current alias. Can be used to embed a language-specific alias within a global alias depending on what kind of file is being edited.
% <i>EnvironmentVariable %</i>	Inserts the value of the environment variable specified.
%lm _maybe_prefix_quote_env <i>EnvironmentVariable %</i>	Inserts the value of the environment variable specified with a leading double-quote if the value contains spaces.
%{ <i>ParameterName</i> }	Parameter Prompting replacement. See Parameter Prompting for more information.
%lp	Inserts parameters from the function that is located beneath it. See Doc Comments for more information.
%lq	Insert the type for the parameters (double, integer, string, etc.) from the function that is located beneath it. Typically used in conjunction with the %lp escape sequence. See Doc Comments for more information.

Escape Sequence	Description
%\r	Inserts the return type from the function that is located beneath it. See Doc Comments for more information.
%\u	Includes this line in the expansion, if there are any function parameters that are expanded. If there are no function parameters expanded, this line is not included in the full expansion. See Doc Comments for more information.
%\v	Includes this line in the expansion, if there are any return types that are expanded. If there are no return types expanded, this line is not included in the full expansion. See Doc Comments for more information.
%\un	If there are any function parameters that are expanded, this line is not included in the full expansion. If there are no function parameters expanded, this line is included in the full expansion. See Doc Comments for more information.
%\vn	If there are any return types that are expanded, this line is not included in the full expansion. If there are no return types expanded, this line is included in the full expansion. See Doc Comments for more information.
%()	Used to separate identifier characters. For example, %\u%()n has the effect of the %\u option followed by a literal "n". It is recommended that %() be used to separate alias escape sequences ending with a letter from other identifier characters so that new aliases escape sequences won't break existing aliases you have. Don't write %\dx. Write %\d%()x instead.

Escape Sequence Examples

The following table contains some examples of using escape sequences in alias values:

Alias Name and Description	Value

Alias Name and Description	Value
Sample formatted dates	<pre>%\m printtime #b. #d, #Y% ==> Apr. 07, 2008 %\m printtime #A, #B #d, #Y% ==> Monday, April 07, 2008 %\m printtime #m/#d/#Y% ==> 04/07/2008</pre>
comment - A header comment to have the date and time inserted.	<pre>/* **** */ /* Date: %\d Time: %\t */ /* **** */</pre>
if - A simple if statement, with indenting, support for surround, and a cursor position.	<pre>if (%\c){ %\i// Comment goes here %\i%\m sur_text% }</pre>
ifelse - An if/else statement with indenting and several cursor hot spots.	<pre>if (%\c){ %\i%\c } else { %\i%\c }</pre>
fori - A for statement with a local indexing variable and multiple cursors.	<pre>for (int %\ =0; %\ < %\c; %\ ++) { %\i%\c }</pre>
wm - 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>

Alias Name and Description	Value
/** - A Javadoc comment.	<pre>/** * * %\c * * @author %\a (%\d) * %\u * @param %\p %\c * %\v * @return %\c%\v */</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, 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 page), 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 options page, 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 options page, 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, use the Parameters section of the Alias options page to 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 a Language-Specific Alias from a Selection

You can create a language-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 language-specific Alias options page appears, from which you can edit the code to fine-tune or add parameters.

API Help

API Help definitions are used to determine what action occurs when you press F1 in a source file. When you press F1 in a source file (**help** command), SlickEdit will attempt to determine the package/namespace/class that the current word belongs. Help actions can be defined to match specific classes and/or members (the current word). For example, on Windows, SlickEdit has defined over 22,000 help actions for the Win32 API for C/C++. This covers most of the Win32 API symbols but not all. When SlickEdit does not find a more specific action defined for a symbol, the default action is to do an internet search for the language, package/namespace/class, and the member (current word). This works quite well for most languages. Often what you are looking for appears in the first few items of the first web page.

There are language specific API Help profiles and global API Help profiles:

- **Tools → Options → Languages → [Language Category] → [Language] → API Help**

Language-Specific API Help - The default language specific API Help profile is processed first. If an action match is found, the action is taken (typically an internet search or specific URL) and further processing stops. An API Help profile can reference, like an include, other API Help profiles. For example, on Windows the C/C++ API Help "Default" profile references the "Win32 API" profile. Defining additional profiles is useful for defining specific API Help actions for libraries.

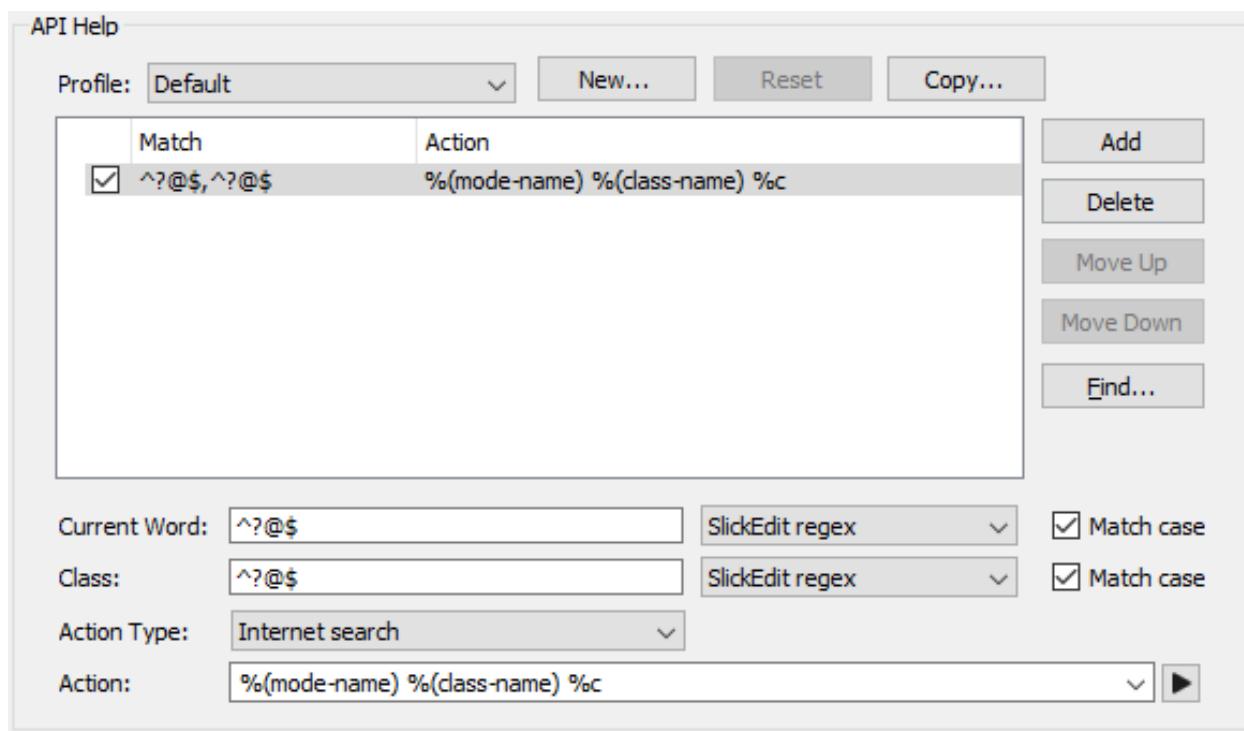
- **Tools → Options → Editing → API Help**

Global API Help - When the language specific API Help profile does not find an action match, the global API Help profile is processed. By default on Unix and macOS "man" is used to attempt to find a man page for the symbol (current word). The last action is to do an internet search for the language package/namespace/class and the member (current word).

API Help Profile Settings

The API Help dialog allows you to set a default profile and define a list of help actions for your API Help profiles.

API Help Profile Settings



The following options are available for setting the default profile and creating, resetting, and deleting profiles:

- **Profile:** - Lists the profiles available. The profile displayed is default profile.
- **New...** - Creates a new profile
- **Reset...** - Resets the **profile** to its original settings (Only displayed for built-in profiles).
- **Delete** - Deletes the **profile**. This button is not displayed for built-in profiles. Built-in profiles may not be deleted.
- **Copy...** - Copies the **profile** under a new profile name.

The following buttons are for adding, deleting, ordering, and searching for API Help items

- **Add** - Appends an API Help item
- **Delete** - Deletes the selected API Help item. Built-in API Help items can't be deleted. You can disable them by unchecking the leading check box.
- **Move Up** - Moves an API Help item up in the list.
- **Move Down** - Moves an API Help item down in the list.

The following are API Help item properties:

- **Current Word:** - Search term for matching the current word. Not used for Action Types Profile or Stop.
- **Class:** - Search term for matching the package/namespace/class. Not used for Action Types Profile or

Stop. To match a class and any member of the class, leave the **Current Word:** blank and specify a **Class:** to match.

- **Action Type:** - Specifies one of the following types:
 - **Internet search** - When matched, performs an internet search for the **Action:** text specified using the default search engine and default search browser.
 - **URL** - When matched, Loads the URL specified by **Action:** in the default search browser.
 - **Profile** - Matches the actions in the API Help profile specified by **Action:**(like an include). Additional profiles are can be used to defining specific API Help actions for a library. For example, on Windows the C/C++ API Help "Default" profile references the "Win32 API" profile.
 - **Slick-C** - When matched, executes the Slick-C command specified by **Action:**. If the command returns 0, the help action matching is stopped.
 - **Stop** - Stops help action matching.
- **Action:** - Usage depends on **Action Type:**. See Action Types above.

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( ) {  
}
```

Syntax Expansion triggers [Dynamic Surround](#) for block-oriented statements. This allows you to expand and collapse the newly inserted block to include more/less code. Additionally, for C, C#, C++, J#, Java, and Slick-C®, after the statement is expanded, you can use the **next_hotspot** command (**Ctrl+I**) to jump the cursor to the next part of the statement. In the case of the **for** loop above, **Ctrl+I** would move the cursor from the group in parentheses to the code block.

The statements **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. You do not have to type the entire keyword for Syntax Expansion to occur. For example:

```
if;      =>    if ( <cursor here> ) <next hotspot>;
```

The statements **if**, **while**, **catch**, **return** and **switch** are also expanded if you type an open paren immediately afterwards. If your programming style is to omit these keywords, you would not be accustomed to typing space after these keywords, so this feature allows you to still get syntax expansion using your normal typing patterns.

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, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **[Language] Formatting Options**. See [Language-Specific Formatting Options](#) for more information.

For further customization, for most languages, you can override the default keyword expansion by

defining an alias for that keyword. See [Language-Specific Aliases](#) for more information.

Syntax Expansion Settings

To access Syntax Expansion settings, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Editing**.

To turn Syntax Expansion on or off, select or clear 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, from the main menu, use the **[Language] Formatting Options** screen.

Tip

SlickEdit® can display Syntax Expansion choices for the word prefix under the cursor. To turn this option on/off, select the **Auto-Complete** language-specific options screen and select/clear the **Syntax expansion** option. See [Completions](#) for more information.

Modifying Syntax Expansion Templates

Syntax Expansion templates are essentially language-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, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Aliases**.
2. Click **New** and then type **for** as the alias name.
3. Type the following lines in the text box to the right of the alias name:

```
for (%\c;;) {  
} /* for */
```

The **%\c** escape sequence above specifies the cursor placement after expansion is performed.

4. Repeat Steps 2 and 3 for the **while**, **if**, and **switch** keywords.
5. 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 options page, or using alias escape sequences, see [Language-Specific Aliases](#).

Adding Syntax Expansion for Other Languages (Pro only)

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 language-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.

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.

Note

Code Templates are for creating new files. To insert code into an existing file, use SlickEdit [Aliases](#).

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 $package$;

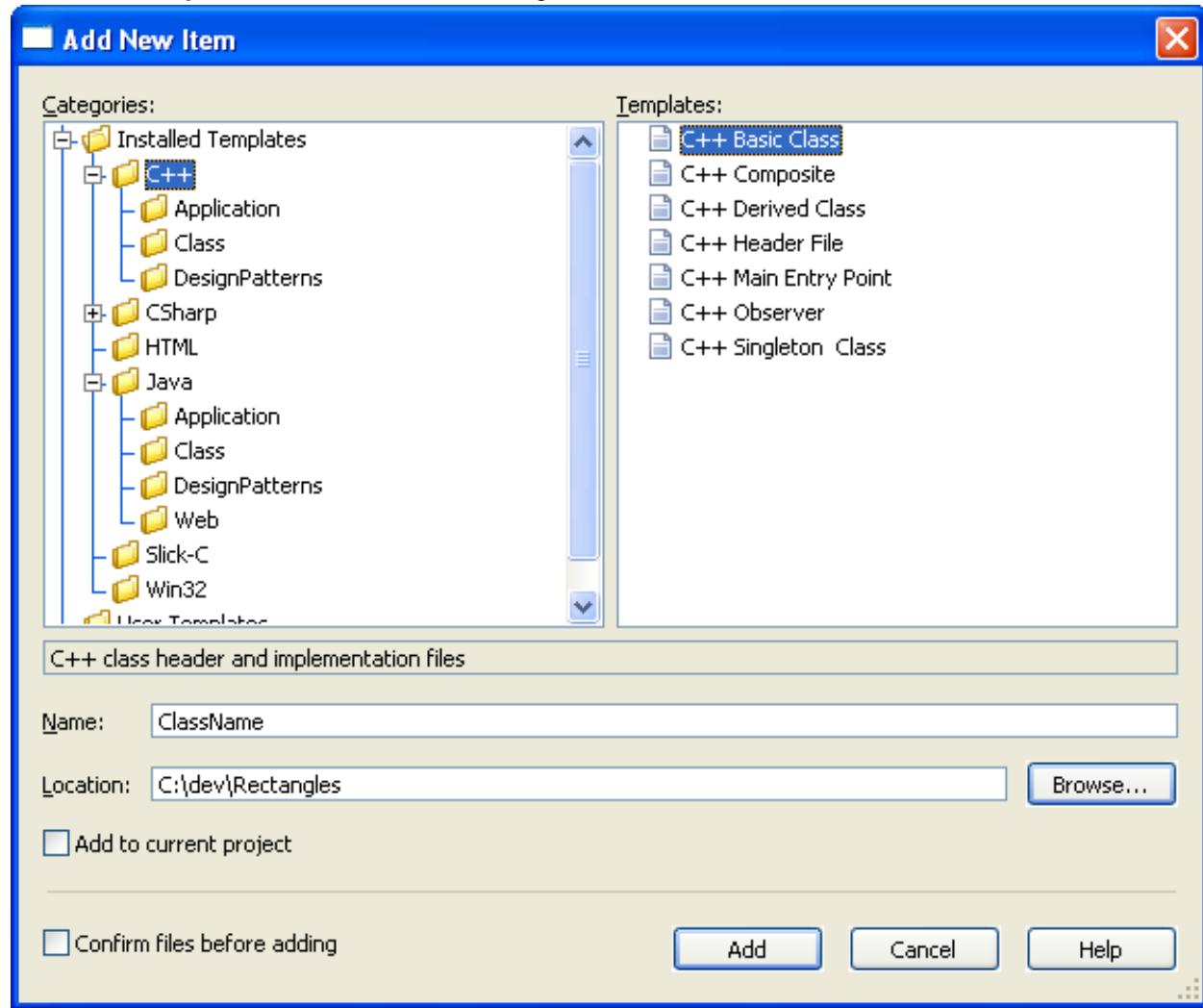
/**

 * @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 listed under **Installed Templates** and are organized into categories by language and then by purpose. The **Template Manager** dialog will not allow you to add, edit, or delete Installed Templates. Use the Template Manager dialog to add, edit, and delete templates under **User Templates**. The Template Manager dialog is accessed by clicking **File → Template Manager**.

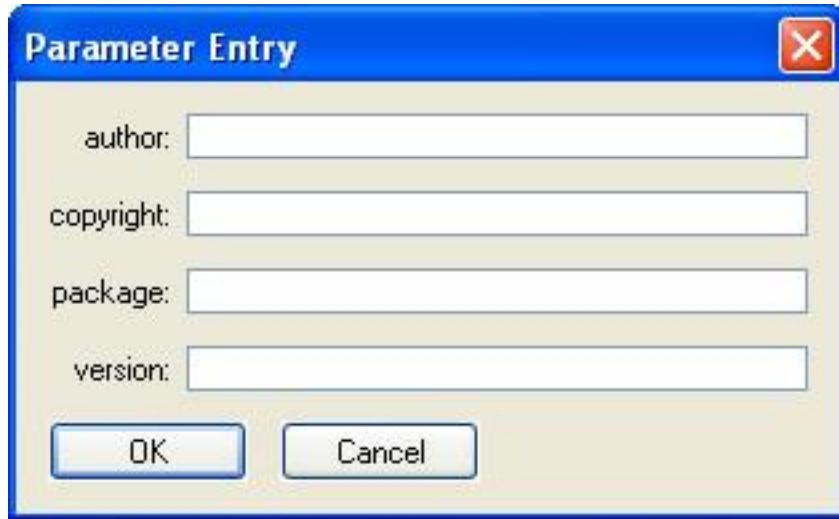
Instantiating a Template

You can add an item to your current project by clicking **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 click **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, **\$safeitemname\$** 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

Click **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 (+)** button 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. Click **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:

Substitution Parameters

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 \$safeitemname\$ would evaluate to My_Custom_Class for a C++ source code file.
\$upcasesafeitemname\$	Same as \$safeitemname\$ with all characters uppercased.
\$lowcasesafeitemname\$	Same as \$safeitemname\$ with all characters lowercased.
\$tempdir\$	Location of operating system temp directory. No trailing file separator.
\$rootnamespace\$	Root namespace or package for the current project. This is typically used for C# and Java projects to find the namespace containing Main() (or main() in the case of Java).
\$ampmtime\$	Time of day in the form <i>hh:mm[am pm]</i> . Example: 11:34pm
\$localtime\$	Time of day in locale-specific format.
\$time\$	Time of day in the form <i>hh:mm:ss</i> .
\$localdate\$	Current date in locale-specific format.
\$date\$	Current date in the form <i>mm/dd/yyyy</i> .
\$projectname\$	Current project name (no path, no extension).
\$safe projectName\$	Current project name (no path, no extension), with all unsafe characters replaced with safe characters. For example, if the project name was: My Project.vpj , then \$safe projectName\$ would

Parameter Name	Description
	evaluate to My_Project for a C++ source code file.
\$workspacename\$	Current workspace name (no path, no extension).
\$safeworkspacename\$	Current workspace name (no path, no extension), with all unsafe characters replaced with safe characters. For example, if the workspace name was: My Workspace .vpw, then \$safeworkspacename\$ would evaluate to My_Workspace for a C++ source code file.
\$projectworkingdir\$	Current project working directory. No trailing file separator.
\$projectbuilddir\$	Current project build (output) directory. No trailing file separator.
\$projectconfigname\$	Current project configuration name.
\$workspaceconfigname\$	Current workspace configuration name. This will be the same as \$projectconfigname\$ except for MS Visual Studio workspace which will have a separate workspace/solution configuration name.
\$projectdir\$	Location of current project file. No trailing file separator.
\$workspacedir\$	Location of current workspace file. No trailing file separator.
\$username\$	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 click **Project → Add New Item**.
5. Verify that your new category appears in the **Categories** list on the Add New Item dialog box.

Caution

The **Template Manager** dialog will not allow you to create new categories or re-organize 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. For information about the location of shipped templates, see [Locating Templates](#).

Template Manager Operations

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

Note

You can only add, edit, and delete templates under the **User Templates** node in the **Template Manager** dialog. If you want to modify a template shipped with SlickEdit, copy it to the `ItemTemplate` subdirectory in your config. See [Locating Templates](#).

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.

Template Manager Dialog

The Template Manager dialog is made up of the following elements:

- **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

The **Details** tab of the Template Manager dialog contains the following:

- **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 of the Template Manager dialog 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 of the Template Manager dialog 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 by clicking the **Options** button.

Global Substitution Parameters

The **Global substitution parameters** area on the Template Options dialog 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. To launch this dialog, right-click on a file in the **Files** tab of the Template Manager dialog, and select **Add File**, or use the **Add File** button. The dialog contains the following:

- **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](#).

The Add Parameter dialog contains the following:

- **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

Used to add an item to your current project, the Add New Item dialog is displayed when you click **Project** → **Add New Item** or **File** → **New Item from Template**.

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 manage your templates from the Template Manager dialog box by choosing **File > Template Manager**.

The Add New Item dialog contains the following:

- **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 to project** - When selected, the new item is added to the project selected.
- **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:

```
[My Documents]\My SlickEdit Core Config\[VERSION]\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 clicking **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 clicking **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. From the main menu, click **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, click **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 (.setemplate). 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 .setemplate file. The item creates C++ class header (.h) and implementation (.cpp) files.

Code Template Metadata File Reference

```
<?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 `.setemplate` extension.

The **SETemplate** element is the root element of a template file.

Summary of metadata elements:

Element	Child Elements	Attributes
Elements	-	-
Description	-	-
File	-	ReplaceParameters, TargetFilena me
Files	File	-
Name	-	-
Parameter	-	Name, Value

Code Template Metadata File Reference

Element	Child Elements	Attributes
Parameters	Parameter	-
SETemplate	TemplateContent,TemplateDetails	Type,Version
SortOrder	-	-
Template Content	Files,Parameters	Delimiter
TemplateDetails	DefaultName,Description,Name,SortOrder	-

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	<p>Optional.</p> <p>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:</p> <p>TargetFilename="\$fileinputname\$.cpp").</p> <p>Possible values are "1" (true) or "0" (false).</p> <p>Defaults to "1" (true).</p>
TargetFilename	<p>Optional.</p> <p>Specifies the actual name of the item that is created from the template.</p> <p>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.</p>

- **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](#).

Code Template Metadata File Reference

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.

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.

Code Template Metadata File Reference

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 "\$".

Code Template Metadata File Reference

Attribute	Description
-----------	-------------

- **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>
```

Quick Brace/Unbrace

Quick Brace makes it easy to convert single line statements into a brace-enclosed blocks, so you can add new lines without having to manually position the cursor and type extra keystrokes. Unbrace removes the braces from a block that contains a single line statement.

Tip

- As you write new code, SlickEdit® automatically expands statement templates for common block structures (such as **if**, **for**, or **while**) when you type the initial keyword. See [Syntax Expansion](#) for more information.
- Quick Brace and Unbrace do not support code blocks containing multiple statements, but you can use [Dynamic Surround](#) and [Unsurround](#) instead.

A single line statement is defined as a single line child statement that is not enclosed in braces, for example:

```
if ( cond ) doSomething();
```

Hanging single line statements are often broken across two lines in the editor:

```
if ( cond )
    doSomething();
```

Using Quick Brace/Unbrace

When you use Quick Brace, SlickEdit attempts to honor your brace style and indent settings. To use Quick Brace, position the cursor where you would normally type the open brace, and type it. Using the preceding code samples, you could position the cursor as follows:

```
if ( cond ) <cursor here> doSomething();
```

or

```
if ( cond )
<cursor here> doSomething();
```

After typing the opening brace, the child statement is moved to the next line if necessary, indented

according to your indent settings, and the closing brace is inserted automatically. The result on the preceding code sample is:

```
if ( cond ) {  
    doSomething();  
}
```

Tip

TIP Indentation and brace style settings are specified on the **Formatting Options** screen specific to your language (**Tools** → **Options** → **Languages** → [Language Category] → [Language] → **Formatting**). For all languages, use the [Language-Specific Formatting Options](#) screen.

Unbrace does the opposite of Quick Brace, removing the braces from a brace-enclosed block that contains a single line statement and moving the statement to the preceding line that contains the parent statement (unless it is just too long). To use Unbrace, simply delete the opening brace. Using the preceding code example, the result is as follows:

```
if ( cond ) doSomething();
```

You can use Unbrace on any brace-enclosed block that contains a single line statement, not just a block that was created with Quick Brace. Unbrace works on any brace style.

Depending on the original brace style and the column location of the open brace, Unbrace either pulls the statement up to the same line or leaves it hanging. The default column threshold is 40, which can be modified by setting the configuration variable **def_hanging_statements_after_col** (**Macro** → **Set Macro Variable**). Set the value to **1** for statements to always remain on the second line. Set the value to a very large number to always pull up statements to the original line. See [Setting/Changing Configuration Variables](#) for more information on setting variables.

Quick Brace and Unbrace work for C/C++ and similar languages that support brace blocks. Statements such as **if**, **while**, and **for** are supported, as well as the handling of **else** clauses for **if** statements and the splicing of the close brace with a trailing **else**.

Disabling Quick Brace/Unbrace

Quick Brace/Unbrace is on by default, and can be enabled/disabled on an language-specific basis. To access this option, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then click **[Language] Formatting Options** and select or clear the option **Quick brace/unbrace one line statements**.

Surrounding and Unsurrounding

SlickEdit® provides two features that allow you to surround existing text with new text. [Dynamic Surround](#) lets you selectively include more or fewer lines in a block structure, like an **if** statement. [Surround With](#) lets you surround any selected text with text predefined in one of the [Surround With Aliases](#). In addition, [Unsurround](#) can be used to remove code block structures.

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 expansion and alias expansion features (see [Syntax Expansion](#) and [Language-Specific Aliases](#)). It 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. By default, Dynamic Surround is turned on for all supported languages. To turn it off, select **Tools** → **Options** → **Languages**, expand your language category and language, then click **Indent** and uncheck **Use Dynamic Surround**.

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.

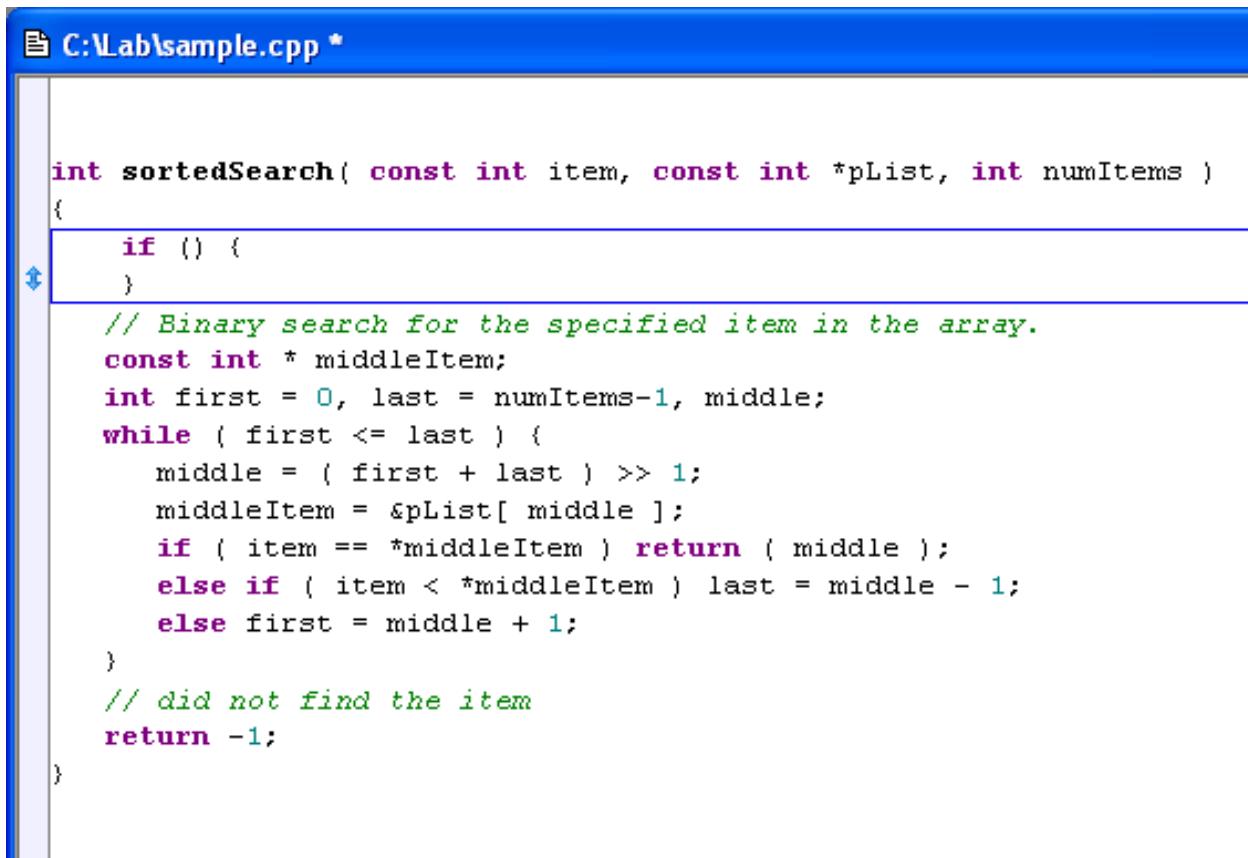
Dynamic Surround stays active until you press **ESC**. During that time auto-completions and symbol preview are unavailable.

You can also invoke Dynamic Surround on an existing block structure using the **dynamic-surround** command. Put the cursor on the line containing the block structure keyword, like an "if" or a "for", press **ESC** to open the SlickEdit command line, then type **dynamic-surround**. If SlickEdit recognizes the block structure, the box will be drawn and you will be able to expand or contract the structure using the **Up**, **Down**, **PgUp**, or **PgDn** keys. By default, this command is not associated with a key binding. See [Creating Bindings](#) for information on creating your own.

The following screen shot shows the Syntax Expansion menu that appears after typing "if" in a C++ file:

```
int sortedSearch( const int item, const int *pList, int numItems )
{
    if|  
    / Syntax Expansion for the specified item in the array.  
    c IFO if( ... ){ ... } eItem;  
    i Keywords     st = numItems-1, middle;  
    w ifstream      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;
}
```

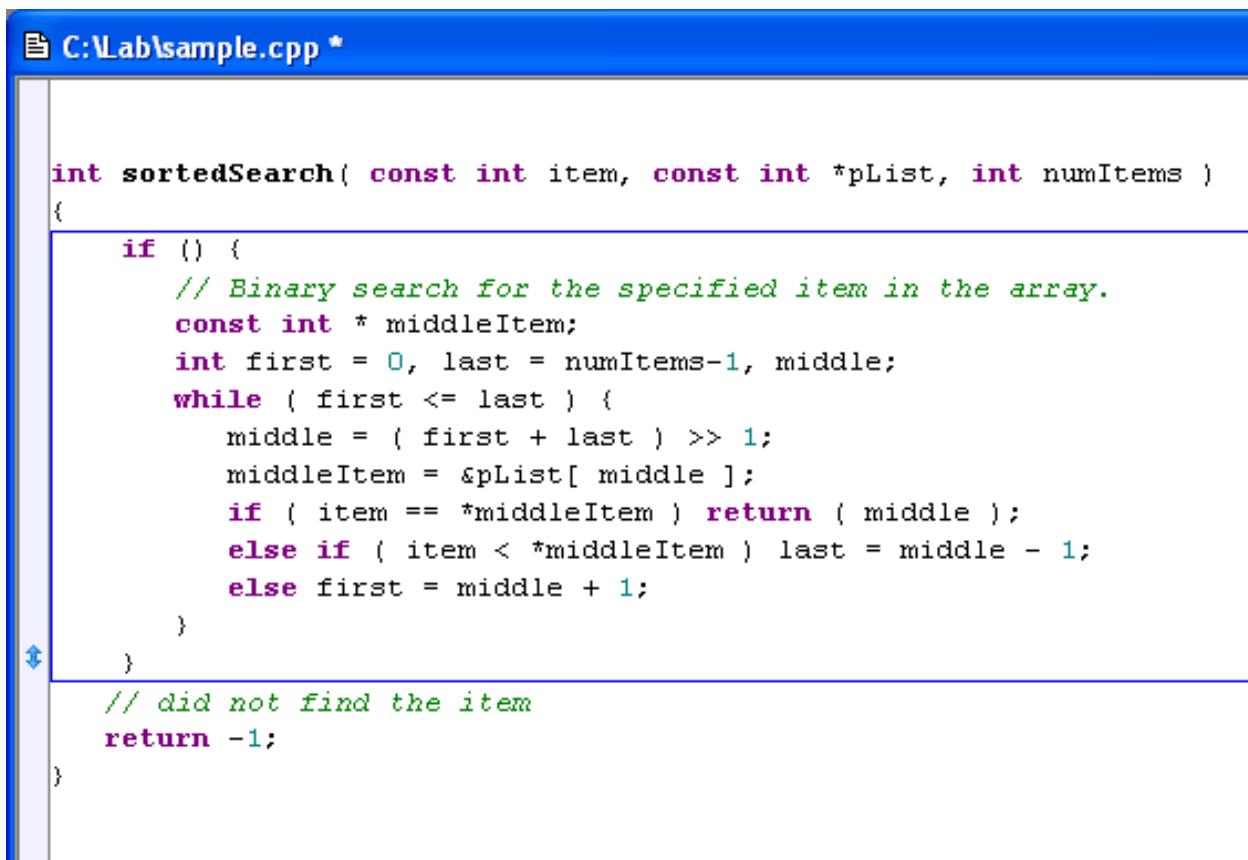
After pressing **Space** to expand the statement, Dynamic Surround is activated, with a blue rectangle drawn around the expanded statement, as shown below:



```
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;
}
```

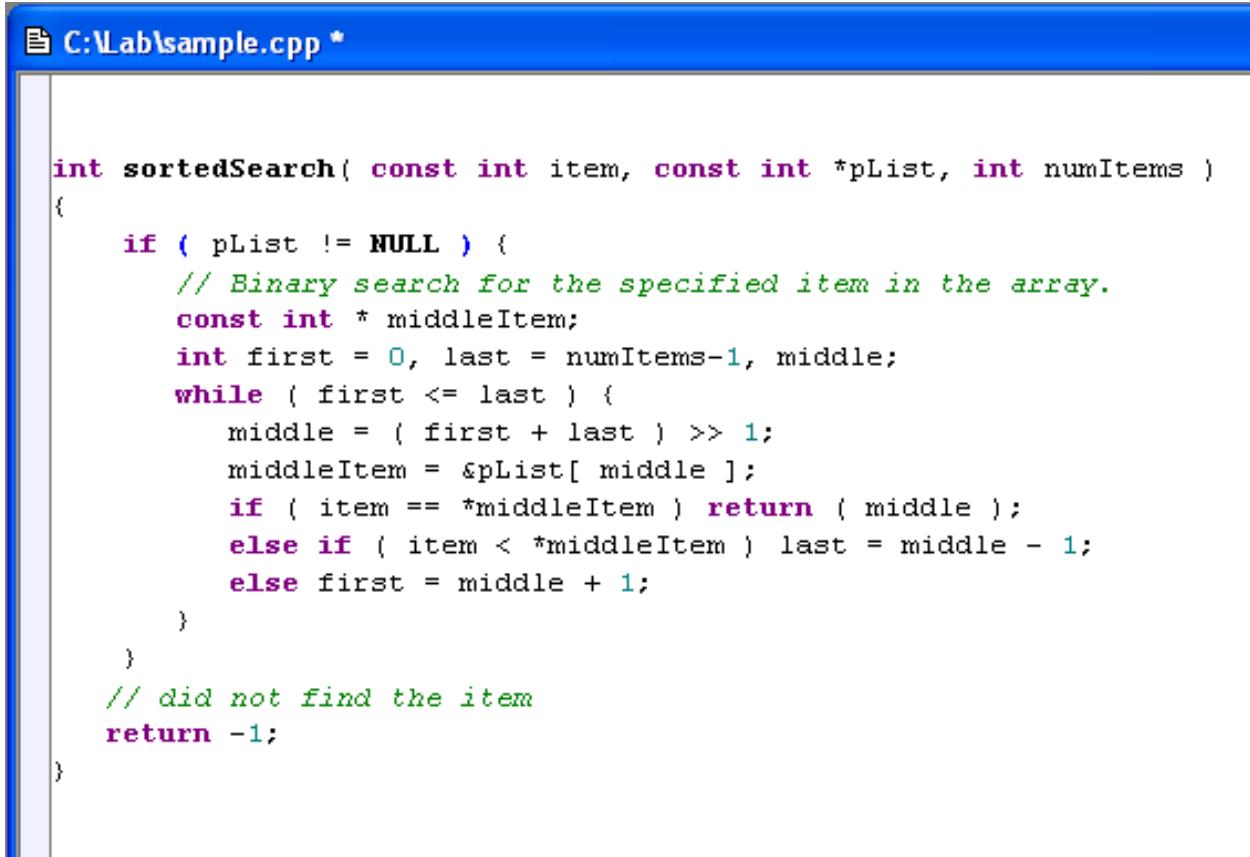
Pressing the **Down** arrow key pulls the code block into the statement, indented to the correct levels, as shown below:



The screenshot shows a code editor window titled "C:\Lab\sample.cpp *". The code is a C++ implementation of a binary search algorithm. It includes a header guard, function declarations, and a main function. The main function contains a nested function "sortedSearch" which performs the search logic. The code uses standard C++ syntax with comments explaining the purpose of each section.

```
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 below:



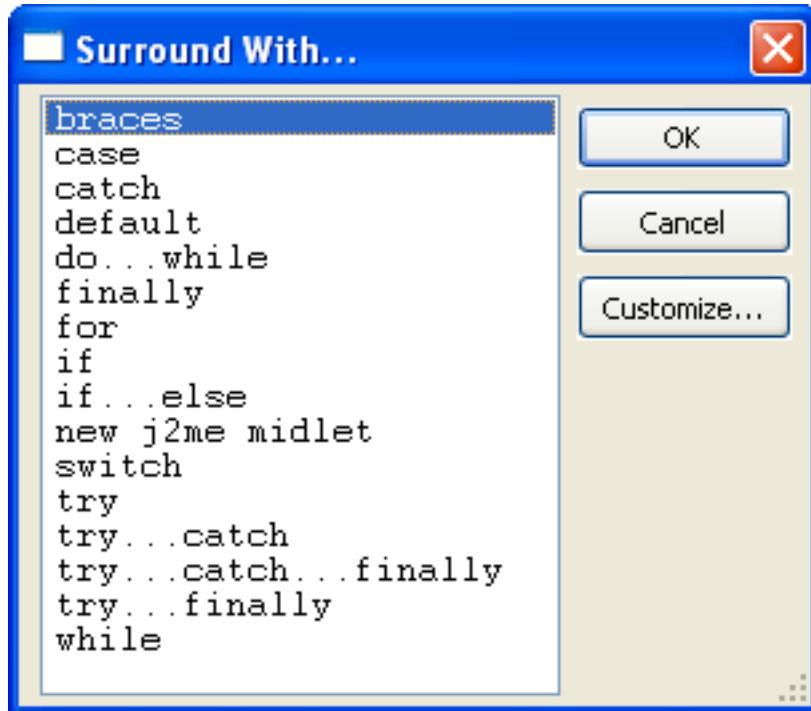
```
int sortedSearch( const int item, const int *pList, int numItems )
{
    if ( pList != NULL ) {
        // Binary search for the specified item in the array.
        const int * middleItem;
        int first = 0, last = numItems-1, middle;
        while ( first <= last ) {
            middle = ( 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 indent settings (see [Syntax Indent](#)). The color of the rectangle box guide is controlled by the **Block Matching** screen element (see [Colors](#)).

Syntax Expansion must be on for Dynamic Surround to work. Both options are on by default. To turn off either of these options, from the main menu, select **Tools** → **Options** → **Languages**, expand your language category and language, then select **Indent**. Clear the option(s) **Use Dynamic Surround** and/or **Syntax expansion**.

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).

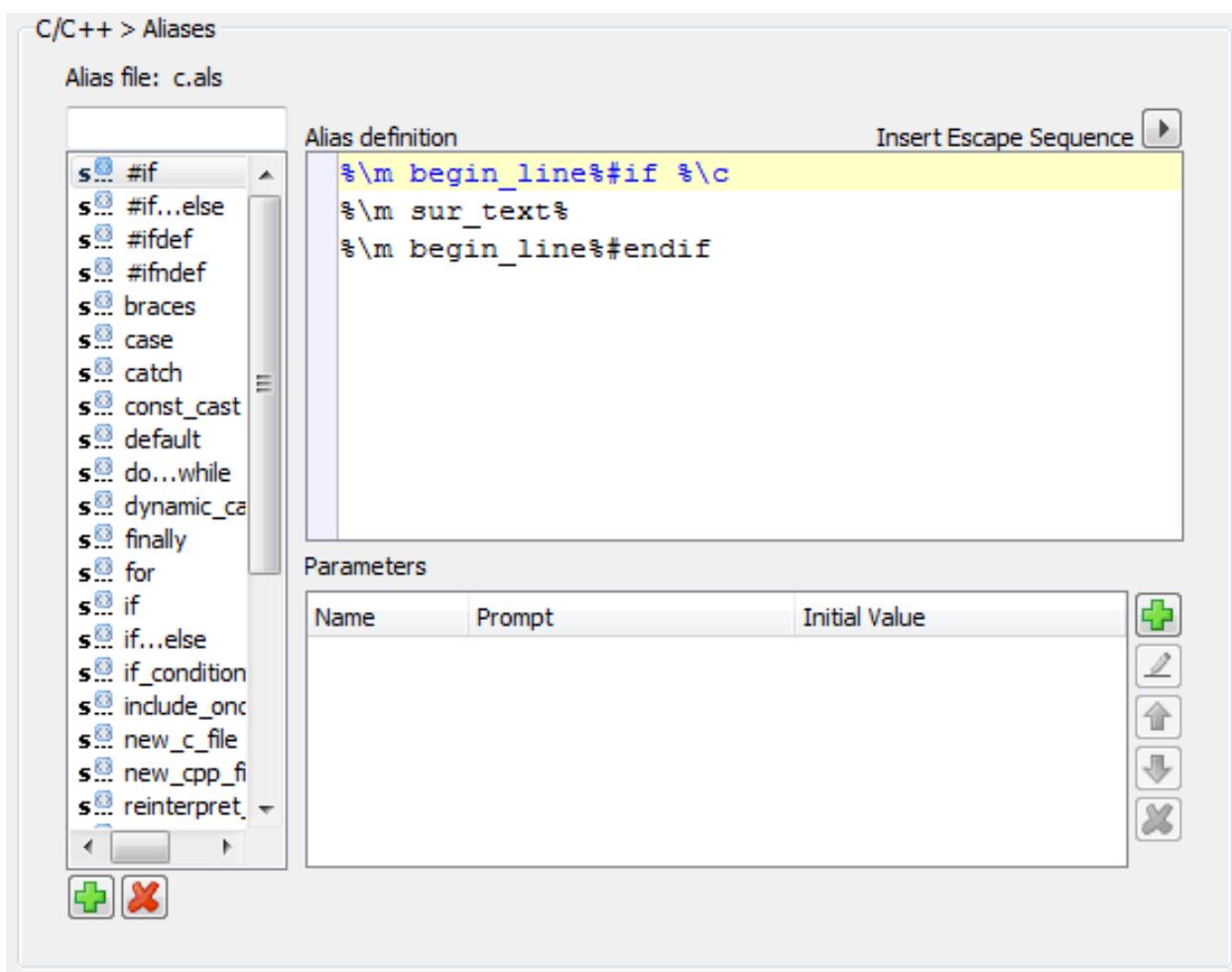
Tip

The **surround_with** command has a button available for toolbar customization. See [Customizing Toolbars](#) for more information about creating your own custom toolbars.

Surround With Aliases

Surround With aliases 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 alias. See [Surround With Commands](#) for more information on **sur_text**.

To view or modify the Surround With aliases, use the **surround_with** command to display the Surround With dialog, then click the **Customize** button. This will display the language-specific Alias options page. As an example, the C/C++ Alias options page is shown below.



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 alias for the structure appears in the text box on the right.
2. Modify the alias to suit your needs. For a list of escape sequences and examples, see [Alias Escape Sequences](#). For more information about using the Alias options page, see [Creating a Language-Specific Alias](#).
3. When you are finished, click **OK** on the Alias options page.
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 alias. It is used to indicate where the selected text should be placed and can be used multiple times within a single Surround With alias. **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 alias 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 alias 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** alias 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 alias name. For instance, calling **surround_with('i')** will prompt you to select the **if**, **if...else**, or **include once** alias. If there is an exact match, that alias will be used. In the case of **surround_with_if**, "if" matches the beginning of both the **if** and **if...else** aliases, but the **if** alias 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. By default, the unsurround command is bound to **Ctrl +Shift +Del**.

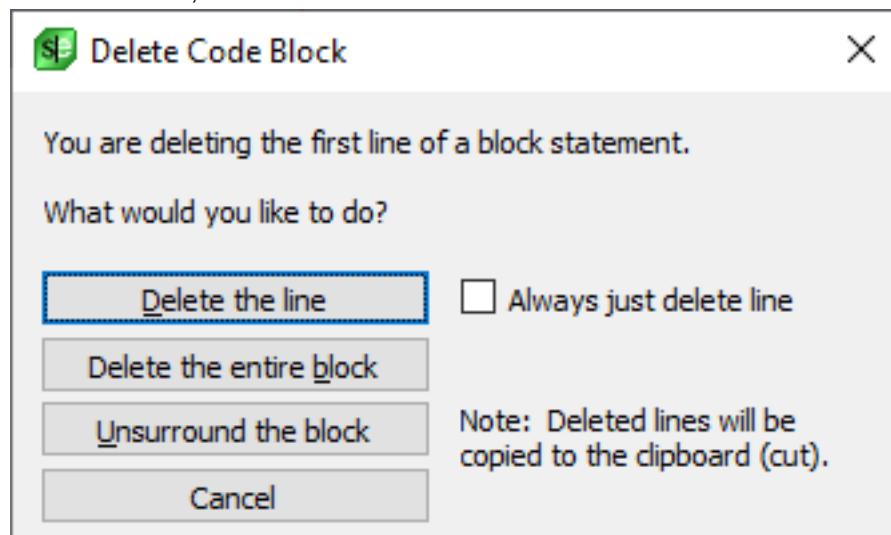
For example, to remove the **if** statement structure from a code block, select the code block or part of the code block, then right-click and select **Unsurround** (or use the **unsurround** command). The entire code block under the cursor is automatically highlighted and a dialog prompt appears to 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 a button 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.



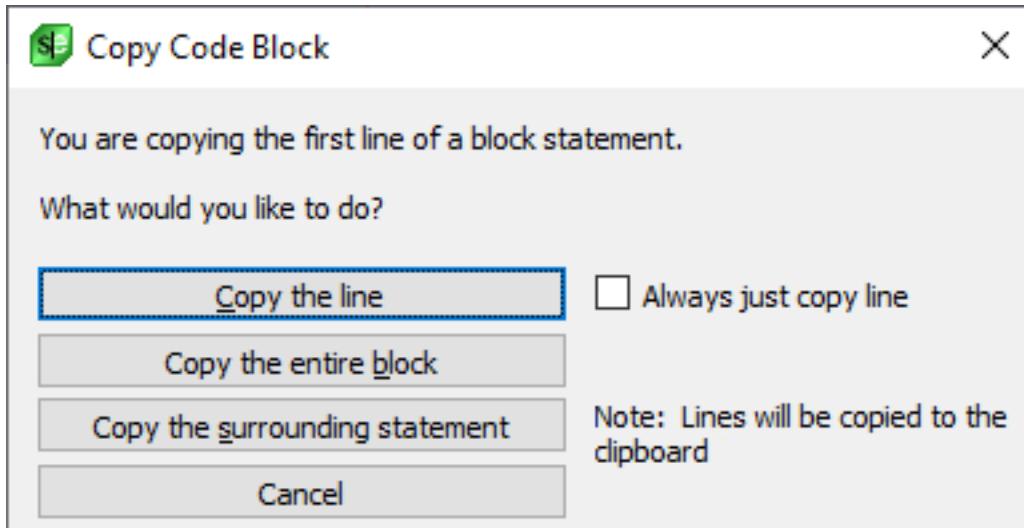
Each of these operations copies the removed text to the clipboard. When choosing the **Unsurround the**

block option, this is useful if you want to paste the statement 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.

Copying Code Blocks

Unsurround is also associated with the **copy-to-clipboard (Ctrl+C)** command. When this command is invoked while the cursor is on the first line of a block statement, the Copy Code Block dialog appears, from which you can choose to just copy the current line, copy the entire block, or copy the outer-most surrounding statement from the block.



Each of these operations copies text to the clipboard. When choosing the **Copy the surrounding statement** option, this is useful if you want to paste the statement 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 Copy Code Block dialog also contains an option to **Always just copy line when copy-to-clipboard** is invoked. Selecting this option will prevent the dialog from appearing when these operations are used. To see the dialog again, use the **copy_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](#) - Uses a green flag to mark long-term, meaningful locations in the code, or to quickly set a temporary, named bookmark on the current line.
- [Pushed Bookmarks](#) - Uses a blue flag to set temporary "breadcrumbs" as you explore the code. Pushed bookmarks are manipulated separately from named bookmarks.



```
// Repeat calculations for r2
r2.printBanner("Testing on r2");
printf("Width = %f\n", r2.getWidth());
printf("Height = %f\n", r2.getHeight());
```

Bookmark options for named and pushed bookmarks can be found on the [Bookmark Options](#) screen of the Options dialog (**Tools** → **Options** → **Editing** → **Bookmarks**).

A set of bookmark controls for named and pushed bookmarks is available for adding to your custom toolbars. See [Customizing Toolbars](#) for more information.

Named Bookmarks

Named bookmarks are great for marking long-term, meaningful locations in your code. For example, if you have a project with a lot of include files, you might want to bookmark the top of the main header file. Or you could bookmark a file with instructional comments about a particular project you're working on.

In addition to menu items and command line operations, the Bookmarks tool window (**View** → **Tool Windows** → **Bookmarks**) is available for creating and managing named bookmarks. See [Bookmarks Tool Window](#) for more information.

See the following sections for more information about working with named bookmarks:

- [Setting Named Bookmarks](#)
- [Navigating Named Bookmarks](#)
- [Deleting Named Bookmarks](#)
- [Using Workspace Bookmarks](#)

Setting Named Bookmarks

There are various ways to use named bookmarks, and the way you set them depends on which way you want to use them:

- **Give them a specific name** - Creating a bookmark and naming it yourself is one of the best ways to mark a location in your code. For example, you could set a bookmark named "main" to save the location of the **main** function. See [Setting a Bookmark With a Specific Name](#).
- **Allow automatic naming** - This is a quicker way to set named bookmarks. This method could be useful to mark locations that you temporarily need to reference, perhaps only in the current editing session. See [Setting a Bookmark With an Automatic Name](#).
- **Use a key binding shortcut for the name** - The quickest way to set and navigate named bookmarks is to name according to a specific key binding. This method lets you create a bookmark with one key binding, and navigate back to the bookmark with a similar key binding. See [Setting a Bookmark With a Key Binding](#).

After setting a named bookmark, a green bitmap is displayed in the left margin of the editor window, indicating the location of the bookmark. Use the **Show set bookmarks** option to enable or disable the display of the indicator (**Tools** → **Options** → **Editing** → **Bookmarks**).

Setting a Bookmark With a Specific Name

There are several different ways to set a bookmark on the current line and give it a name:

- Using the [Bookmarks Tool Window](#), click the **Create a New Bookmark** button.
- From the main menu, click **Search** → **Bookmarks** → **Set Bookmark**. The [Bookmarks Dialog](#) is displayed. Type the name of the bookmark in the combo box, then click **Add**.
- On the SlickEdit® command line, use the **set_bookmark** or **sb** command. The **sb** command is a shortcut for **set_bookmark**, so you can use whichever you prefer. If you use the command without arguments, the [Bookmarks Dialog](#) is displayed. Or, you can append **sb** or **set_bookmark** with any character or text string, and a bookmark will be instantly set using that value for the name. This works best in conjunction with the **goto_bookmark** (or **gb**) command, because you can use **sb 1** on the command line to create an instant bookmark named "1", and then navigate back to that bookmark at any time by using **gb 1**. See [Navigating Named Bookmarks](#) for more information.

Setting a Bookmark With an Automatic Name

Most of the methods described in the section for [Setting a Bookmark With a Specific Name](#) display a dialog that prompts for the name of the new bookmark. In each case, SlickEdit® prepopulates the name field with an automatic name that you can use if you don't want to specify your own name.

The automatic name will be 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®.

While you can use the methods described in [Setting a Bookmark With a Specific Name](#), the quickest method of setting a named bookmark with an automatic name is to use the Toggle Bookmark feature, which instantly sets an automatically named bookmark and lets you toggle it on and off. To use this feature, from the main menu, click **Search** → **Bookmarks** → **Toggle Bookmark**, press **Ctrl+Shift+J**, or use the **toggle_bookmark** command.

Setting a Bookmark With a Key Binding

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" through "9", and navigating back to them. See [Working with Key Binding Ranges](#) for more information.

Note

Different emulations have different default assignments for **alt_bookmark** and **alt_gtbookmark**. Use the menu item **Help → Where Is Command** to see what keys are set up by default in your emulation. See [Using the Command Line to View Key Binding Associations](#) for more information.

Navigating Named Bookmarks

To jump to and navigate between your named bookmarks, use one of the following methods:

- From the main menu, click **Search → Bookmarks → Previous Bookmark** or **Search → Bookmarks → Next Bookmark**(or use the **prev_bookmark** and **next_bookmark** commands, respectively). The order of navigation matches the order in which the bookmarks were created. These operations are also available on the [Bookmarks Tool Window](#).
- Use the Go to Bookmark feature: From the main menu, click **Search → Bookmarks → Go to Bookmark**, or, use the **goto_bookmark** command or the **gb** command. The **gb** command is a shortcut for **goto_bookmark**, so you can use whichever command you prefer. This displays the [Bookmarks Tool Window](#), from which you can select a bookmark to navigate to.

Tip

When you use the Go to Bookmark dialog to jump to a named bookmark, SlickEdit® pushes a bookmark in the process. To get back to where you were, press **Ctrl+Comma**. See [Pushing and Popping Bookmarks](#).

- You can also append `goto_bookmark` or `gb` with the name of the bookmark to go directly to that bookmark's location in the code. This works best in conjunction with the `set_bookmark` (or `sb`) command. For example, you can set a bookmark named "1" by using `sb 1` on the SlickEdit command line, then use `gb 1` to navigate back to that location. [Command Line Completion](#) is supported to assist you with typing the name of the bookmark. See also [Setting a Bookmark With a Specific Name](#).

Deleting Named Bookmarks

To remove a named bookmark, use one of the following methods:

- Using the [Bookmarks Tool Window](#), select the bookmark to delete and click the **Delete Selected Bookmark** button or press the **Delete** key.
- When the cursor is on the bookmark line, use the Toggle Bookmark feature to toggle that bookmark off (in effect, deleting it): From the main menu, click **Search** → **Bookmarks** → **Toggle Bookmark**, or, press **Ctrl+Shift+J** or use the `toggle_bookmark` command. Use the feature again to toggle the bookmark back on.
- On the SlickEdit® command line, use the `delete_bookmark` command with the name of the bookmark to delete. For example, if the bookmark to delete is named "1", type `delete_bookmark 1` on the command line. [Command Line Completion](#) is supported to assist you with typing the name of the bookmark.
- On the [Bookmarks Dialog](#) or [Go to Bookmark Dialog](#), select the bookmark to remove and click **Delete**.

To remove all named bookmarks at once:

- Using the [Bookmarks Tool Window](#), click the **Delete All Bookmarks** button.
- On the SlickEdit® command line, use the `clear_bookmarks` command.

Using Workspace Bookmarks

By default, named bookmarks are global and are not associated with a specific workspace (see [Managing Workspaces](#)). However, you can choose to associate your bookmarks with a workspace instead. When you use workspace bookmarks, the [Bookmarks Tool Window](#) (and other dialogs that show bookmarks) only displays bookmarks that are associated with the current workspace. You can set bookmarks for files that are not in the current workspace. To enable workspace bookmarks, set the **Use workspace bookmarks** option to **True** (**Tools** → **Options** → **Editing** → **Bookmarks**).

Relocatable Code Markers

Named bookmarks use relocatable code markers to store their location within the source code. This allows SlickEdit to find the new location if someone makes changes to the file externally, like modifying 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 bookmark. If the code has changed too much, SlickEdit may not be able to find the new location. Instead, the bookmark will be placed at the line number where it was last known to be.

Pushed Bookmarks

Pushed bookmarks are used to set temporary "breadcrumbs" as you move throughout your code. For example, you may have multiple spots in your code that you want to examine. You can push a bookmark (drop a breadcrumb) at each location, one right after the other. When you're done examining the code and pushing bookmarks, you can backtrack to where you first started by popping the bookmarks (picking up the breadcrumbs).

Pushed bookmarks are stored on the bookmark stack, which is simply an internal list of pushed bookmarks. When you push a bookmark, the current line is placed on top of the bookmark stack. Popping a bookmark removes the top bookmark from the stack, and navigates the cursor to the location of the previous bookmark.

Pushed bookmarks are deleted when you close SlickEdit®.

See the following sections for more information:

- [Pushing and Popping Bookmarks](#)
- [Viewing Pushed Bookmarks](#)
- [Pushed Bookmark Options](#)

Pushing and Popping Bookmarks

To push a bookmark for the current line, placing it on top of the bookmark stack, use one of the following methods:

- From the main menu, click **Search** → **Bookmarks** → **Push Bookmark** (or use the **push_bookmark** command).
- Use the Go to Definition or Go to Reference feature: Press **Ctrl+Dot** (bound to the **push_tag** command) to move the cursor from a symbol to its definition, or **Ctrl+/** (bound to the **push_ref** command) to navigate from a symbol to its reference, pushing a bookmark in the process. See [Symbol Navigation](#) for more information.
- To pop a bookmark, from the main menu, click **Search** → **Bookmarks** → **Pop Bookmark**, press **Ctrl+Comma**, or use the **pop_bookmark** command. The top bookmark on the stack is removed and the cursor jumps to the location of the previous bookmark.

To pop all pushed bookmarks at once, use the **pop_all_bookmarks** command.

Tip

When you use the Go to Bookmark dialog to jump to a named bookmark (see [Navigating Named Bookmarks](#)), SlickEdit pushes a bookmark in the process. This way you can quickly go back to your previous location.

Viewing Pushed Bookmarks

In most use cases, you will never need to see pushed bookmark locations or a list of pushed bookmarks, because they are intended to act as temporary "breadcrumbs" as you explore your way through code.

However, there are two ways to see visual representations of pushed bookmarks:

- **Enable the visual indicator** - SlickEdit® can display a blue bitmap in the left margin of editor windows at the location of each pushed bookmark. To enable display of the indicator, set the **Show pushed bookmarks** option to **True** (**Tools** → **Options** → **Editing** → **Bookmarks**).
- **Use the Bookmark Stack dialog** - SlickEdit provides a Bookmark Stack dialog that shows a list of all pushed bookmarks that are currently set. To display it, from the main menu, click **Search** → **Bookmarks** → **Bookmark Stack**, or use the **bookmark_stack** command. The Bookmark Stack dialog can also be used to navigate between and delete pushed bookmarks. See [Bookmark Stack Dialog](#) for more information.

Pushed Bookmark Options

There are several options for pushed bookmarks:

- When using [Symbol Navigation](#) (Go to Definition or Go to Reference), if SlickEdit® opens a file that was not previously open and you navigate away from it, SlickEdit prompts to close the visited, unmodified file. To remove the prompt and specify the default action, set the option **Automatically close visited files** (**Tools** → **Options** → **Editing** → **Bookmarks**). See [Automatically Closing Visited Files](#) for more information.
- To delete pushed bookmarks automatically when a file is closed, set the **Close deletes pushed bookmarks** option to **True** (**Tools** → **Options** → **Editing** → **Bookmarks**). SlickEdit automatically deletes pushed bookmarks when the editor is closed.
- SlickEdit can automatically 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 enable this behavior, set the **Top/bottom buffer pushes bookmark** option to **True** (**Tools** → **Options** → **Editing** → **Bookmarks**). Note that this option corresponds to the **def_top_bottom_push_bookmark** configuration variable.

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 Options dialog opens to the language-specific **Comments** screen for the current language. You can also open this screen by clicking **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Comments**.

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 [Language-Specific Comment Options](#).

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 activate 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 Options dialog is displayed, open to the **Comments** option screen for the current language. You can also open this screen by clicking **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Comments**.
2. In the **Doc comments** box, check the option **Automatically expand doc comments**.
3. Click the **Edit expansion** button to configure the start characters and comment templates for the doc comment style you plan to use for the selected language. For comments formatted in Javadoc, select **/***. For XMLDoc, select **///**. For Doxygen, select **/*!** or **!!**.
4. Optionally, click the **Edit expansion** button to view or edit the doc comment template that is inserted when you type the selected start characters. See [Modifying Doc Comment Templates](#) for more information.
5. Click **OK** on the Options dialog.

Tip

If you modify a function signature, you can update the associated doc comment by running the **update_doc_comment** command from the SlickEdit command line.

Doc Comment Examples

Javadoc Format

To use the Javadoc commenting format for the selected language, 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:

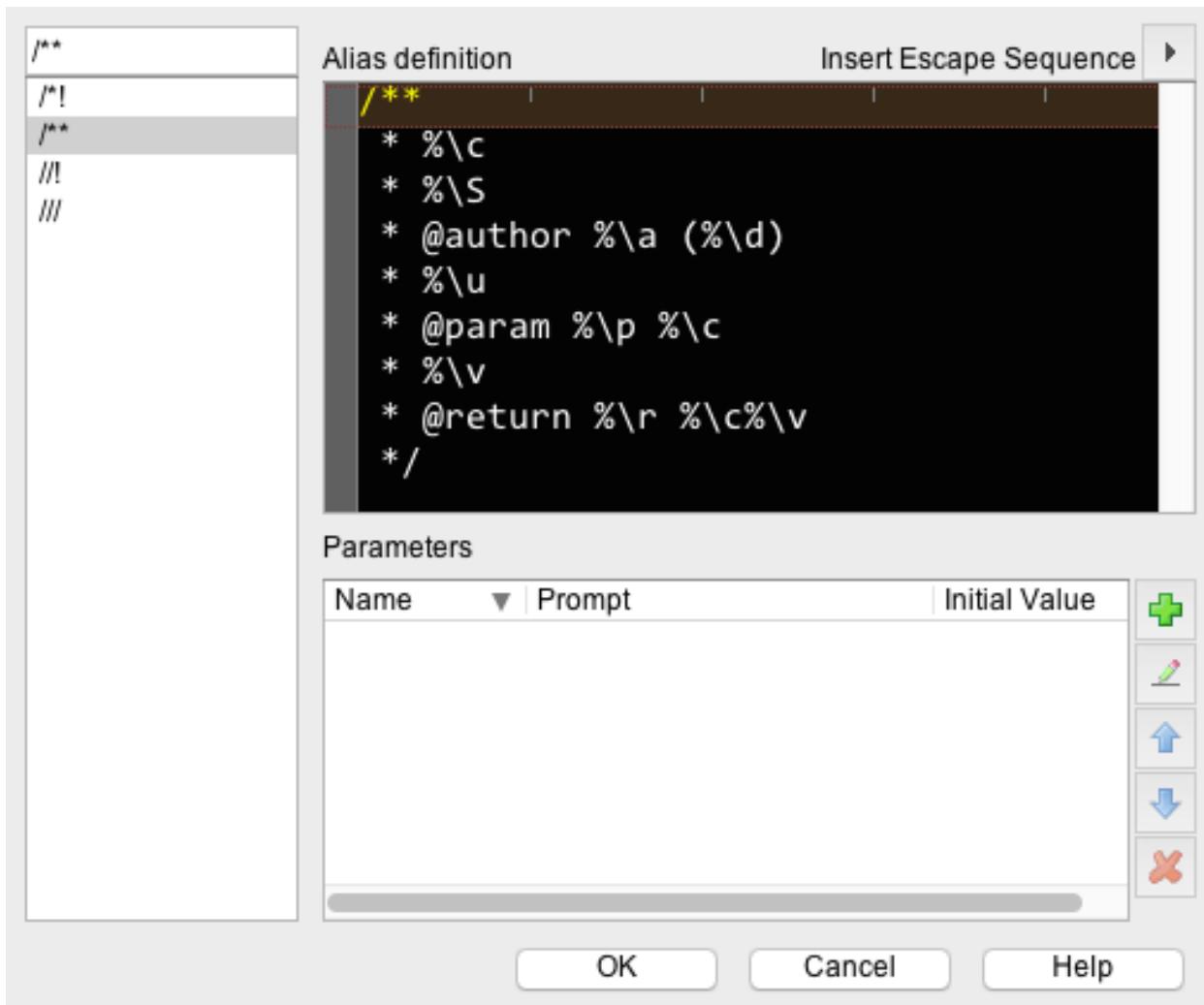
```
/*! [CURSOR_HERE]*/
int setDimensions(int length, int width, int height) {
    ...
}
```

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) {
    ...
}
```

Modifying Doc Comment Templates

To modify a doc comment template, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, and select Comments. Then click the **Edit expansion** button. The Doc Comment Editor dialog for the selected language opens. Click on the start characters for style of comments you want to use to view and edit the associated comment template.



The box on the left contains a list of doc comment start characters. The edit window on the right contains the expansion for the selected start characters. See [Alias Escape Sequences](#) for a list of special escape characters you can use inside doc comment templates, for example, to insert local function param names, types, and return types. See [Doc Comment Examples](#) for an example of each comment style.

Tip

You cannot add or delete doc comment templates using the Doc Comment Editor. You can, however, add a new doc comment expansion as a regular language-specific alias. See [Creating a Language-Specific Alias](#) for more information. All of the doc comment escape sequences will work as long as you expand the alias on a blank line above a function or class declaration.

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, click **Document** → **Comment Setup** (`comment_setup` command). The Options dialog is displayed open to the **Comments** screen. Select the option **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 activate comment wrapping, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Comment Wrap**. Select 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** screen 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 [Language-Specific Comment Wrap Options](#).

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, click **Document** → **Reflow Comment**. For more information on the available options, see [Reflow Comment Dialog](#).

Code Annotations (Pro only)

Code Annotations Overview (Pro only)

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 (Pro only)

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 (.vpw) 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 located 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

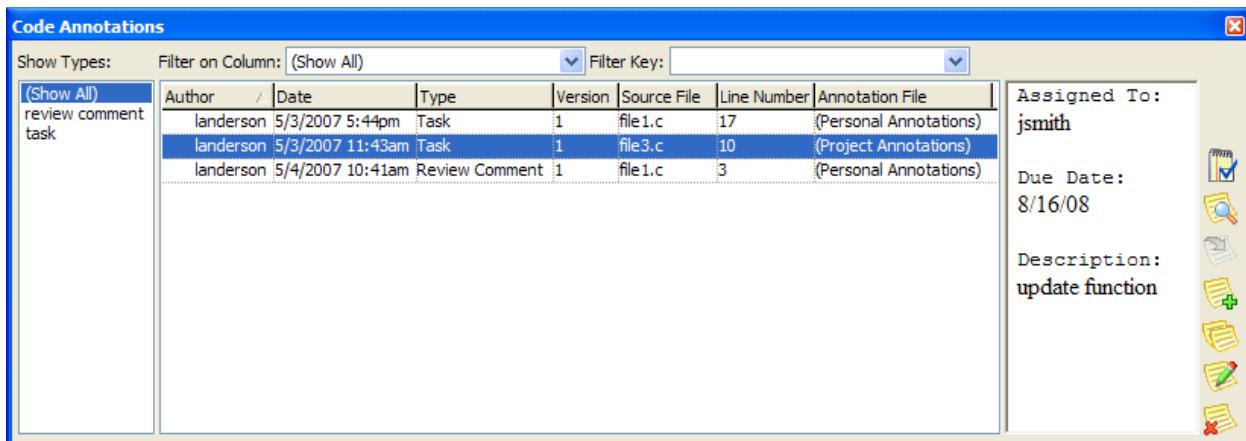
Managing Annotations (Pro only)

annotation files you like. See [Managing Annotation Files](#) for more information.

Managing Annotations (Pro only)

The Code Annotations tool window provides a detailed view of annotations that you have created as well as operations for adding, modifying, moving, and removing annotations. From this window, you can also manage annotation files and create your own annotation types.

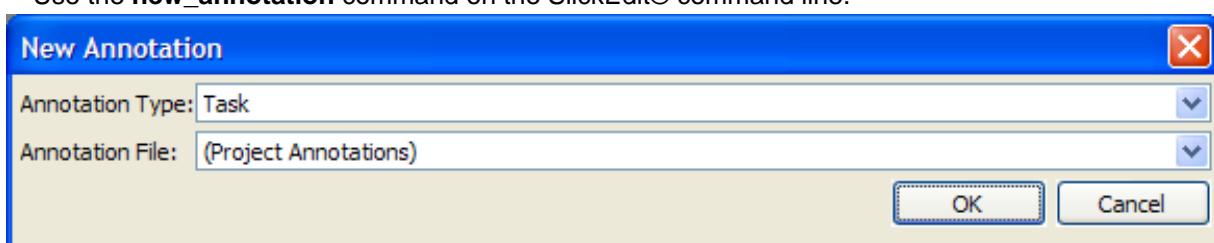
To display the tool window, from the main menu click **View → Tool Windows → Code Annotations**, or use the **annotations_browser** command on the SlickEdit command line.



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:
 - Right-click and select **Create Code Annotation**.
 - Click the **Add**  button on the Code Annotations tool window.
 - 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:
 - **Comment** - The Comment Annotation dialog provides a box for typing a text comment.
 - **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.
 - **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 Code 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.

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.

Filtering Code Annotations

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.

Note

The columns listed in the **Filter on Column** drop-down list depend on the annotation types selected. If you have selected **(Show All)**, then only the columns shared by all annotation types will be listed. If you don't see the column you want to filter by, then you may need to select a different annotation type.

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

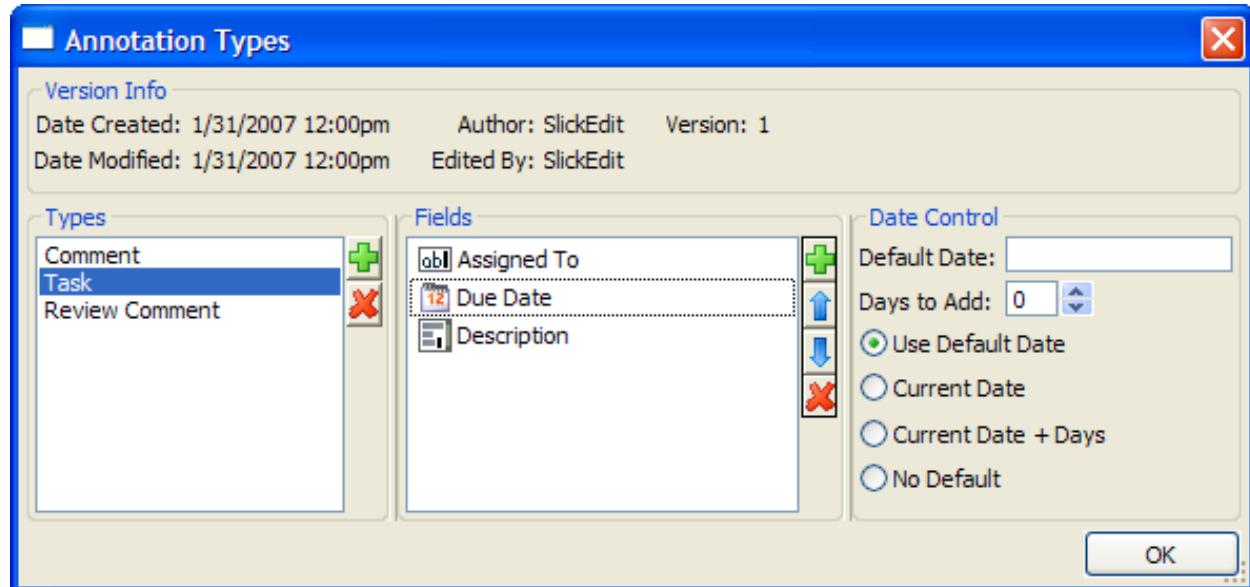
To delete an annotation, from within the Code Annotations tool window, select the annotation to delete then click the **Delete**  button. You can also right-click and select **Delete** or use the **delete_annotation**

Managing Annotations (Pro only)

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.

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.

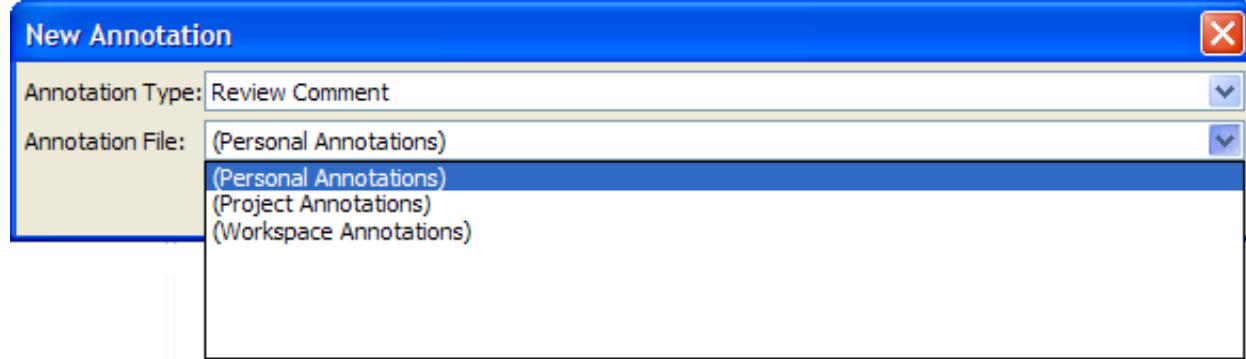
To avoid conflicts in your personal annotation types, you may want to create your own types. Conflict resolution is performed using the name of the annotation type to uniquely identify a type. If the types you use in your personal annotations have their own, unique names then you can avoid conflicts 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

Managing Annotations (Pro only)

specify the location when you create new code annotations by selecting from the Annotation File dropdown 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`.

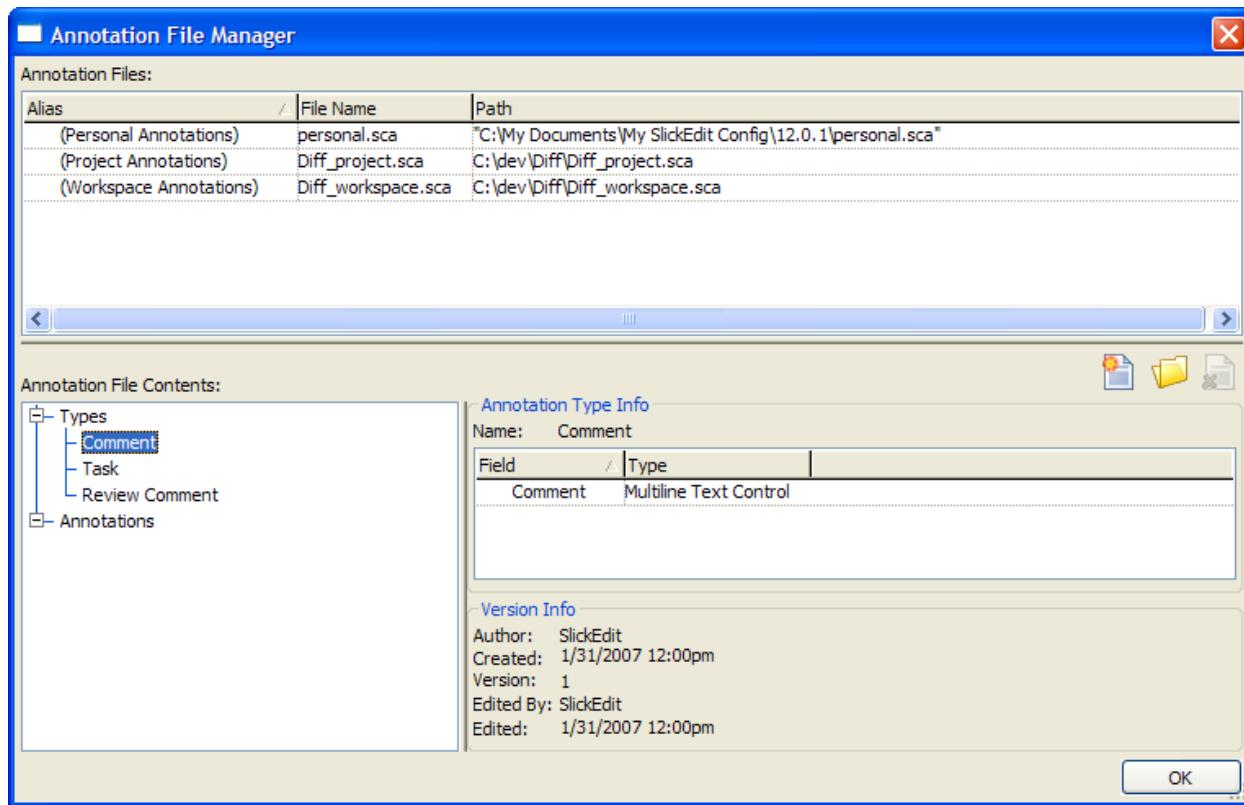
User-Defined Annotations

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.

Managing Annotations (Pro only)



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 (Pro only)

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 (Pro only)

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 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.

Message List (Pro only)

Message List is a feature that shows output messages from processes running in SlickEdit®, such as build warnings and errors.

Messages are automatically displayed in the Message List tool window, docked in the bottom tab group of the editor by default. The tool window shows messages in tabular format, with columns for the message type (warning, error, etc.), source file and line number associated with the message, a description, the message origin (Build, Java Live Errors, XML validation, etc.), and the date. Messages can be filtered and sorted. You can clear the tool window of all messages, or clear only messages with a certain creator or type.

Messages are associated with a location in the code. When the cursor is on a line with an error or warning in the source code, the corresponding message in the Message List is highlighted. You can also navigate to the message in the source code from within the Message List by double-clicking on it (or right-click on a selected message and click **Go to code location**).

Processes That Use Message List

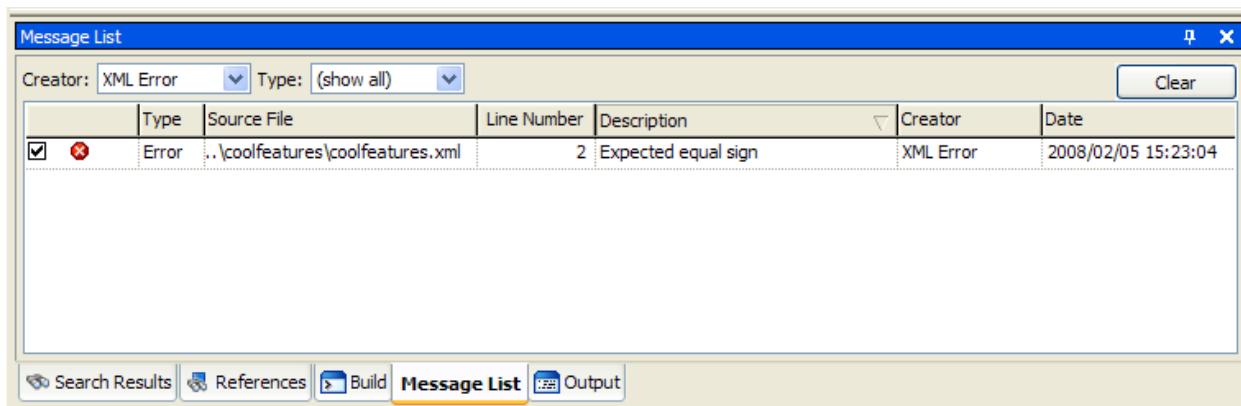
The following processes originate messages that are automatically displayed in the Message List:

- Build operations (such as **Build → Compile** or **Build → Build**) show Warning and Error messages. See [Working with Build Errors](#) for more information.
- Live Errors, when it is enabled, shows Caution and Error messages. See [Java Live Errors](#) and [Overview](#) for more information.
- XML validation shows Error messages. SlickEdit® automatically validates XML files that you open in the editor. You can force a validation check at any time with the **xml_validate** command.

Message List Tool Window (Pro only)

The Message List tool window is docked to the bottom tab group of the editor by default. Display of the tool window can be toggled on/off by clicking **View → Tool Windows → Message List** or by using the **toggle_messages** command. To display the tool window on demand, use the **activate_messages** command.

Message List Tool Window (Pro only)



Messages are shown in tabular format with the information divided into columns. Click on a column header to sort by that column in ascending order, or click again to sort in descending order. An arrow in the column header indicates the ascending or descending sort order. Drag the column separators to resize columns.

The columns are:

- **Indicator** - The first column in the Message List window shows the icon that corresponds to the message in the code. The same icon is used in editor window margins to indicate lines that contain messages. This column can also be used as a type of "To Do" checklist. Click inside a cell in this column to place a checkmark next to the message, and click again to clear it. This could be useful if you just want to change the state of a message, rather than removing the message.
- **Type** - This column shows the type of message, such as Error, Warning, etc.
- **Source File** - Messages are associated with a location in the code. This column shows the complete path to the source file containing the message. As you resize this column, the path is elided to keep the file name visible.
- **Line Number** - Shows the line number containing the message.
- **Description** - The text of the message.
- **Creator** - The Creator is the process that originated the message, such as Build, Java Live Errors, XML validation, etc.
- **Date** - This column contains the date and timestamp showing exactly when the message was generated. This could be useful if a process is lengthy, to see the order in which messages were generated.

As you move the mouse over messages the Message List window, a tool tip shows the message detail. You can also see a preview of the location of the message in the source code in the [Preview Tool Window](#), assuming it is docked so that both are visible, by single-clicking on a message in the Message List.

When the cursor is on an error or warning in the source code, the corresponding message in the Message List is highlighted. Double-click on a message in the Message List and the cursor moves to that message in the source code (or select the message and choose **Go to Message** from the right-click

context menu).

For messages originating from Build processes, the Build tool window shows Build output for a selected message when you choose **Go to Build Output** from the right-click context menu.

Filtering and Removing Messages

Use the **Creator** and **Type** drop-down boxes at the top of the tool window to filter the Message List to only show messages with the selected Creator and/or Type. The default value for both is **(show all)**.

The **Clear** button is used to remove all visible messages from the Message List. Clicking **Clear** also resets the filters to **(show all)**, displaying all messages except for the ones you just cleared. The messages do not appear again until/unless the originating operation regenerates them.

Remove selected messages in the Message List by choosing **Delete** from the right-click context menu.

Document Overview Bar

SlickEdit features an overview bar that will be positioned alongside vertical scrollbars to indicate the position of important items in the current document relative to the current scroll position in the document. This allows you to glance at the scrollbar and get a quick overview of where in the document different items exist. To quickly scroll a marked location into view, you can drag the scrollbar slider to the mark, or click on the mark itself.



The screenshot shows the SlickEdit edit window with the file "Document.m" open. The code editor displays Objective-C code with line numbers from 113 to 130. A vertical scrollbar is on the right side of the editor. A red dot marks the position of line 123, which contains the assignment of an NSTextStorage object to the variable "text". A green arrow points to the top of the scrollbar, indicating the position of the marked line.

```
113 */
114 - (BOOL)readFromURL:(NSURL *)absoluteURL ofType:(NSString *)typeName error:(NSError **)outError
115 {
116     NSMutableDictionary *options = [NSMutableDictionary dictionaryWithCapacity:5];
117     NSDictionary *docAttrs;
118     id val, paperSizeVal, viewSizeVal;
119     NSTextStorage *text = [self textStorage];
120     [fileTypeToSet release];
121     fileTypeToSet = nil;
122     [[self undoManager] disableUndoRegistration];
123     [options setObject:absoluteURL forKey:NSBaseURLDocumentOption];
124 }
```

The following items are marked up on the SlickEdit edit window scrollbar:

- Find output when **List all occurrences** or **Highlight all matches** is on. These options can be found on the [Find Tab](#). See [Find and Replace](#) for more information about searching within the application.
- Compiler errors and warnings
- [Named Bookmarks](#)

The Document Overview bar is also used in the [DIFFzilla®](#) edit window, shown below.

Highlighting

The screenshot shows the DIFFzilla Pro interface comparing two files: f:\gnudemo\Rectangle.cpp and f:\gnudemo\Rectangle_brace2.cpp. The left file contains code for a Rectangle class with methods for width and height. The right file is identical but includes brace matching. A red oval highlights the scrollbar area, which features three distinct markers: a red square for 'Modified line', a green square for 'Inserted line', and a yellow square for 'Imaginary line'. The status bar at the bottom displays file paths, line numbers (4/85 and 5/86), and various toolbar buttons.

The following items are marked up on the DIFFzilla window scrollbar:

- Modified lines
- Inserted lines
- Imaginary (deleted) lines

For larger files, the scrollbar slider will not be an exact representation of the viewable portion of the screen since there is a minimum size for the scrollbar slider. In these cases, clicking on the markers will be more useful.

Highlighting

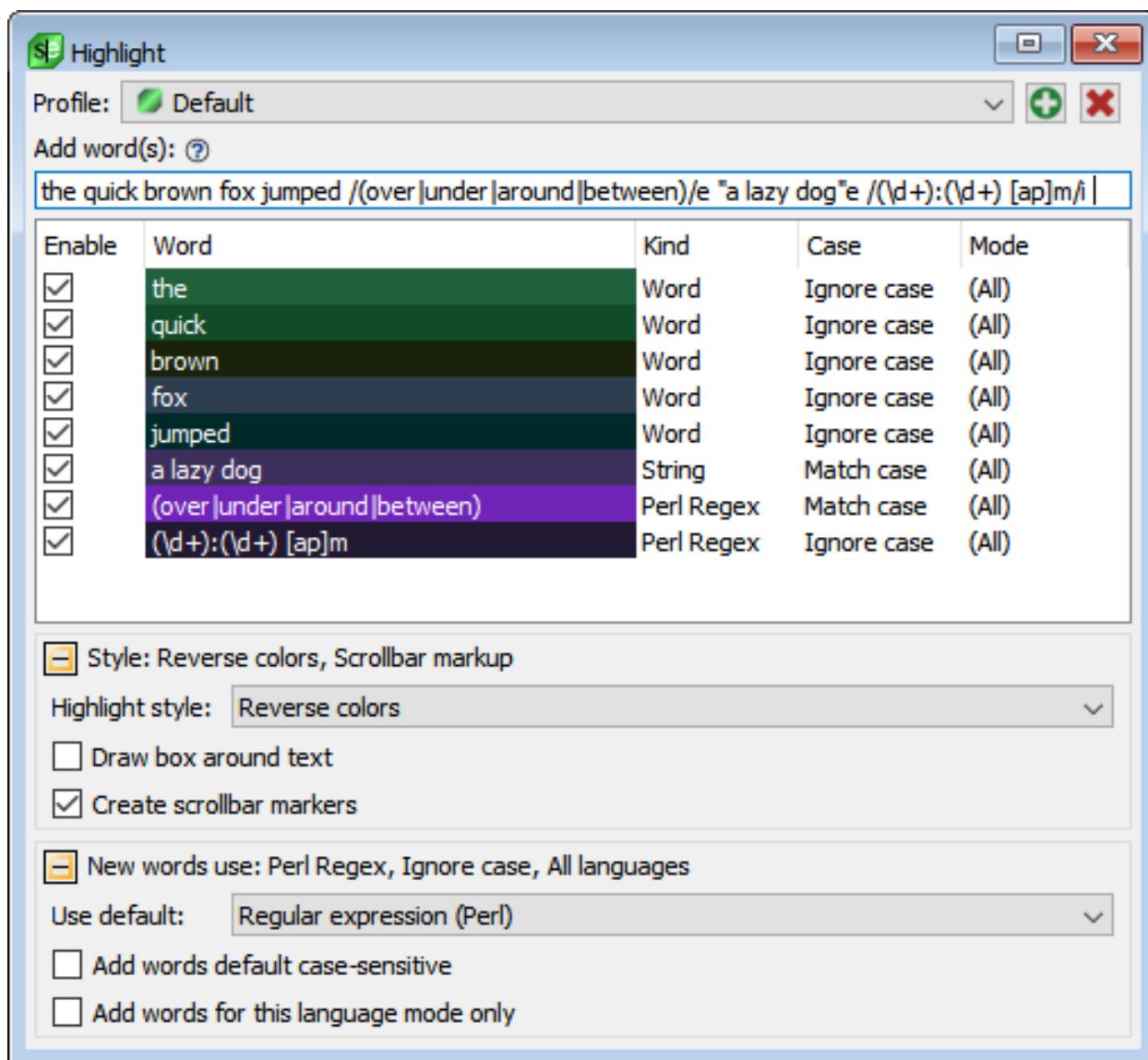
Highlighting allows you to define a set of word patterns to highlight using an array of colors. The word patterns can be simple word matches, substring matches, or regular expressions. There are a variety of options for how to display the highlighted matches. You can also define custom sets of word patterns (Highlight profiles).

Highlight Tool Window

The Highlight tool window is not docked by default. It is a vertically oriented tool window and works best docked to the left or right. It can also be used in a very small horizontal mode where you only expose the Profile name and the "Add word(s)" text box for quick one-off highlighting of word patterns.

Display of the tool window can be toggled on/off by clicking **View → Tool Windows → Highlight** or by using the **toggle_highlight** command. To display the tool window on demand, use the **activate_highlight** command.

Highlight Tool Window



The following example shows how the word patterns shown above would be displayed in the editor window.

The quick brown fox jumped over a lazy dog at 12:40 pm.

The Highlight tool window allows you to select from a set of highlighting profiles which are sets of word patterns to highlight. There are a few built-in profiles, and you can add your own by clicking on the add (+) button to create a new profile. The delete button (x) can be used to delete a custom profile or to reset a built-in profile back to defaults. The built-in **Default** profile can not be deleted, and does not contain any word patterns by default. The built-in **(None)** profile is used to indicate that highlighting is currently disabled.

Highlighting word patterns are shown in tabular format with the information divided into columns. The columns are:

- **Enable** - The first column in the Highlight tool window shows a check box to indicate if the word pattern is enabled or disabled. Click on the check box to toggle its state.
- **Word** - This column shows the actual word pattern. When word patterns contain special characters, such as tabs, the special characters will be displayed here using standard C/C++ string escape sequences, for example \t for a tab character. Click in this column to edit the word pattern.
- **Kind** - The kind of word pattern to use. One of **Word** for a simple word match, **String** for a substring match which does not depend on word boundaries, **Vim Regex**, **Perl Regex** or **SlickEdit Regex** for a regular expression pattern match. Click in this column to select a specific match type using the combo box.
- **Case** - The case-sensitivity to be used for this word pattern. One of **Match case** for case-sensitive, **Ignore case** for case-insensitive, and **Language case** to use language mode specific case-sensitivity. Click in this column to select a specific mode using the combo box.
- **Mode** - The language mode this word pattern is restricted to (defaults to **All**). Click in this column to select a specific language mode using the combo box.

The **Add word(s)** text box is used for adding new word patterns to the word highlighting. There are three different kinds of word highlighting patterns recognized here.

- **Words** - Words are the simplest type of word pattern and are expected to match on word boundaries.
- **String** - Substring matches are entered in double quotes or single quotes, and optionally can be followed by an **E** or **I** modifier to indicate whether the match should be case-sensitive or not, as you can see in the example above, where "a lazy dog" is specified as a substring match using exact case matching. Substring matches do not have to match on word boundaries. Double-quoted strings support basic C-style escape sequences, including ` `. Single quote strings are raw strings. Two adjacent single quotes can be used to escape a single quote.
- **Regular expression** - Regular expression matches are entered surrounded by slashes or backticks, much like Perl language syntax for regular expression matching. If the expression contains a forward slash, it can optionally be escaped using a backslash if necessary. They are optionally followed by a **E** or **I** modifier to indicate case sensitivity, and/or a **L**, **V**, or **R** modifier to indicate the regular expression type.

The Highlight tool window allows you to select from a variety of styles for the word highlighting. The **Style** panel can be expanded or collapsed to save space by clicking on the [+ / -] button.

- **Highlight style** - Select from **Reverse colors**, **Color text**, **Bold text**, **Underline text**, **Strikethrough**, or **Highlight color**. The **Highlight color** uses the background color from the corresponding color in your color profile. All other modes use colors selected from the extended color palette. See [Color Options](#) for more information.
- **Draw box around text** - Draw a box around the highlighted matches.
- **Create scrollbar markers** - Create editor scrollbar markup for highlighted matches.

The Highlight tool window also allows you to select a variety of options for how a word pattern is added by

default. The **New words** panel can be expanded or collapsed to save space by clicking on the [+ / -] button.

- **Use default** - Set this option to select one of the following types of regular expression search syntax to use when a regular expression word pattern is added. The following types are supported: **Regular expression (SlickEdit®)**, **Regular expression (Perl)**, or **Regular expression (Vim)** See [Using Regular Expressions in SlickEdit®](#) for more information.
- **Add words default case-sensitive** - This check box has three states. Checked indicates new word patterns will be case-sensitive (**Match case**). Unchecked indicates new word patterns will be case-insensitive (**Ignore case**). The in-between case indicates that new word patterns should be added using language mode specific case-sensitivity options.
- **Add words for this language mode only** - If checked, new word patterns will be added to the highlighting profile restricted to the current language mode instead of being highlighted in all language modes.

Highlighting Commands

The following commands are available for controlling highlighting. None of these commands are bound to keys by default. See [Managing Bindings](#) to learn how to bind a command to a key. See [Toolbar Customization dialog](#) to learn how to add a toolbar button for a command.

- **activate_highlight** - Activates the Highlight tool window.
- **toggle_highlight** - Toggles the display of the Highlight tool window.
- **toggle_highlight_pinned** - Toggles the display of the Highlight tool window between pinned and unpinned states.
- **highlight_delete_profile** - Delete the given highlight profile name.
- **highlight_toggle_enabled** - Toggle word pattern highlighting entirely on or off.
- **highlight_clear_all** - Turn word pattern highlight entirely off.
- **highlight_cycle_add_color** - Cycle the color for the word pattern under the cursor to the next color. If there is no word pattern highlighted under the cursor, create a new word pattern for the word or selection under the cursor and add it to the current highlighting profile.
- **highlight_cycle_next_color** - Cycle the color for the word pattern under the cursor to the next color.
- **highlight_cycle_prev_color** - Cycle the color for the word pattern under the cursor to the previous color.
- **highlight_toggle_word_enabled** - Toggle the word pattern under the cursor between the enabled and disabled states.
- **highlight_toggle_word_case** - Toggle the case-sensitivity options for the word pattern under the cursor.

- **highlight_toggle_word_or_string** - Toggle the search options between doing a word match and doing a substring match for the word pattern under the cursor.
- **highlight_toggle_word_language** - Toggle the language-specific attribute for the word pattern under the cursor between the current language mode and all languages.
- **highlight_toggle_word** - Toggle whether the word pattern under the cursor is part of the current highlight profile or not. If there is no matching word pattern highlighted under the cursor, add a new word pattern to the profile.
- **highlight_add_word** - Add the word under the cursor to the current highlighting profile.
- **highlight_delete_word** - Remove the word pattern under the cursor from the current highlight profile.

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.

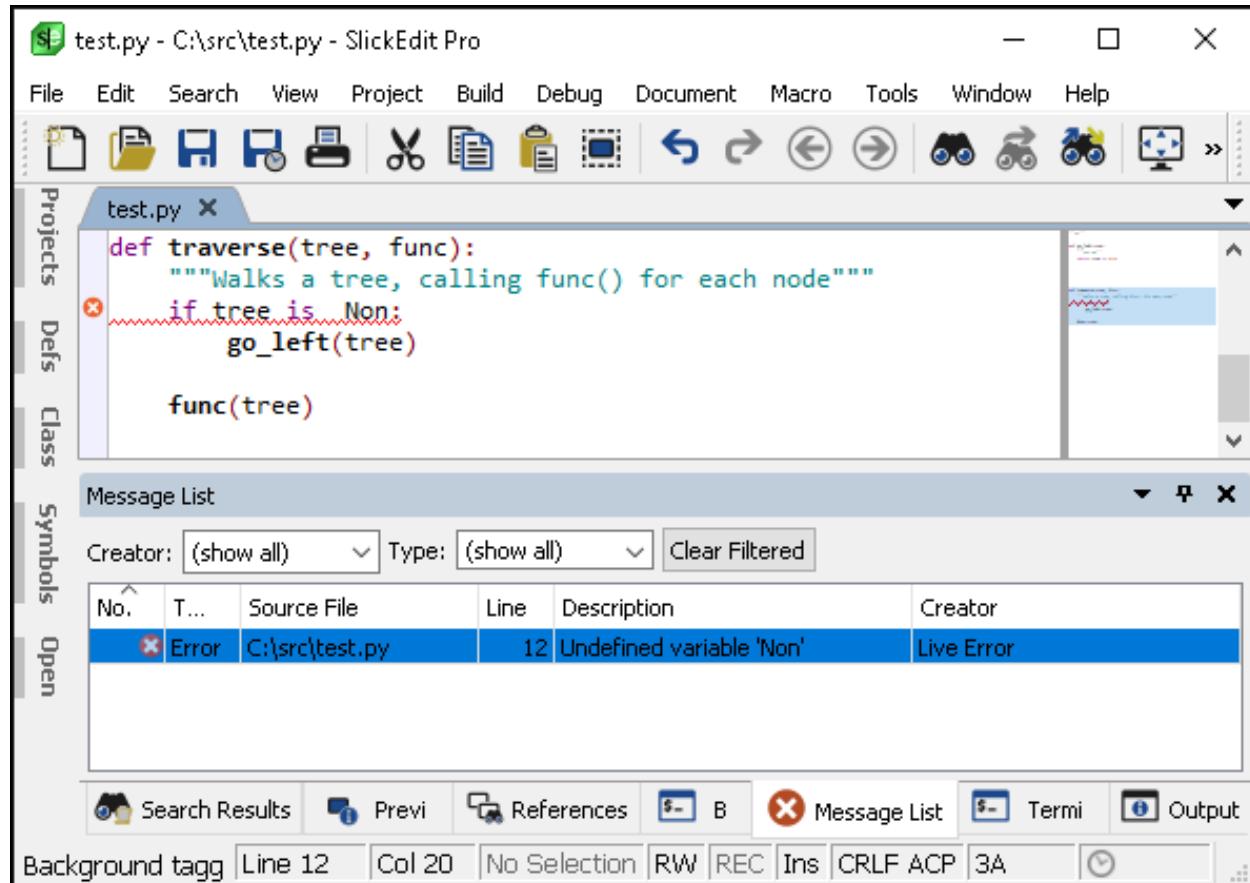
To beautify selected lines of code, or to beautify the entire buffer, from the main menu, click **Tools** → **Beautify** (or use the **gui_beautify** command). A dialog box is displayed with functions specific to the type of project that is active. If an HTML project is active, then the HTML Beautifier 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.

- C/C++, Objective-C, Java, C#, Python, JavaScript, VBScript, PHP, SystemVerilog, Verilog, Groovy - These languages use beautifiers accessible through the Language options. See [Beautifiers](#).
- HTML, CFML, XML, and XSD - These beautifiers contain the same options and settings. See [HTML and XML Beautifiers](#).
- Javadoc - See [Javadoc Beautifier Options Dialog](#).

Live Errors

Overview

The Live Errors system provides error and warning feedback as you edit the source code, using user configurable syntax checking programs to generate the errors.



Languages with well known checkers, such as Python & Pylint, are shipped as pre-configured profiles for convenience.

Configuring Live Errors

Live Errors profiles are grouped by source language. See [Language-Specific Live Errors Profiles](#) for details.

Reflowing Text

To reflow text in the current paragraph according to your margin settings, click **Document** → **Reflow Paragraph** or use the **reflow_paragraph** command. Margin settings are defined on the language-specific Word Wrap options screen (see [Language-Specific Word Wrap Options](#)).

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 changed (**Tools** → **Options** → **Editing** → **General**). If **Reflow next** is set to **Cursor on next paragraph**, 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.

Quick Refactoring (Pro only)

Refactoring is a code editing technique used to "clean up" and improve the understandability of source code without affecting the code's external behavior. Quick Refactoring is a feature set that provides several fast and easy-to-use refactoring methods:

- [Quick Rename](#) ® Rename a symbol under the cursor or any symbol selected in the **Defs** or **Symbols** tool windows.
- [Quick Extract Method](#) ® Create a new method using currently selected lines as the body and any undeclared variables as parameters.
- [Quick Modify Parameter List](#) ® Use to add, delete, and re-order parameters for a selected function.
- [Quick Replace Literal with Constant](#) ® Replace a selected literal with a constant.

Quick Refactoring performs refactorings using [Context Tagging Features](#) rather than a formal language parser. Quick Refactorings are supported for C++, C#, Java, and Slick-C®.

Available Refactorings (Pro only)

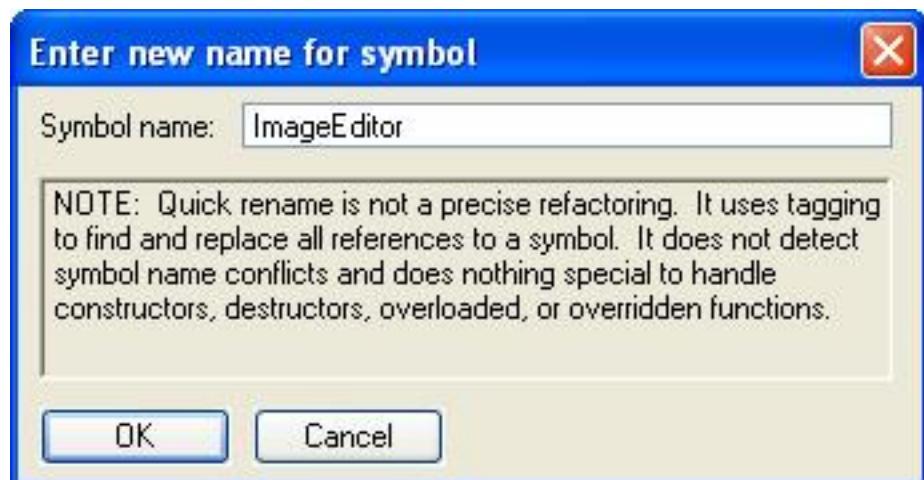
To access Quick Refactoring, from the main menu, click **Tools** → **Quick Refactoring**. Quick Refactoring menu can also be accessed from the right-click context menus in the **Symbols** and **Defs** tool windows.

Tip

Refactoring operations can modify more than one file. You can undo all of the refactoring modifications in one step by clicking **Edit** → **Undo Refactoring**.

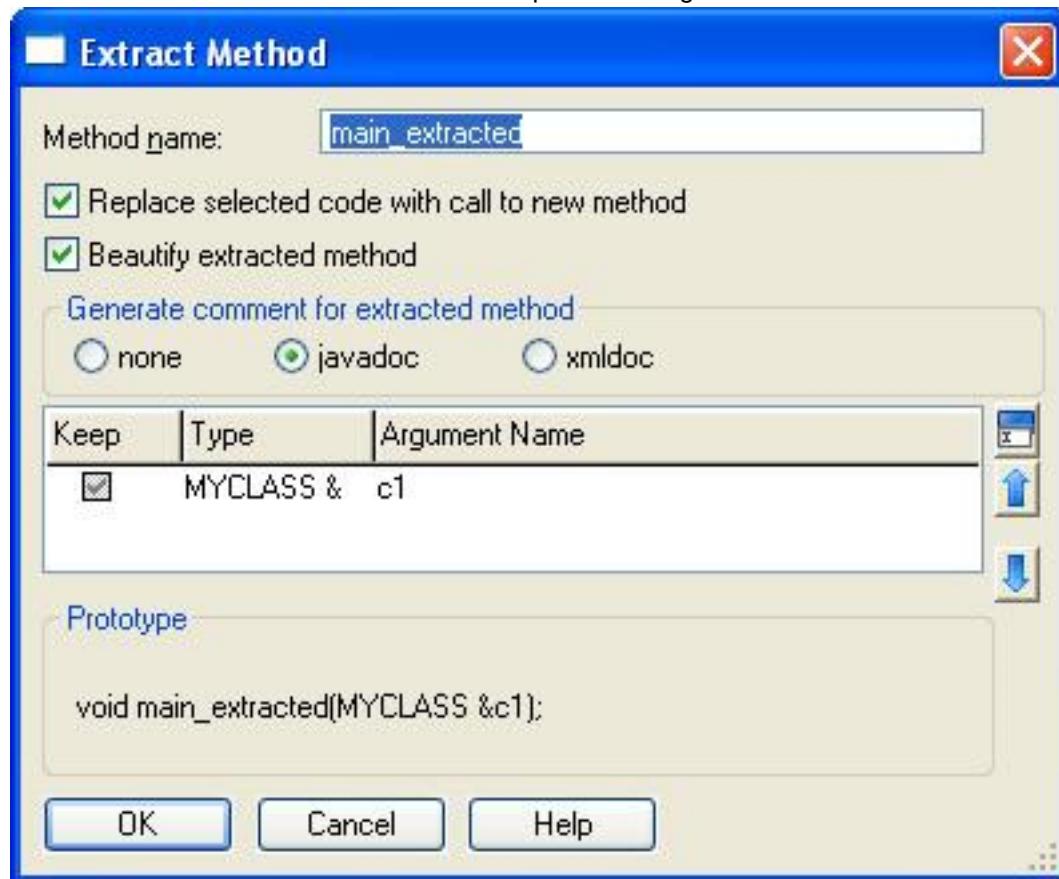
Quick Rename

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. 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 Modify Parameter List

This refactoring allows you to add, delete, and re-order parameters for a selected function. The

Reviewing Refactoring Changes (Pro only)

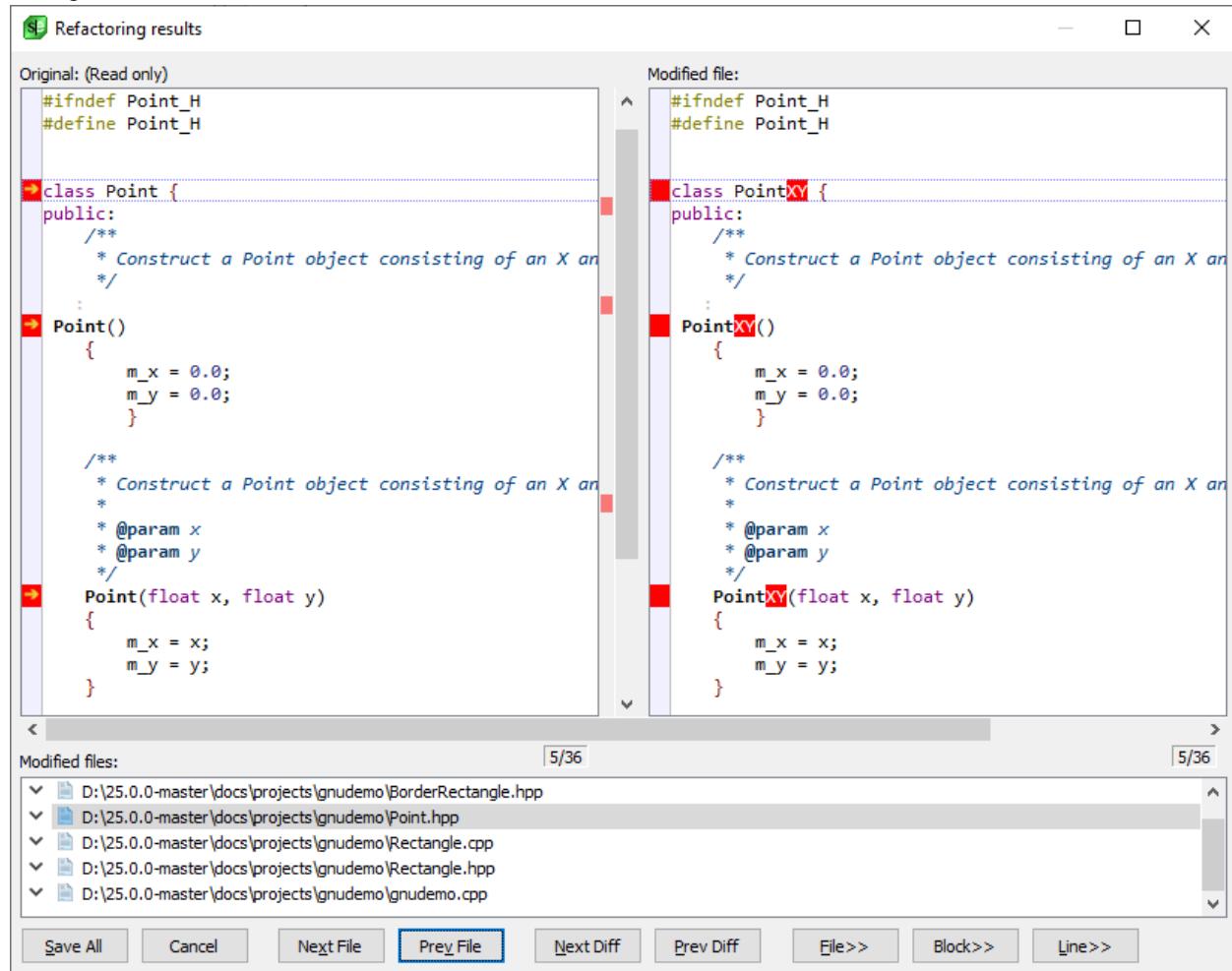
refactoring will modify the parameter list for the selected function and all of its counterparts within the class hierarchy.

Quick Replace Literal with Constant

Replaces the selected literal with a constant, replacing use of the literal with the new constant.

Reviewing Refactoring Changes (Pro only)

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.

Note

NOTE If you prefer to view the modified file on the left-hand side, there is an option to reverse the left and right panes. See [Refactoring Options](#) and [Search Options](#) for more information.

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 File** or **Prev File** to advance to the next or previous file in the list of files.

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 currently 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 [Quick Modify Parameter List](#), may require further user input to complete the refactoring operation. In this case each input request 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.

Language-Specific Editing

This chapter describes the language-specific editing features of SlickEdit.

Introduction to Language-Specific Editing

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-Specific Options](#) and [Language Editing Mode](#) below for more information.

Language-Specific Options

Options for language-specific features can be set through the Options dialog (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** or **config** command). A shortcut method to access language options for the current buffer is to use the **Document** → **[Language]** → **Options** menu item, or the **setupext** command. This will open the Options dialog to the **General** language-specific option screen for that language. See [Language Options](#) for more information.

Language Editing Mode

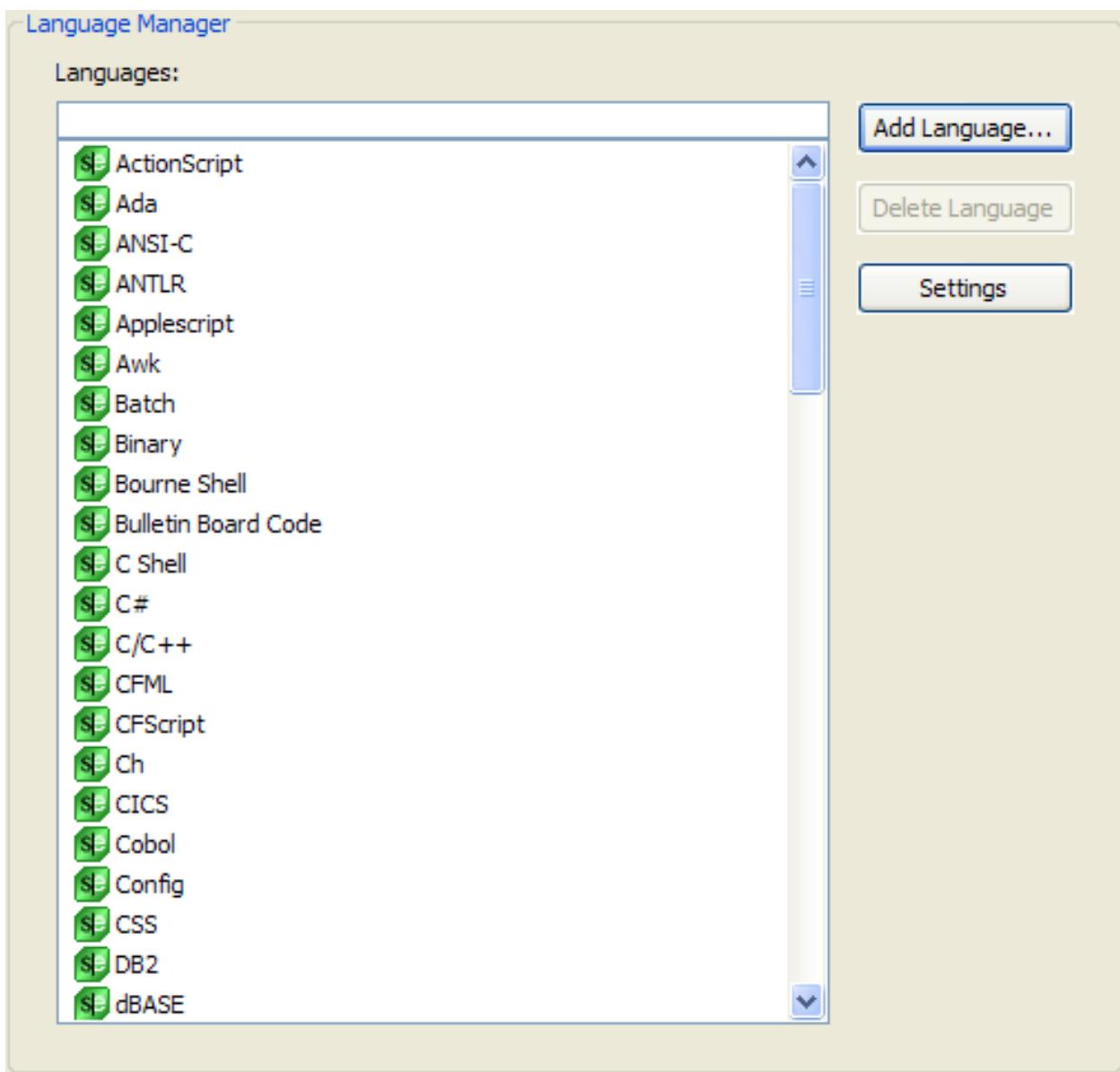
SlickEdit® uses the extension of the current file to determine what language you are using, thereby only making available the options and features that are possible in that language.

Manually Setting the Language Mode

If you have a file with a non-standard extension or no extension at all, you will need to manually specify the language editing mode. To specify a mode, from the main menu click **Document** → **Select Mode** (or use the **select_mode** command). The Select Mode dialog is displayed with a list of modes from which to select.

Managing Languages

Supported languages are listed in the Language Manager (**Tools** → **Options** → **Languages** → **Language Manager**). You can use this tool to add languages and delete languages you have added.

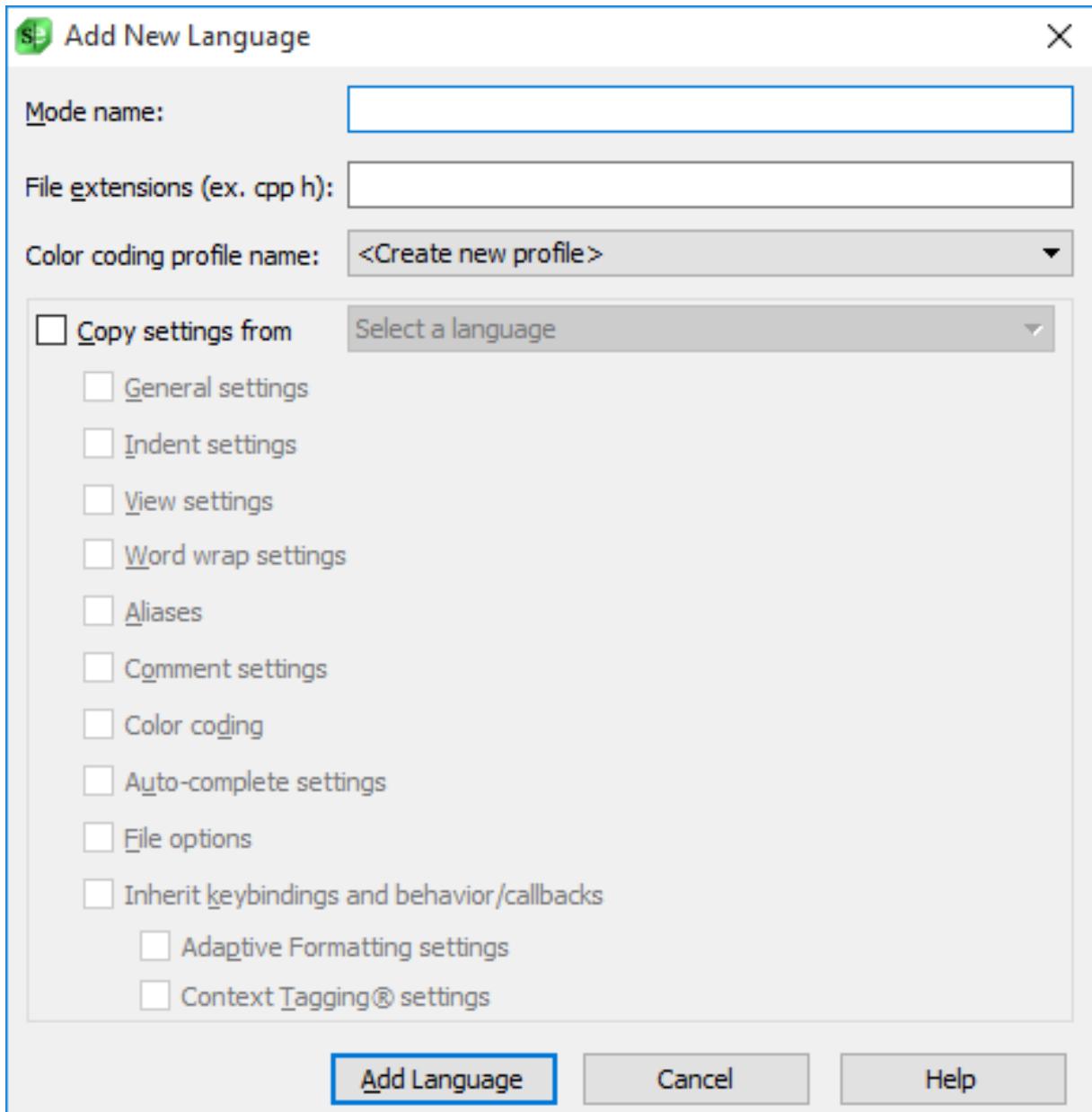


Installed languages are denoted in the list with a SlickEdit bitmap. Use the filter box at the top of the language list to search the list incrementally as you type. Use the **Add Language** and **Delete Language** buttons to add and remove languages (see [Adding and Removing Languages](#) below). Click **Settings** to jump to the [Language-Specific General Options](#) screen for the selected language.

Adding and Removing Languages

To add a language, complete the following steps:

1. Click **Add Language** on the **Language Manager** screen. The Add New Language dialog is displayed.



2. In the **Mode name** field, type a name for the new language (for example, C/C++, Java, etc.).
3. In the **File extensions** field, type the file extension(s) associated with this language. Separate each extension with a space, and do not include the dot character (for example: c h cc cpp). The extensions you list here, if not already defined, are added to the File Extensions list on the File Extension Manager. If you specify an extension that already exists and is associated with another language, a confirmation prompt is displayed.

Tip

To see a list of file extensions that are associated with a language, see the language-specific

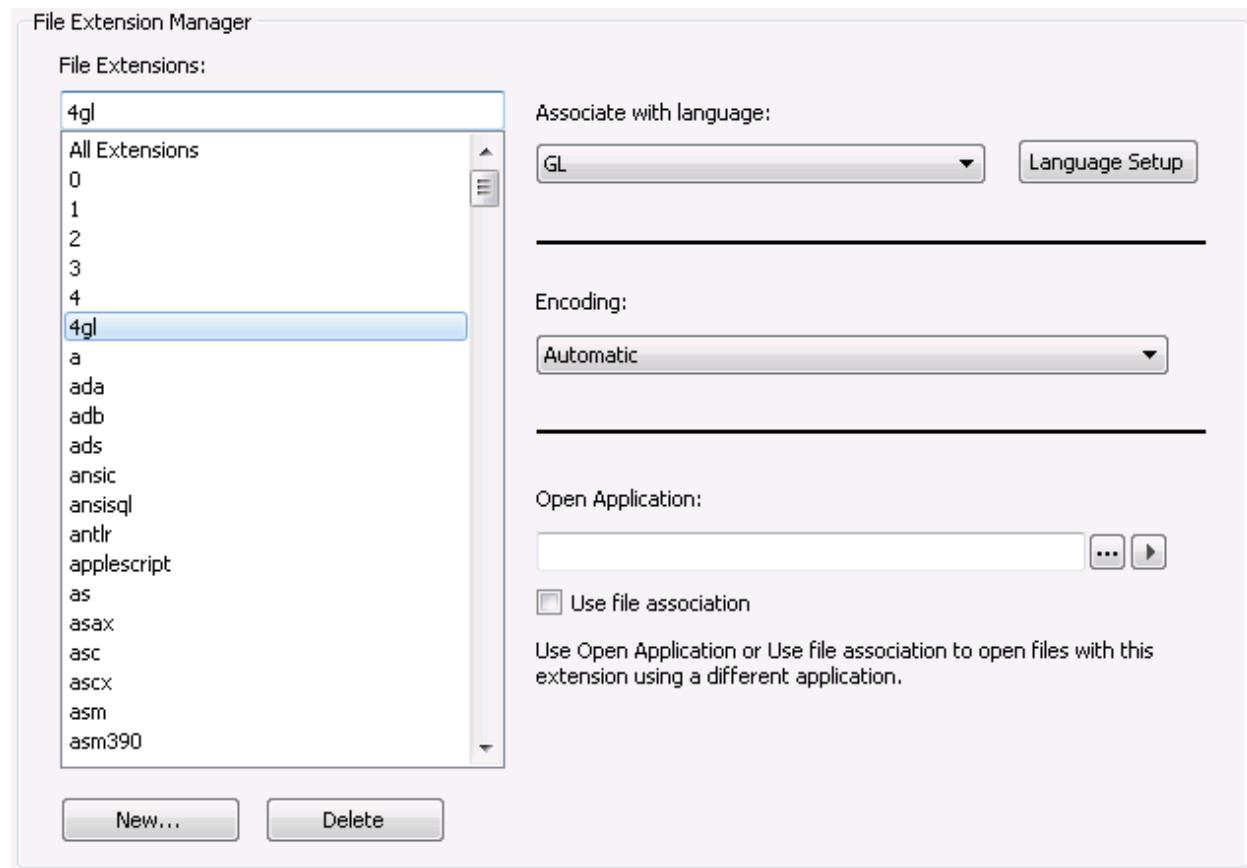
General options page (**Tools** → **Options** → **Languages** → [Language Category] → [Language] → **General**).

4. In the **Color coding profile name** field, specify the language identifier so that SlickEdit® knows what elements to color. Use the drop-down list to select the profile.
5. If you wish to copy settings from an existing language, check the **Copy settings from** checkbox and select the language from the combo box. Then check any boxes that correspond to settings you wish to copy from the selected language to your new language. These options are organized by their respective nodes in the options dialog. For more information, see [Language Options](#).
6. Click **Add Language**. The new language will be displayed in the Languages list as well as the list of document language modes on the Select Mode dialog (**Document** → **Select Mode**).

To delete a language you have added, select it in the Language list and click **Delete Language**. Installed languages cannot be deleted.

Managing File Extensions

The File Extension Manager (**Tools** → **Options** → **Languages** → **File Extension Manager**) is used to add and work with file extensions in SlickEdit®.



Recognized file extensions are listed in the File Extensions list. There is also an **All Extensions** item in the list to allow you to set certain settings for all known extensions. See [Adding and Removing File Extensions](#) for more information.

The settings on the File Extension Manager screen are described as follows:

- **Associate with language** - This drop-down shows the language that is associated with the selected file extension. Associations are created when you add a new language using the Language Manager. You can use this field to change the language association.

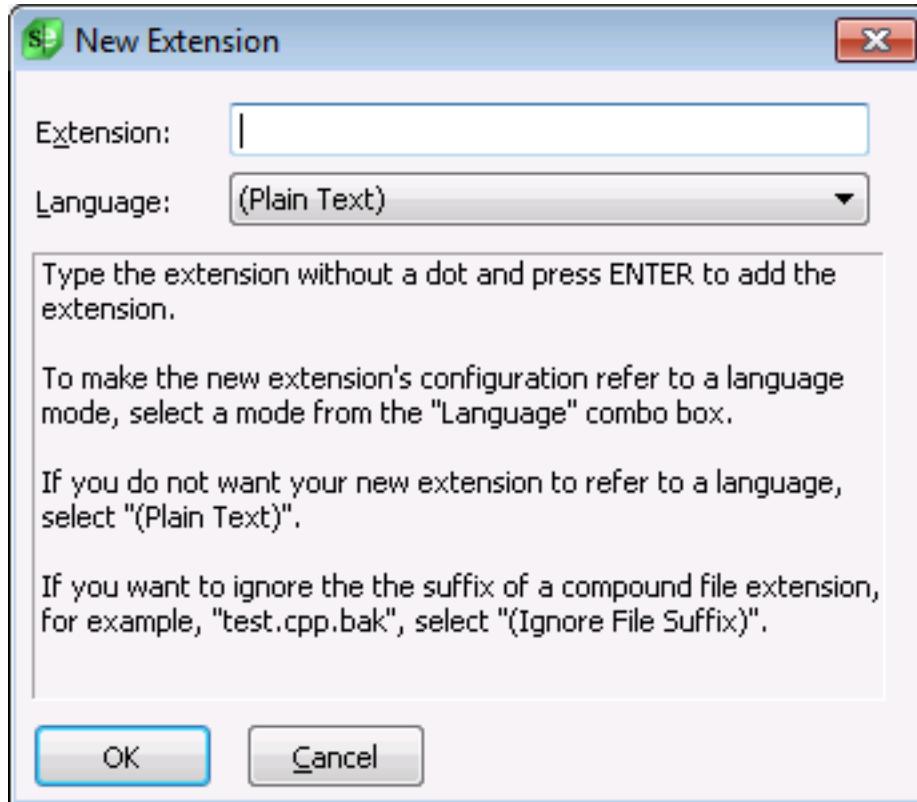
The "**(Plain Text)**" option at the top of the list is an easy-to-find alias for the default "**Plain Text**" mode for text files.

The "**(Ignore File Suffix)**" option instructs SlickEdit to ignore this file extension and attempt to map the filename to a language mode based on the rest of the file name. For example, if you have a file named "FileName.ada.orig", and ".orig" is set to "**(Ignore File Suffix)**", then SlickEdit will use "FileName.ada" to determine the language mode (e.g., **Ada**). This option is very useful to help SlickEdit intelligently select the document mode for backup copies of files.

- **Language Setup** - Click this button to jump to the language-specific **General** options screen, which shows a list of file extensions associated with the selected language and provides general language-specific options.
- **Encoding** - Each extension can have its own encoding specification. Both the language-specific and global option 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 on the [Load File Options](#) screen (**Tools** → **Options** → **File Options** → **Load**) is used. You can set the encoding for all extensions at once by selecting **All Extensions** in the **File Extensions** list and then selecting the encoding you want. Note that Unicode support is required to work with encodings. For more information about working with encodings and Unicode, see [Encoding](#).
- **Open Application** and **Use file association** - These options are mutually exclusive. **Open Application** is used to specify the application in which to open files when you use the Projects tool window to open them. Use the arrow to the right of this field to insert escape sequences that are used as arguments. On Windows, if the option **Use file association** is selected, the association specified in your operating system is used instead, overriding **Open Application**. When **Use file association** is selected, SlickEdit checks the Windows registry for the application associated with the selected file extension and invokes that application to display the file, rather than the one specified by **Open Application**. Note that **Use file association** is a Windows-only option. To open a selected file from the Projects tool window, double-click on it or right-click and select **Open** from the context menu.

Adding and Removing File Extensions

If SlickEdit® does not provide a file extension that you need to use, you can add it. If you have added a new language to SlickEdit with a file extension that was not already defined, the new extension is added to the File Extension list automatically. Or you can just add a new extension by clicking **New**, and the New Extension dialog is displayed.



Enter the new file extension in the **Extension** box (without the dot character), then select the associated language from the **Language** drop-down list and click **OK**. If the language does not exist, cancel this dialog and add it using the Language Manager first (see [Adding and Removing Languages](#)).

If you do not want your new extension to refer to a language, select "**(Plain Text)**" in order to treat the file as a plain text file.

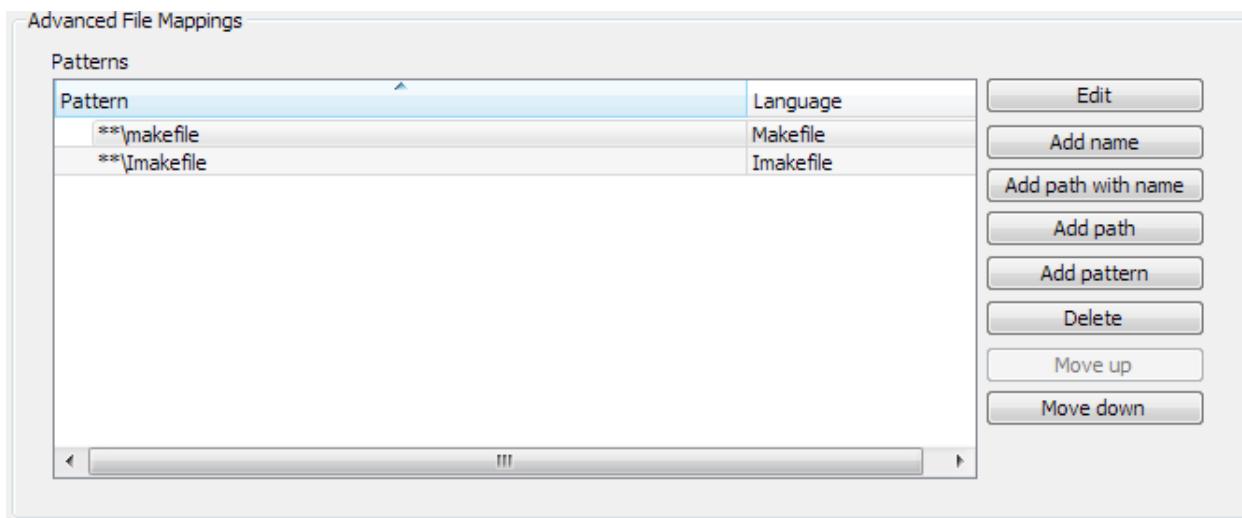
If you want SlickEdit to treat file files with the new extension as binary files (and display them in hex mode by default), select "**binary**".

If you want to ignore the suffix of a compound file extension, for example, "\test.cpp.bak\", select "**(Ignore File Suffix)**".

To delete the selected extension from the File Extensions list, click **Delete**.

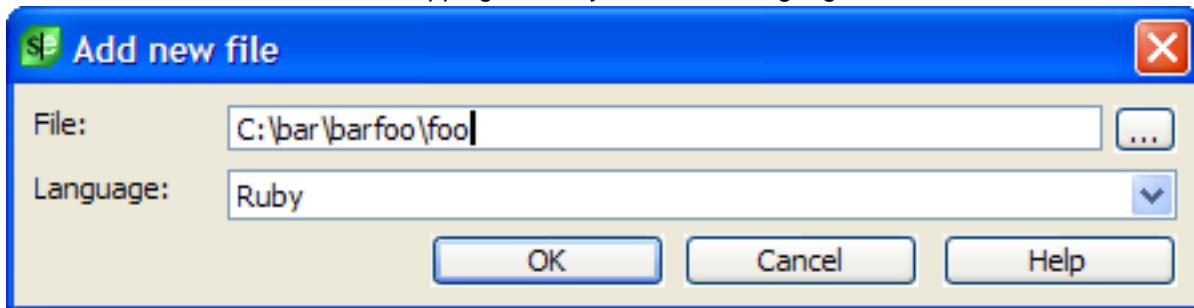
Managing Extensionless Files

When you open a file within the application, the extension is used to determine which language mode should be associated with the file. You can also use **Advanced File Mappings** to associate languages to files that do not have extensions or need to be recognized by location/filename. **Advanced File Mappings** can be accessed by going to **Tools** → **Options** → **Languages** → **Advanced File Mappings** and is shown below:



File Mapping

If you wish to select a specific file and map it to a language, add it to the **Files** list. Use the **Add** button next to the **Files** list to add a file mapping. Select your file and language and click **OK**.

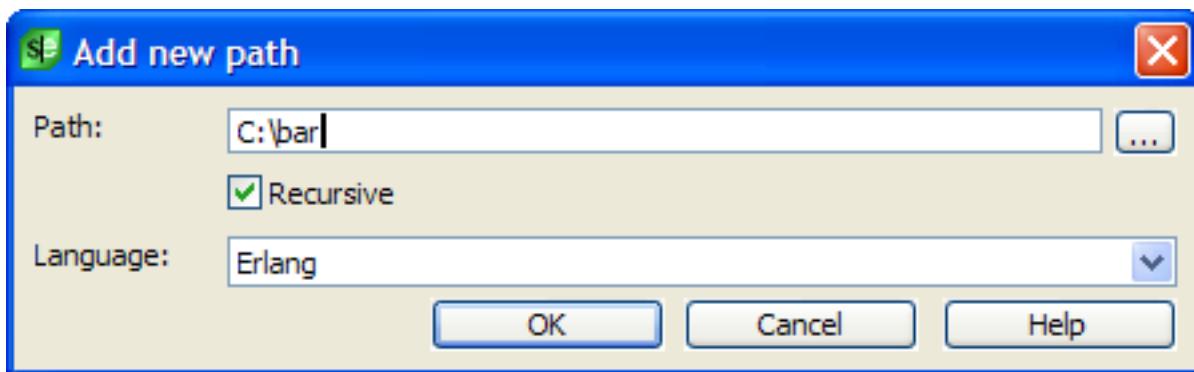


Now that file is treated as a file of the language you specify. You can edit or delete a file mapping using the appropriate buttons.

Pattern Mapping

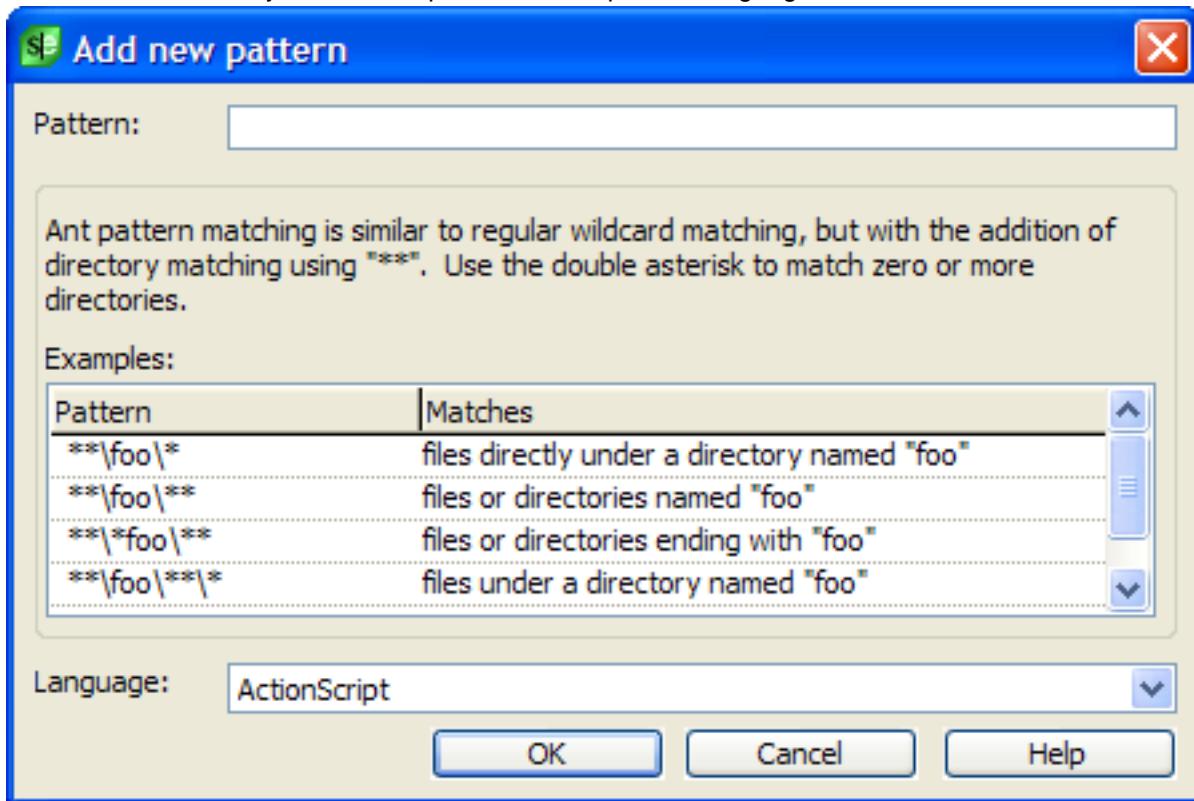
You can also specify mappings for files that match a certain pattern. For instance, if you want all files that are named "foo" or all files located in "C:\bar\" to be mapped to a specific language, you can set up a pattern mapping.

To create a pattern map that matches a filename, click the **Add filename** button. You can select a file using the ... button or you can simply type in a filename. You can use * as a wildcard in the filename. Select the language you want. When you click **OK**, the filename pattern will be added to the list. Notice that the filename will be translated into an Ant pattern that will match any file in any directory with that name.



To create a pattern map that matches a directory, click the **Add path** button. Select a path using the ... button or type one in. Use the **Recursive** checkbox to specify if you want all files below that directory or only files directly under it. Select the language you want and click **OK**. When the item is added to the list, it will be translated into an Ant pattern that matches files under that directory.

If you wish to create a more complex pattern than a simple filename or path match, use the **Add pattern** button. You can enter your own Ant pattern and map it to a language.



The mappings in the **Patterns** list can be re-ordered. When an extensionless file is encountered, the top mapping is checked first. If the file matches the pattern, then the associated language is used. If not, then the second pattern is checked. Each pattern is tested against the file until a match is found. Use the **Move up** and **Move down** buttons to reorder the patterns in order of desired precedence. Use the **Edit** and **Delete** buttons to do the respective actions.

Ada

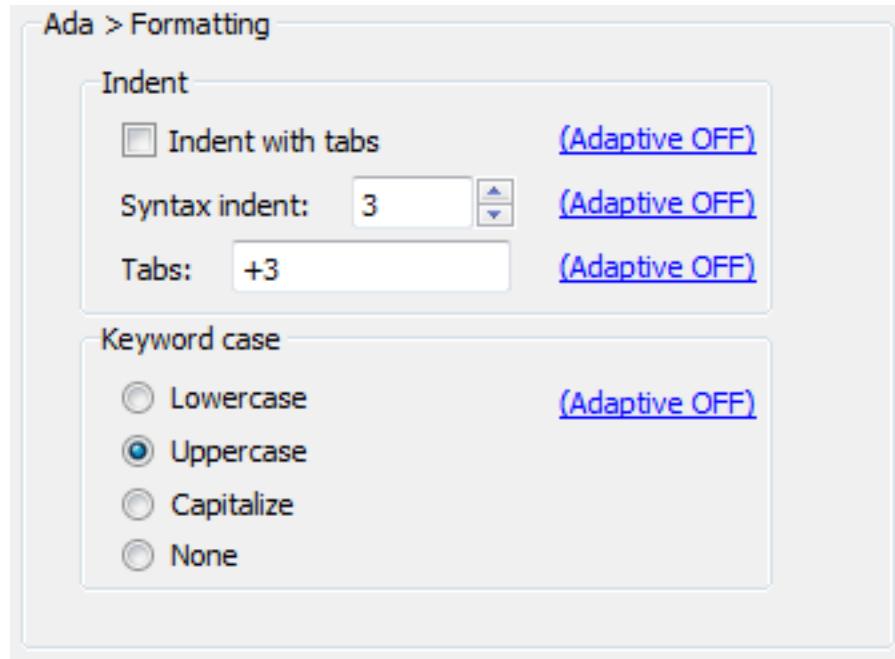
This section describes some of the options that are available for Ada.

Ada Formatting Options (Standard or Community only)

To access the Ada Formatting Options, from the main menu, click **Tools** → **Options** → **Languages**, expand **Application Languages > Ada**, then click **Ada Formatting Options**.

Note

Languages similar to Ada may have similar Formatting Options screens that are not specifically documented.



The following options are available for Ada:

- **Indent with tabs** - Determines whether **Tab** key, **Enter** key, and paragraph reformat commands indent with spaces or tabs. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Indenting with Tabs](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Syntax Indent](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.

- **Keyword case** - The **Keyword case** option specifies the case of keywords used by Syntax Expansion. For example, when you type the word "procedure" and the **Keyword case** is set to **Upper case**, the editor changes "procedure" to "PROCEDURE". The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.

Ada Beautifier (Pro only)

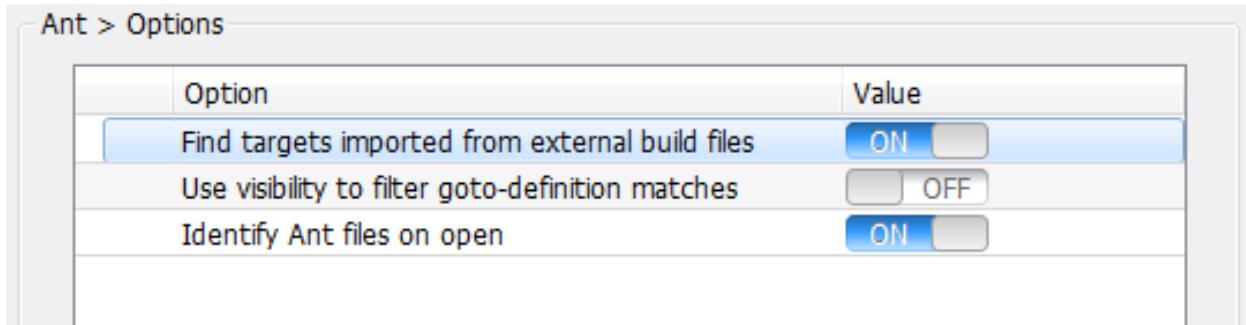
See [Beautifiers](#) for more information.

Ant

This section describes some of the features and options that are available for Ant.

Ant Options

There are several settings which are specific to editing Ant files. These options can be configured by going to **Tools** → **Options** → **Languages** → **XML/Text Languages** → **Ant** → **Options** and are pictured below.



The following options are available:

- **Find targets imported from external build files** - When set to **On**, SlickEdit will find targets imported into the selected file from other Ant files.
- **Use visibility to filter goto-definition matches** - When set to **On**, Ant goto-definition match results will be filtered based on visibility.
- **Identify Ant files on open** - When set to **On**, all XML files are parsed when opened to identify Ant build files.

C and C++

This section describes some of the advanced features and options that are available in SlickEdit® for C and C++, including language-specific formatting options, the C/C++ Beautifier, compiler settings, and preprocessing.

- [Working with ANSI-C](#) - Read this to configure SlickEdit what file extensions should be interpreted as ANSI-C vs C/C++
- [Beautifiers](#) - Detailed information about C and C++ beautifiers, as well as for related languages.
- [C/C++ Compiler Settings](#) - Information about C and C++ compiler configuration options.
- [C/C++ Advanced Options](#) - Information about C and C++ advanced Context Tagging® and parsing options.
- [C/C++ Preprocessing](#) - Information about how to configure C and C++ parsing to handle C preprocessing macros.
- [C/C++ Documentation Comments](#) - Information about using Javadoc and XMLDoc comments in C and C++.
- [Adding #includes](#) - Describes how to automate adding a #include dependency while editing C and C++ source code.

Working with ANSI-C

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.

Beautifiers (Pro only)

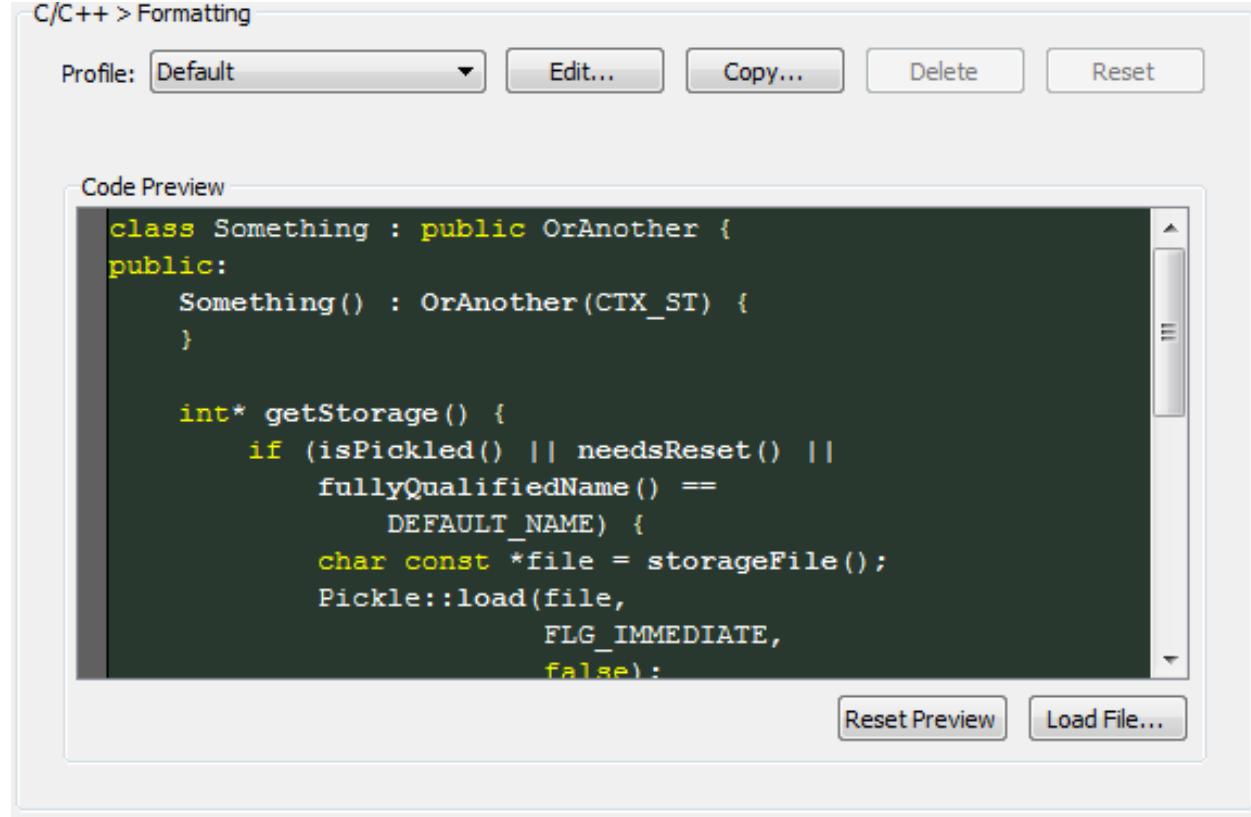
Most of SlickEdit's beautifiers have been updated to allow more control over source formatting details, and to allow formatting settings to be grouped into profiles for easier management over multiple projects.

You can use the commands **beautify** or **beautify_selection** to instantly beautify the file or the selection according to the settings on the Beautifier dialog.

In addition to being an on-command beautifier, the updated beautifiers can also format your text as you type. You can control when the beautifier will be automatically invoked as you code by setting the beautifier-related options found on the [Language-Specific General Options](#). See [Editing Beautify Options](#) for more information.

Beautifier Profiles

The C++, Object-C, Java, C#, Python, JavaScript, VBScript, PHP, HTML, XML Formatting, SystemVerilog, Verilog, Groovy, Ada, and Slick-C options allow you to pick which formatting profile you want to be in effect, edit or delete existing profiles, and create new profiles. To access these Beautifier settings, go to **Tools** → **Options** → **Languages** → [Language Category] → [Language] → **Formatting**.



The Formatting page has the following controls:

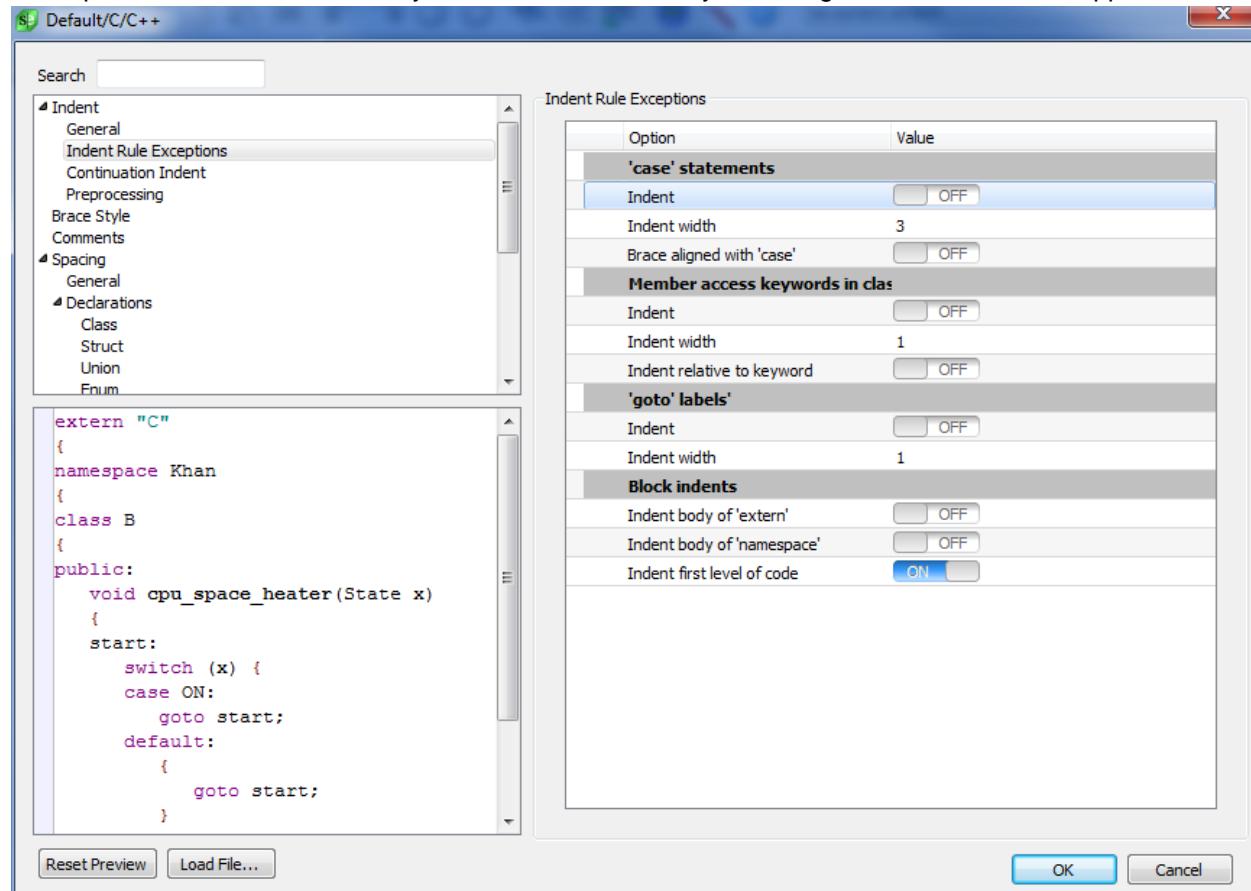
- **Profile Combo Box** - allows you to select which beautifier profile is in effect. The preview window below the combo box will show how the profile would beautify a snippet of code. Once you've selected a profile, and hit OK, the profile's settings are used for both formatting as you're editing code, and as the default profile to use for the language if you beautify the source using the **beautify** command, or by going to **Tools** → **Beautify**.
- **Edit...** - Allows you to edit the settings of an existing profile. Profiles that shipped with the system are read-only, but will allow you to save modified versions under a different name. Clicking this button will take you to the [Beautifier Profile Editor](#).
- **Copy...** - Creates a copy of the currently selected profile, after prompting you for a name. This is how

you can create new profiles, by selecting a profile that's closest to the formatting that you want, and creating a copy of it that you can modify.

- **Delete** - Deletes a profile. Profiles that shipped with the product can not be deleted.
- **Reset** - Clears changes to a built-in profile. This button is only enabled if changes have been made to a built-in profile.
- **Load File** - loads a different file as the example code snippet in the preview window.
- **Reset Preview** - Resets the contents of the preview window back to the default code snippet.

Beautifier Profile Editor

The profile editor allows you to change the formatting options for a beautifier profile. Every editor page has a preview window that allows you to see the effects of your changes on source code snippets.



Most options have an **Enabled** checkbox to the left of the option description. If the checkbox is cleared, the option is disabled, which means the beautifier will leave the source code normally targeted by the option unchanged. As an example, there's an option for padding the parenthesis of an 'if' statement. Assuming it's enabled, it will either force padding in all 'if' statements, or removes the padding from all if statements. If you disable it by clearing the checkbox, then the padding for if statements will be left alone, leaving whatever type of padding that already exists in the original source.

The following controls are available on the Beautifier Profile Editor:

- **Search** - Like the options dialog, there are a lot of settings, the search box allows you to type in search terms to only show options that match the search term.
- **Load File** - Allows you to load a different example file into preview window.
- **Reset Preview** - Resets the preview window back to the default example code snippet.
- **Beautify** - This button will only appear if the profile editor was launched from a menu or button. Clicking this button will beautify the active buffer in the editor with the settings from the profile editor.

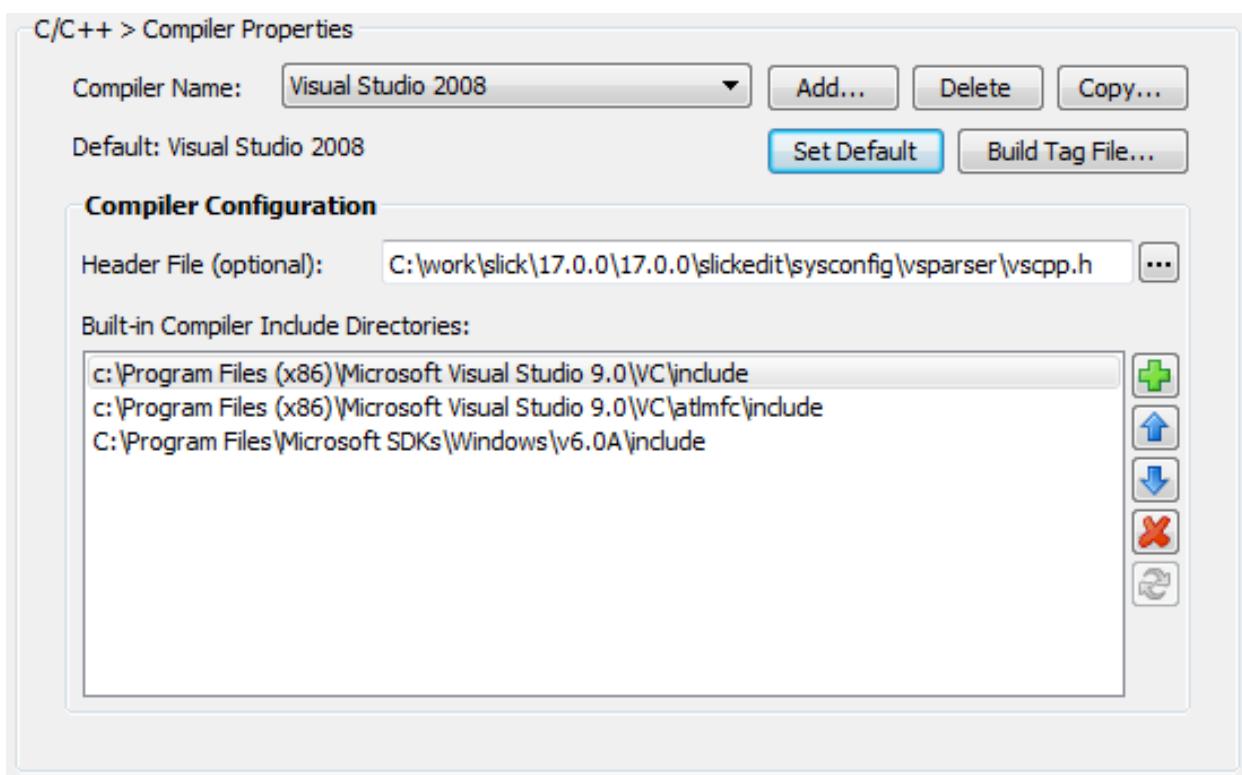
C/C++ Compiler Settings (Pro only)

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.

These properties can be specified using the C/C++ Compiler Properties options screen or the C/C++ Compiler Properties dialog. The interfaces contain the same fields and options so you can make changes using the one you prefer:

- From the main menu, click **Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Compiler Properties**.
- With a C/C++ project open, from the main menu, click **Project** → **Project Properties**. Select the **Compile/Link** tab, then click the **Ellipsis** button to the right of the **Compiler** combo box. The C/C++ Properties dialog is displayed.

C/C++ Compiler Settings (Pro only)



The interface shows the default compiler and its associated header file and include directories, known collectively as 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++ Advanced Options

The C/C++ advanced parsing options allow you to fine-tune the behavior of the C/C++ parser in order to better handle parsing your code. The C/C++ advanced Context Tagging® options allow you to enable advanced editing and symbol completion features specific to C/C++.

- **Parsing Options**
 - **Tag function prototypes** - Tag prototypes (function declarations) in C/C++ style code. We recommend leaving this option on, otherwise you will not be able to navigate to function declarations.
 - **Tag function prototypes with no semicolon** - Do not skip over function definitions/prototypes that do not have a semicolon or open brace following the parameter list. Set this option when you do not want old C-style function definitions skipped.
 - **Tag function prototypes with no return type** - Do not skip over old style C/C++ prototypes that do not have an explicit return type. This should be off in order for local-variable search to work properly, otherwise it is very difficult to distinguish a function call from a prototype.
 - **Treat close brace in column 1 as function end** - Do not break out of parsing a function if we see a brace in column 1. We normally do this as a safeguard against parsing past the end of a function when the braces mismatch.
 - **Ignore stray identifiers that may be preprocessing** - Ignore stray identifiers that may be preprocessing. This only works in very specific parsing contexts.
 - **Tag Visual C++ bracketed attributes** - Tag Visual C++ bracketed attributes in C++ code.
 - **Dynamically expand local #defines** - Dynamically expand local #defines within the current file as if they were defined in the C/C++ Preprocessing options. Note that this feature only works within the current file and does not work for local variable tagging. It can also have the side-effect of dumbing down code by replacing preprocessing constant names with their values in certain contexts. This feature is off by default because it can negatively effect performance for files that contain extremely large amounts of preprocessing.
 - **Dynamically expand project #defines** - Dynamically expand preprocessing defines in the project properties for the current project and build configuration, as if they were defined in the C/C++ Preprocessing options. Note this feature gets defines from the current project and build configuration only. If your workspace contains multiple projects, when you switch projects or build configuration, changes to preprocessing options will not be fully in effect until you rebuild your workspace tag file. On account of this, this feature is a partial solution, off by default, designed to be helpful, but not

comprehensive, without having a negative impact on performance.

- **Unscoped nested structs and unions (as in ANSI-C)** - When working with mixed source (ANSI-C and C++), there may be ANSI-C headers which are treated as C++. This setting allows you to override the nesting of structs, unions, and enumerated types so that they are seen globally as they are scoped in ANSI-C and K&R C code.
- **Editing Options**
 - **Auto-Correct '.' to '->' for pointers** - When you type '.' after a variable which evaluates to a pointer type, automatically translate '.' to '->' to dereference the pointer type.

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
```

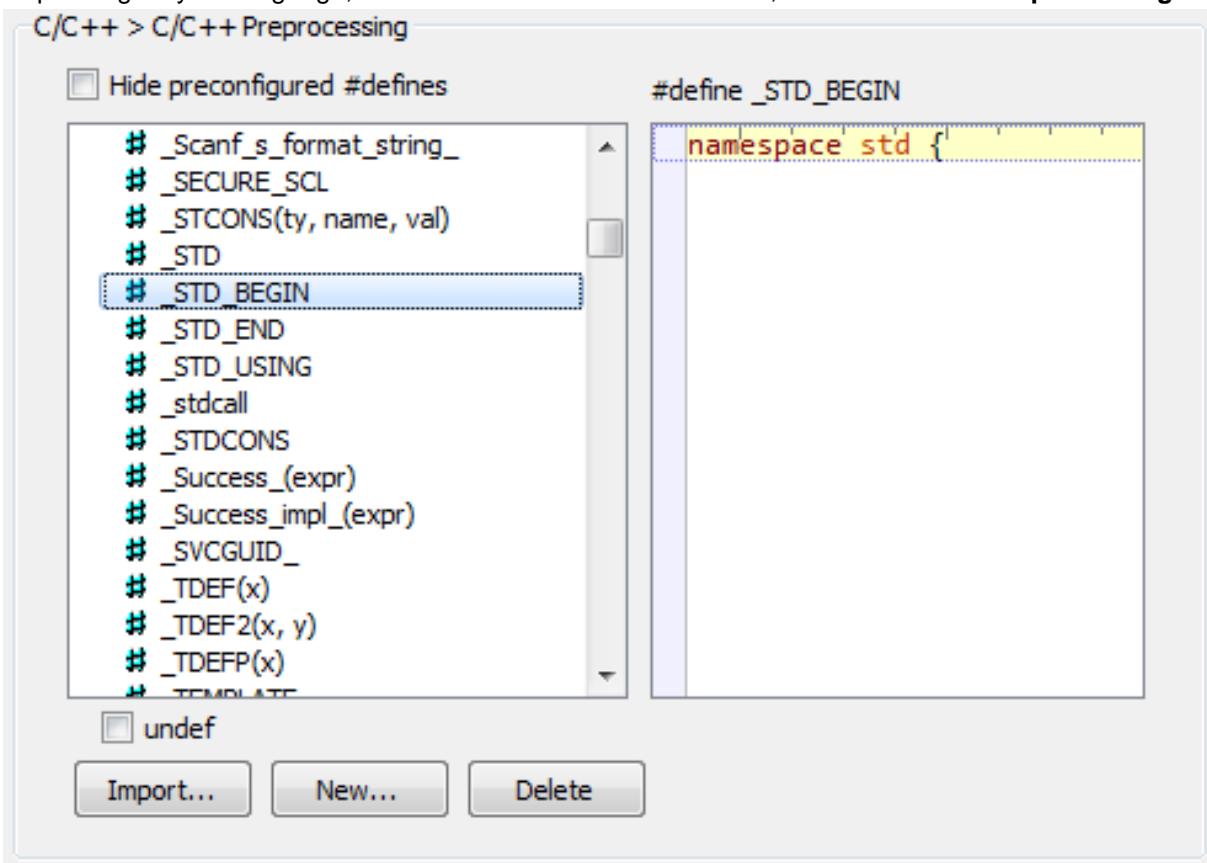
This example uses the following preprocessor macros:

```
#define MYNAMESPACEDECL(name) namespace name {
#define MYPACKEDMACRO __packed
#define MTYPELESS 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, click **Tools** → **Options** → **Languages** and expand the **Application Languages** node in the tree.

2. Depending on your language, select **ANSI-C** or **C/C++** in the tree, then click **C/C++ Preprocessing**.



3. Click **New** to add new preprocessing macros. Arguments are allowed; for example, **mymacro(a,b,c)**

4. When finished, click **OK**.

5. A prompt appears asking whether to rebuild your workspace tag file. Click **Yes**.

Preprocessor macros are stored in `usercpp.h`, (`unxcpp.h` on Unix and macOS), 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` or `unxcpp.h` file should only be used for #defines and #undefs not #includes.

Each workspace may have a `[workspace]_cpp.h` file which can be used for the same purpose as `usercpp.h` or `unxcpp.h`, except that the configuration is for the corresponding workspace only. This file may be edited using the **C/C++ Preprocessing** dialog accessible from **Project → Workspace Properties**.

C/C++ Documentation Comments

Several features are available to help you enter and format Javadoc and XMLDoc comments. See [Doc Comments](#) for more information.

Add #include (Pro only)

Adding #include

To add a #include statement for a symbol name under the cursor in C, C++, Objective-C, or Slick-C code, move the cursor to the symbol name you want to import, then from the main menu, click **Tools** → **Imports** → **Add #include**, or from the right-click context menu, select **Imports** → **Add #include**. This feature works for types (classes, structs, etc), functions, and constants. Alternately, use the **refactor_add_import** command. This command can also be used to generate using statements for symbols imported from namespaces.

To jump to the #include statement for the symbol name under the cursor, move the cursor to the symbol name, then from the main menu, click **Tools** → **Imports** → **Go to #include**, or from the right-click context menu, select **Imports** → **Go to #include**. Alternately, use the **refactor_goto_import** command. This command is helpful for identifying which package an unqualified class name comes from.

Note

C and C++ #include dependencies can be very delicate. The Add #include feature works by finding the location of a symbol's declaration in a header file and attempts to add the specific header file physically containing the symbol's declaration to the list of #include dependencies for the current file. For very cleanly written code, this strategy will generally work very well, however, it should be recognized that SlickEdit will occasionally select a header file to include which isn't the entry point that you would want. This can happen when there are header file wrappers or if you have a more general header file you prefer to #include rather than the individual files where a symbol is actually declared. Furthermore, many header files can not stand on their own because they depend on declarations pulled from other header files which are typically included first in the dependency chain. These and many other reasons mean that Add #include may or may not produce useful results in your code. Always verify the results when using it in C and C++ code.

Note

The Add #include feature will generate #include statements using double quotes for header files that are part of your project, and angle brackets for symbols coming from system header files. This is best practice for C and C++ code, thus there is no option to generate the #include statements differently.

COBOL

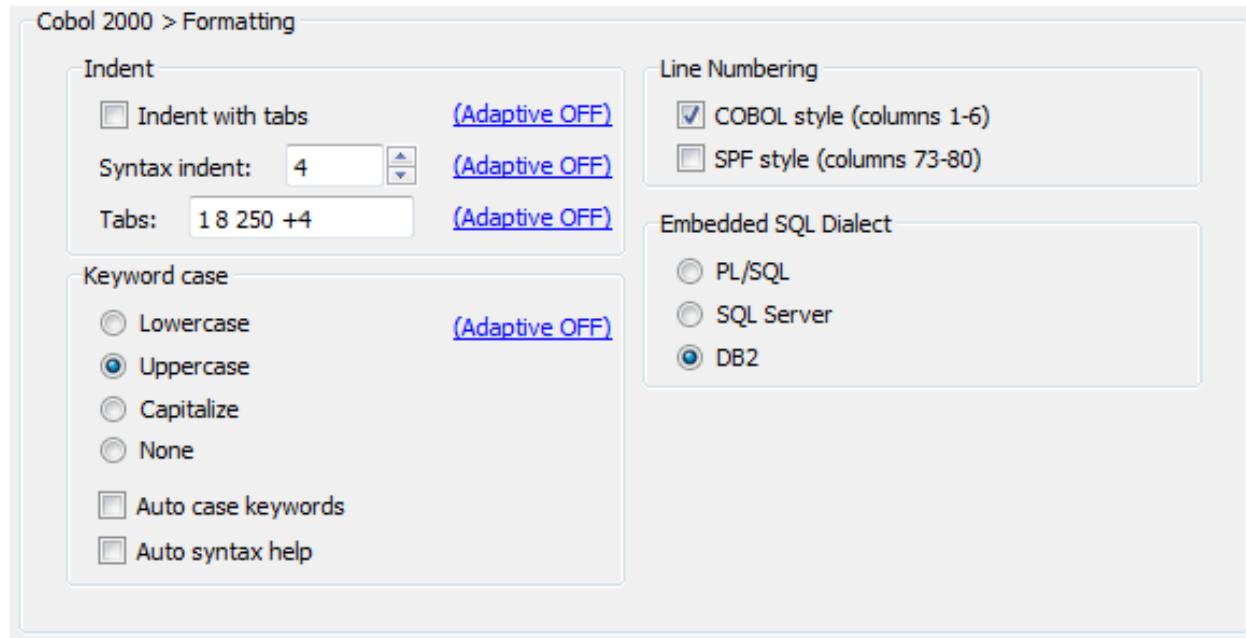
This section describes some of the advanced options that are available for COBOL.

COBOL Formatting Options

Options are available for COBOL for changing [Syntax Indent](#) and [Syntax Expansion](#) styles. To access these options, from the main menu, click **Tools** → **Options** → **Languages**, expand **Mainframe Languages > Cobol**, then click **Cobol Formatting Options**.

Note

Languages similar to COBOL may have similar Formatting Options screens that are not specifically documented.



The following options are available:

- **Indent with tabs** - Determines whether **Tab** key, **Enter** key, and paragraph reformat commands indent with spaces or tabs. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Indenting with Tabs](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Syntax Indent](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.

- **Keyword case** - Specifies the case of keywords used by Syntax Expansion. 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". The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **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.

Java

SlickEdit® provides a full-featured Java development environment, allowing you to edit, build, and debug Java programs. Topics in this section:

Note

NOTE There are numerous compilers which are compatible with the JVM environment, including compilers for Ruby (JRuby), Clojure, Groovy, Python (Jython), Kotlin, Scala, JavaScript, COBOL, Common Lisp, Cypher, Mercury, Oberon, Pascal, PHP, Prolog, Perl 6, R, Rexx, Scheme, Smalltalk, Tcl, and Visual Basic. Many of these can be used with SlickEdit® by customizing a **Java - Other** project to use the compile and build commands required by the given compiler. See [Creating Custom Project Types](#) for more information.

- [Initial Setup](#) - Read this to configure SlickEdit for your JDK and other settings needed for compiling and debugging.
- [Java-Specific Features](#) - Information about features designed specifically for Java programmers.
- [Java-Specific Interfaces](#) - Detailed descriptions of dialogs, tool windows, and option screens specific to Java programming.

Initial Setup (Pro only)

SlickEdit® relies on an installed Java Development Kit (JDK) for compiling and debugging. After you have installed the JDK on your computer, the following steps will configure SlickEdit to use it. The steps are divided into three categories:

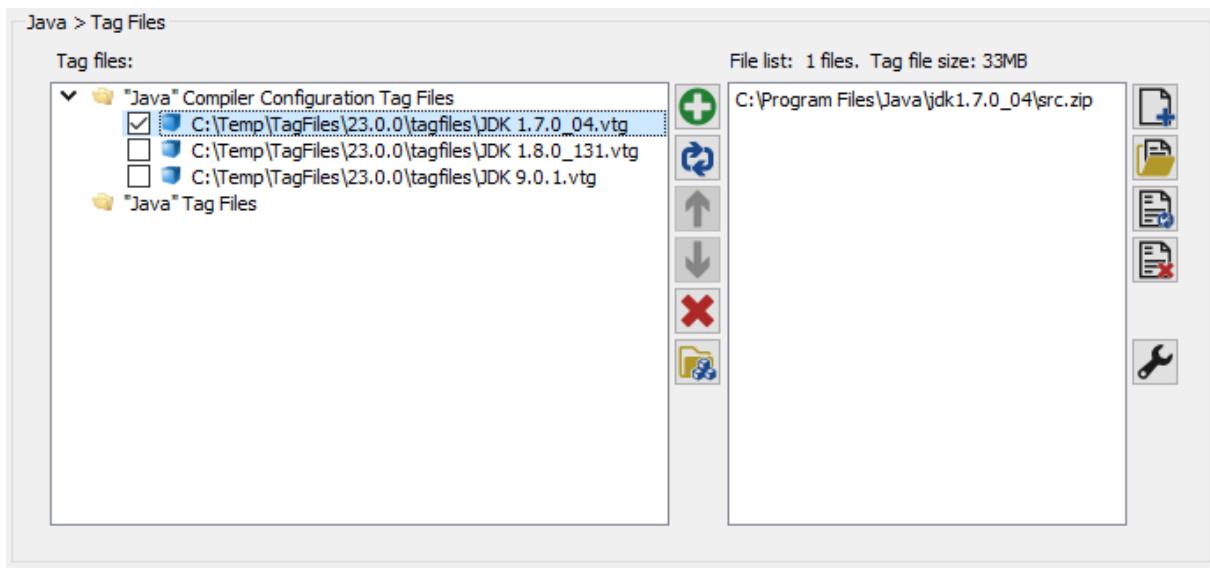
- [Context Tagging® for Java](#)
- [Setting Up a Java Workspace and Project](#)
- [Configuring Java Build and Runtime Options](#)

Context Tagging® for Java (Pro only)

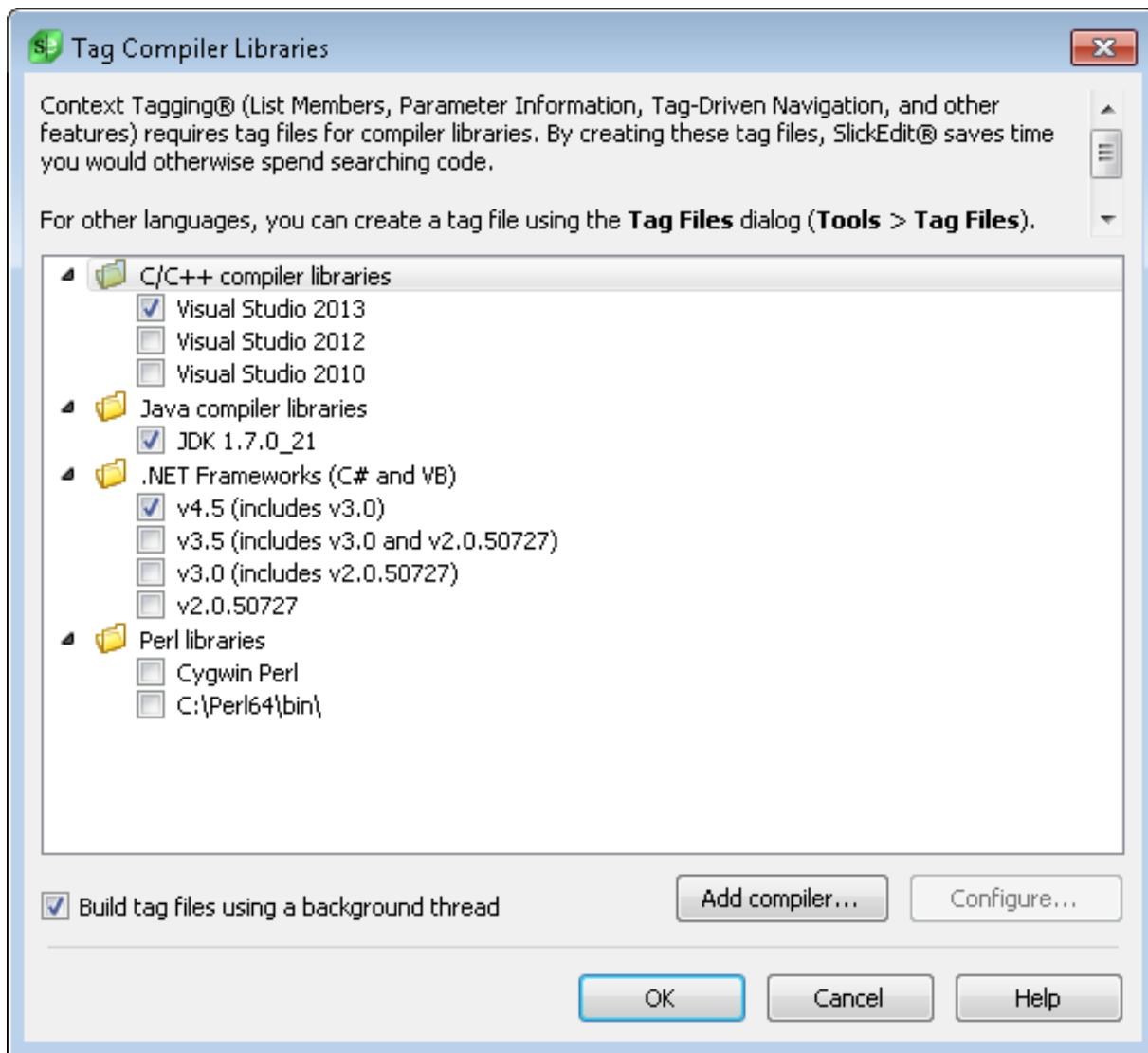
SlickEdit® needs to tag the Java libraries to provide symbol completions and other Context Tagging features for those classes (see [Context Tagging Features](#)). When you first run SlickEdit after an installation, you are prompted with a dialog to create these tag files. Complete the steps below if you did not create tag files at that time or to configure additional JDKs.

1. Open the [Context Tagging - Tag Files Dialog](#) by selecting **Tools** → **Tag Files**.

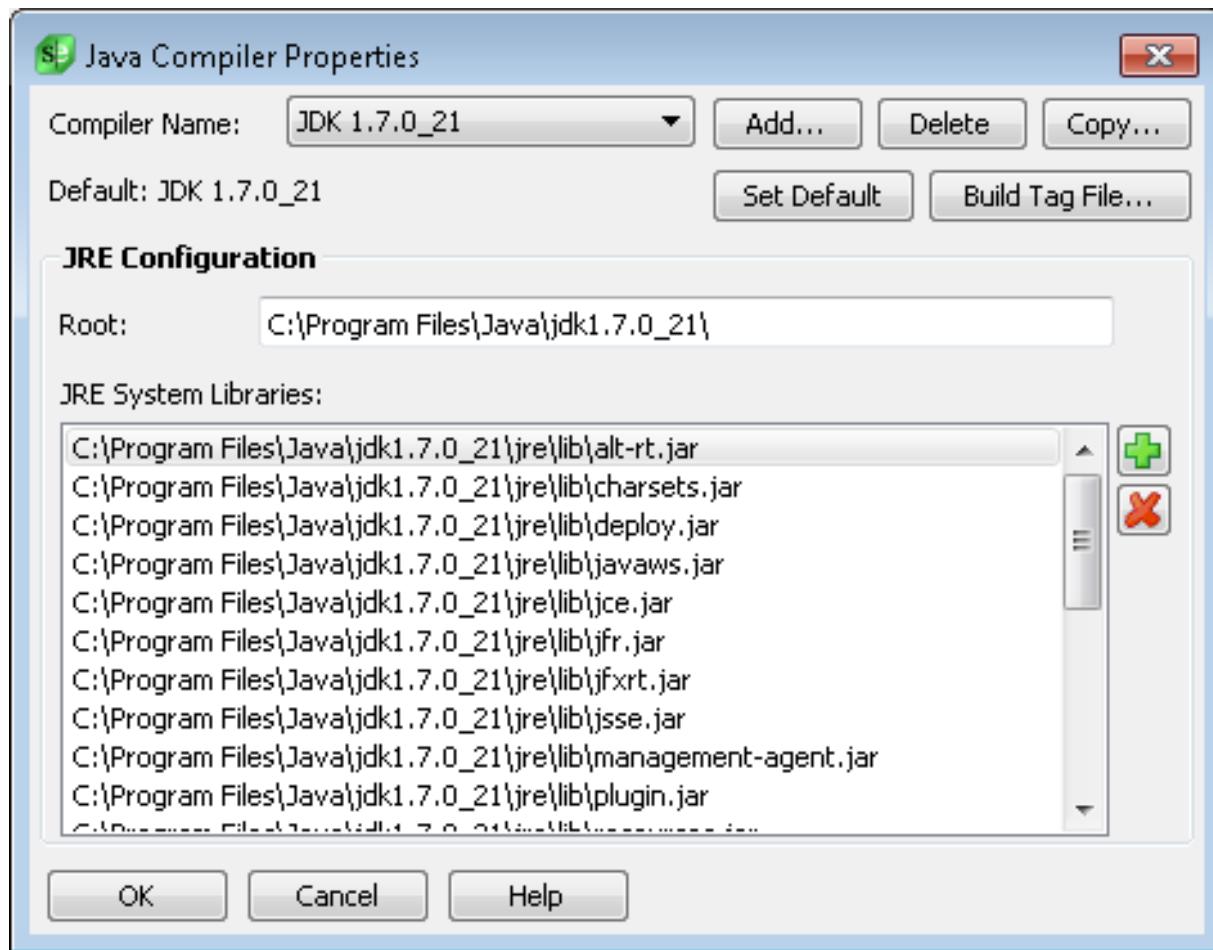
Initial Setup (Pro only)



2. Click the **Auto Tag** button to open the Tag Compiler Libraries dialog.



3. SlickEdit may detect that you have installed the JDK. If so, the section for Java will be filled out. If not, you will have to configure this manually.
4. Click the **Configure** button to open the **Java Compiler Properties Dialog**. You can have multiple JDKs installed on your computer at the same time and configure SlickEdit to use different JDKs for each project. This dialog provides the name and location for each JDK so that you can select it for tagging or building.



5. Click the **Add** button to browse to the root of the desired JDK (or JRE). If SlickEdit recognizes the Java vendor and version, it will automatically set the appropriate properties. If not, you will be prompted for the configuration name. Give it a name that represents the associated JDK, like "JDK 1.6".
6. If a default JDK has not been specified, click the **Set Default** button to set this JDK as the default.
7. When finished, click the **OK** button to return to the auto tag dialog.
8. Make sure there is a check in the **Create tag file for Java compiler libraries** check box. Depending on your environment, there may be checks in the check boxes for C++ and .NET. Leave those checked if you have not already tagged those libraries. If you just want to tag the Java libraries, uncheck the other check boxes.
9. Click the **Create tag file(s)** button.
 - 1 SlickEdit will display a progress bar while your libraries are being tagged. When finished, SlickEdit will
 - 0 display the Context Tagging - Tag Files dialog. You can close this if you have no other libraries to tag.

Setting Up a Java Workspace and Project (Pro only)

In SlickEdit®, files are contained in projects, and projects are contained in workspaces. Except for the

most basic editing, you should always work within a workspace and project in SlickEdit. Context Tagging® relies on having your source files contained in a project. See [Workspaces and Projects](#) for more information.

Editing options are determined by the file extension and accessed by selecting **Tools** → **Options** from the main menu, then selecting the corresponding language in the options hierarchy. Editing options control how your code is formatted and key editing behaviors as you type.

The project type determines your build environment and provides options specific to that project type. For Java, this includes specifying which JDK to use, setting up the debugger, and configuring [Java Live Errors](#). To create a new Java project or to see a list of the available Java project types, select **Project** → **New** from the main menu, then expand the Java node in the tree under **Project type**.

After you have created a Java project, you can set the build options by selecting **Project** → **Project Properties** from the main menu, selecting **Build** in the **Tool name** list, then clicking the **Options** button. The options displayed are specific to the project type of the active project. If you are in a Java project, you will see Java options. If you are in a C++ project, you will see options for the C++ compiler.

Note

The **Options** button is only available if you have selected **Build without a makefile (dependencies automatically checked)** on the **Build** tab of [Project Properties Dialog](#). If you select either of the other two options, SlickEdit uses an external command to launch the build. The **Options** button is not available in **Java - Other** projects.

You can change the build options for Java by selecting **Build** → **Java Options** from the main menu (or the corresponding item for other languages).

You can change the execution and debugging options for Mono by selecting the **Run/Debug** tab on the **Project** → **Project Properties** dialog.

Configuring Java Build and Runtime Options (Pro only)

The Java Options dialog contains settings used when you build or execute a Java project. Most of the settings are stored for the particular Java project and configuration selected. You can set different values for different projects and for different configurations of the same project. For example, you might have different settings for the Debug configuration than from the Release configuration, allowing you to turn on optimizations used for release that are incompatible with debugging. See [Java Options Dialog](#) for more information.

Java-Specific Features

SlickEdit® provides many features that work across several languages including Java, and Java-specific information is described throughout the documentation where applicable. The following are additional features designed specifically for Java developers:

- [Javadoc Comments](#)

- [Organize Java Imports](#)
- [Java Live Errors](#)
- [JUnit Testing](#)

Javadoc Comments

Several features are available to help you enter and format Javadoc comments (as well as other doc comment formats). See [Doc Comments](#) for more information.

Organize Imports (Pro only)

Organize 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, click **Tools** → **Imports** → **Organize Imports**, or from the right-click context menu, select **Imports** → **Organize Imports**. Alternately, use the **refactor_organize_imports** command.

Adding Imports

To add an import statement for the class name under the cursor in Java code, move the cursor to the class name you want to import, then from the main menu, click **Tools** → **Imports** → **Add Import**, or from the right-click context menu, select **Imports** → **Add Import**. Alternately, use the **refactor_add_import** command.

Go to Import

To jump to the import statement for the class name under the cursor in Java code, move the cursor to the class name, then from the main menu, click **Tools** → **Imports** → **Go to Import**, or from the right-click context menu, select **Imports** → **Go to Import**. Alternately, use the **refactor_goto_import** command. This command is helpful for identifying which package an unqualified class name comes from.

Import Options

Several options are available on the Options dialog to control the behavior of Organize Imports. See [Organize Java Imports Options Interface](#) for details.

Java Live Errors (Pro only)

Java Live Errors is a feature that 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.

To activate Live Errors, open a Java project then complete the following steps:

1. Open the Java Options dialog by selecting **Build** → **Java Options** from the main menu.

2. Select the **Compiler** tab and enter the **Source Compliance level**.
3. Select the **Live Errors** tab and check **Enable Live Errors**.
4. In the **Path to JDK 6 or later** field, specify the root of the JDK 6 (or compatible JDK) installation. There is no requirement that you build your code with JDK 6, only that it is available to Live Errors.

Note

If you have Live Errors running and wish to specify a different JDK 6 (or other compatible JDK) root, after changing the path on the Java Options dialog, you must restart SlickEdit®.

After activating 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 (see [Creating Bindings](#)).

See [Live Errors Tab](#) for more information about the fields and options on this tab.

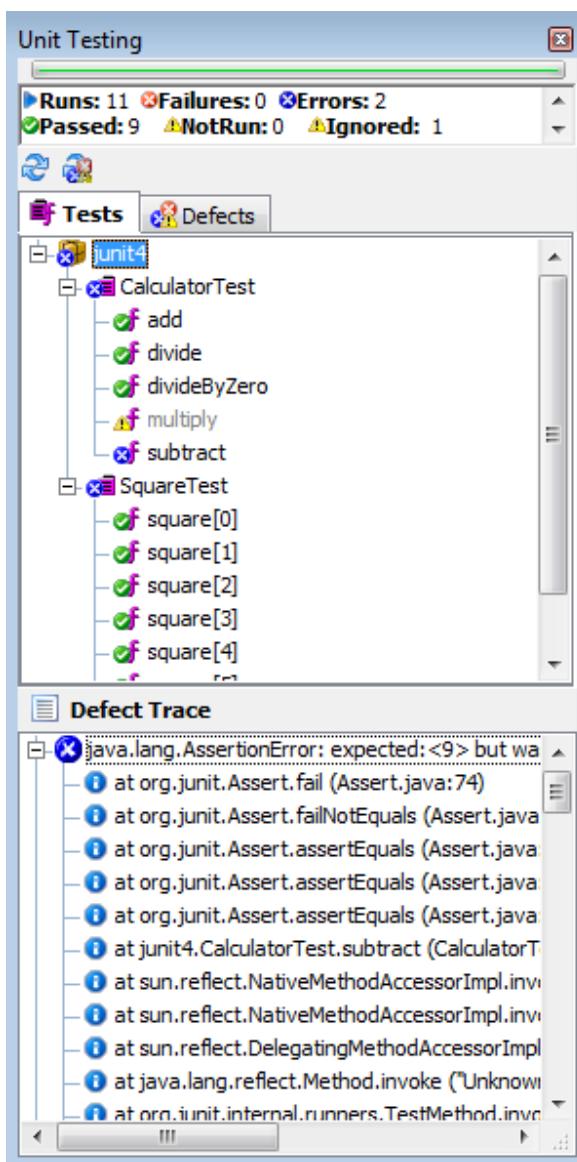
JUnit Testing (Pro only)

JUnit tests can be run from within SlickEdit®. The results can be viewed and code that fails the testing can be easily reconciled.

Note

JUnit support in SlickEdit® requires Java 8 or later. JUnit4 or Junit5 tests are both supported. JUnit dependencies are included with SlickEdit installation located in the `vsjunit`, and are automatically added to classpath when running unit tests. If you want to include different or more recent JUnit dependent jars, add them to the project classpath in the Java Options dialog (**Build** → **Java Options**).

To run a JUnit test, in the Projects tool window, select the project, package or file that you want, then on the right-click context menu, select **Unit Test** → **Run (junit command)** or **Unit Test** → **Debug (junit_debug command)**. The results are displayed in the Unit Testing tool window.



The Unit Testing tool window displays the number of tests that ran, failed, passed, had errors, and were not run. Double-click the items found on the **Tests** or **Defects** tab to be redirected to the code that needs to be debugged.

A tree control displays the defect trace(s) for the currently selected test item. Buttons are available above the tabs to rerun tests:

- To rerun the last set of tests, click **Run Current TestCases**.
- To rerun only the tests with defects from the last set of tests run, click **Run Current TestCases with Defects**.

Java-Specific Interfaces

This section provides detailed information about the following dialogs, tool windows, and option screens that are specific to Java programming:

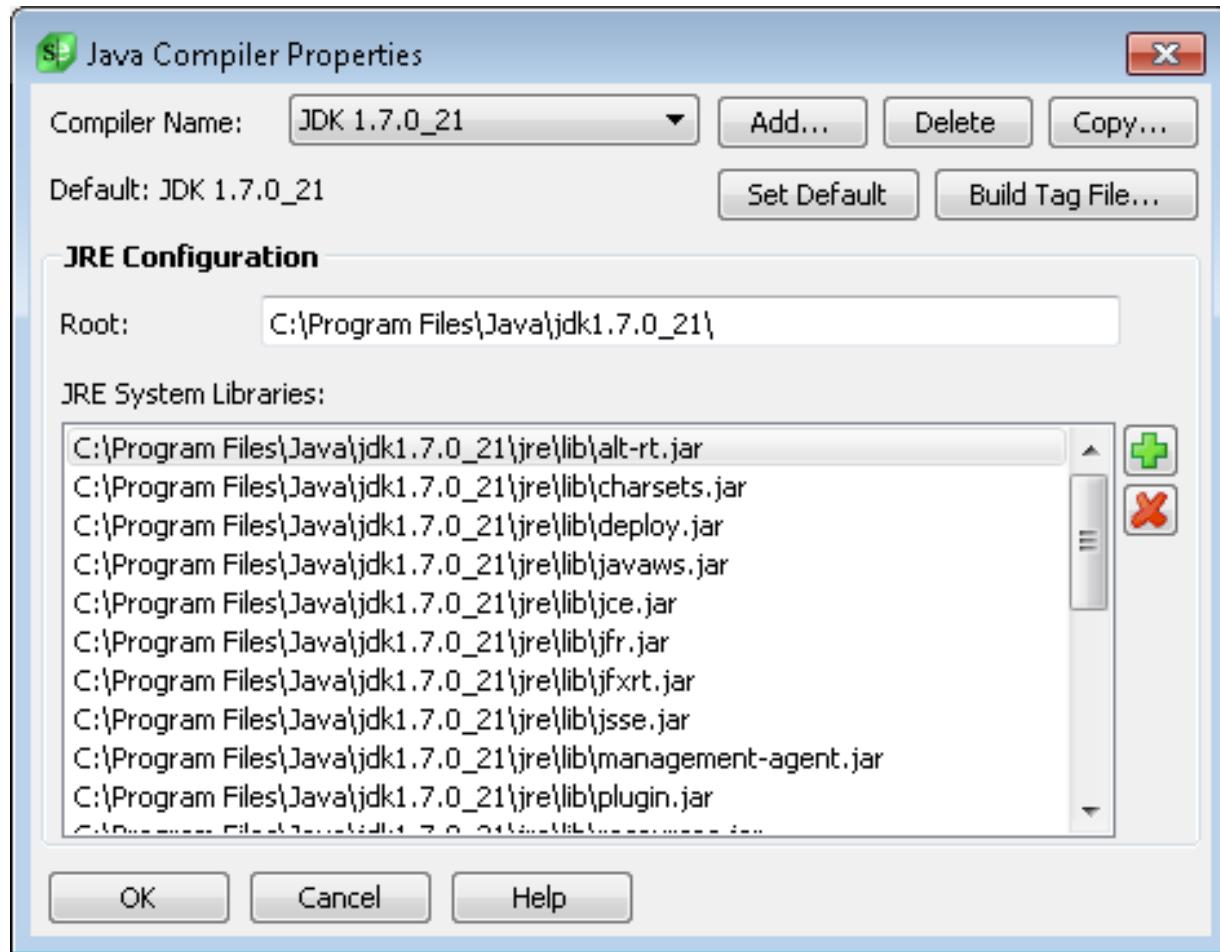
- [Java Compiler Properties Dialog](#)
- [Java Options Dialog](#)
- [Organize Java Imports Options Interface](#)
- [Beautifiers](#)
- [Javadoc Editor Dialog](#)
- [Javadoc Beautifier Options Dialog](#)

Java Compiler Properties Dialog (Pro only)

In order to correctly perform symbol analysis and cross-referencing, SlickEdit® needs to know which JDK you are using and where the system libraries are located.

These properties can be specified using the Java Compiler Properties option screen or the Java Compiler Properties dialog. The interfaces contain the same fields and options so you can make changes using the one you prefer:

- From the main menu, click **Tools** → **Options** → **Languages** → **Application Languages** → **Java**, then select **Compiler Properties**.
- With a Java project open, from the main menu, click **Project** → **Project Properties**. Select the **Compile/Link** tab, then click the **Ellipsis** button to the right of the **Compiler** combo box. The Java Properties dialog is displayed.

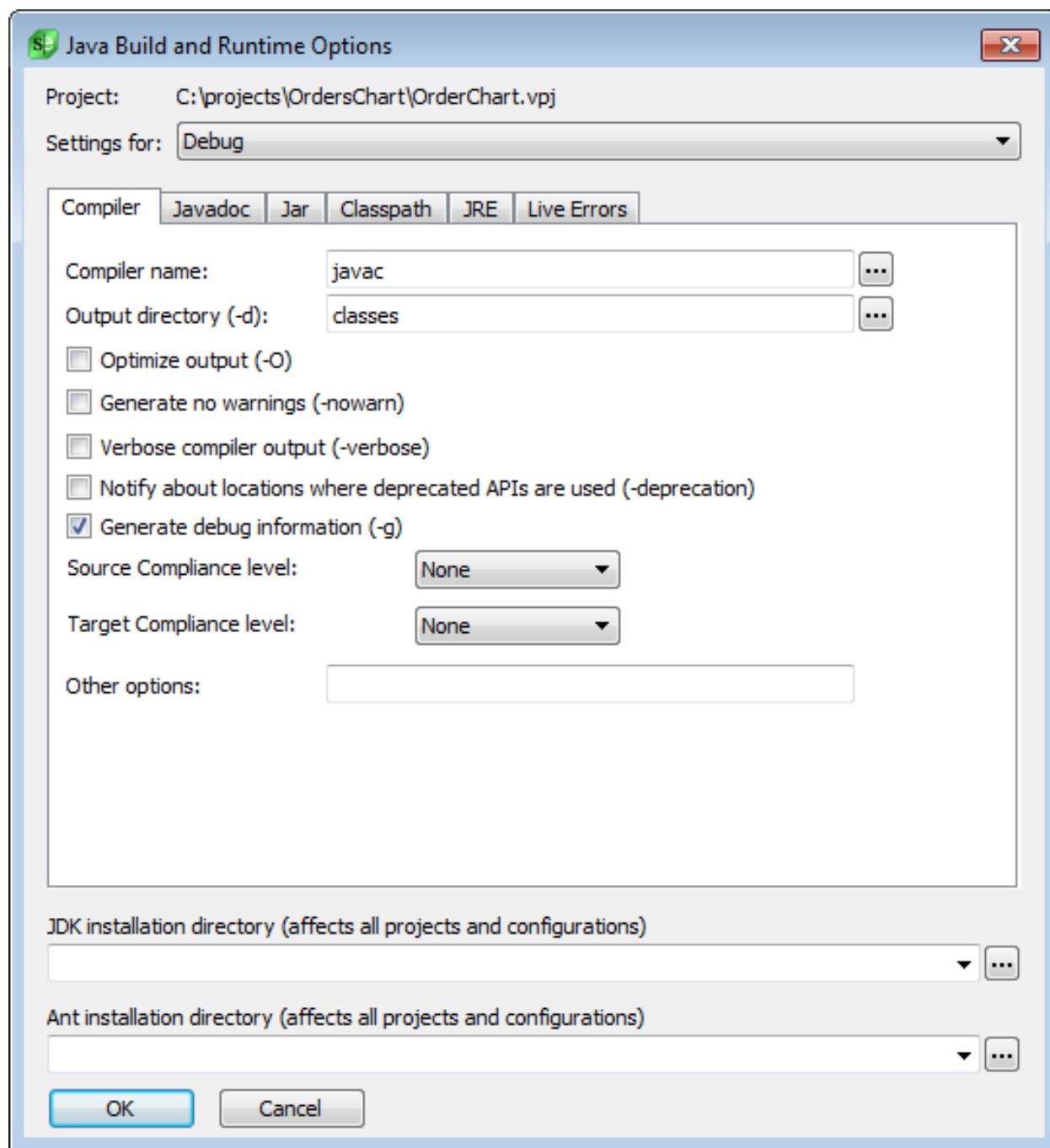


For Java, you can have multiple JDKs installed on your computer at the same time and configure SlickEdit to use different JDKs for each project. The interface provides the name and location for each JDK so that you can select it for tagging or building. See [Language-Specific Compiler Properties](#) for more information about these options.

Java Options Dialog (Pro only)

The Java Options dialog contains settings used when you build or execute a Java project. Most of the settings are stored for the particular Java project and configuration selected. You can set different values for different projects and for different configurations of the same project. For example, you might have different settings for the Debug configuration than from the Release configuration, allowing you to turn on optimizations used for release that are incompatible with debugging.

To access the Java Options dialog, first make sure you have a Java project or file open, then from the main menu, click **Build → Java Options** (or, use the **javaoptions** command).



There are three settings on the dialog that apply to all tabs:

- **Settings for** - The **Settings for** drop-down list at the top of the dialog is used to specify the project configuration you want to affect with the option settings. This is the same field that is on the Project Properties dialog, where you can also create new configurations. See [Project Configurations](#) for more information.
- **JDK installation directory (affects all projects and configurations)** - Specifies the full path to the root of the JDK used to build and execute Java programs. This value is shared by all Java projects and

all configurations. Click the button to the right of the field to browse. Use the drop-down button to select a recently used entry.

- **Ant installation directory (affects all projects and configurations)** - Specifies the full path to the root of the Ant installation directory. Ant is a commonly used build tool for Java. It is not shipped with SlickEdit, so you need to specify where it is installed. Click the button to the right of the field to browse. Use the drop-down button to select a recently used entry.

The other settings and options are divided into the following tabs:

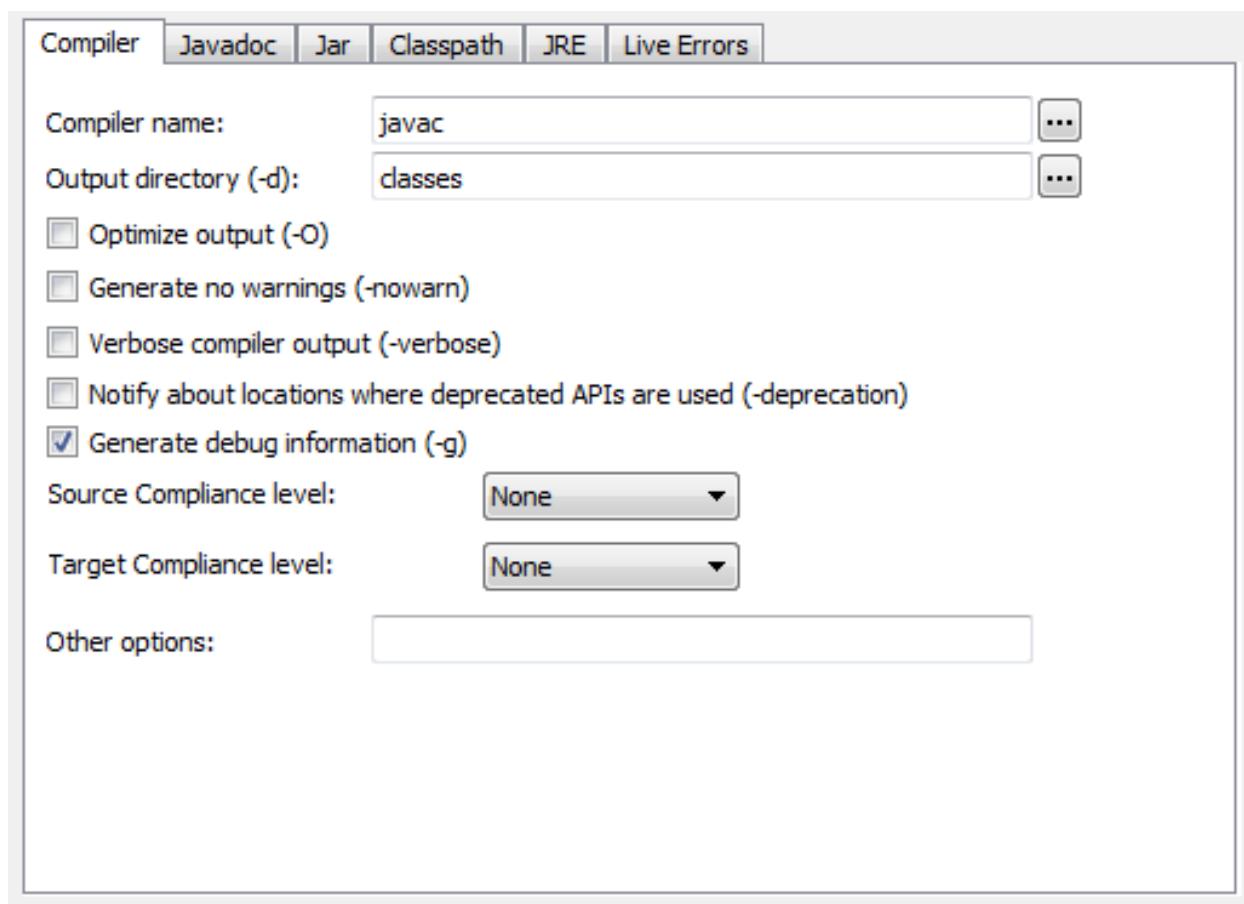
- [Compiler Tab](#)
- [Javadoc Tab](#)
- [Jar Tab](#)
- [Classpath Tab](#)
- [JRE Tab](#)
- [Live Errors Tab](#)

Note

- Options that are self-explanatory are not described in the documentation.
 - Prior to making changes on the tabs of the dialog, be sure that your desired project configuration is selected in the **Settings for** drop-down list at the top of the dialog.

Compiler Tab

The Compiler tab on the Java Options dialog (**Build → Java Options**) is shown below.



This tab contains the following fields and options:

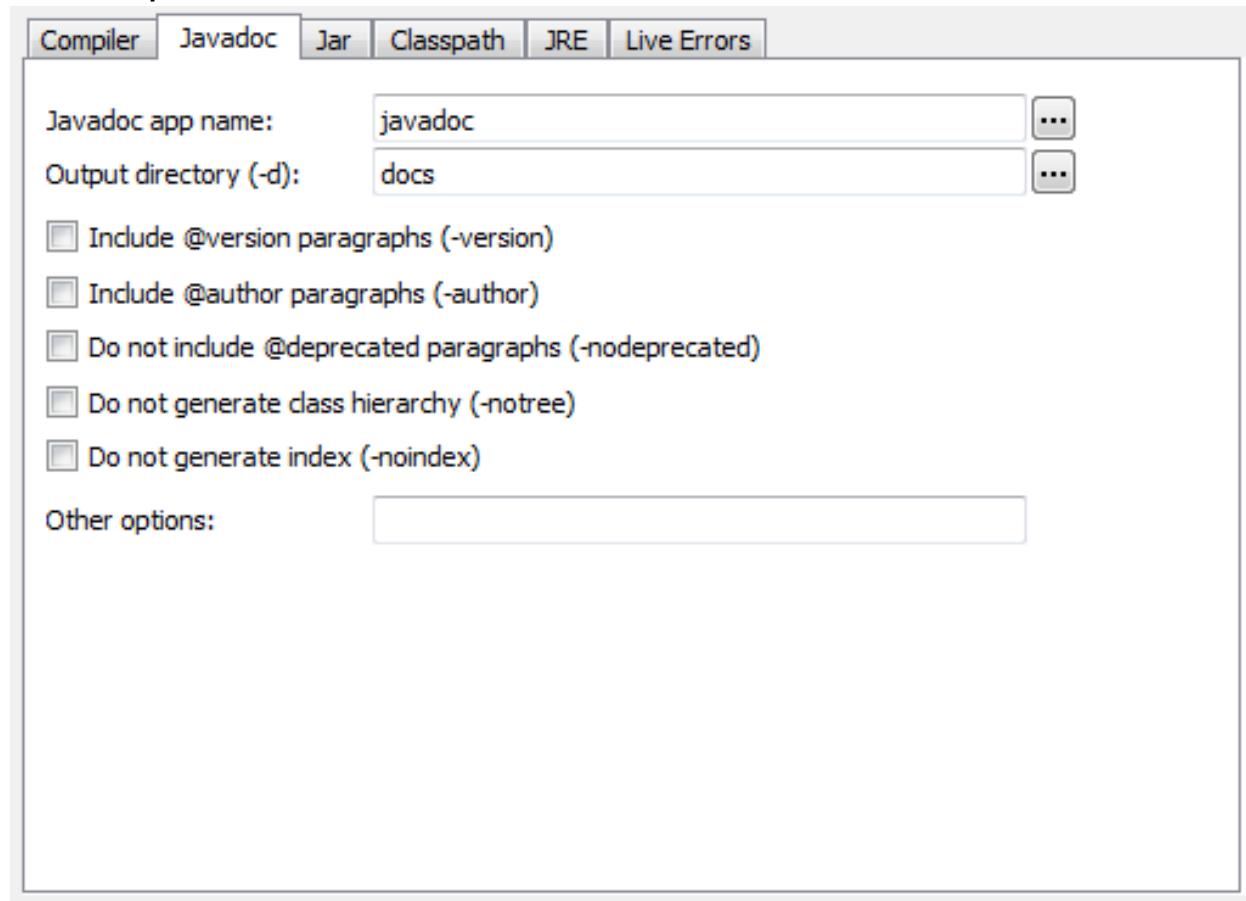
- **Compiler name** - Specifies the name of the compiler executable without the file extension. Click the **Ellipsis** button to browse for the file.
- **Output directory (-d)** - Specifies the full path to a directory to save the generated class files. Click the **Ellipsis** button to browse for the directory.
- **Optimize output (-O)**
- **Generate no warnings (-nowarn)**
- **Verbose compiler output (-verbose)**
- **Notify about locations where deprecated APIs are used (-deprecation)**
- **Generate debug information (-g)**
- **Source Compliance level** - Specifies the JDK version number to use when parsing the code. For example, JDK 1.6 can still parse the syntax for code written using JDK 1.4. Use this field to tell the compiler which Java syntax to use. This is the same as setting the **-source** option on the javac command line.
- **Target Compliance level** - Setting this value will generate code that will run on the specified version of

the Java VM. This is the same as setting the **-target** option on the javac command line.

- **Other options** - Specify additional command line options in this text field. For example, you can use this field to enter a value like "-sourcepath c:\dev\src\BigProject".

Javadoc Tab

The Javadoc tab on the Java Options dialog (**Build → Java Options**) contains options to configure the application that processes the Javadoc comments in your code to produce project documentation. By default, SlickEdit® uses the javah program. Many of the options are specific to that processor. If you choose to use a different program, uncheck the command line arguments and enter your options using the **Other options** field.



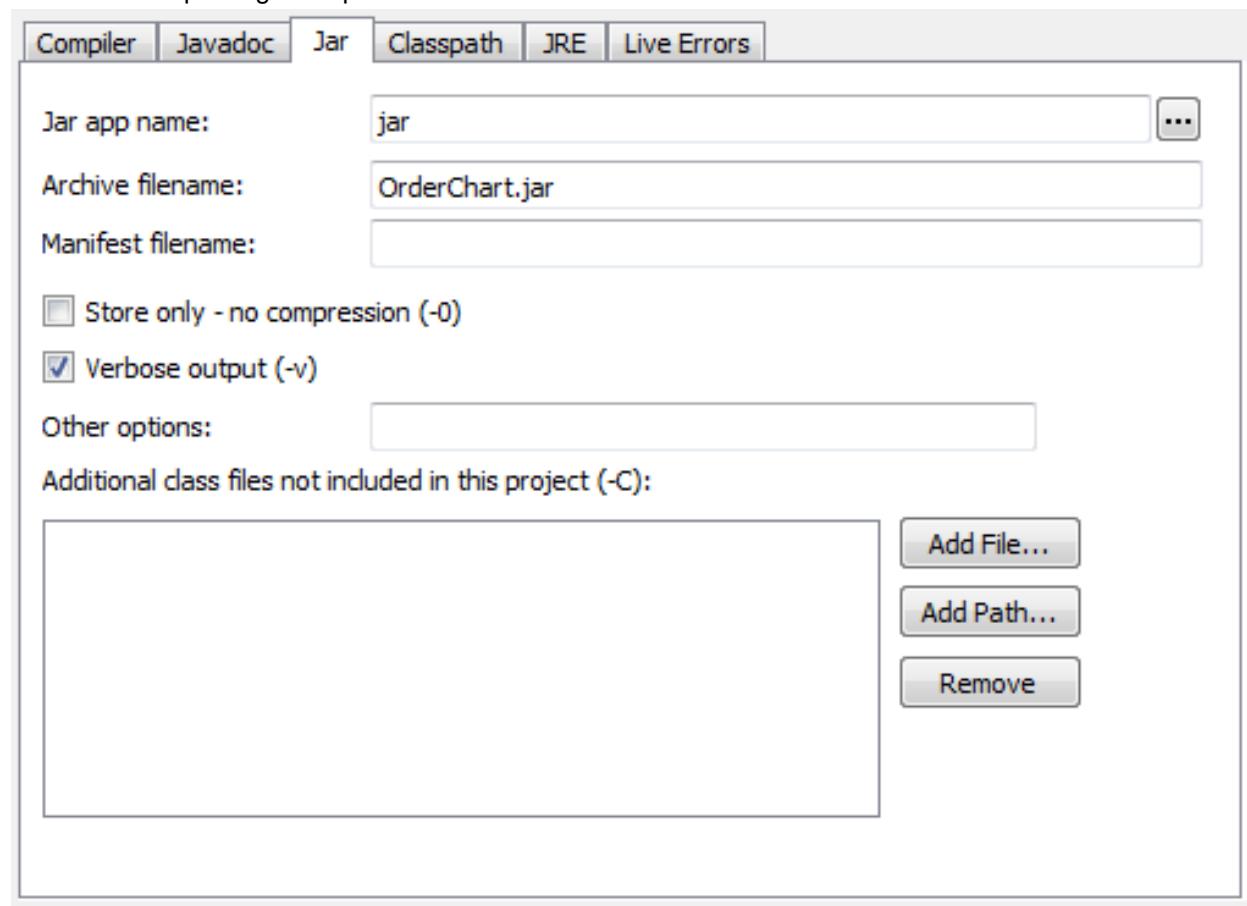
This tab contains the following fields and options:

- **Javadoc app name** - Specifies the name of the executable used to process the Javadoc comments in your source code. By default, this is set to **javadoc**. The extension is not needed. Click the **Ellipsis** button to browse for the file.
- **Output directory (-d)** - Specifies the directory in which to save the generated Javadoc. You can specify a full path or relative path. Relative paths are interpreted relative to the project location, specified when you created the project. For example, a value of **docs** will create a subdirectory in the project directory to store the output from the Javadoc processor.

- **Include @version paragraphs (-version)**
- **Include @author paragraphs (-author)**
- **Do not include @deprecated paragraphs (-nodeprecated)**
- **Do not generate class hierarchy (-notree)**
- **Do not generate index (-noindex)**
- **Other options** - Specify additional command line options in this text field to be passed to the Javadoc processor.

Jar Tab

The Jar tab on the Java Options dialog (**Build → Java Options**) is used to configure the Jar application that is used to package compiled Java classes.



This tab contains the following fields and options:

- **Jar app name** - Specifies the name of the Jar executable, without the extension. Click the **Ellipsis** button to browse for the file.
- **Archive filename** - Specifies the name of the archive to create.

- **Manifest filename** - If you want to create a manifest, specify the name of the manifest file here. If necessary, use this file to specify a main class for your application.
- **Store only - no compression (-0)**
- **Verbose output (-v)**
- **Other options** - Specify additional command line options to pass to the Jar application.
- **Additional class files not included in this project (-C)** - Used to specify additional classes you would like to include in the archive.
 - **Add File** - Click to add a new class file to the archive.
 - **Add Path** - Click to specify a directory. All of the classes in that directory will be added.
 - **Remove** - Click to remove the selected file or directory.

Classpath Tab

In Java, the classpath defines a search path for compiled Java classes. The elements are searched in the order specified and the first matching class is used. The **Classpath** tab on the Java Options dialog (**Build** → **Java Options**) allows you to configure the Java classpath used in SlickEdit® for running and debugging programs. This does not affect the classpath set in the operating system.

Warning

If your classpath contains unnecessary files or directories, it could slow down the launching of the Java debugger using the step-into command. If you experience this problem, remove any elements not needed by this specific project and avoid using the system Classpath.

If you have different classpaths for different projects, configuring the classpath inside of SlickEdit is very useful. If you have a single classpath and use it with other external tools, it is best to configure it in the operating system and then put a reference to the external classpath in SlickEdit by using the **Add Path** button on this tab.

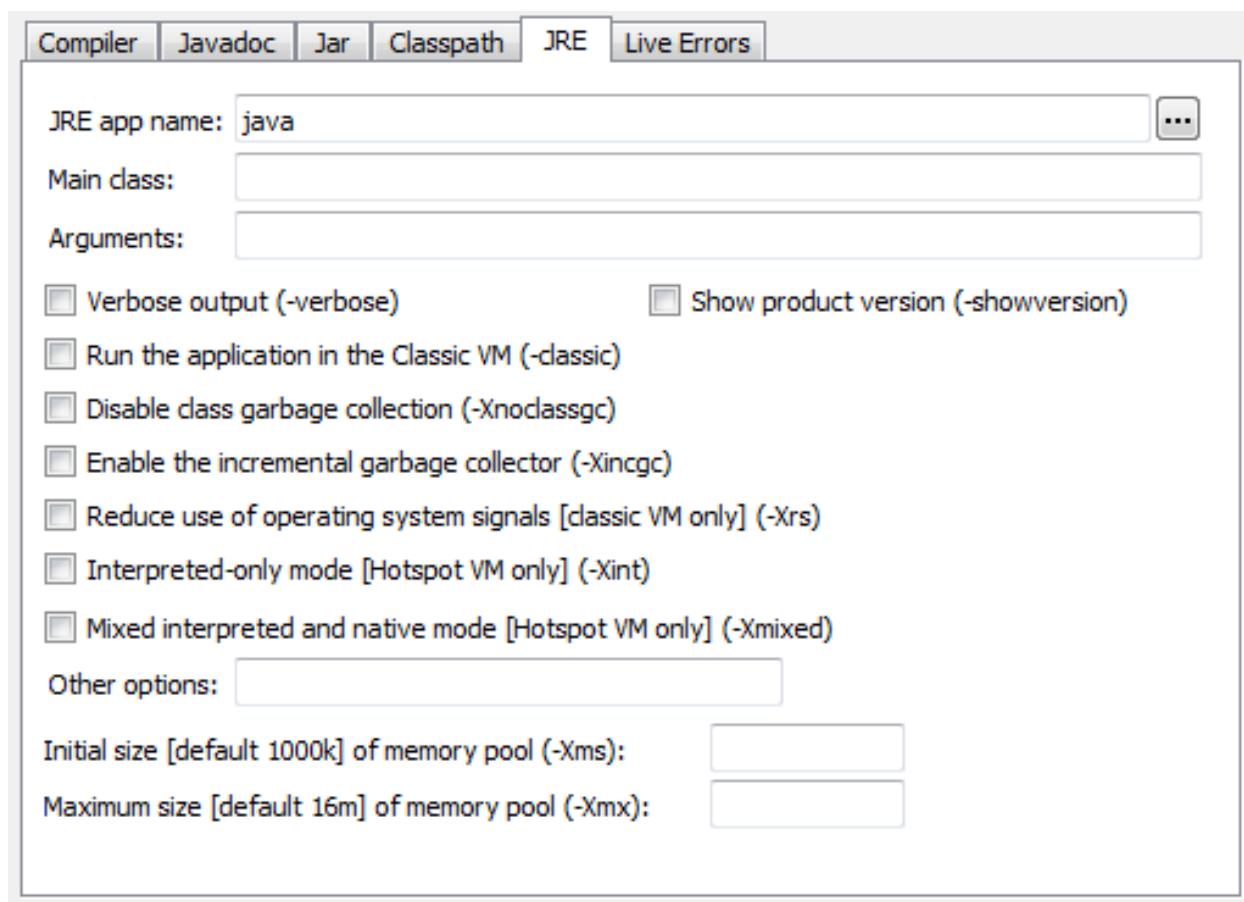


This tab contains the following fields and options:

- **Add Path** - Click to browse for a directory. This directory is added to the classpath.
- **Add Jar File** - Click to add a JAR file to the classpath.
- **Add Class Path** - Click to insert the environment variable that contains the classpath defined in the operating system.
- **Edit** - Click to edit the selected classpath element.
- **Delete** - Click to delete the selected classpath element.
- **Up/Down** - Use the **Up** and **Down** buttons to move the selected item up and down in the list.
- **Use Classpath settings for antmake commands** - When this option is selected, the SlickEdit classpath is passed to Ant. See [Language-specific Build Methods](#) for more information about Ant support.

JRE Tab

The **JRE** tab on the Java Options dialog (**Build** → **Java Options**) is used to configure options for the Java Runtime Environment. These are used when executing a Java program inside SlickEdit®.



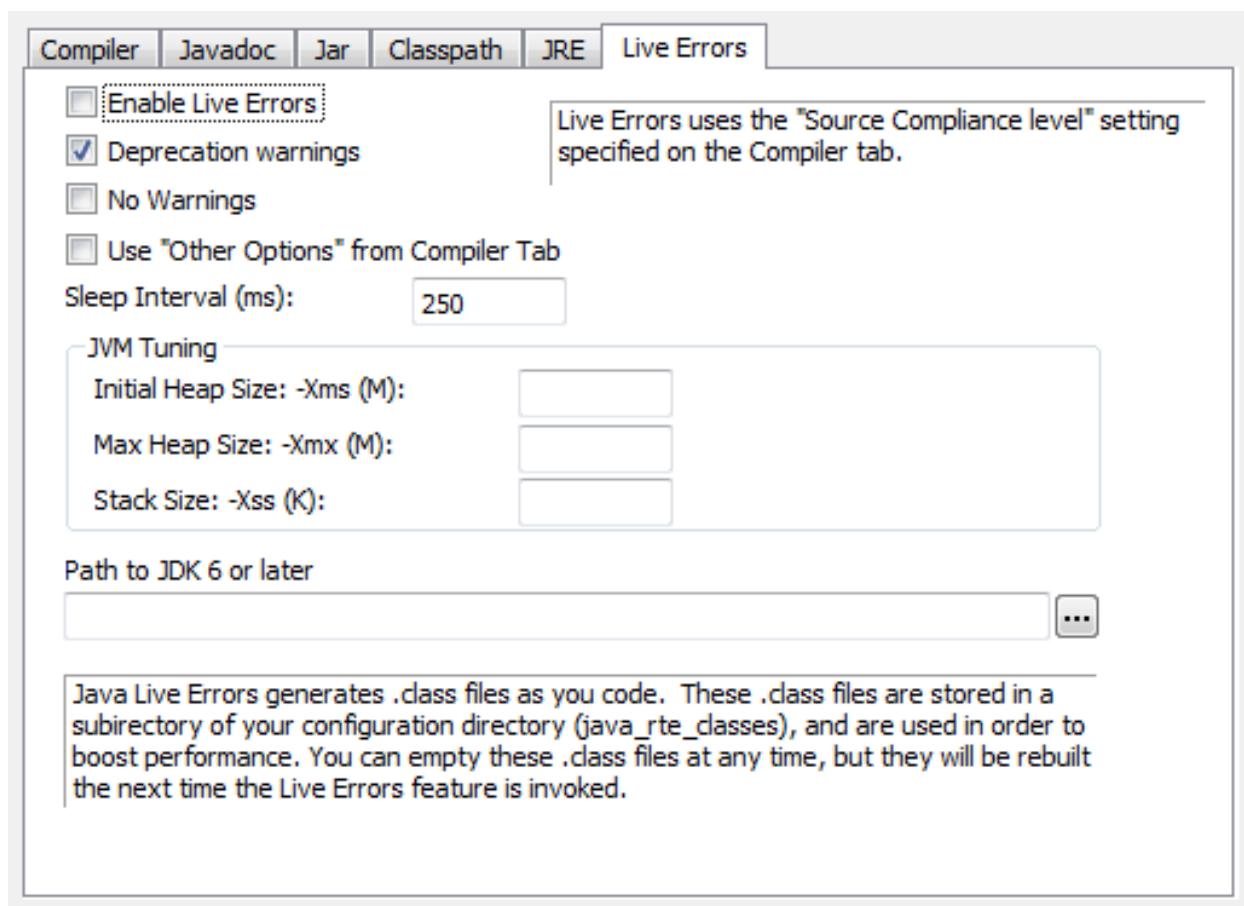
This tab contains the following fields and options:

- **JRE app name** - Specifies the name of the executable for the Java interpreter, **java** by default. The extension is not needed. Click the **Ellipsis** button to browse for the executable.
- **Main class** - Specifies the name of the class to begin execution. In Java, each class can contain a main function. This value determines where execution begins when you select **Build** → **Execute** for this project.
- **Arguments** - Specifies parameter values passed to the main function contained in the main class.

The remaining options are common configuration parameters to pass to the Java interpreter. Each displays the command line switch used. For more information, look up the corresponding switch in the JRE documentation. If these values are not supported by the JRE you are using, uncheck them and use **Other options** to pass arguments to the JRE.

Live Errors Tab

The Live Errors tab on the Java Options dialog (**Build** → **Java Options**) is used to configure the Live Errors feature. Live Errors identifies syntax errors as you type. SlickEdit compiles your code in the background and highlights errors directly in the editor.



This tab provides the following fields and options:

- **Enable Live Errors** - When selected, the Live Errors feature is activated. This setting is checked by default if SlickEdit has detected a valid JDK 6, or compatible, installation on your system, the first time a Java project is opened. For some coding, you may want to disable Live Errors since it will point out all syntax errors, which are common in incomplete code.

After activating 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.

Note

- For Java 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 in the **JDK installation directory** field 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 Java Live Errors running and wish to specify a different JDK 6 (or compatible JDK) root, after changing the JDK path on the Java Options dialog, you must restart SlickEdit.

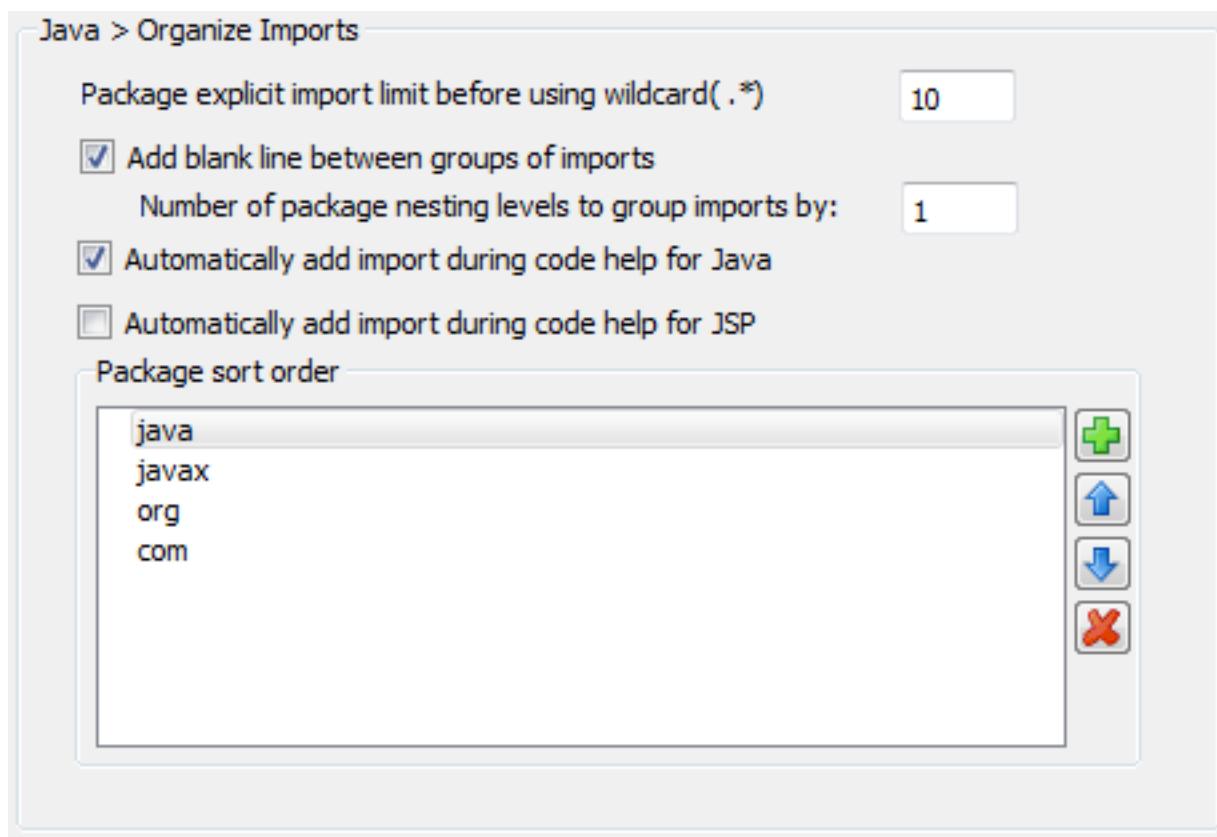
- In order to boost performance, Java Live Errors generates .class files as you code. These files are stored in the `java_rte_classes` subdirectory of your config. You can remove these files, but they will be rebuilt the next time Java Live Errors is invoked.
- To specify the source compliance level, use the **Source Compliance level** setting on the [Compiler Tab](#).

- **Deprecation warnings** - When this option is selected, a **Warning** bitmap appears when Live Errors encounters a keyword that has been deprecated. This is useful for programmers coding to strict Java standards.
- **No Warnings** - When this option is selected, warning notices from Live Errors are not displayed.
- **Use "Other Options" from Compiler Tab** - When this option is selected, any options specified in the "Other Options" field on the **Compiler** tab will be used for Live Errors. Live Errors will automatically parse out those javac options which are not supported by the Java Compiler API.
- **Sleep Interval(ms)** - Specifies the amount of time, in milliseconds, in which a Live Errors thread sleeps before checking for errors (during times when an error check is not forced). Use this to tune performance. A larger value will give the editor more time between builds and should reduce any performance issues.
- **JVM Tuning** - These options control how much memory is allocated to the JVM used for Live Errors. SlickEdit will require a restart in order for new values to take effect.
 - **Initial Heap Size: -Xms (M)** - Initial heap size, in megabytes. Must be greater than or equal to 2.
 - **Max Heap Size: -Xmx (M)** - Maximum heap size, in megabytes.
 - **Stack Size: -Xss (K)** - Stack size for each thread, in kilobytes.
- **Path to JDK 6 or later** - Specifies the root directory of a valid JDK 6 or compatible JDK installation. Live Errors requires JDK 6 or a compatible JDK to run, and will not activate if this path is not correct. SlickEdit attempts to populate this field for you the first time a Java project is opened. Click the **Ellipsis** button to browse for the directory.

Organize Java Imports Options Interface (Pro only)

Options are available on the Options dialog to configure the Organize Imports feature (**Tools** → **Options** → **Languages** → **Application Languages** → **Java** → **Organize Imports**). A more direct route to the options is to use one of the following methods:

- From the main menu, click **Tools** → **Imports** → **Options**
- Select **Tools** → **Imports** → **Options** from the right-click context menu in the editor window.
- Use the **refactor_organize_imports_options** command.



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 **Ellipses (...)** button to add a new package. Use the **Up** and **Down** arrow buttons to move items. Use the **X** button to delete the currently selected package from the list.

Javadoc Editor Dialog

Use the Javadoc Editor to generate Javadoc syntax comments for Java, C, C++, JavaScript, and Slick-C®. To access the Javadoc Editor, choose **Document** → **Edit Documentation Comment**.

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>.

Note

NOTE SlickEdit® provides powerful capabilities to create and edit Javadoc comments within the editor. See [Commenting](#) for more information.

Javadoc Beautifier Options Dialog

To beautify Javadoc comments or set up Javadoc Beautifier options, first invoke the Javadoc Editor by choosing **Document** → **Edit Documentation Comment**. 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.

Mono

SlickEdit® provides a full-featured Mono development environment, allowing you to edit, build, and debug C#, F#, and Visual Basic programs using Mono. Topics in this section:

Note

NOTE There are numerous compilers which are compatible with the Mono environment, including compilers for C#, F#, Mono, Scala, Boo, Visual Basic.NET, Python.NET, JavaScript, Oberon, PHP, Lua, Cobra, Synergy-DBL, and Object Pascal. Many of these can be used with SlickEdit by customizing a C# Mono project to use the compile and build commands required by the given compiler. See [Creating Custom Project Types](#) for more information. Note that only C# and Visual Basic have been tested and verified by the SlickEdit team.

- [Initial Setup](#) - Read this to configure SlickEdit for your Mono installation and other settings needed for compiling and debugging.
- [C# Organize Imports](#) - Information about the Organize Imports feature for C#.
- [XMLDoc Comments](#) - Information about features designed specifically for C# and Visual Basic programmers.

Initial Setup (Pro only)

SlickEdit® relies on an installed Mono development kit for compiling and debugging. After you have installed Mono on your computer, the following steps will configure SlickEdit to use it. The steps are divided into three categories:

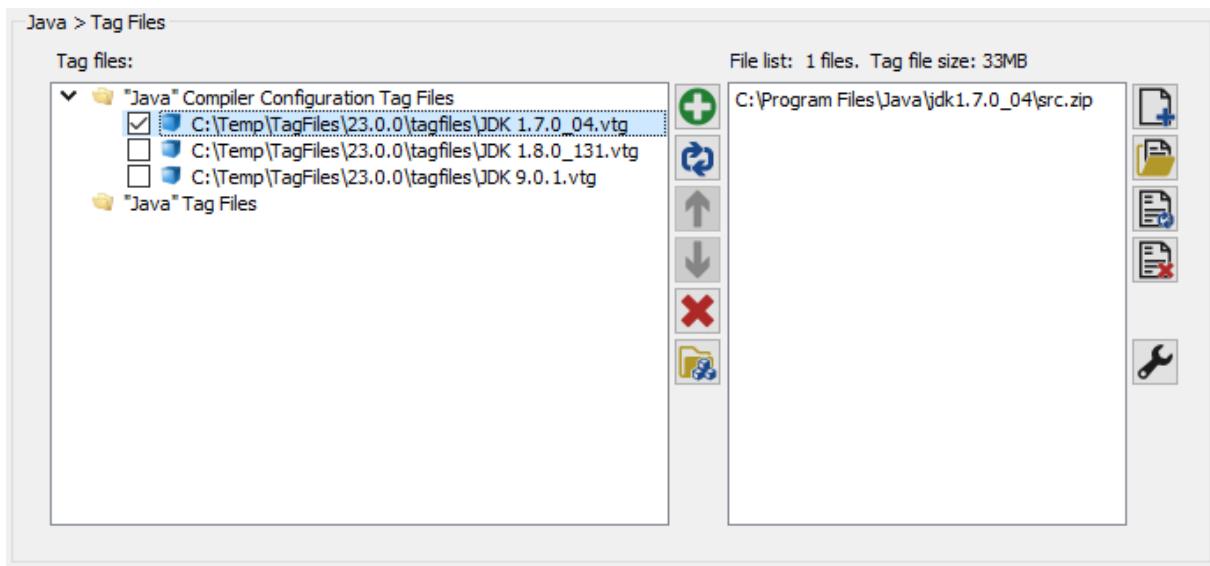
- [Context Tagging® for .NET](#)
- [Setting Up a Mono Workspace and Project](#)
- [Mono Options Dialog](#)

Context Tagging® for .NET(Pro only)

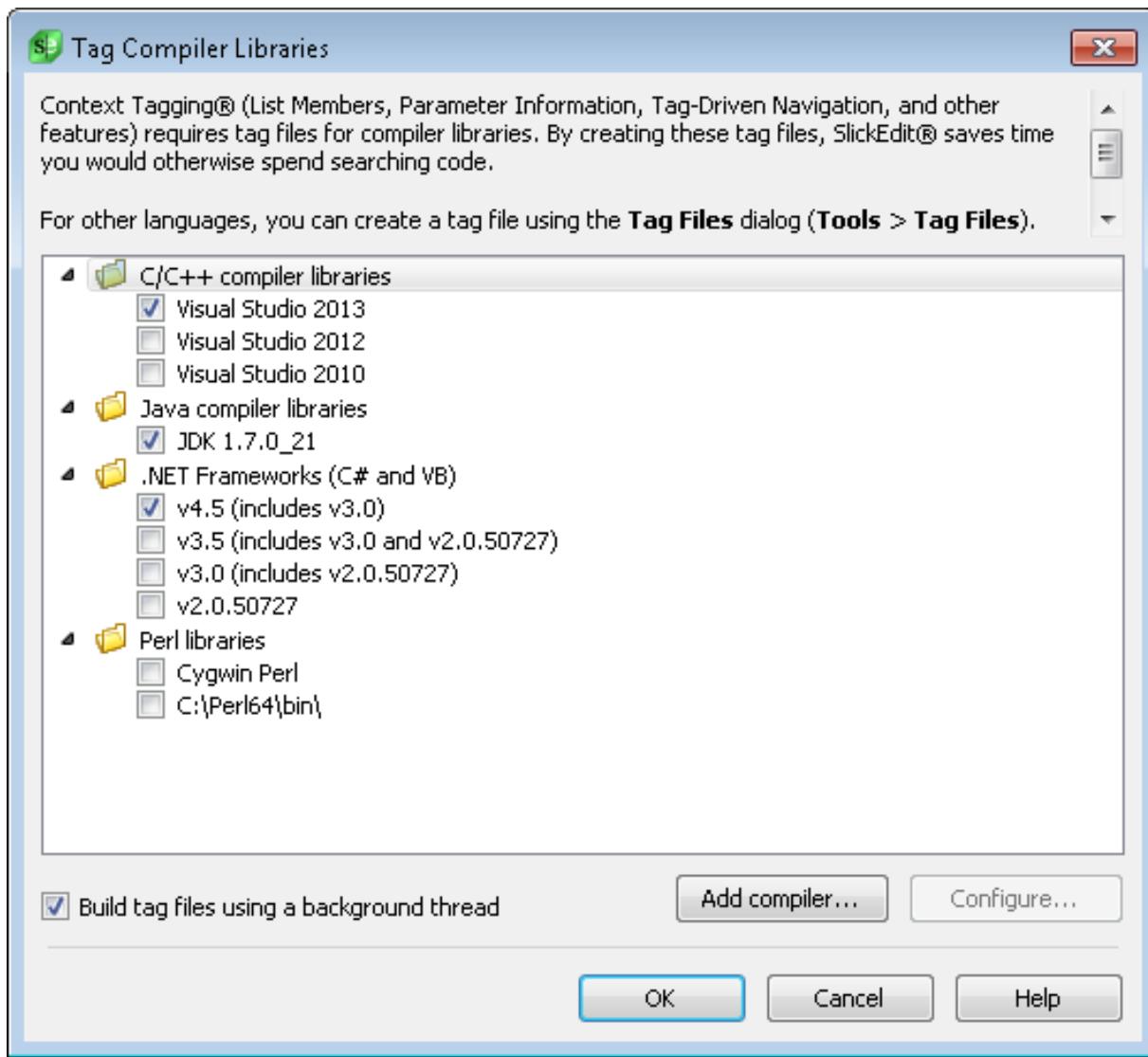
SlickEdit® needs to tag the .NET framework libraries to provide symbol completions and other Context Tagging features for those classes (see [Context Tagging Features](#)). When you first run SlickEdit after an installation, you are prompted with a dialog to create these tag files. Complete the steps below if you did not create tag files at that time or to configure additional C# or Visual Basic libraries.

1. Open the [Context Tagging - Tag Files Dialog](#) by selecting **Tools** → **Tag Files**.

Initial Setup (Pro only)



2. Click the **Auto Tag** button to open the Tag Compiler Libraries dialog.



3. SlickEdit may detect that you have installed Mono or Microsoft .NET. If so, the section for .NET Frameworks will be filled out. If not, you will have to configure this manually.
4. Make sure there is a check in the **.NET Frameworks (C# and VB)** check box. Depending on your environment, there may be checks in the check boxes for C++ and Java. Leave those checked if you have not already tagged those libraries. If you just want to tag the .NET libraries, uncheck the other check boxes.
5. Click the **Create tag file(s)** button.
6. SlickEdit will display a progress bar while your libraries are being tagged. When finished, SlickEdit will display the Context Tagging - Tag Files dialog. You can close this if you have no other libraries to tag.

Setting Up a Mono Workspace and Project (Pro only)

In SlickEdit®, files are contained in projects, and projects are contained in workspaces. Except for the

most basic editing, you should always work within a workspace and project in SlickEdit. Context Tagging® relies on having your source files contained in a project. See [Workspaces and Projects](#) for more information.

Editing options are determined by the file extension and accessed by selecting **Tools** → **Options** from the main menu, then selecting the corresponding language in the options hierarchy. Editing options control how your code is formatted and key editing behaviors as you type.

The project type determines your build environment and provides options specific to that project type. For Mono, this includes specifying which Mono interpreter to use, along with arguments, and setting up the debugger. To create a new Mono project or to see a list of the available project types, select **Project** → **New** from the main menu, then expand the C#, F#, or Visual Basic node in the tree under **Project type**. Note that, on Windows, there are separate project types both for Mono and Visual Studio.

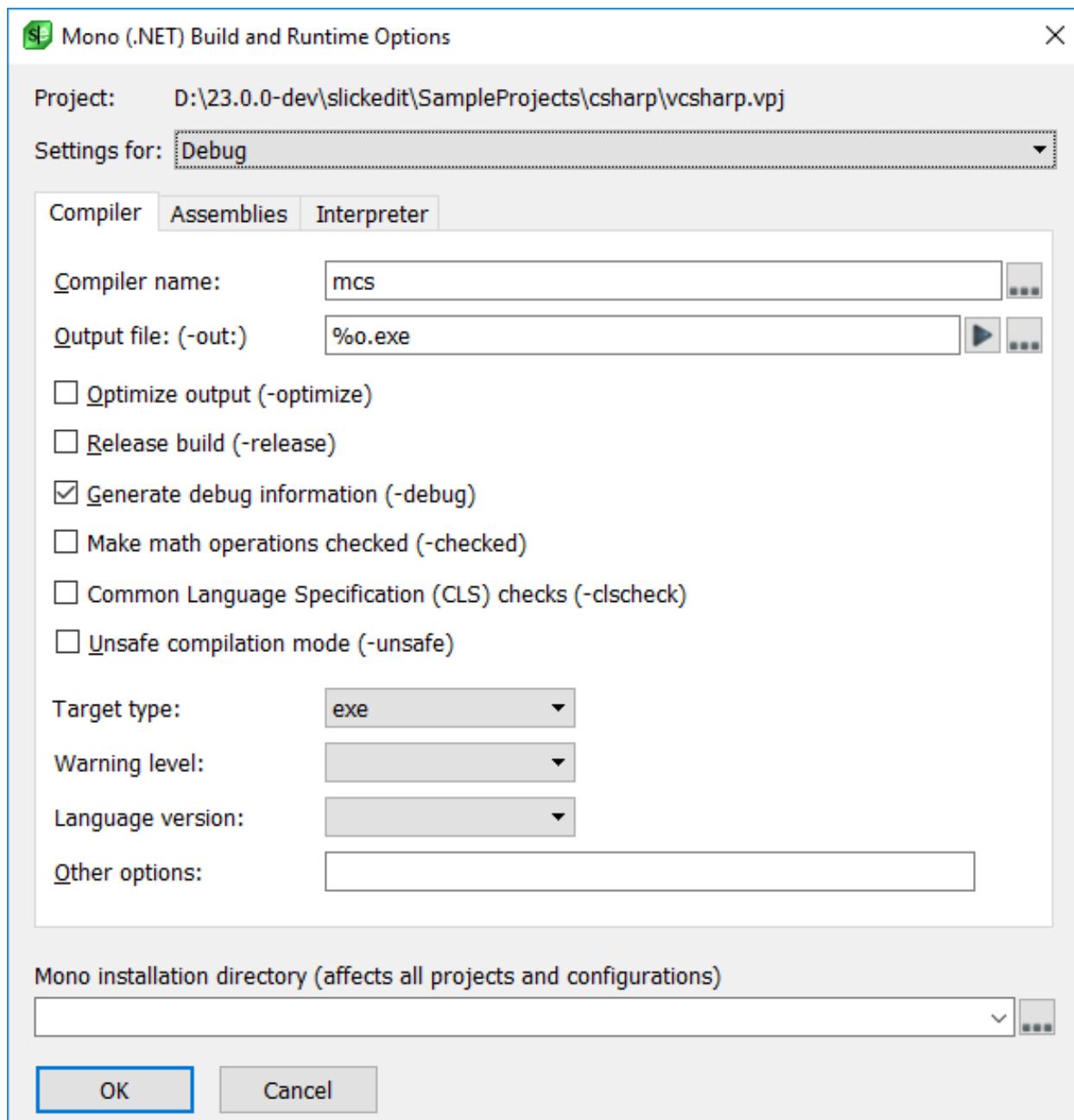
After you have created a Mono project, you can set the build options by selecting **Project** → **Project Properties** from the main menu, selecting **Tools** and selecting an item in the **Tool name** list to edit specific command parameters.

You can also change the execution and debugging options for Mono by selecting the **Run/Debug** tab on the **Project** → **Project Properties** dialog.

Mono Options Dialog (Pro only)

The Mono Options dialog contains settings used when you build or execute a Mono project. Most of the settings are stored for the particular Mono project and configuration selected. You can set different values for different projects and for different configurations of the same project. For example, you might have different settings for the Debug configuration than from the Release configuration, allowing you to turn on optimizations used for release that are incompatible with debugging.

To access the Mono Options dialog, first make sure you have a Mono project or C# or Visual Basic file open, then from the main menu, click **Build** → **Mono Options** (or, use the **mono_options** command).



There are three settings on the dialog that apply to all tabs:

- **Settings for** - The **Settings for** drop-down list at the top of the dialog is used to specify the project configuration you want to affect with the option settings. This is the same field that is on the Project Properties dialog, where you can also create new configurations. See [Project Configurations](#) for more information.
- **Mono installation directory (affects all projects and configurations)** - Specifies the full path to the root of the Mono installation used to build and execute Mono programs. This value is shared by all Mono projects and all configurations. Click the button to the right of the field to browse. Use the drop-down button to select a recently used entry.

The other settings and options are divided into the following tabs:

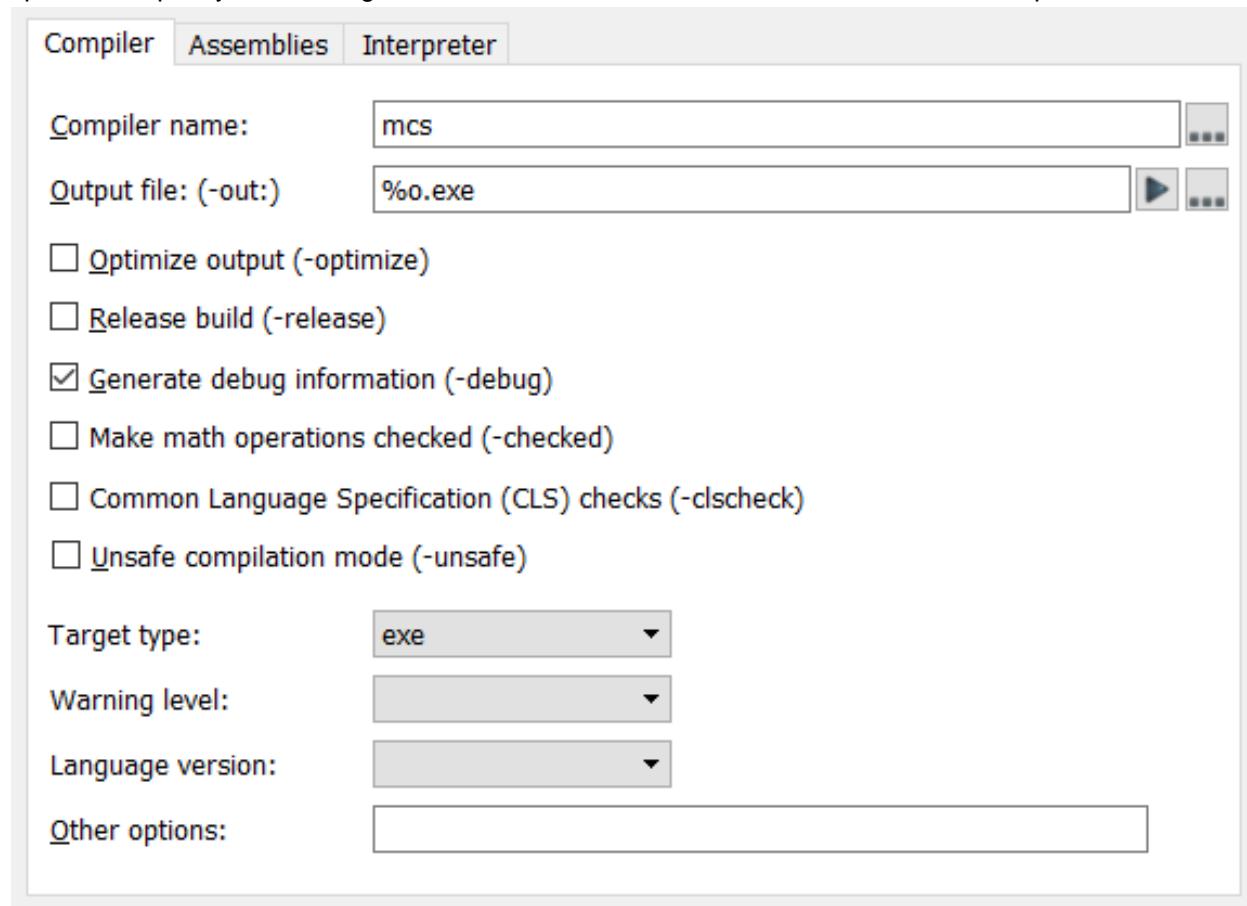
- [Compiler Tab](#)
- [Assemblies Tab](#)
- [Interpreter Tab](#)

Note

- Options that are self-explanatory are not described in the documentation.
 - Prior to making changes on the tabs of the dialog, be sure that your desired project configuration is selected in the **Settings for** drop-down list at the top of the dialog.

Compiler Tab

The Compiler tab on the Mono Options dialog (**Build → Mono Options**) is shown below. It provides quick access to the most common C# compiler options. Additional options and options that are different for alternate compilers can be specified in **Other Options**. If any of these options are not supported by the specific compiler you are using, uncheck it or leave it blank to take them out of the compile command.



This tab contains the following fields and options:

- **Compiler name** - Specifies the name of the compiler executable without the file extension. Click the **Ellipsis** button to browse for the file.
- **Output file (-out:)** - Specifies the full path to the output file to be created by this build. Click the **Ellipsis** button to browse for the file location.
- **Optimize output (-optimize)**
- **Release build (-release)**
- **Generate debug information (-debug)**
- **Make match operations checked (-checked)**
- **Common Language Specification (CLS) checks (-clscheck)**
- **Unsafe compilation mode (-unsafe)**
- **Target type:**
 - **Warning level:**
 - **Language version:**
- **Other options** - Specify additional command line options in this text field. For example, you can use this field to enter a value like "-linkresource ..."

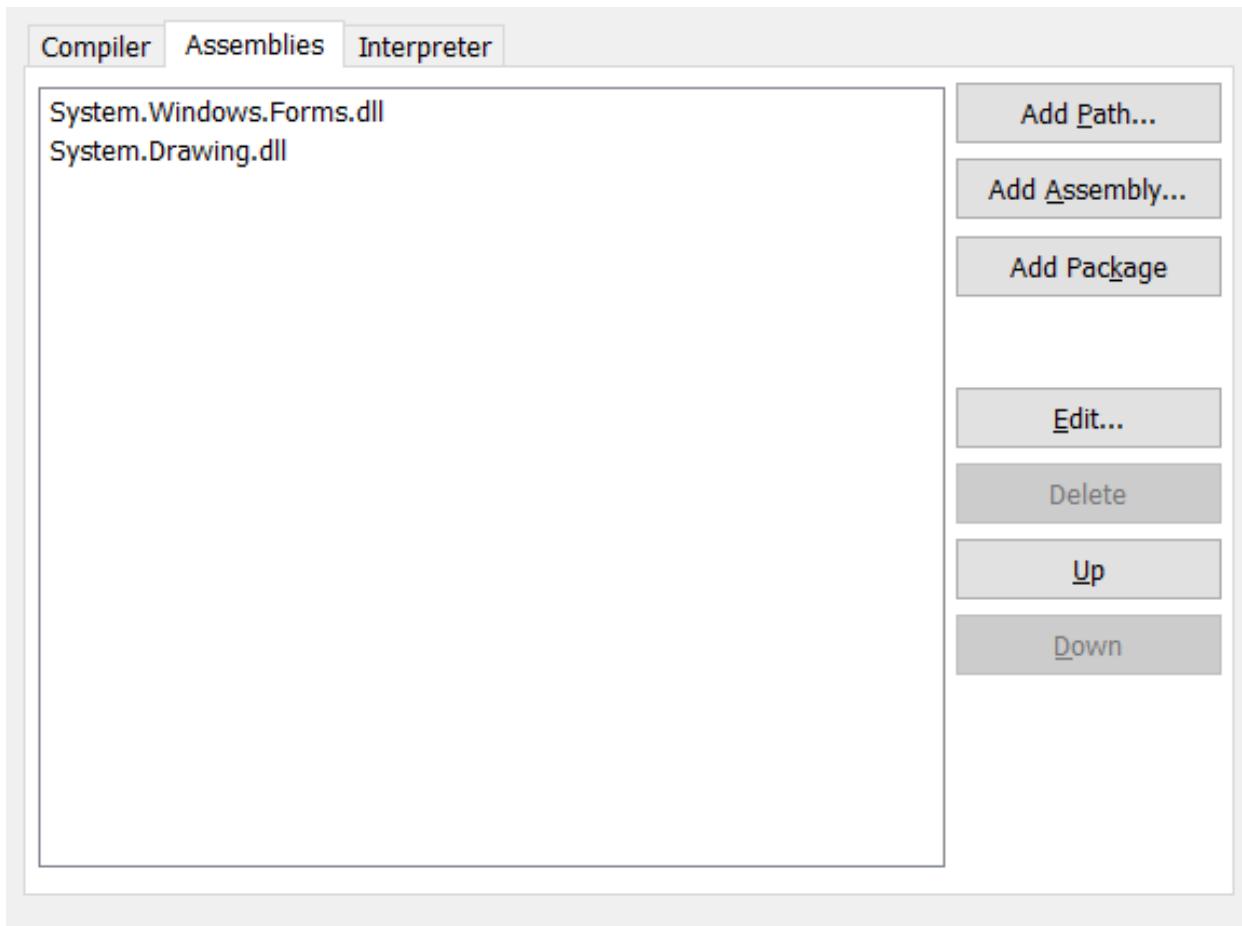
Assemblies Tab

In the .NET framework, the path defines a search path for compiled Mono assemblies. The elements are searched in the order specified and the first matching assembly is used. The **Assemblies** tab on the Mono Options dialog (**Build → Mono Options**) allows you to configure the path used in SlickEdit® for compiling, running and debugging programs. This does not affect the path set in the operating system.

Warning

If your path contains unnecessary files or directories, it could slow down the launching of the Mono debugger using the step-into command. If you experience this problem, remove any elements not needed by this specific project.

If you have different paths for different projects, configuring the path inside of SlickEdit is very useful. If you have a single path and use it with other external tools, it is best to configure it in the operating system and then put a reference to the external classpath in SlickEdit by using the **Add Path** button on this tab.

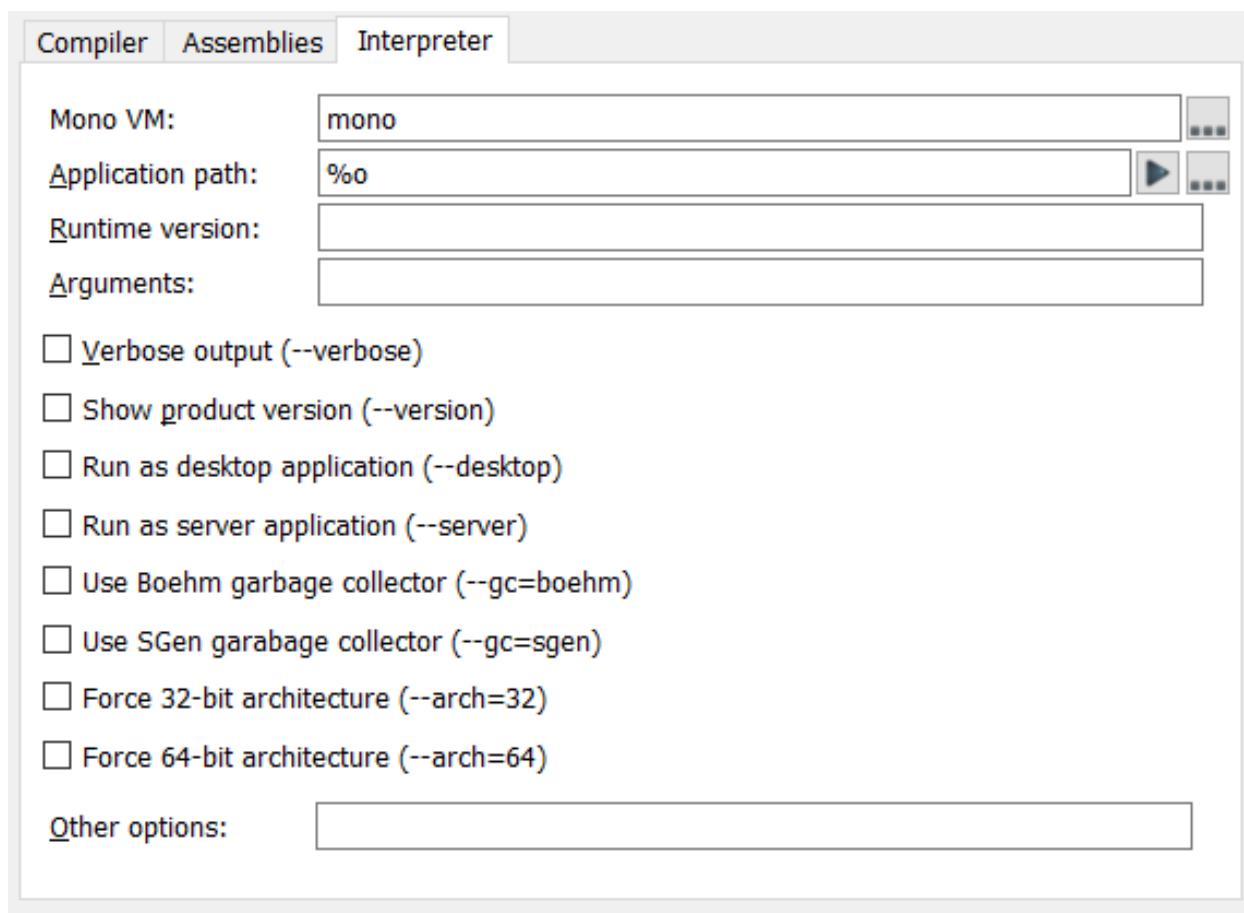


This tab contains the following fields and options:

- **Add Path** - Click to browse for a directory. This directory is added to the search path.
- **Add Assembly** - Click to add a .NET Assembly to the path.
- **Add Package** - Click to add a .NET package to the path.
- **Edit** - Click to edit the selected path element.
- **Delete** - Click to delete the selected path element.
- **Up/Down** - Use the **Up** and **Down** buttons to move the selected item up and down in the list.

JRE Tab

The **Interpreter** tab on the Mono Options dialog (**Build → Mono Options**) is used to configure options for the Mono runtime environment. These are used when executing a Mono program inside SlickEdit®.



This tab contains the following fields and options:

- **Mono VM:** - Specifies the name of the executable for the Mono interpreter, **mono** by default. The extension is not needed. Click the **Ellipsis** button to browse for the executable.
- **Application path:** - Specifies the name of the executable to launch to run this project. This will normally be the executable you are building for executables, but for library projects, it can be another executable that you launch in order to run the code in the library.
- **Runtime version:** - Specifies the version of the Mono or .NET runtime to run the application using.
- **Arguments** - Specifies parameter values passed to the main function contained in the main class.

The remaining options are common configuration parameters to pass to the Mono interpreter. Each displays the command line switch used. For more information, look up the corresponding switch in the Mono documentation. If these values are not supported by the version of Mono you are using, uncheck them and use **Other options** to pass arguments to the interpreter.

- **Verbose output (--verbose)**
- **Show project version (--version)**
- **Run as desktop application (--desktop)**

- Run as server application (`--server`)
- Use Boehm garbage collector (`--gc=boehm`)
- Use SGen garbage collector (`--gc=sgen`)
- Force 32-bit architecture (`--arch=32`)
- Force 64-bit architecture (`--arch=64`)
- Other options - Specify additional Mono command line options in this text field.

C# Organize Imports

SlickEdit® provides many features that work across several languages including C#, and C#-specific information is described throughout the documentation where applicable. The following are additional features designed specifically for C# developers:

- [Organize C# Imports](#)
- [Organize C# Imports Options Interface](#)

Organize Imports (Pro only)

Organize Imports automates the management of using statements in C# source files. This feature minimizes the amount of time that it takes to compile code by only importing the classes that are used. Existing using statements are also sorted in a readable format and are more consistent between different C# namespaces in the same project. Organizing of using statements is applied to an entire file.

To organize using statements, from the main menu, click **Tools** → **Imports** → **Organize Imports**, or from the right-click context menu, select **Imports** → **Organize Imports**. Alternately, use the **refactor_organize_imports** command.

Adding using statements

To add a using statement for the class name under the cursor in C# code, move the cursor to the class name you want to import, then from the main menu, click **Tools** → **Imports** → **Add using statement**, or from the right-click context menu, select **Imports** → **Add using statement**. Alternately, use the **refactor_add_import** command.

Goto using statement

To jump to the import statement for the class name under the cursor in C# source code, move the cursor to the class name, then from the main menu, click **Tools** → **Imports** → **Go to using statement**, or from the right-click context menu, select **Imports** → **Go to using statement**. Alternately, use the **refactor_goto_import** command. This command is helpful for identifying which namespace an unqualified class name comes from.

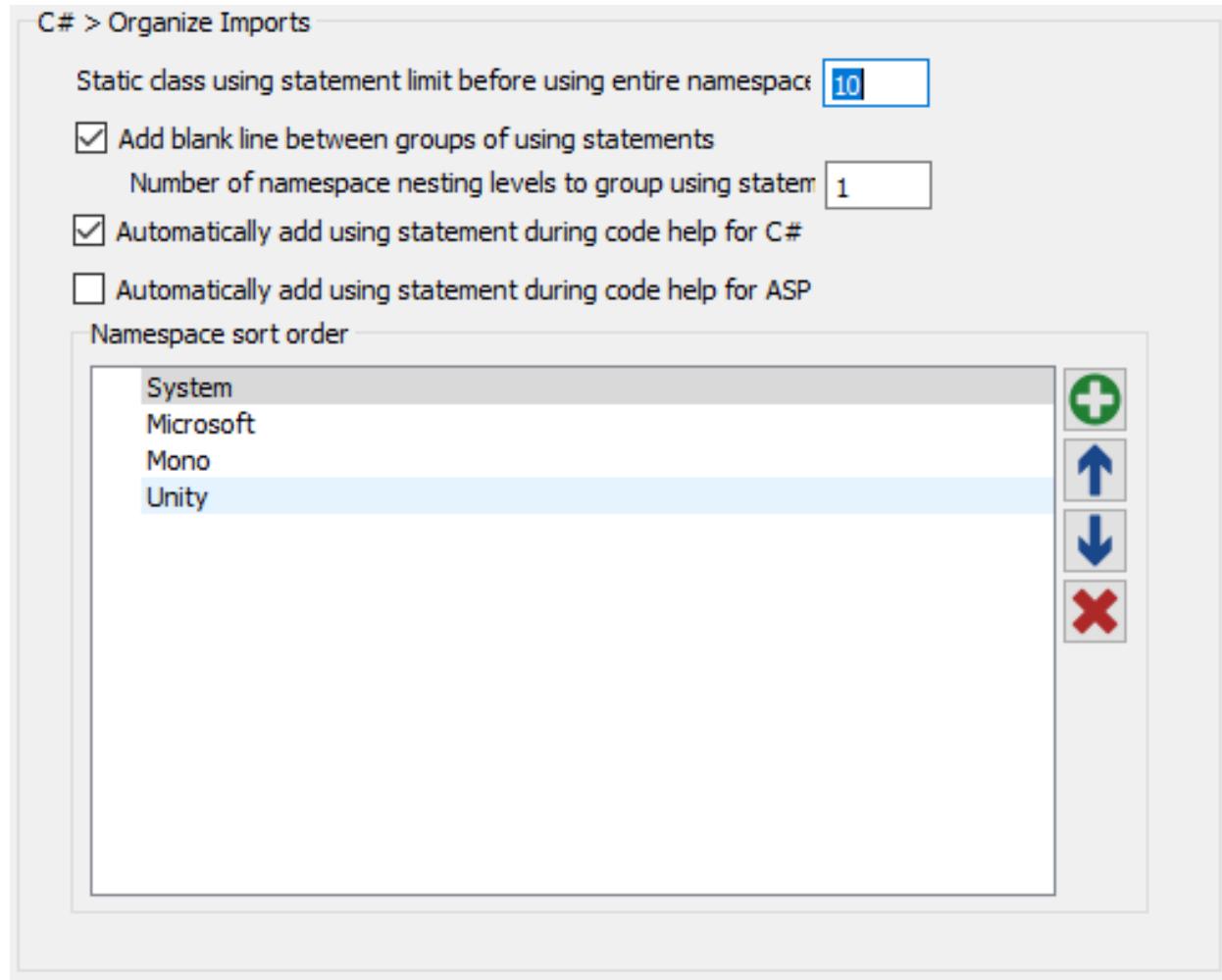
Import Options

Several options are available on the Options dialog to control the behavior of Organize Imports. See [Organize C# Imports Options Interface](#) for details.

Organize C# Imports Options Interface (Pro only)

Options are available on the Options dialog to configure the Organize Imports feature (**Tools** → **Options** → **Languages** → **Application Languages** → **C#** → **Organize Imports**). A more direct route to the options is to use one of the following methods:

- From the main menu, click **Tools** → **Imports** → **Options**
- Select **Tools** → **Imports** → **Options** from the right-click context menu in the editor window.
- Use the **refactor_organize_imports_options** command.



The following settings are available:

- **Static class using statement limit before using entire namespace** - If more than this number of classes are explicitly imported from the same namespace in one file, the imports will be replaced with a single namespace using statement.

- **Add blank line between groups of using statements** - Organize Imports will group using statements by namespace name or top-level namespace name. Select this option to force Organize Imports to add a blank line between these groups instead of having just one flat list of using statements.
- **Number of namespace nesting levels to group using statements by** - If this is set to **1**, using statements will be grouped by top-level namespace name only. For example, all your using statements from `System.` namespaces would be in a separate group from your using statements from `Microsoft.` namespaces. If set to **2**, using statements will be grouped by second level namespace names. For example, all your using statements from `System.Windows` would be in a separate group from your using statements from `System.Web`.
- **Automatically add using statement during code help for C#** - If selected, SlickEdit® will attempt to automatically add using statements as you edit C# code.
- **Automatically add using stement during code help for ASP** - If selected, SlickEdit will attempt to automatically add using statements as you edit C# code embedded in HTML. ASP using statements are added using the following notation: `<%@ page import="System.Web"%>`.
- **Namespace sort order** - This list specifies the order in which namespace groups are sorted. Use the **Ellipses (...)** button to add a new package. Use the **Up** and **Down** arrow buttons to move items. Use the **X** button to delete the currently selected namespace from the list.

XMLDoc Comments

- [Editing XMLDoc Comments](#)
- [XMLDoc Editor Dialog](#)
- [XMLDoc Beautifier Options Dialog](#)

Editing XMLDoc Comments

Several features are available to help you enter and format XMLDoc comments (as well as other documentation comment formats). See [Doc Comments](#) for more information.

XMLDoc Editor Dialog

Use the XMLDoc Editor to generate XMLDoc syntax comments for C#, Visual Basic, F#, Java, C, C++, and JavaScript. To access the XMLDoc Editor, choose **Document** → **Edit Dococumentation Comment**.

To add a custom or unsupported tag, append the tag with optional attributes and content into the **Description** text box.

For more information, see Microsoft's XMLDoc documentation at
<https://docs.microsoft.com/en-us/dotnet/csharp/codedoc>.

Note

NOTE SlickEdit® provides powerful capabilities to create and edit XMLDoc comments within the editor. See [Commenting](#) for more information.

XMLDoc Beautifier Options Dialog

To beautify XMLDoc comments or set up XMLDoc Beautifier options, first invoke the XMLDoc Editor by choosing **Document** → **Edit Documentation Comment**. Then click the **Options** button. The XMLDoc Beautifier Options dialog is displayed. The following settings are available:

- **Add blank line after parameter comment** - If checked, a blank line is added if a tag follows an <param> tag.
- **Add blank line after last parameter** - If checked, a blank line is added if a tag follows the last <param> tag.
- **Add blank line after return comment** - If checked, a blank line is added if a tag follows the <returns> 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 XMLDoc tag.
- **Add blank line after example** - If checked, a blank line is added if a tag follows the <example> tag.
- **Add blank line after remarks** - If checked, a blank line is added if a tag follows the <remarks> tag.

Pascal

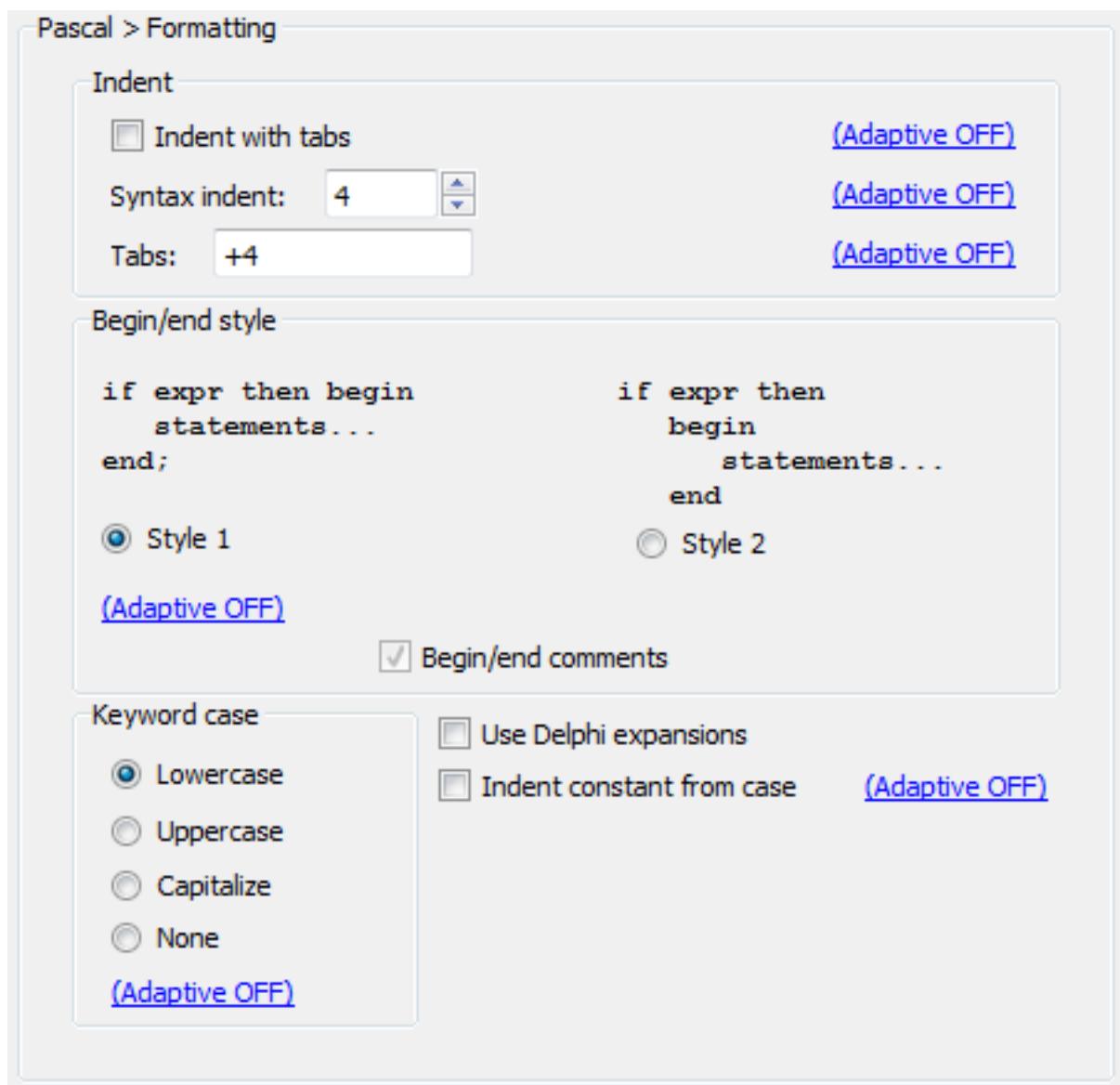
This section describes some of the advanced options that are available for Pascal.

Pascal Formatting Options

Options are available for Pascal for changing [Syntax Indent](#) and [Syntax Expansion](#) styles. To access these options, from the main menu, click **Tools** → **Options** → **Languages**, expand **Application Languages** > **Pascal**, then click **Pascal Formatting Options**.

Note

Languages similar to Pascal may have similar Formatting Options screens that are not specifically documented.



The following options are available:

- **Indent with tabs** - Determines whether **Tab** key, **Enter** key, and paragraph reformat commands indent with spaces or tabs. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Indenting with Tabs](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Syntax Indent](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **Begin-end style** - Specify the begin/end style used by [Syntax Indent](#) and [Syntax Expansion](#). The

hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. 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 Syntax Expansion. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **Indent constant from case** - Specifies whether constants of a case statement are indented or aligned to the case keyword. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **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.

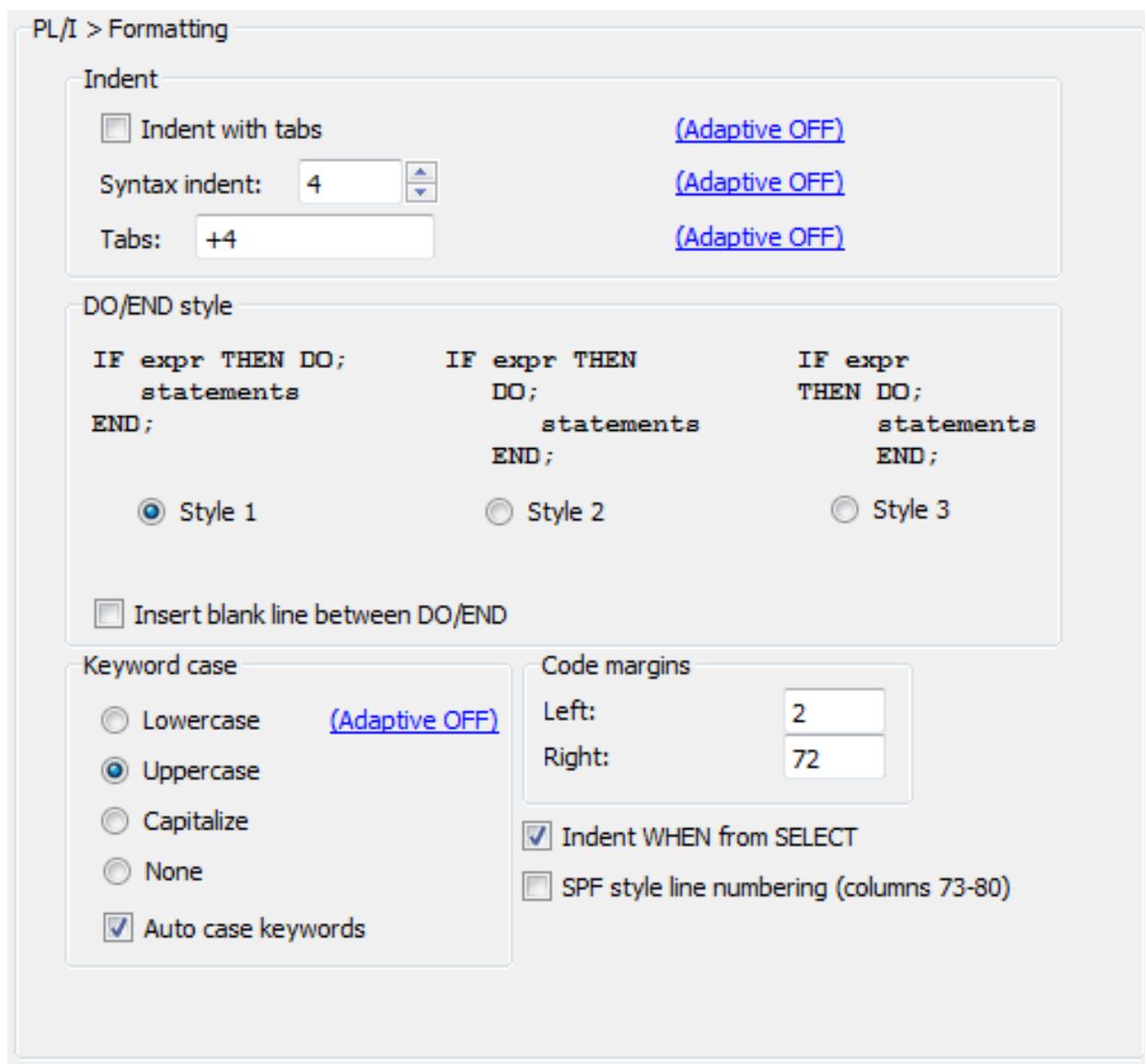
PL/I Formatting Options

Options are available for PL/I for changing [Syntax Indent](#) and [Syntax Expansion](#) styles. To access these options, from the main menu, click **Tools** → **Options** → **Languages**, expand **Mainframe Languages > PL/I**, then click **PL/I Formatting Options..**

Note

Languages similar to PL/I may have similar Formatting Options screens that are not specifically documented.

PL/I Formatting Options



The following options are available:

- **Indent with tabs** - Determines whether **Tab** key, **Enter** key, and paragraph reformat commands indent with spaces or tabs. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Indenting with Tabs](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Syntax Indent](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **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 Syntax Expansion. 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.
- **Code margins** - Indicates where the margins are for PL/I source. These values are set to 2 and 72 by default. Any code, comments, sequence numbers, or printer control characters outside of these margins will be ignored by the language support in SlickEdit. This setting will be overridden for a particular file should the file contain a preprocessor "MARGINS" statement.

Python

Python support includes many advanced features such as a built-in debugger, Context Tagging®, smart indenting, SmartPaste®, a beautifier, and project support. Those features are described elsewhere. This section describes some Python specific features not described elsewhere.

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 click **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 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.

Verilog and SystemVerilog

This section describes some of the advanced features and options that are available in SlickEdit® for Verilog and SystemVerilog, including language-specific formatting options, the Verilog Beautifier, and Verilog Preprocessing.

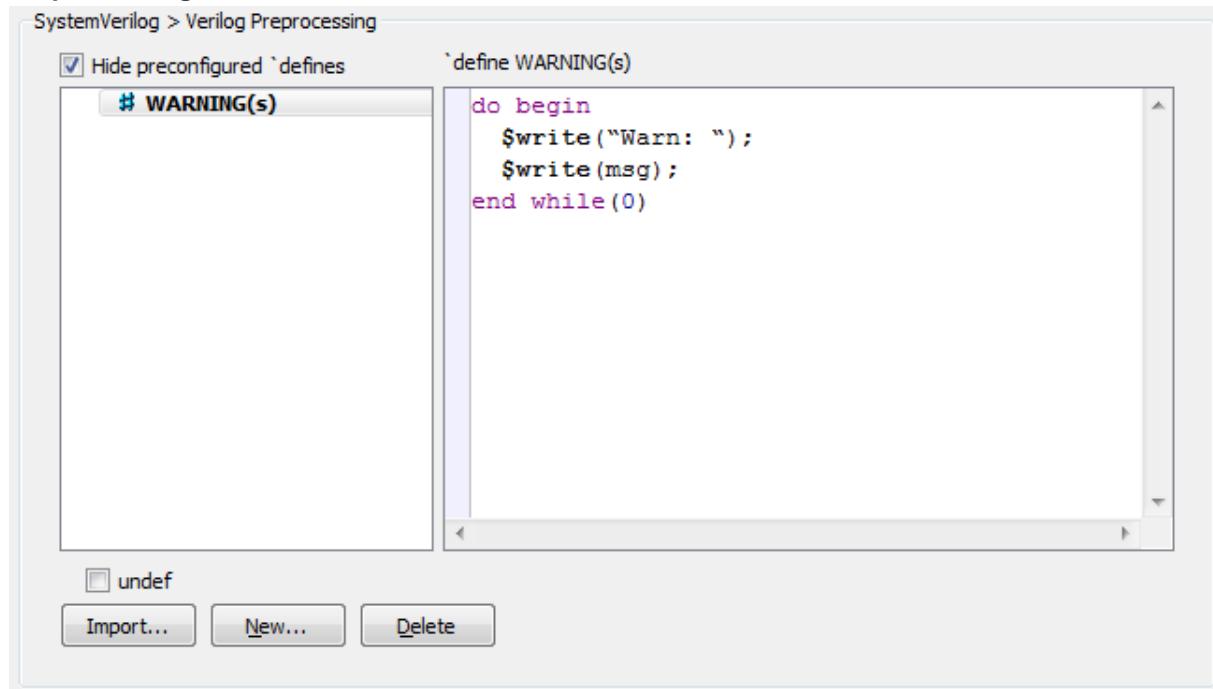
Verilog Beautifiers (Pro only)

The Verilog and SystemVerilog Beautifiers are described in the C and C++ language-specific options page. They work and are configured in the same manner using very similar dialogs. Refer to [Beautifiers](#) for more details.

Verilog 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 Verilog or SystemVerilog syntax can cause the parser to miss symbols. In addition, macros defined in Verilog or SystemVerilog frameworks, such as UVM need to be pre-configured.

1. From the main menu, click **Tools** → **Options** → **Languages** and expand the **Hardware Description Languages** node in the tree.
2. Depending on your language, select **Verilog** or **SystemVerilog** in the tree, then click **Verilog Preprocessing**.



3. Click **New** to add new preprocessing macros. Arguments are allowed; for example, **mymacro(a,b,c)**
4. When finished, click **OK**.
5. A prompt appears asking whether to rebuild your workspace tag file. Click **Yes**.

Preprocessor macros are stored in `userverilog.vh` and/or `usersystemverilog.svh`, 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 configuration file should only be used for `defines and `undefs not `includes.

Verilog Parsing Options

The Verilog parsing options allow you to fine-tune the behavior of the Verilog and SystemVerilog parsers in order to better handle parsing your code.

- **Parse `include files inline** - When set to On, the Verilog and/or SystemVerilog tagging parsers will recursively parse quoted include if the file is found relative to the current file or on the include path.

Note

Note that the parser does not look for bracketed include files because those are typically system headers and this feature is intended to provide more complete parsing of user code.

- **`include file search path** - Specifies a list of directories, in addition to the directory relative to the current file, to search for `include files.

XML and HTML

Features for XML and HTML are described below. See also [HTML and XML Beautifiers](#).

XML

XML features in SlickEdit® include Context Tagging® (Pro only), validation, well-formedness checking, a beautifier(Pro only), 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 [Language-specific Build Methods](#).

XML Validation

You can optionally configure SlickEdit to validate XML documents when opened, select **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Formatting** and check **Auto validate on open**.

You can manually check the validity of an XML document using the **xml_validate** command. You can manually check if the document is well-formed using the **xml_wellformedness** command. There are buttons on the XML toolbar for these two operations. See [XML Toolbar](#) for more information.

If the XML file being edited references a DTD or schema, SlickEdit will attempt to access it. This is used for validating the XML file and for color coding. You can customize this capability in the following ways:

- If the location of the DTD or schema is not accessible, you can map a local directory to that URL. From the main menu, select **Tools** → **Options** → **Network & Internet Options** → **URL Mappings**. Then add the URL for this DTD/schema and specify a directory where that file can be found.
- Even if you turn off auto-validation, the DTD/schema will be loaded for color coding. To prevent it from being loaded, you can add the extension of the file specifying the DTD/schema to the **def_xml_no_schema_list** variable. For more information, see [Setting/Changing Configuration Variables](#).

Tip

If you don't want to suppress the loading of DTDs or schema files for all files of a particular extension, you can define an empty DTD or schema and put it in a directory, then map the URL for that file to the directory as described above.

XML Toolbar

The XML toolbar is available for quickly accessing common XML operations. To display the XML toolbar, from the main menu, click **View** → **Toolbars** → **XML**. By default, three buttons are available:

- **Beautify selection or entire buffer** - Use this button to instantly beautify the current file according to

the Beautifier settings. See [HTML and XML Beautifiers](#) 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.

(Standard or Community only) XML Formatting Options

See [\(Standard and Community only\) HTML and XML Formatting Options](#).

XML Formatting Options (Pro only)

Content in XML and HTML files may be set to automatically wrap and format as you edit by turning on the **Beautify while typing** check box (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Editing**).

Many beautifier options including tag and attribute options are set in your XML beautifier profile. To access these options, from the main menu, click **Tools** → **Options** → **Languages**, expand **XML/Text Languages** > **XML**, then click **Formatting**. See [HTML and XML Beautifiers](#) for more information.

Tip

If you are currently editing an XML file, you can access your XML beautifier profile settings more quickly here (**Tools** → **Beautify** → **Edit Current Profile**).

XMLDoc Editor

Use the XMLDoc Editor to generate Microsoft XML syntax comments for C#, C, C++, Java, and JavaScript. 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.

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®(Pro only) 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 Mapping

SlickEdit® provides a way to map URLs to different locations. 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 elsewhere, 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. Similarly, if you wanted to test changes to a DTD without modifying every XML document's DTD references, you can map the URL to the test DTD location.

Optionally, you can specify a default lookup directory that contains all of your DTDs and namespace schemas files. Every mapping doesn't need to be explicitly configured. You can also create mappings for namespace URIs as well as DTD files.

To map a URL, complete the following steps.

1. From the main menu, click **Tools** → **Options** → **Network & Internet Options** → **URL Mappings**.
2. Click the **Add** button, and a new line opens.
3. In the **From** field, type in the URL that will be mapped to a different location.
4. Press Tab or click in the **To** field and type in the location to use for this URL.
5. Optionally, use the **Search directory** field to specify the default lookup directory for DTDs and namespaces. Files in this directory are searched prior to validation.
6. Click **OK** to save the changes and close the Options dialog.

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 language-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+J**.

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, click **View → Toolbars → HTML**.

To invoke a Web browser or to display the current file in a browser, use the **Web Browser** button 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, click **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 click **Tools → Options → Network & Internet Options → Web Browser Setup**. See [Web Browser Setup Options](#) for more information.

(Standard or Community only) HTML and XML Formatting Options

Various formatting options may be specified for XML and HTML files (**Tools → Options → Languages → [Language Category] → [Language] → Formatting**).

General Tab

- **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 since this is terminology we use elsewhere.
- **Indent with tabs** - Determines whether the **Tab** key and **Enter** key indent with spaces or tabs.
- **Tab size:** - Specifies the tab size for displaying tabs.
- **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.
- **Auto Symbol Translation** - Enables or disables auto symbol translations. Auto Symbol Translation automatically converts a character or sequence of characters to the appropriate entity reference, saving you from having to repeatedly guess at the correct entity or look up reference charts. This feature works

automatically as you type, so you don't need to press a special key or key sequence to trigger the translation. For example, type **>>**, and SlickEdit® automatically converts the **>>** sequence to **>**, which is the entity reference for the right angle bracket (**>**). Typing **&&** translates to **&**, the entity reference for the ampersand symbol (**&**).

- **Edit Translations** - Displays the Symbol Translation Editor dialog which is used to configure symbol aliases.

Tags Tab

- **Standalone** - 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 one 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.
- **End-tag** - When on, the selected tag has an end tag. For example, the tag **<TD>** has an 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>**.
- **Parent Tag** - Allows tag options to be set (inherited) from a parent tag. This is useful when many tags have the same options as a particular tag. When you change the options in the parent tag, all child tags are changed as well.

Case and Quoting Tab

- **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>**.

HTML Formatting Options (Pro only)

Content in XML and HTML files may be set to automatically wrap and format as you edit by turning on the **Beautify while typing** check box (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Editing**).

Many beautifier options including tag and attribute options are set in your HTML beautifier profile. To access these options, from the main menu, click **Tools** → **Options** → **Languages**, expand **Web Authoring Languages** > **HTML**, then click **Formatting**. See [HTML and XML Beautifiers](#) for more information.

Tip

If you are currently editing an HTML file, you can access your HTML beautifier profile settings more quickly here (**Tools** → **Beautify** → **Edit Current Profile**).

HTML and XML Beautifiers (Pro only)

To beautify an HTML or XML document, open the document you want to beautify, then from the main menu, click **Tools** → **Beautify** (or use the **gui_beautify** command). The HTML/XML Beautifier dialog will be displayed, which allows you to make settings for how the code will be beautified.

Tip

Content in XML and HTML files may be set to automatically wrap and format as you edit by turning on the **Beautify while typing** check box (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Editing**).

You can use the commands **beautify** or **beautify_selection** to instantly beautify the file or the selection according to the settings on the Beautifier dialog.

Note

The CFML and XSD beautifiers contains the same options and settings as the HTML and XML beautifiers.

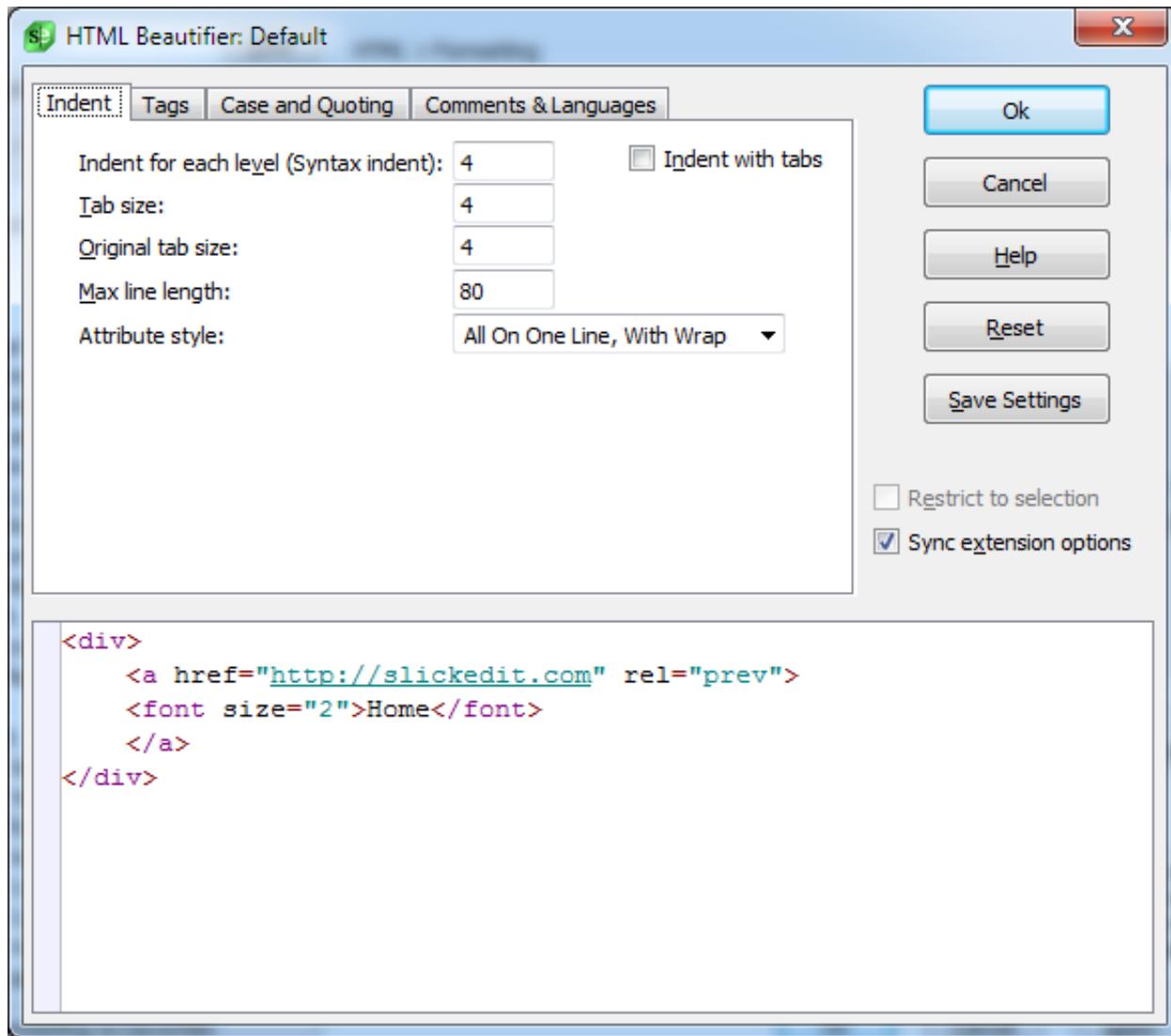
The following buttons and options are available on the Beautifier:

- **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 language 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:

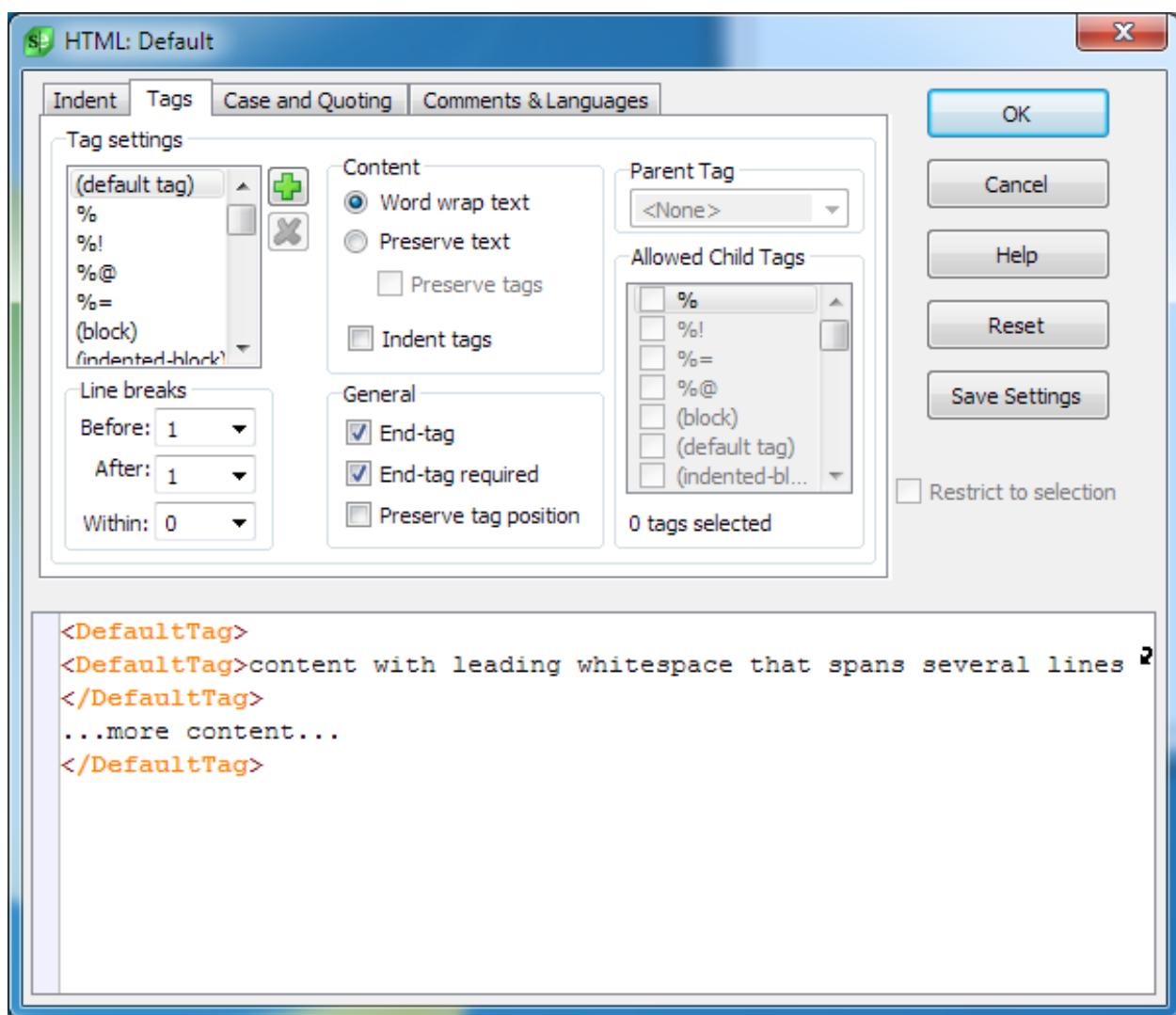
- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level. We have put the words "Syntax indent" in parenthesis to help indicate that this field has the same value as the **Syntax indent** text box on the language-specific **Formatting** options screen (see [Language-Specific Formatting Options](#)). 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 on the language-specific **Formatting** options screen (see [Language-Specific Formatting Options](#)).
- **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 on the language-specific **Formatting** options screen (see [Language-Specific Formatting Options](#)).
- **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.
- **Attribute style** - Specifies how attributes for tags are formatted. Specifies one the following:
 - **All On One Line, With Wrap** - Place tag and attributes on one line and wrap at the maximum line length.
 - **All On One Line** - Place tag and attributes on one line but don't wrap at the maximum line length.
 - **One Line If One Attr** - Place tag and attribute on one line if there is one attribute. Otherwise, place the each attribute on separate lines.
 - **Preserve Layout, Reindent** - Preserve line breaks and white space of attributes but reindent them.
 - **One Per Line** - Place each attribute on a separate line from the tag.
 - **Preserve Layout** - Preserve line breaks and white space for attributes

Tags Tab

The **Tags** tab on the HTML Beautifier is pictured below.



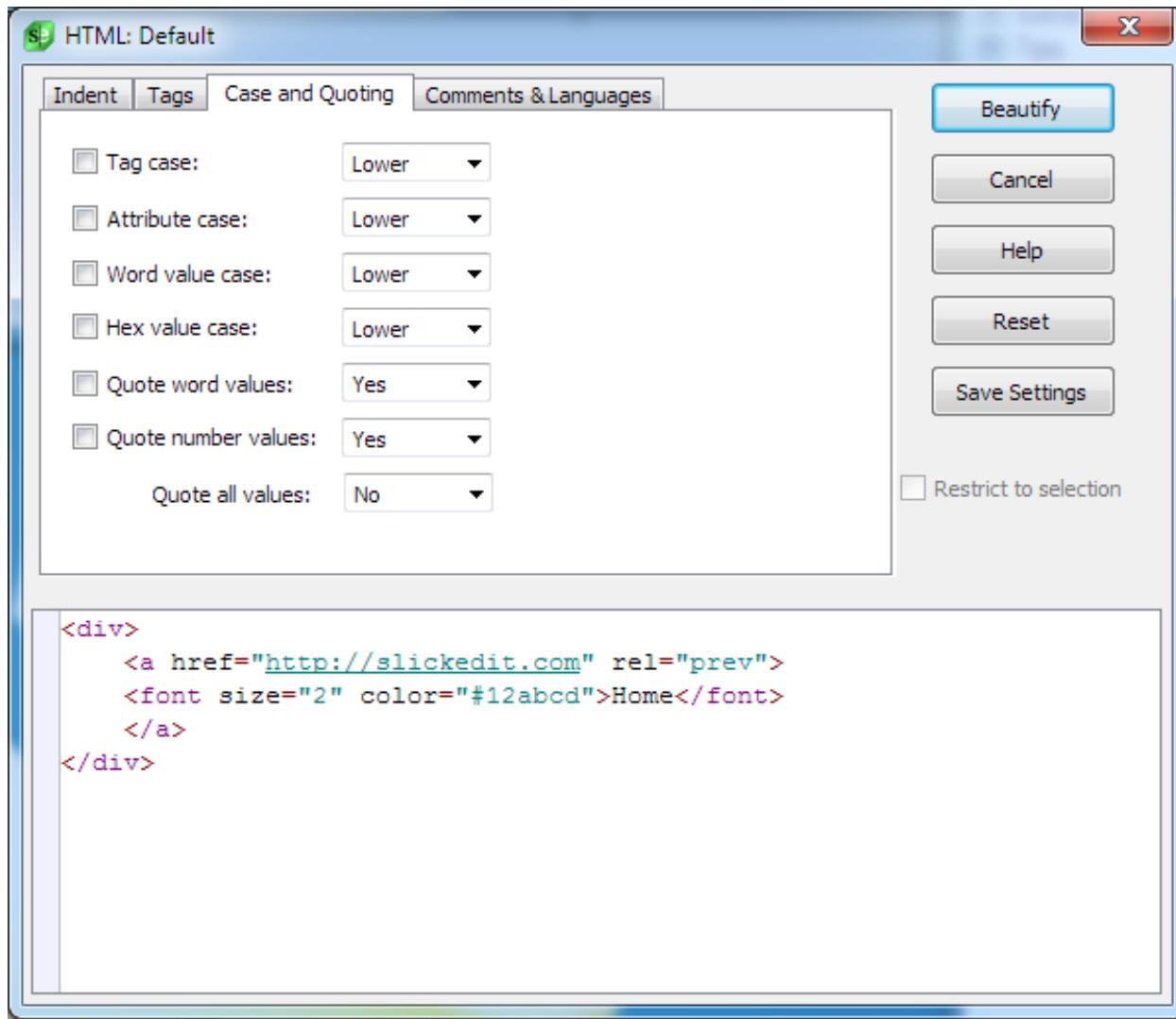
The **Tags** tab contains the following options and settings:

- **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.
- **+ Icon** - Displays the Add Tag dialog. This dialog allows you to add a tag definition to the list and specify how it will be beautified.
- **X Icon** - Removes selected tag.
- **Content** - Specify how to beautify content from the following options:
 - **Word wrap text** - When on, text inside tag is word wrapped
 - **Preserve text** - When on, text inside is not word wrapped.
 - **Preserve tags** - When on, child tags are preserved.

- **Indent Tags** - When on, child tags are indented.
- **End-tag** - When on, the selected tag has an end tag. For example, the tag <TD> has an 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 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** - Specify the number of line breaks before the opening tag.
 - After** - Specify the number of line breaks after the close tag.
 - Within** - Specify the number of line breaks after the opening tag and before close tag.
- **Parent Tag** - Allows tag options to be set (inherited) from a parent tag. This is useful when many tags have the same options as a particular tag. When you change the options in the parent tag, all child tags are changed as well.
- **Allowed Child Tags** - Since HTML tags may have an optional end-tag, these settings indicate which child tags are allowed inside the selected tag.

Case and Quoting Tab

The **Case and Quoting** tab of the HTML and XML Beautifier is pictured below.



The **Case and Quoting** tab contains the following settings:

- **Tag case** - Specifies how you want tags cased. For example, if you choose **Upper**, then `<body bgcolor="#ffffff">` would be beautified to `<BODY bgcolor="#ffffff">`.
- **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`

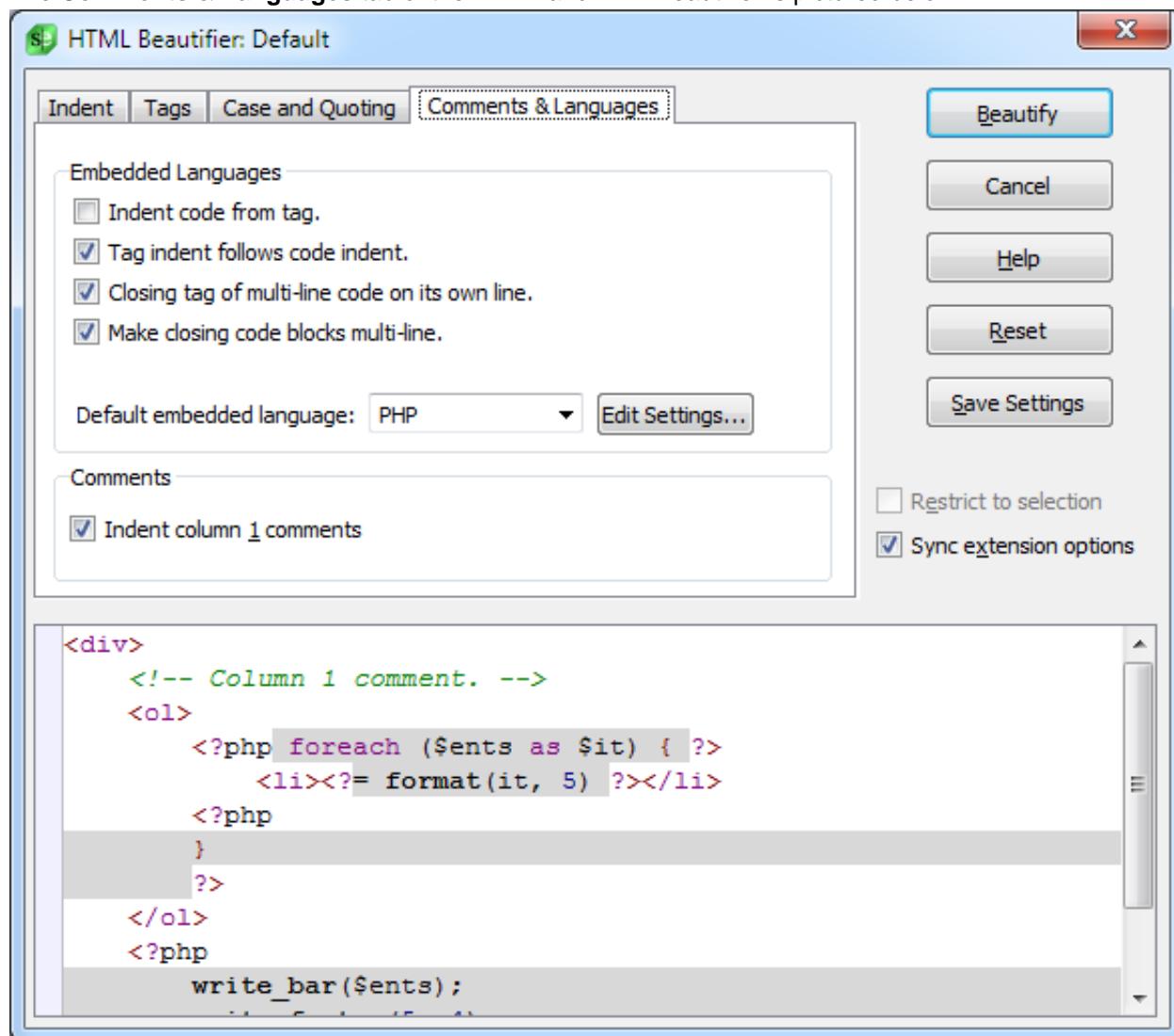
>. Select **Preserve** if you want word values left alone.

- **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">.

When the check box to the left of the labels above is unchecked, that item is preserved. For example, if **Tag case:** is unchecked, then the tag case will be preserved on beautification.

Comments & Languages Tab

The **Comments & Languages** tab of the HTML and XML Beautifier is pictured below.



The **Comments & Languages** tab contains the following options and settings:

- **Indent code from tag** - When on, indents the code from the start column of the enclosing tag. This only effects embedded languages which don't have beautifiers.
- **Tag indent follows code indent** - When on, the indent of tags is affected by the brace indent of surrounding embedded code. For the following snippet, it would control whether the `....` is indented from the foreach above it, or is left in the same column:

```
<?php foreach ($ents as $it) { ?>
    <li><?= format(it, 5) ?></li>
<?php
}
?>
```

- **Closing tag of multi-line code on its own line** - When on, for a tag like `?php` where the code is inside the angle brackets, the trailing `?>` will be placed on a line by itself.
- **Make closing code blocks multi-line** - When on, single line right brace like "`<? } ?>`" will be transformed to:

```
<?
}
?>
```

- **Indent column 1 comments** - When on, comments which start in column 1 are indented. Otherwise, they are left alone.

Auto Symbol Translation

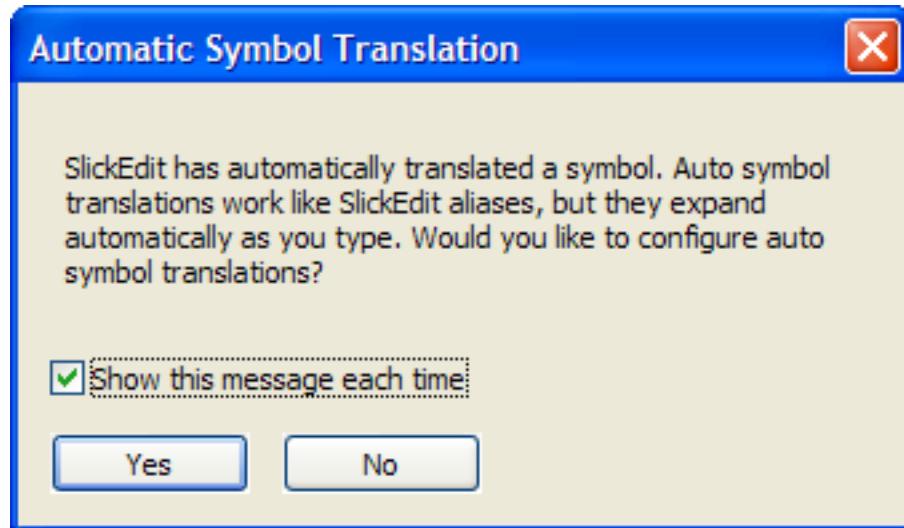
Auto Symbol Translation automatically converts a character or sequence of characters to the appropriate entity reference, saving you from having to repeatedly guess at the correct entity or look up reference charts. This feature works automatically as you type, so you don't need to press a special key or key sequence to trigger the translation. For example, type `>>`, and SlickEdit® automatically converts the `>>` sequence to `>`, which is the entity reference for the right angle bracket (`>`). Typing `&&` translates to `&`, the entity reference for the ampersand symbol (`&`).

Auto Symbol Translation uses the alias mechanism in SlickEdit to expand escape sequences in the text for each alias. SlickEdit comes with some predefined symbol aliases. You can view these, customize them, and create your own by using the Symbol Translation Editor dialog. The first time Auto Symbol Translation is triggered, a prompt appears that describes the feature and lets you open the Symbol Translation Editor.

Enabling/Disabling Auto Symbol Translation

Auto Symbol Translation is enabled by default. The first time you type a symbol alias and automatic translation occurs, a prompt is displayed that explains the feature and provides a button to access the

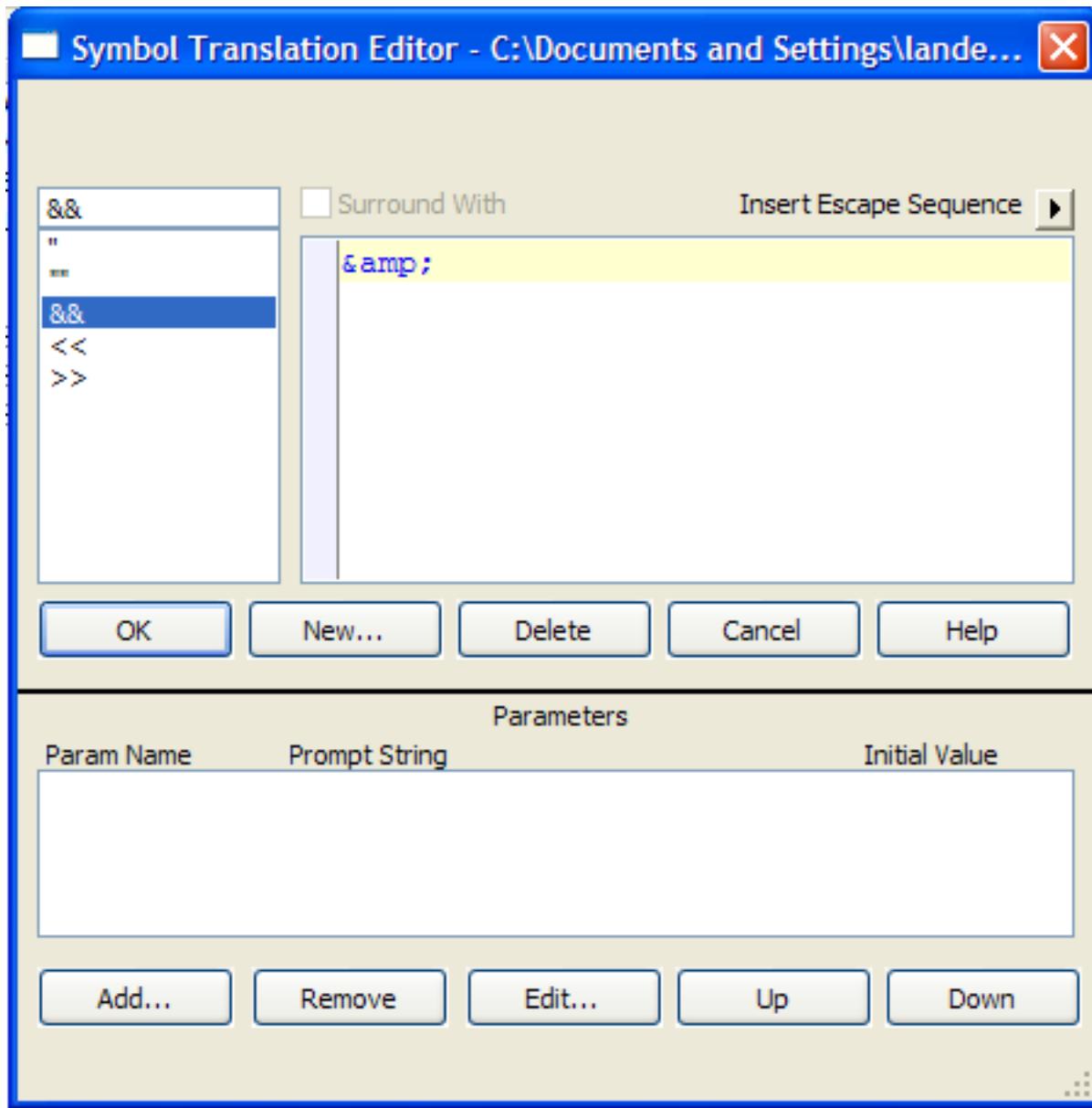
Symbol Translation Editor. You can also choose to prevent the prompt from appearing again in the future.



To turn off Auto Symbol Translation, from the main menu, click **Tools** → **Options**. Expand **Languages** and your language category, then select your language and click **[Language] Formatting**. Clear the **Auto symbol translation** option.

Configuring Symbol Aliases

The Symbol Translation Editor dialog is used to configure symbol aliases. It can be displayed by clicking **Yes** on the prompt that appears the first time a symbol translation occurs, or by clicking the **Settings** button on the Formatting Options screen for your language (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Formatting**).



The box on the left shows a list of symbol aliases. The edit window on the right contains the translation for the selected alias. Click **New** to enter a new symbol alias, or click **Delete** to remove a selected alias. Working with symbol aliases is the same as working with language-specific aliases. You can even use escape sequences and Parameter Prompting for some interesting translation results. See [Creating a Language-Specific Alias](#) for more information.

Outline View for XML

For XML documents, the Defs Tool Window also provides a more customizable way of representing any XML file. This allows you to set up rules for how you want to see any type of XML node that is presented in the tree. There are several ways that you can customize a particular element's appearance:

- Determining whether or not we want the element to be shown.
- Using static text, attribute values or the node's value itself to build a format string for each node type.

Formatting Rule Sets

Formatting rule sets define how each element in an XML file is displayed in the tree. Each rule in the set represents an XML element that may be found in the XML file, and each is assigned a formatting string. This formatting string works very similarly to formatting strings in programming; you may use static text, and you may also include replaceable text, or aliases, for values that come from each specific node. There are currently two aliases that are available in for a format string:

- (**%attribute**): This alias retrieves the value of a specific attribute on an XML node. For example, (%src) would retrieve the "src" attribute value.
- (**%\v**): This alias retrieves the actual value of the node between the tags.

For instance, if we have a **sect1** node like the following:

```
<sect1 xreflabel="Introduction">Hello world!</sect1>
```

and a format string defined for **sect1** like this:

```
Section 1 (%xreflabel) : "(%\v)"
```

then the XML Outline View representation for this **sect1** node would look like this:

```
Section 1 (Introduction) : "Hello world"
```

By controlling the way each XML element is displayed in the tree, we can create a much more readable presentation of the XML file.

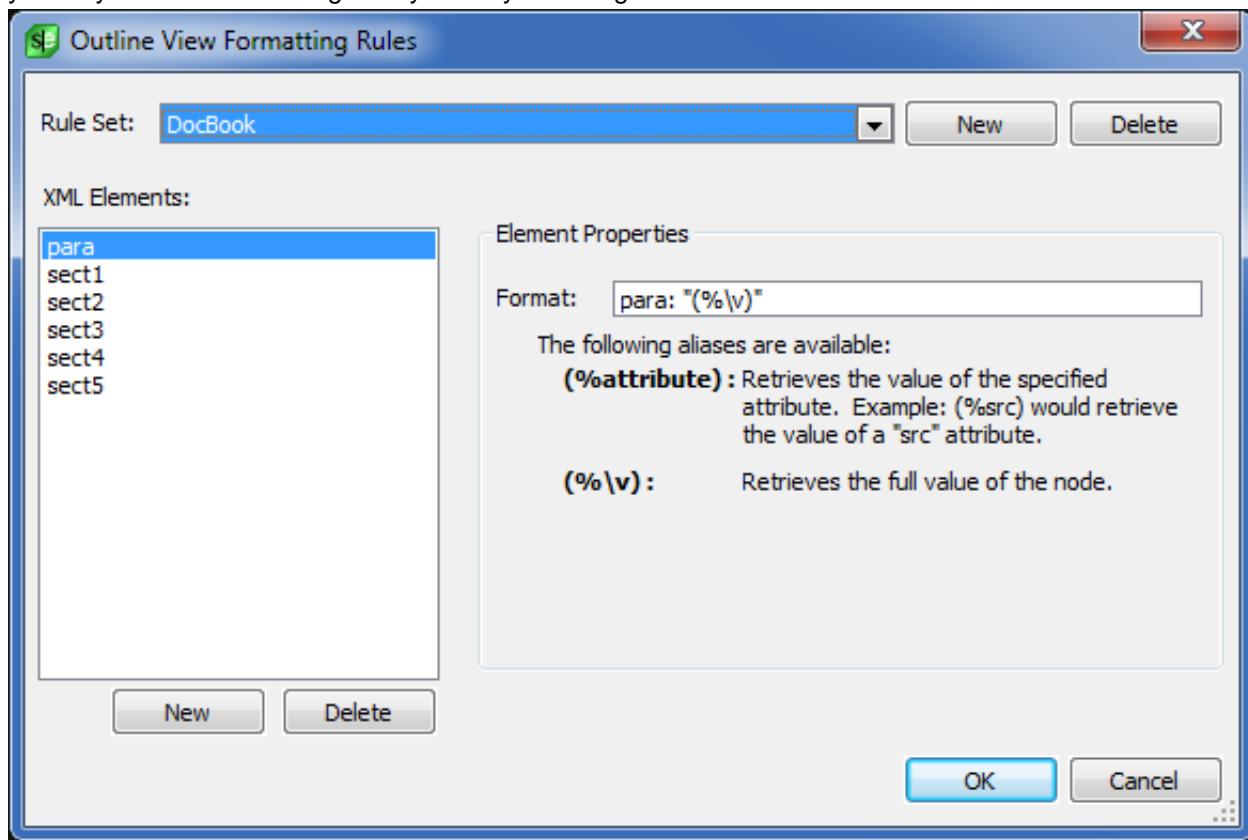
Activating the XML Outline View

To activate the XML Outline View, right click in the Defs Tool Window and select **Outline View → Use Outline View**. This will only be enabled if the current document is an XML document. If you have not used the Outline View before, and you open an XML document, you will be prompted for whether or not you would like to use it, and whether you would like to configure a set of formatting rules for the current document. If you don't want to see the XML Outline View anymore, you may simply right click the tree and uncheck **Outline View → Use Outline View** to toggle it off.

Formatting Rule Set Configuration

The following dialog allows you to set up the formatting rule set you want to use for any specific XML document. You may be prompted to set up formatting rules the first time you view an XML document, or

you may return to this dialog at any time by selecting **Outline View → Edit Format Rules**.



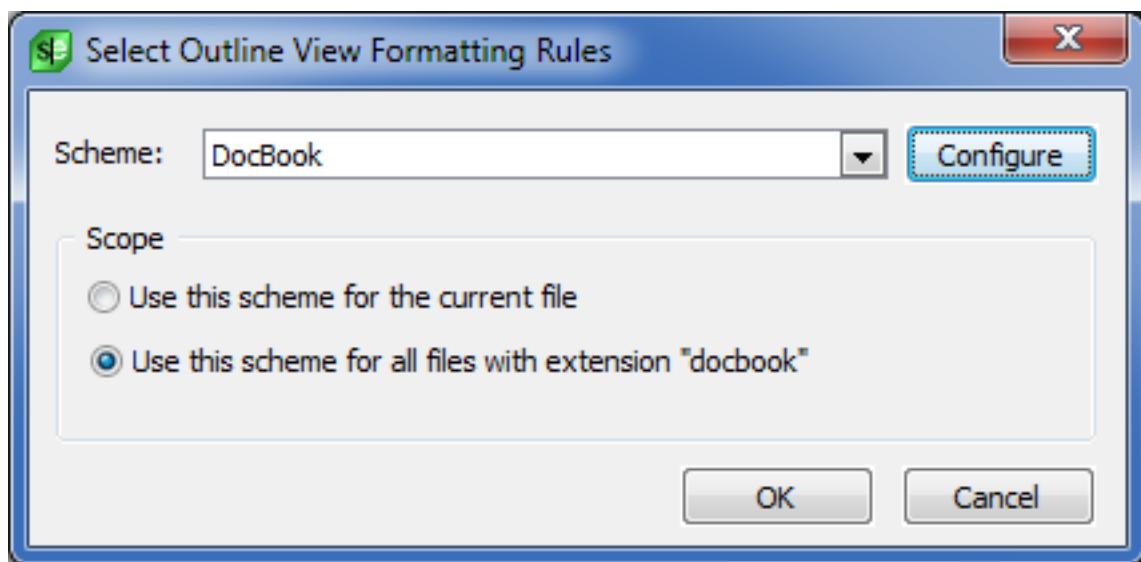
The current rule set is shown at the top of the dialog and represents the rule set you are currently editing. In the screen shot, which shows a rule set for docbook files, there is a rule for the **para** element, and a rule for the **sect1** to **sect5** elements.

To add a new element to the current rule set, click the **New** button underneath the XML element list. You will be prompted to select the name of the element from a combo box (which is populated with unique element names from the current document) or you can enter your own name if it's not in the list. Next, you will enter a formatting string for the element. You may also delete the current rule by clicking the **Delete** button underneath the XML element list. If an element is not in the list, it will not be represented in the XML Outline View.

If you would like to create a new formatting rule set, you may click the **New** button next to the rule set drop down. You will be prompted to enter the name of the new rule set, and then you may begin adding rules to it. If you no longer want a rule set, you may delete it by clicking the **Delete** button next to the rule set drop down.

Applying Formatting Rules to XML Files

Once you have a formatting rule set to apply to one or more of your XML files, you can make that assignment by right clicking the tree and selecting **Outline View → Select Format Rules**. This will bring up the following dialog:



Here, you may select the formatting rule set that you wish to use with the current file. You may also select the scope to which it is applied. You may:

- Use the scheme for the current document.
- Use the scheme for all XML files with the same extension as the current document.

From this dialog, you may also click the **Configure** button to go back to the [Formatting Rule Set Configuration](#) dialog.

Tools and Utilities

This chapter describes the tools and utilities provided by SlickEdit that help while coding.

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).
- If you like non-GUI, no focus change, old school (SlickEdit/Emacs) incremental searching see [Incremental Searching](#) below.
- Use the Mini Find and Replace dialog (see [Mini Find and Replace Dialog](#) below) if you like incremental searching with a small GUI similar to a web browser.
- 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](#), [Undoing/Redoing Replacements](#), and [Match Highlighting](#).

Default Search Options

The behavior for all of the search mechanisms in SlickEdit® is controlled by the Search Options located in the Options dialog (**Tools** → **Options** → **Editing** → **Search**). The options specified here are used each time a search is performed, except for the [Find and Replace Tool Window](#), which has controls to override these settings.

The value of the settings in the Options dialog are used to initialize the corresponding controls in the Find and Replace tool window. Once the value is changed in the tool window it is remembered and used the next time the Find and Replace tool window is launched. So changing a value in the Search Options may not have any effect on the Find and Replace tool window. Changing a setting in the Find and Replace tool window will not change the settings in the Search Options. See [Search Options](#) for more information.

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 context 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 Search always uses the default search options (see [Search Options](#)).

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. Quick Replace always uses the default search options (see [Search Options](#)).

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**). Incremental Search always uses the default search options (see [Search Options](#)).

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. See Find Symbol Tool Window or Regular Expressions for more information.
Ctrl+W	Toggle word searching on and off. To change the word characters for a specific language, use the Word chars field on the language-specific General options screen (see Language-Specific General Options).

Keys	Function
Ctrl+Shift+W	Copy complete word at cursor to search string.
Ctrl+C	Toggle case sensitivity. The key bound to the Brief emulation command case_toggle 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.

Mini Find and Replace Dialog

The Mini Find and Replace dialog performs incremental searching and highlights matches as you typed. Unlike the non-GUI incremental searching ([Incremental Searching](#)), this dialog offers many of the advanced find and replace features found in the much larger Find and Replace tool window (see [Find and Replace Tool Window](#)) including replace, matching based on color coding, multi-file searching, and multi-file replace. The Find and Replace tool window has more features which are especially useful for multi-file search/replace operations.

Perform the following steps to configure the new Mini Find/Replace dialog to work like the big Find and Replace Tool window:

- Turn on **Close on Enter Key** (**Tools** → **Options** → **Editing** → **Search**)
- Display the mini Find/Replace dialog (Ctrl+F or **Search** → **Find**). click the down arrow menu button to the right of the "< >" buttons. Uncheck **Incremental Search** and **Search Highlighting**.

Note

If you prefer your key bindings for gui_find and gui_replace (and Find/Replace menu items) to display the Find and Replace tool window, set your **Default find and replace GUI** to **Find and Replace tool window** (**Tools** → **Options** → **Editing** → **Search**)

The following key combinations are useful hot keys for the Mini Find and Replace dialog:

Note

Mac: The **Alt** hot keys can be supported on macOS but you must set "Mac Option/Alt key behavior" to "Use as Windows-style Alt key modifier" (**Tools** → **Options** → **Keyboard and Mouse** → **Advanced**). Keep in mind that you can't enter extended ASCII characters using the **Alt** shift key once you make this change.

Keys	Function
Esc	Dismiss Mini Find.
Enter	Find next match.
Shift+Enter	Find previous match.
Alt+C, Command+E	Toggle search case sensitivity on/off.
Alt+W, Command+W	Toggle whole word searching on/off. To change the word characters for a specific language, use the Word chars field on the language-specific General options screen (see Language-Specific General Options)
Alt+T, Alt+U, Command+T, Command+U	Toggle regular expression pattern matching on/off. See Find Symbol Tool Window or Regular Expressions for more information.
Ctrl+F, Command+F	Collapse mind find dialog or display the Find and Replace tool window. This is not a hard coded key binding. When you press the key bound to the gui-find when the Mini Find and Replace dialog has focus, the Find and Replace tool window is

Mini Find and Replace Dialog

Keys	Function
	displayed.
Ctrl+W	Copy current word at cursor to search string.
Ctrl+Shift+Space	Copy (complete) more word text at cursor to search string.
Alt+0-9, Command+0-9	Find all occurrences of the search string and output results to corresponding search tab 0-9.
Alt+O, Command+O	Toggle color coding search on/off.
Alt+Shift+O, Command+Shift+O	Display color coding settings menu.
Ctrl+R, Command+N	Expand mind find dialog or display Find and Replace tool window. This is not a hard coded key binding. When you press the key bound to the gui-replace when the Mini Find and Replace dialog has focus, the Find and Replace tool window is displayed.
Alt+R, Command+R	Replace next
Alt+A, Command+A	Replace all occurrences
Alt+I, Command+I	Preview replace all occurrences
Alt+V, Command+J	Toggle preserve case replace on/off.
Alt+P, Command+P	Toggle wrap search on/off.
Alt+B, Command+B	Toggle backward search on/off.
Alt+H, Command+H	Toggle hidden text search on/off.
Alt+L, Command+L	List all occurrences using last search output tab.
Alt+Shift+L, Command+Shift+L	List all occurrences using new search output tab.
Alt+Shift+I, Command+Shift+I	List all occurrences using auto increment search output tab.
Alt+G, Command+Q	Highlight all matches.

Keys	Function
Alt+M, Command+M	Bookmark all matches.
Alt+S, Command+S	Set multiple cursors.
Alt+F, Alt+Enter, Command+D	Display Find tools menu.
Alt+Shift+H, Command+Shift+Q	Toggle highlight replaced text
Alt+Shift+U, Command+Shift+U	Toggle list replace matches
Alt+K, Command+K	Set focus in Look in combo box (i.e [<Current Buffer>]).
Alt+Shift+R, Command+Shift+R	Display regex menu
Alt+Shift+K, Command+Shift+K	Keep matching lines.
Alt+Shift+X, Command+Shift+X	Delete matching lines.
Ctrl+\, Ctrl+Shift+\, Command+\, Command+Shift+\	Cycle next/prev item in Look in combo box (i.e [<Current Buffer>]).
Ctrl+/, Command+/-	Toggle incremental search highlighting.

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 / command is:

```
/SearchString[/OptionCharacters]
```

The syntax of the **find** or **I** command is:

```
find /SearchString[/OptionCharacters]
```

```
1 /SearchString[/OptionCharacters]
```

The **find** or **I** command is typically used when your search string contains a / character and you need to change the delimiters. The first non-blank character is used as the delimiter. The example below uses \$ as the delimiter:

```
1 $SearchString[$OptionCharacters]
```

Note

When changing the delimiter, there must be a space between the command name (**find** or **I**) and the delimiter. Otherwise, no space is needed

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. See Find Symbol Tool Window or Regular Expressions for more information.
L	Interpret search string as Perl regular expression. See Find Symbol Tool Window or Regular Expressions .
~	Interpret search string as Vim regular expression. See Find Symbol Tool Window or Regular Expressions .
U	Interpret search string as Perl regular expression. Unix syntax regular expressions are no longer supported.
B	Interpret string as a Perl regular expression. Brief syntax regular expressions are no longer

Option Character(s)	Description
	supported.
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 language, use the Word chars field on the language-specific General options screen (see Language-Specific General Options).
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.
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.

Find and Replace Commands

Option Character(s)	Description
X <i>CCLetters</i>	<p>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:</p> <ul style="list-style-type: none"> • O - Other • K - Keyword • N - Number • S - String • C - Comment • P - Preprocessing • L - Line number • 1 - Symbol 1 • 2 - Symbol 2 • 3 - Symbol 3 • 4 - Symbol 4 • F - Function color • V - No save line • A - Attribute
C <i>CCLetters</i>	<p>Requires the first character of search string to be one of the color coding elements specified. See <i>CCLetters</i> above.</p>
*	<p>Used with the "Search hidden text" (H) or "Highlight matches" (#) options to find all matches and un-hide or highlight them.</p>
&	<p>Use Wildcard regular expression syntax (*, ?).</p>
#	<p>Highlight matched patterns with highlight color.</p>

Option Character(s)	Description
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.
\$	(Replace commands only) Replaced occurrences are highlighted with modified color.

If you don't specify *OptionCharacters* when using the **find**, **I**, and **/** commands, the default search options are applied. See [Search Options](#) for more information.

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 /*SearchString*/*ReplaceString*[/*OptionCharacters*]

replace /*SearchString*/*ReplaceString*[/*OptionCharacters*]

The available *OptionCharacters* are the same as for the **find**, **I**, and **/** commands (see [Find and Slash \(/\) Commands](#) above).

The first non-blank character is used as the delimiter. The example below uses \$ as the delimiter:

c \$*SearchString*\$*ReplaceString*[\$*OptionCharacters*]

Note

When changing the delimiter, there must be a space between the command name (**c** or **replace**) and the delimiter. Otherwise, no space is needed

You can perform one of the following actions with the replace command (**c**) by pressing the corresponding key:

Find and Replace Commands

Key	Action
Y or Space	Make change and continue searching.
N or Backspace	No change and continue searching.
L or Dot	Make change and stop searching.
G or !	Make change and change the rest without prompting.
Q or Esc	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 click Tools → Options → Editing → Search and set the Restore cursor after replace option to True .
Ctrl+G	Exit command and restore cursor to its original position.
Ctrl+R	Search in reverse for next occurrence of search string.
Ctrl+S	Search forward for next occurrence of search string.
Ctrl+T	Toggle regular expression pattern matching on/off. The key bound to the Brief emulation command re_toggle will also toggle regular expression pattern matching.
Ctrl+W	Toggle word searching on/off. To change the word characters for a specific language, use the Word chars field on the language-specific General options screen (see Language-Specific General Options).
Ctrl+C	Toggle case-sensitivity. The key bound to the Brief emulation command case_toggle will also toggle the case-sensitivity.
Ctrl+M	Toggle searching within selection.
F1 or ?	Display Help on Find and Replace tool window.

Replace Command Search Examples

The table below provides examples of using command line replace.

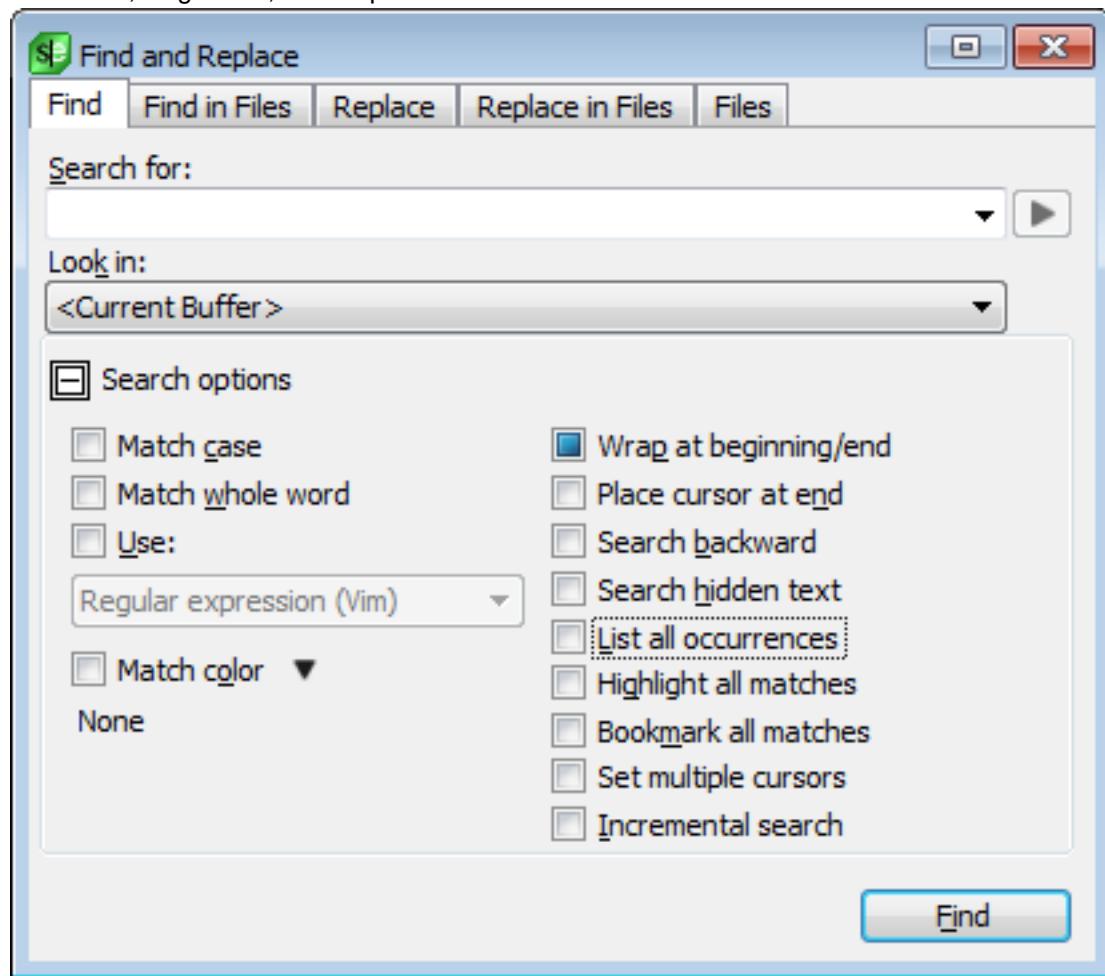
Example	Description
c \$\$\\$	Replace occurrences of forward slashes with back slashes.
c/x/y/m	Replace occurrences of "x" in the selected area with "y" using default search case sensitivity.
c \$x\$y\$m	Replace occurrences of "x" in the selected area with "y" using default search case sensitivity. The 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	Replace occurrences of the word "test" with the word "temp", with the case preserved. For example: <ul style="list-style-type: none">Occurrences of "Test" are replaced with "Temp".Occurrences of "test" are replaced with "temp".Occurrences of "tesT" are replaced with "TEMP" (because a mixed case will retain the actual replacement that you typed).Occurrences of "TEST" are replaced with "TEMP".

Find and Replace Tool Window

You can use the Find and Replace tool window (**Ctrl+F**, **Search → Find**, or **View → Tool Windows → Find and Replace**) to specify search and replace options and conduct search and replace operations on

Find and Replace Tool Window

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 cleared or until one or more other elements are selected. Putting a check in any check box essentially clears 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 clear 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.
- **Cleared** - 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**. See [Search Options](#) for more information. For information on the individual options on the Find and Replace tool window, see [Search Dialogs and Tool Windows](#).

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 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.

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, symbol pattern, 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 [Search Dialogs and Tool Windows](#) provides this same capability, it cannot match the speed of Find Symbol, nor restrict its results just to symbol definitions and declarations.

See [Find Symbol Tool Window](#) for information about the options that are available.

Find and Replace with Regular Expressions

SlickEdit® supports three types of regular expression syntax that you can use for finding and replacing when regular search and replace operations are too limiting:

- [Perl Regular Expressions](#)

- [SlickEdit® Regular Expressions](#)
- [Vim Regular Expressions](#)
- Wildcards

See [Regular Expressions](#) for more information.

Undoing/Redoing Replacements

To undo a replacement, click **Edit** → **Undo**, press **Ctrl+Z**, or use the **undo** command. To redo a replacement, click **Edit** → **Redo**, press **Ctrl+Y**, or use the **redo** command.

To undo replacements in multiple files, click **Edit** → **Multi-File Undo** or use the **mfundo** command. To redo replacements in multiple files, click **Edit** → **Multi-File Redo** or use the **mfredo** command.

Match Highlighting (Pro only)

Cursor on Symbol Shows All Uses in File

SlickEdit® can highlight all occurrences of the current symbol under the cursor. This makes it easy to see, at a glance, all uses of a symbol in a file. This option can be set on a language-specific basis. To enable it, from the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Context Tagging®**. On the options screen, select **Highlight matching symbols under cursor**.

The highlight color is controlled by the **Symbol Highlight** screen element (**Tools** → **Options** → **Appearance** → **Colors**). To change the color, see [Setting Colors for Screen Elements](#). See [Language-Specific Context Tagging® Options](#) for information about other options on the options screen.

This feature includes two advanced options that affect all languages that have **Highlight matching symbols** enabled. These can be set through configuration variables (**Macro** → **Set Macro Variable**):

- **def_highlight_symbols_max_bufsize** - This variable sets the maximum buffer size, in bytes, for symbol highlighting. If the buffer size is greater than the max size, highlighting is restricted to the visible lines.
- **def_highlight_symbols_max_matches** - This variable sets the maximum number of matching occurrences for symbol highlighting.

See [Setting/Changing Configuration Variables](#) for more information.

Comparing and Merging

SlickEdit® provides several ways to compare and merge files:

- [DIFFzilla®](#)
- [3-Way Merge](#)
- [The Compare Command](#)

DIFFzilla

DIFFzilla® provides powerful differencing capabilities that let you compare files or directories(Pro only) and view the differences side-by-side.

(Pro only) 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.

Dynamic Difference Editing (Pro only)

Unlike most diff tools, DIFFzilla® allows you to edit your code while viewing differences. Undo, copy/paste, Syntax Expansion/indenting, SmartPaste®, Auto List Members, Auto Parameter Info and many emulation key mappings work when editing in DIFFzilla. When you type or make any edit, lines are re-diffed (compared again) so that you can view the new intra-line differences easily.

Source Diff (Pro only)

With Source Diff, DIFFzilla ignores whitespace and carriage returns when comparing two files. This allows you to see real differences in the code while ignoring differences in formatting. For example, look at the two code samples, below. They are identical, except for the brace style used. Most diff tools will tell you that they are different. **Source Diff** will tell you that these two are the same.

```
Rectangle::Rectangle() {  
}
```

```
Rectangle::Rectangle()  
{  
}
```

Using **Source Diff**, DIFFzilla presents the Path 2 file with its formatting adjusted to match that of the Path 1 file. We insert stream markers to indicate whitespace that was added or removed. We make a copy of the file specified in Path 2, so no actual changes to the file are made.

The screenshot shows a comparison between two C++ files. The left file, titled 'title', contains code with a brace adjustment highlighted by a red circle. The right file, titled 'x', shows the corrected code where braces are on separate lines. The DIFFzilla interface includes tabs for 'Read only' status, line numbers (4/85 and 5/86), and various diffing options like 'Source Diff', 'Next Diff', and 'Prev Diff'.

In this screen shot, you can see a green squiggle (highlighted in the red circle) that indicates where the formatting was adjusted. The file on the right is actually formatted with the braces on separate lines, but **Source Diff** adjusts them to match the formatting of the file on the left. These adjustments are skipped by **Next Diff** and **Prev Diff**, allowing you to focus on meaningful differences, like the extra definition in the file on the right.

Source Diff also has the ability to ignore specific changes at the token level. For example, if you rename several symbols, you can set up a token mapping table to tell **Source Diff** to ignore those differences. This makes it easier to focus on the significant changes in your source code. See [File Compare Options](#) to learn how to configure this option.

Tip

For example, suppose you made the following changes, you can set up a token mapping as seen in the table below:

Modification	Left
Renamed the symbol Rect to Rectangle	Rectangle
Renamed a local variable r to radius	radius
Renamed a local variable d to diameter	diameter
Cleaned up code to use nullptr instead of NULL	nullptr
Cleaned up code to use nullptr instead of 0	nullptr

Modification	Left
Changed several variables from <code>int</code> type to <code>bool</code>	<code>bool</code>
Cleaned up code to use <code>true</code> instead of <code>1</code> for booleans	<code>true</code>
Cleaned up code to use <code>false</code> instead of <code>0</code> for booleans	<code>false</code>

Using DIFFzilla

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](#).

- [Comparing Two Files](#)
- [Comparing Two Folders](#)
- [Comparing Symbols](#)
- [Generating File Lists](#)
- [Automatic Directory Mapping](#)
- [Diffing File History](#)
- [Launching DIFFzilla from the Operating System](#)

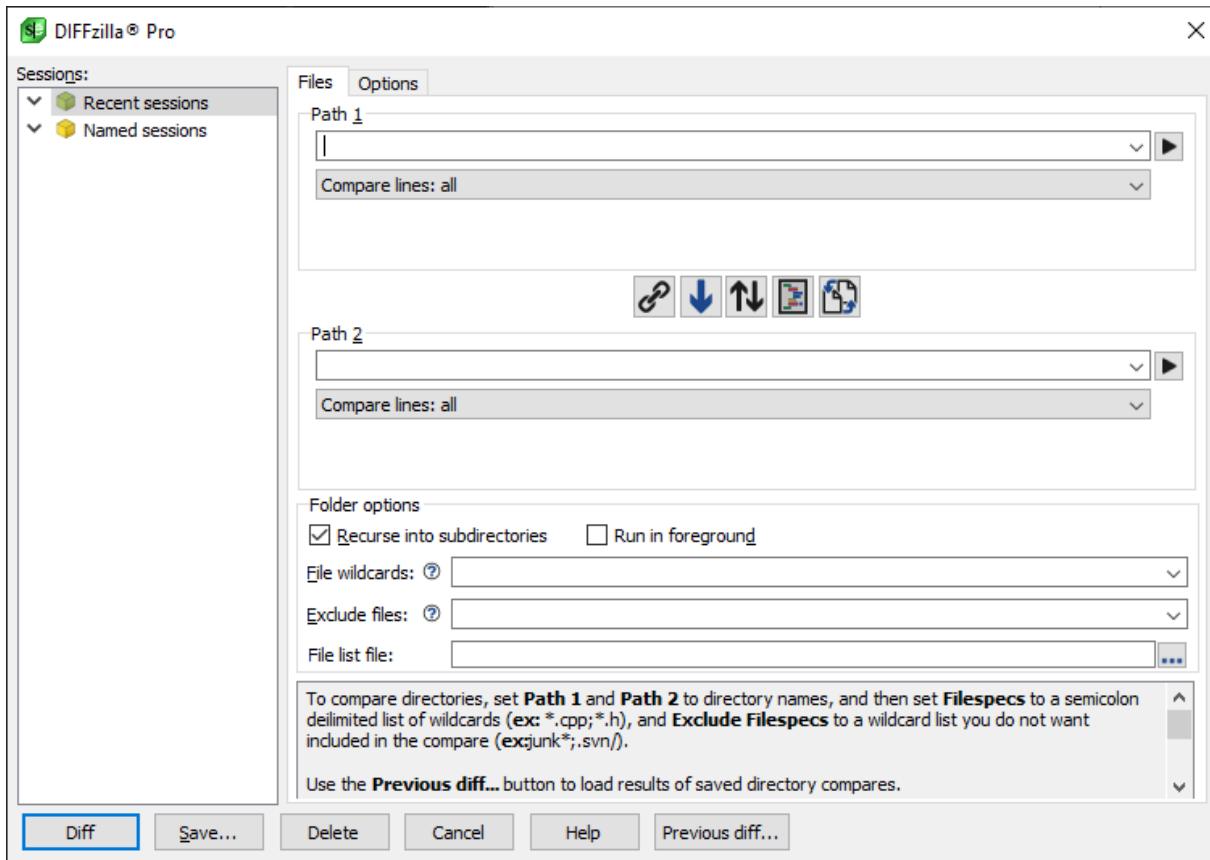
Launching DIFFzilla from the Operating System

DIFFzilla can be launched from the operating system using the **vsdiff** executable. This command actually launches SlickEdit but in a mode where just DIFFzilla is visible. This command includes an implicit **+new** option so that it will not interfere with existing instances of SlickEdit already running. For more information see [Running Multiple Instances](#).

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 **Compare Two Files** 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 Two Directories (Pro only)

You can difference two source trees to determine what files have been added or removed and generate a list of file names. 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, click **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. Alternately, set the **File list file:** text box with a list of relative filenames you want to diff.
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 tool tip 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](#).

Comparing Symbols (Pro only)

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, as some symbol blocks are not yet picked up correctly.

To diff a selected range of lines from two source files, complete the following steps:

1. From the main menu, click **Tools** → **File Difference**.
2. Select the **Compare Two Files** diff type.
3. Enter the first file in the **Path 1** text box.
4. Select **Compare symbols: all**, in the second drop-down list.
5. Enter the second file in the **Path 2** text box.

Comparing Parts of Files

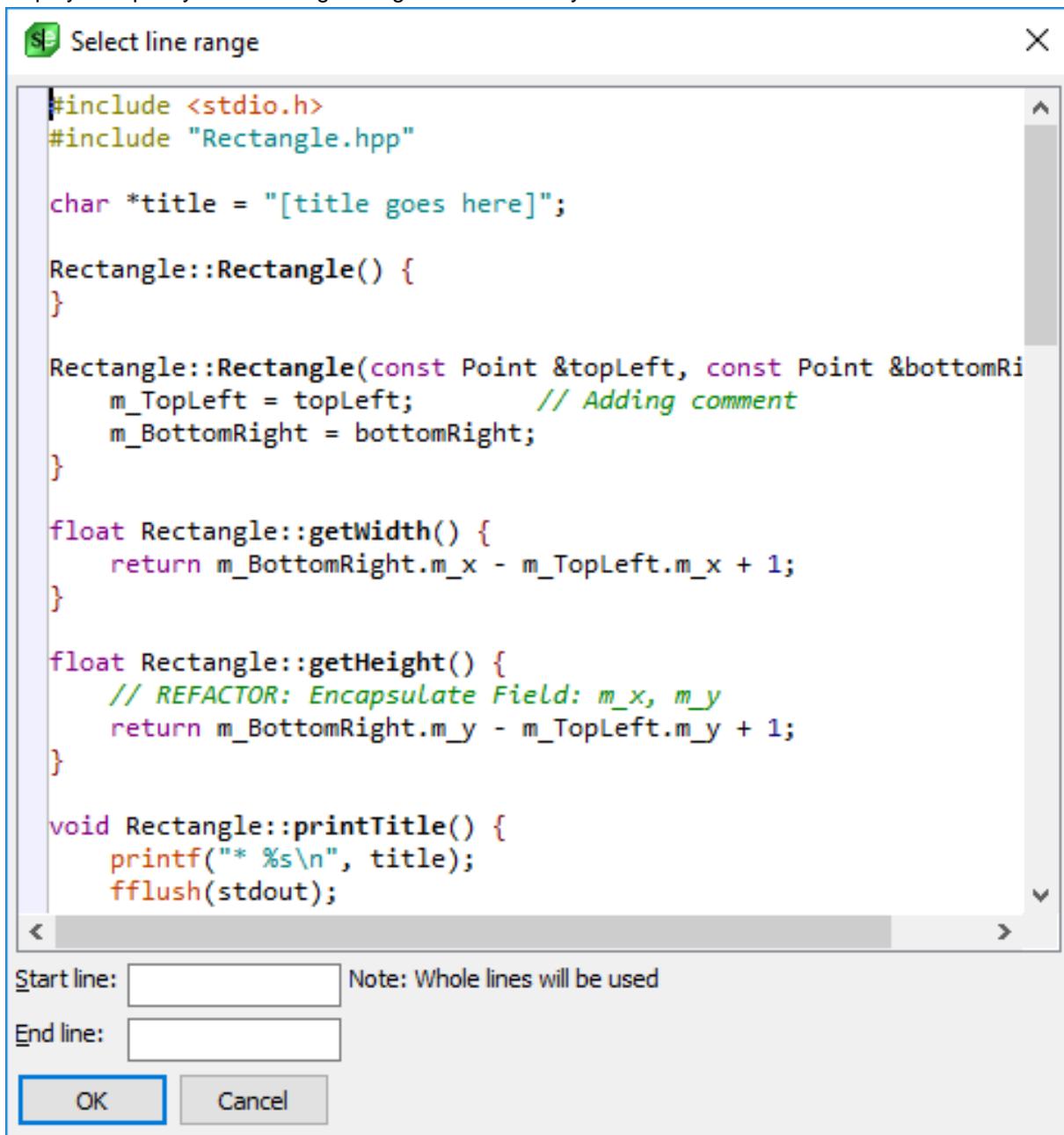
To diff a selected range of lines from two source files or from a single source file, complete the following steps:

Tip

You can compare line ranges from within a single file. This can be useful when working with XML or data files.

1. From the main menu, click **Tools** → **File Difference**.

2. Select the **Compare Two Files** diff type.
3. Type the name of the first file in the **Path 1** text box.
4. Select **Compare lines: range**, in the second drop-down list. The **Select line range** dialog will be displayed. Specify the line range using a selection or by enter the start and end line numbers.

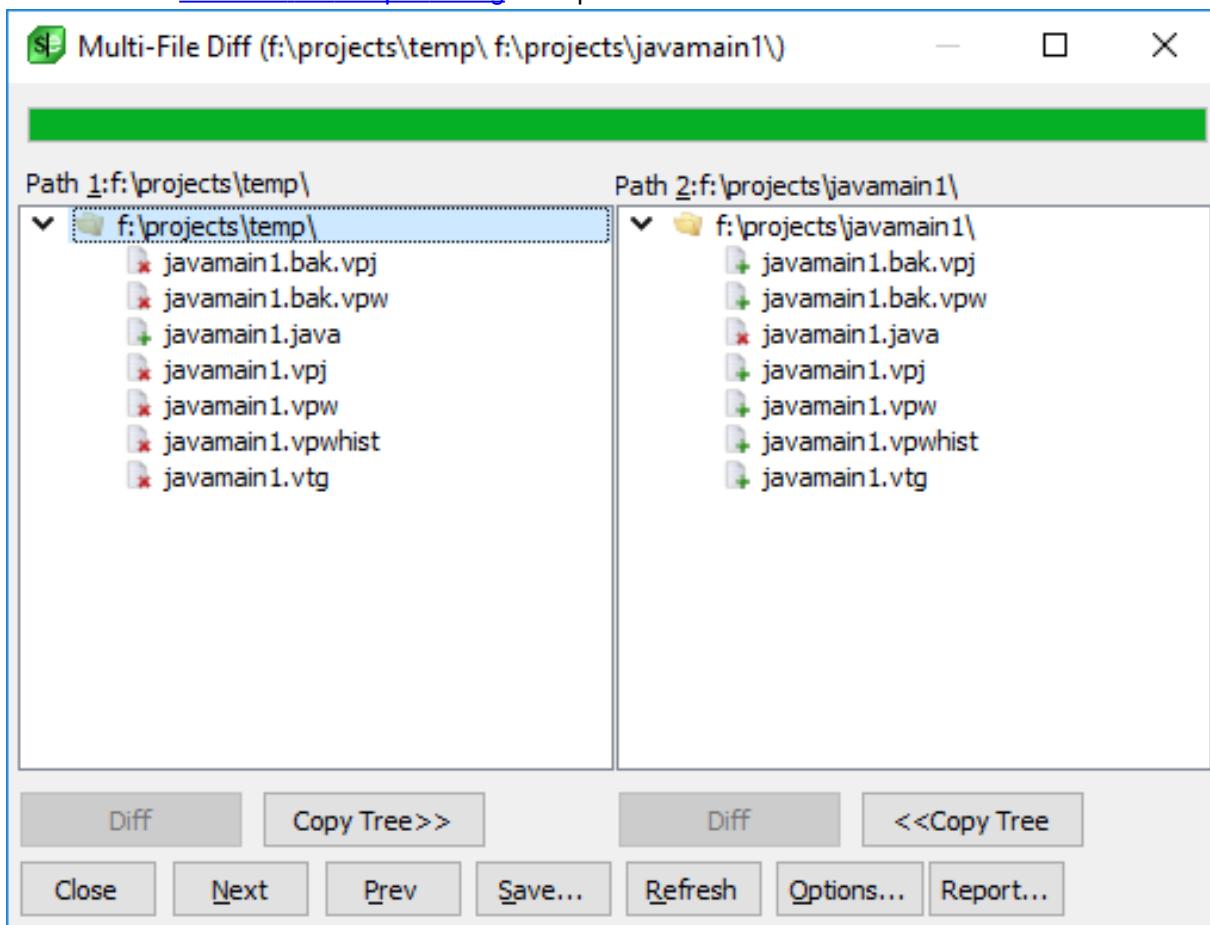


5. Repeat the previous steps for the second file using Path 2.
6. Click **OK** to begin the comparison.

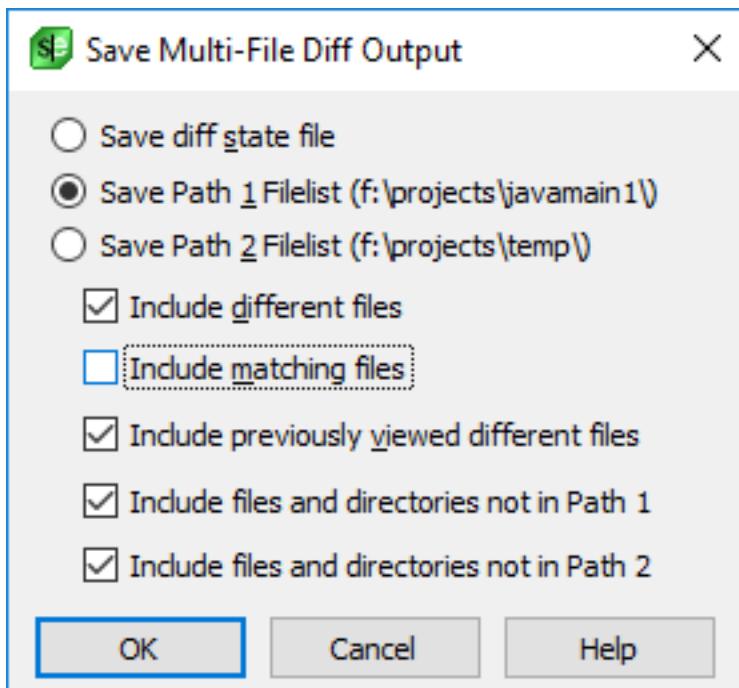
Generating File Lists (Pro only)

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, click **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.



6. Click **Save**. The Save Multi-File Output dialog box opens.



7. Select **Save Path 1 Filelist**, **Include different files**, and **Include files not in Path2**. All other check boxes should be clear.
8. Click **OK** and select an output file for the list. The file you save will have the **.lst** extension appended to the output file name.
9. Zip the files if you want.

Automatic Directory Mapping (Pro only)

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.

1. From the main menu, click **Tools** → **File Difference**, or use the **diff** command.
2. Select the **Options** tab.
3. Click **Dialog Setup**.
4. Clear the **Automatic directory mapping** check box.

Differing 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. For more information about this working with Backup History, see [File Backups](#).

Launching DIFFzilla from the Operating System

DIFFzilla can be launched from the operating system using the **vsdiff** executable. This command actually launches SlickEdit but in a mode where just DIFFzilla is visible. This command includes an implicit **+new** option so that it will not interfere with existing instances of SlickEdit already running. For more information see [Running Multiple Instances](#).

The command line syntax for invoking DIFFzilla® is as follows:

```
vsdiff [-r1][-r2] [-wc wildcard] [-x wildcard] FileOrPath1 FileOrPath2
```

If the files have the same name, you only need to give the path for *FileOrPath2*.

The table below shows a list of available invocation options.

Invocation Option	Description
-? or -h[elp]	Display this help for invocation options.
+t	Recurse directories.
-t	Do not recurse directories.
-wc wildcard	Wildcard for multi-file diff. Multiple instances of -wc may be specified.
-x wildcard	Wildcard for multi-file diff. Multiple instances of -x may be specified.
-filelist <i>listfile</i>	Specifies a line delimited list of relative filenames to be diffed in the specified directories.
-showdifferent	Show different files.
-hidendifferent	Hide different files.
-shownotinpath1	Show files missing from path 1
-hidenotinpath1	Hide files missing from path 1
-shownotinpath2	Show files missing from path 2
-hidenotinpath2	Hide files missing from path 2

Invocation Option	Description
-showviewed	Show files already viewed in diff
-hideviewed	Hide files already viewed in diff
-sc config_path	Specifies the configuration directory. This directory will be used to find and save configuration files.

Examples:

```
-- Display the DIFFzilla® dialog
vsdiff

-- Diff two files
vsdiff file1 file2

-- Diff .cpp and .h files recursively in the two directories
vsdiff -wc *.cpp -wc *.h c:\path1\ c:\path2\

-- Diff all files recursively in the two directories
-- excluding files in the backup directory
vsdiff -wc *.* -x backup\ c:\path1\ c:\path2\

-- Diff .cpp and .h files non-recursively in the two directories
vsdiff -t -wc *.cpp -wc *.h c:\path1\ c:\path2\
```

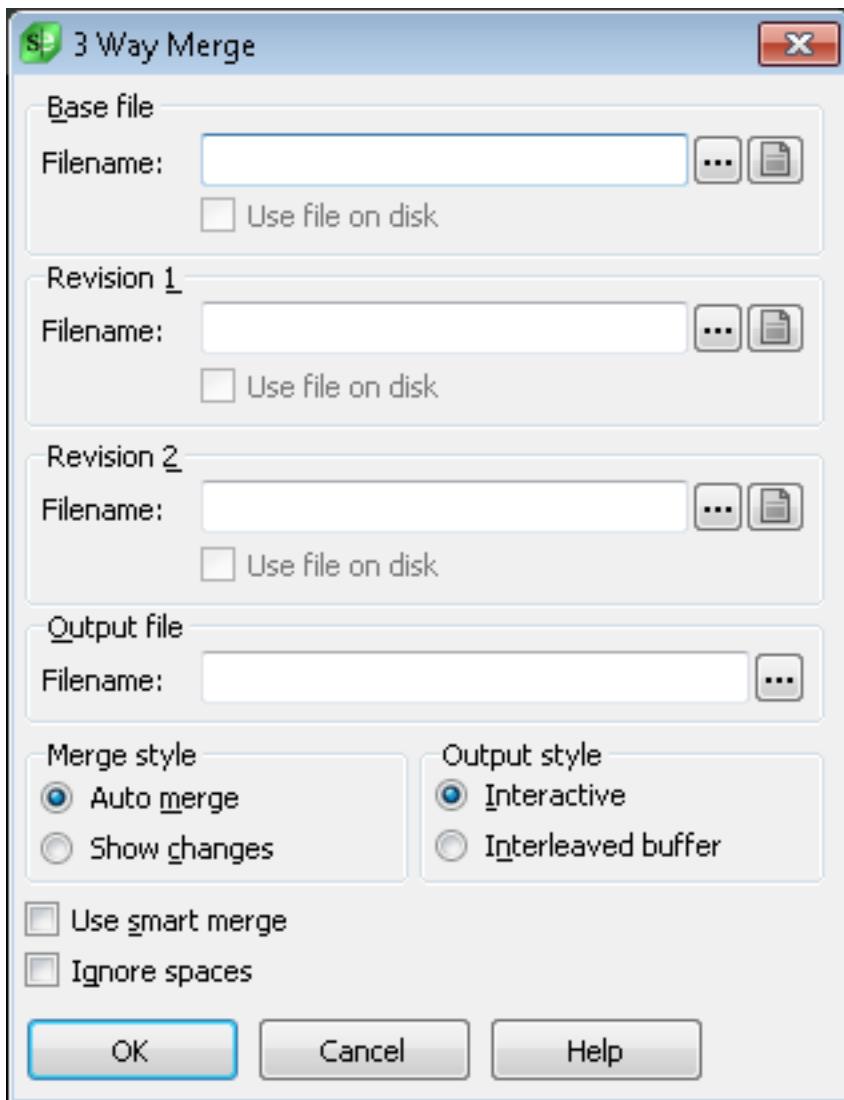
3-Way Merge (Pro only)

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.

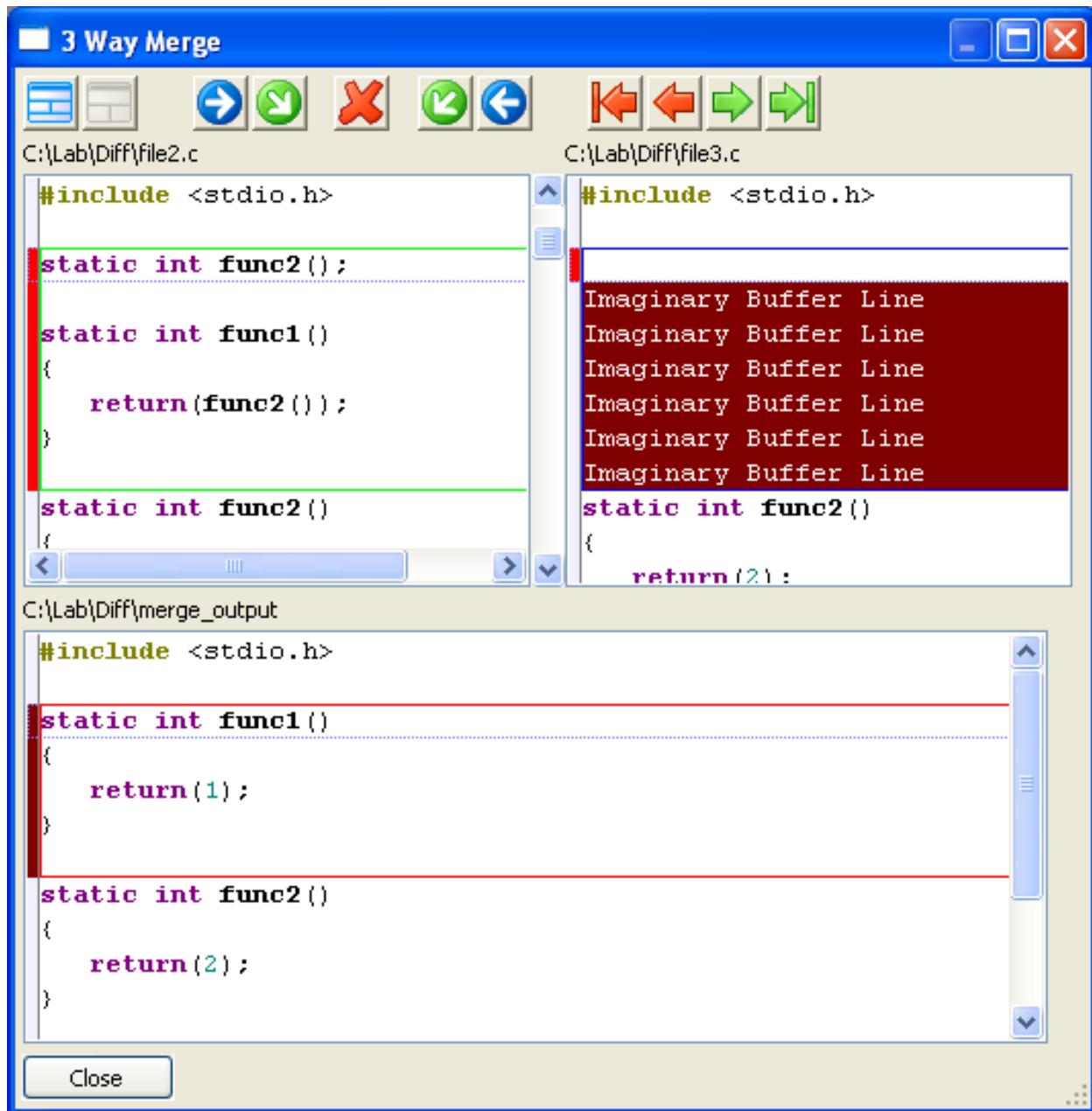
Performing a Three-Way Merge

To perform a three-way merge, complete the following steps:

1. From the main menu, click **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** button to the right of the text box to select files. Click the **B**utton 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 **Ellipses** button 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:



Launching 3-Way Merge from the Operating System

The 3-Way Merge can be launched from the operating system using the **vsmerge** executable. This command actually launches SlickEdit but in a mode where just 3-Way Merge is visible. This command includes an implicit **+new** option so that it will not interfere with existing instances of SlickEdit already running. For more information see [Running Multiple Instances](#).

3-Way Merge Settings

For descriptions of the options on the 3-Way Merge Setup dialog, see [Tools Dialogs and Tool Windows](#).

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 (Pro only)

Overview of Version Control (Pro only)

Version control is accessed from the **Version Control** menu, by right-clicking within a file or buffer, by right-clicking on an item in the Files list of the Open tool window, or by right-clicking on an item in the Project tool window. The operations on the menu vary depending on the version control system.

SlickEdit supports 3 categories of version control systems

- **Integrated Systems** - These systems use the most tightly integrated with SlickEdit, including Context Tagging and DIFFzilla, a comprehensive history dialog, repository browsers, so that you can see exactly what happened when in your code. They are:
 - [Git](#)
 - [Subversion](#)
 - [Perforce](#)
 - [CVS](#)
 - [Mercurial](#)
- **SCC Systems** - (Windows only) - SlickEdit supports the SCC standard and will work with any version control system that supports it. These will be listed if they are installed
- **Legacy Systems** - These are older systems where SlickEdit runs a command line that parses in pieces of your filenames using escape sequences. They include:
 - ClearCase
 - GNU RCS
 - MKS RCS
 - PVCS
 - SCCS
 - TLIB

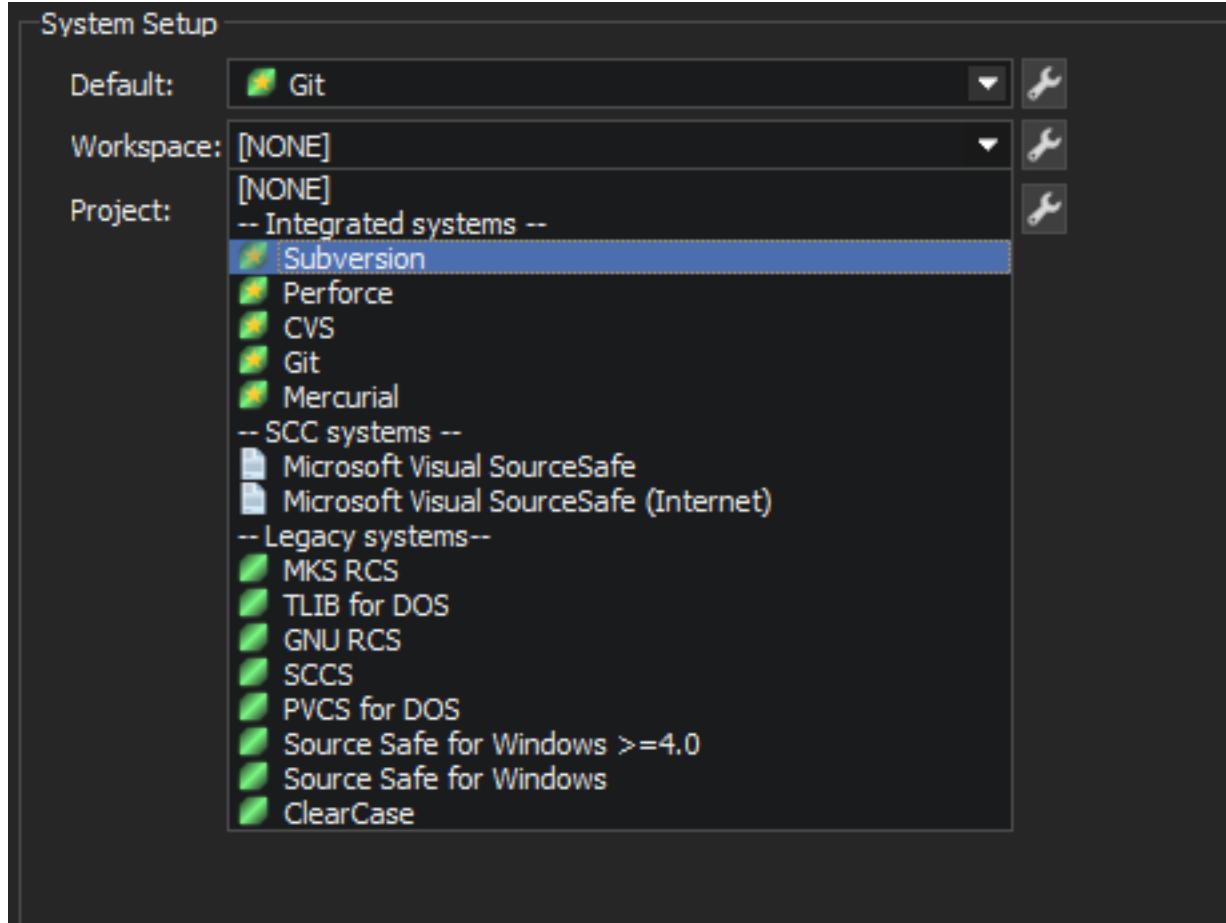
Selecting a Version Control System (Pro only)

SlickEdit supports version control at the global, workspace, and project level. This allows you to support projects that use different systems if an older project did not change from one system to another.

If you are in a team of people all using slickedit and you check in your workspace and project files, you

may want to set your version control up at the workspace or project level. Otherwise, just setting the default will work well

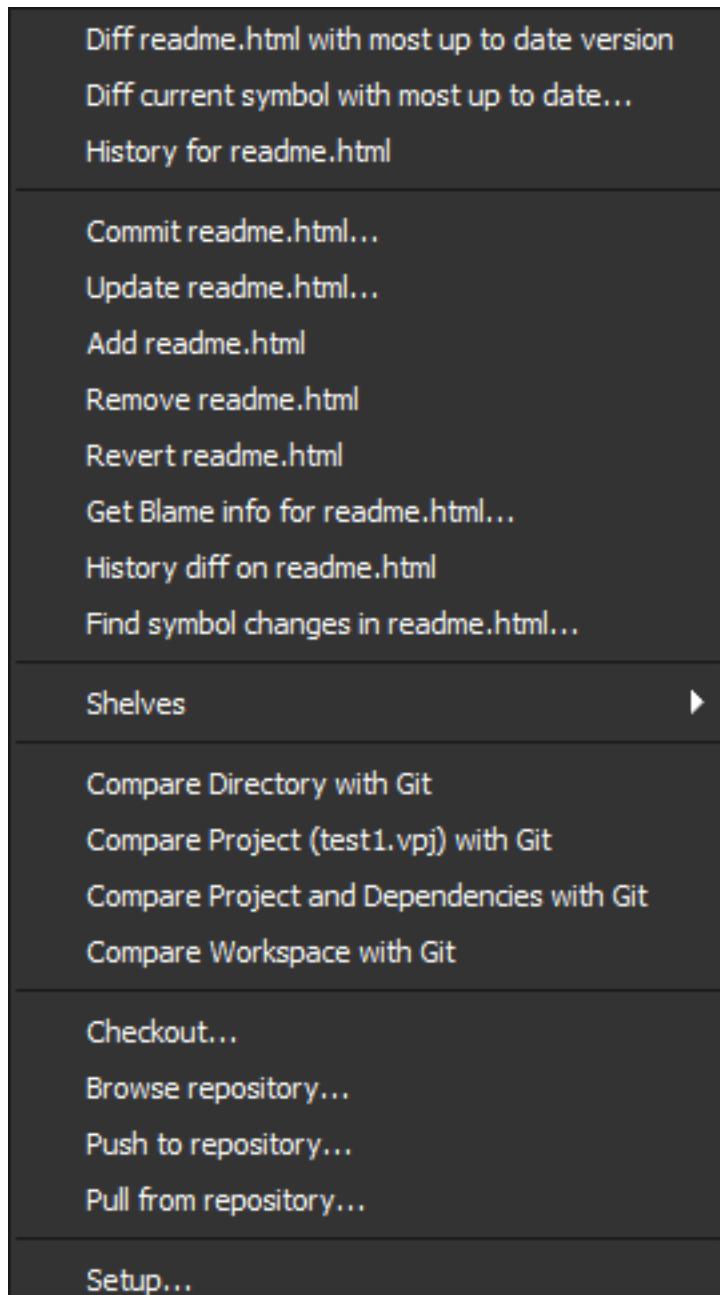
SCC Systems can only be set up for the project. No other version control system may be set up at the same time.



Using Version Control (Pro only)

Version control operations can be accessed from the main menu by clicking **Version Control**, and then choosing an operation. Version control operations are also accessible from the context menu of the Project tool window. The operations on the menu vary depending on the version control system.

If you are using Git, Subversion, Perforce, CVS, or Mercurial, you will see a menu like this:



Since different systems use different terminology for similar commands, there are some differences:

Only Subversion shows the **Lock** menu item.

Git and Mercurial have menu items for **Push to Repository** and **Pull from Repository**.

Perforce and CVS do not have the **Browse Repository...** menu item.

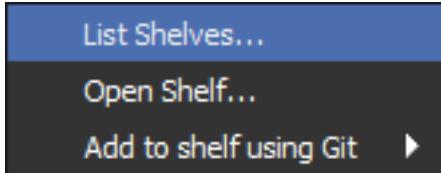
Perforce has **Submit** instead of **Commit** menu item.

CVS does not have the **History Diff** menu item.

Perforce and CVS do not have the **History Diff** menu item.

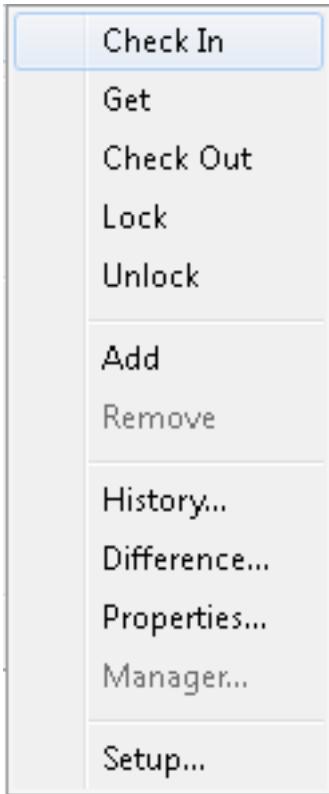
Most of these menu items are obvious since they are common version control operations. Here are some details on the less obvious menu items:

- **Diff current symbol with most up to date version** - This compares the symbol that the cursor is currently in with the most up to date version. This is great when you do not want to view all the differences in the file, but you want to see the differences in the function you are currently viewing. Only SlickEdit can offer this sort of version control feature because it uses SlickEdit's powerful context tagging engine.
- **History diff on readme.html** - Will bring up a special diff with a list of versions of the file in version control. You can see all the regular history information, and diff the selected version with either your local file or the version before it.
- **Find symbol changes in readme.html** - You can pick any symbol Context Tagging recognizes and see all of the versions where it changed.
- **Compare with version control** - This displays a directory tree and lists files that either you modified or someone else modified. It also lists files that are not in the version control repository so they can be added. Modified files can be updated, committed, reverted, or resolved.
- **Browser repository...** - This dialog allows you to browse modules and revisions in your version control repository. You can checkout a specific module or revision.
- If you expand the Shelves menu, you will see a menu like this:



- **List shelves...** - This dialog lists shelves you have created. For more information, see [Shelving](#)
- **Open shelf..** - Opens a shelf you have previously created and allows you to revert/merge some or all of the files in the shelf with files on disk. For more information, see [Shelving](#)
- **Add to shelf using Git** - Will add the current file to one of your existing shelves.

If you are not using an SCC or legacy system, your menu will look more like this:



For some SCC systems, you can 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 **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. 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. For example, when using the **Check In** command, you can click **Available** to view all files that can currently be checked in.

- **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 click **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 click **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 click **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 click **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 click **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 click **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 click **Version Control** → **Remove**, or use the **vcremove** command. Select the files in the list that you wish to remove, then click the **Remove** button.

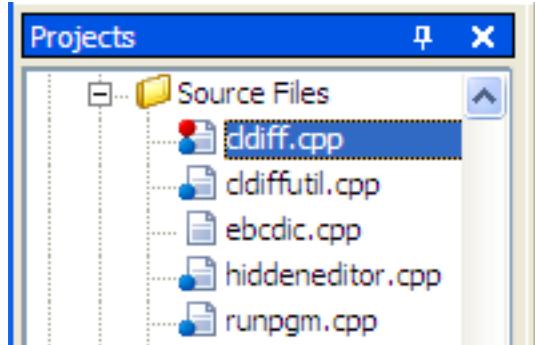
Version Control Status Icons

SlickEdit displays the version control status of a file on the document icon in the following tool windows:

- Projects tool window
- Files tool window

- Open tool window

A red ball will appear on the upper left side of files that are locally modified. A blue ball will appear on the lower left side of files that need to be updated.



Note

This feature is supported only for Git, Subversion and Perforce, currently.

You can turn this feature on/off and configure the frequency of updates on the Git, Subversion or Perforce options screen. Select **Tools** → **Options** and set the values under **Show file status..**

Note

If the version control system returns more than 8 megabytes of data, this feature will be turned off for performance reasons. To change that threshold, set **def_vc_max_status_output_size**.

Configuring Version Control (Pro only)

Before using Version Control, you should configure your setup. To access these options, from the main menu, click **Version Control** → **Setup**, or use the **vcsetup** command. Click the wrench next to your selected version control system. This will bring up that system's specific options.

For legacy (command line) systems, click **Tools** → **Options**.

For more information, see [Version Control Setup Options](#).

Advanced Setup Options

Advanced setup options are available for each version control system that supports them. From the Version Control Setup options screen (**Tools** → **Options** → **Tools** → **Version Control** → **Version Control Setup**), select the version control system you want to set up, then 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 Setup Options](#) 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, click **Tools** → **Options**, expand **Tools**, then select **Version Control Setup** (or use the **vcsetup** command).
2. Select the a command line system from one of the combo boxes
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**.

SCC Support (Pro only)

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, click **Tools → Options**, then select **Version Control Setup** (or use the **vcsetup** command).
2. Select the version control system to be used.
3. Click the wrench icon next to the SCC system you've selected. This will launch the SCC dialog.
4. Click **Initialize Provider**. This launches the SCC provider. If it is already running, this button will be grayed out.
5. Click **Open Project** and complete the Open Project dialog box. This dialog differs from provider to provider. Typically, this is where you enter things like user name, password, and information about the server.
6. 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, click **Tools → Options**, then select **Version Control Setup** (or use the **vcsetup** command).
2. Select an SCC System from one of the combo boxes. For more information on SCC, see [Specific Version Control Support](#) above.
3. Click the wrench icon next to the SCC system you've selected. This will launch the SCC dialog.
4. Click **Initialize Provider**.
5. 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.

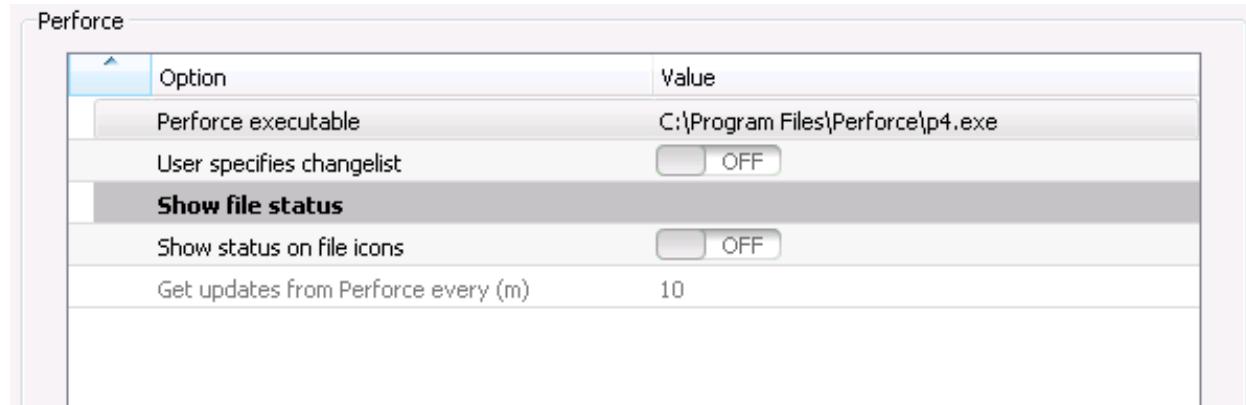
After you open an SCC version control project, it is bound to the currently active project. When you restart the project, or switch to this project from another project, this version control project is automatically activated.

Perforce

Perforce support provides a way to use the editor as a front-end to Perforce, as well as integrating Perforce actions with your regular editor use. To use Perforce, go to **Tools** → **Options** → **Tools** → **Version Control** → **Version Control Setup** and select **Perforce** from the list of command line systems.

Perforce Options

Perforce-specific options are provided under the Version Control Providers options (**Tools** → **Options** → **Tools** → **Version Control** → **Version Control Providers** → **Perforce**). The available options are shown below.



The following options are available:

- **Perforce executable** - Specifies the path to the Perforce executable that you wish to use.
- **User specifies changelist** - When set to **On**, the user will be prompted for possible changelists when checking out or submitting.
- **Show file status** - The following options relate to showing the Perforce status for a file in various tool windows and dialogs.
 - **Show status on file icons** - When set to **On**, dialogs that display file icons in the tree will show the file's Perforce status.
 - **Get updates from Perforce every (m)** - This option specifies, in minutes, how often the file status

updates should be retrieved from Perforce

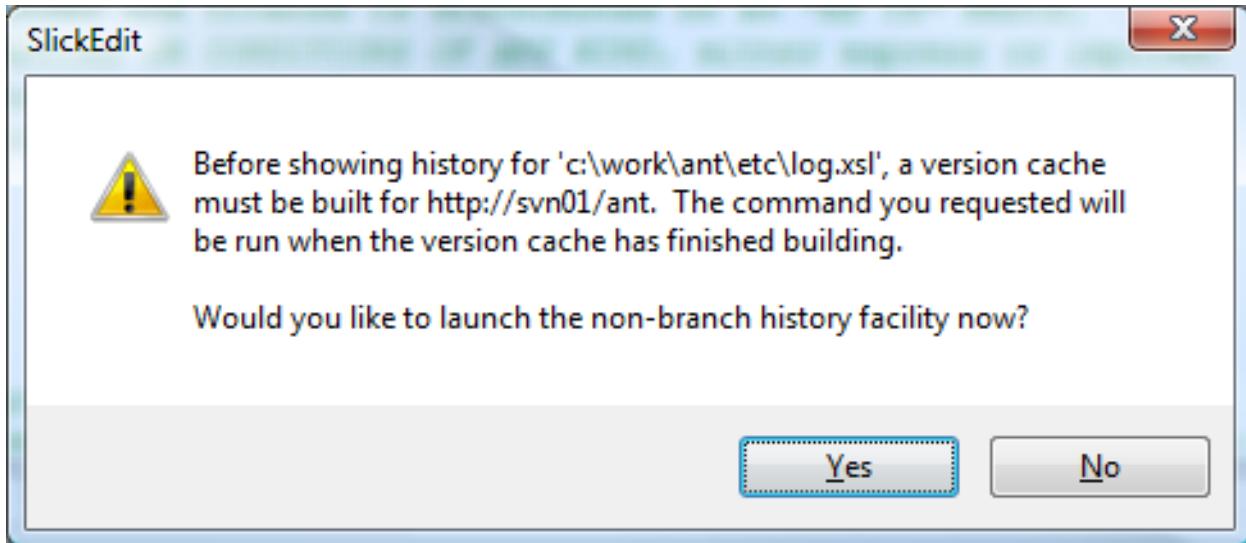
Subversion

Subversion support provides convenient access to information about the files with which you are working, and also a GUI checkout dialog. To get started, from the main menu, click **Tools** → **Options**, expand **Tools**, then select **Version Control Setup** (or use the **vcsetup** command). Set the **Command line system** to **Subversion**. After you activate this setting, 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 click **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.

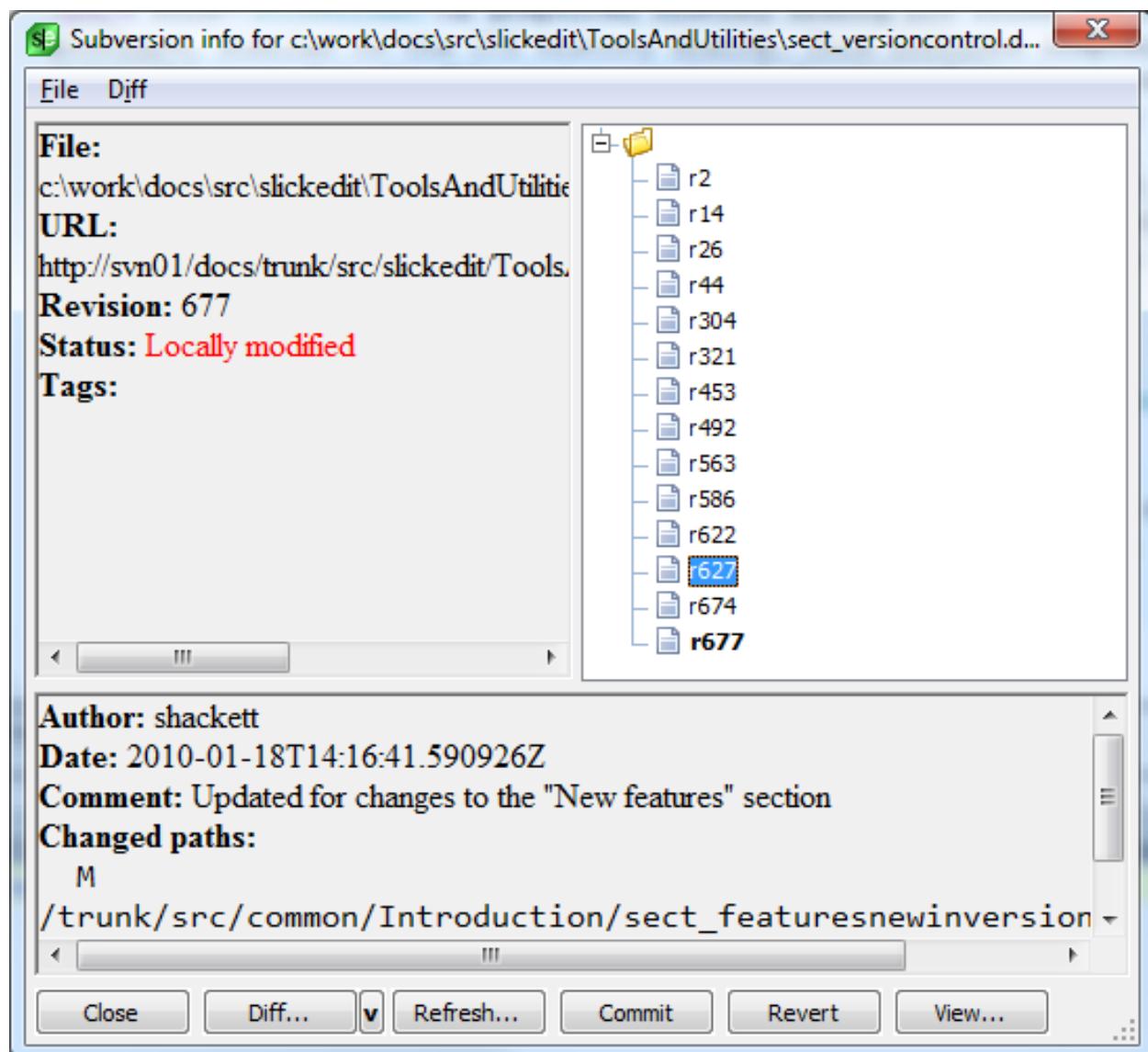
SVN History Dialog

A graphical history dialog for files checked out from SVN is also provided, similar to the CVS history dialog. This includes displaying the current state of the file, all tags for the file and a graphical display of the branches.

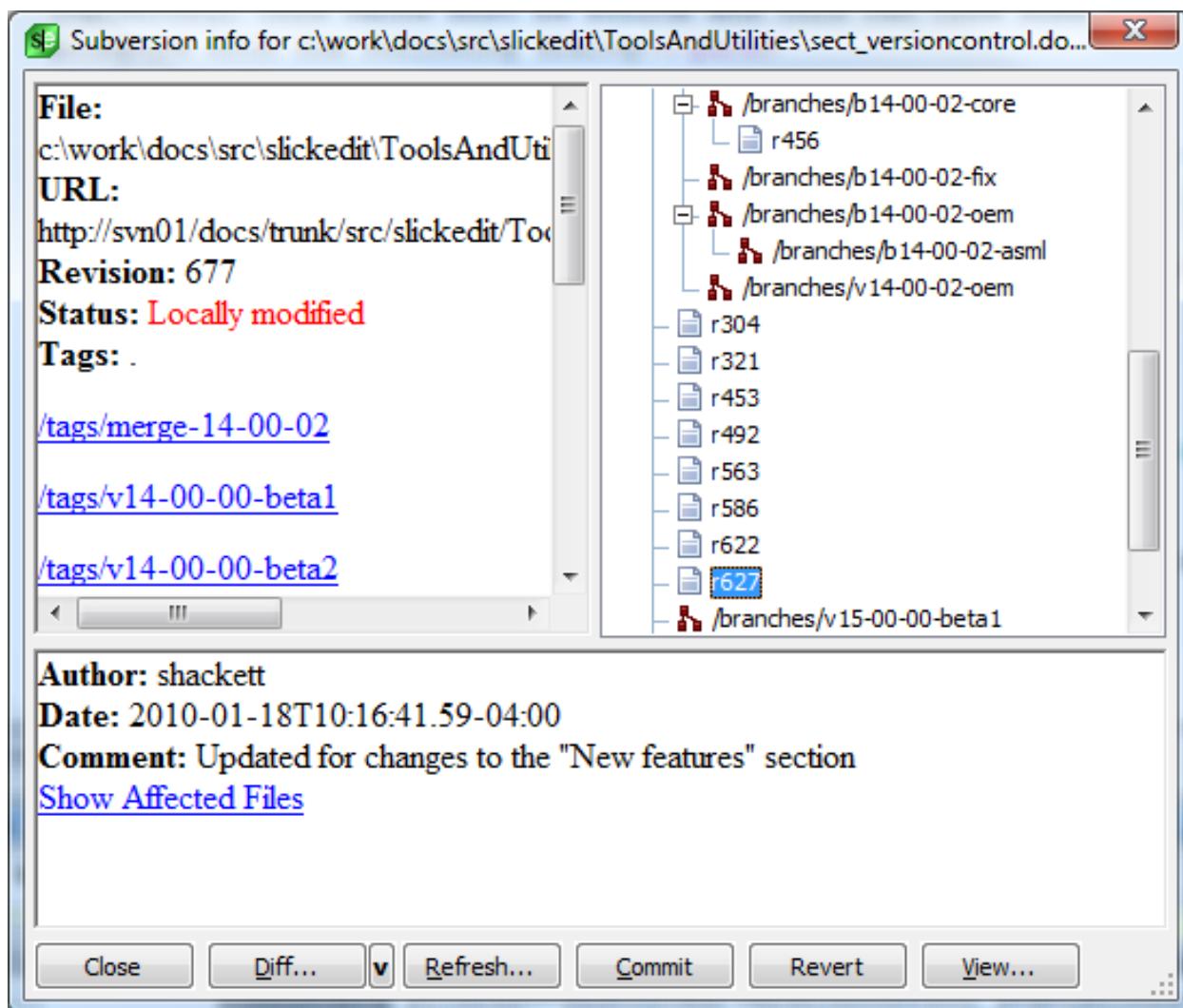
Because SVN cannot provide a full history for all branches for a file, SlickEdit uses a "version cache" to store this information. Details of each revision are stored in this cache and can be queried for a full history of a file, and not just that of the current branch. The first time you choose to view the history of a file in an SVN repository, this cache must be built. This can take some time to do, and you will be prompted with the following dialog:



If you select "Yes", an SVN history will come up right away with a flat revision history for the current branch back to the trunk. This can be seen in the following screenshot:



Once the version cache has been built, the following SVN history dialog will be shown:



This view includes a hierarchy of all revisions for a file, nested by the branches they exist on. All further requests for SVN history will happen very quickly. Only the initial construction of the version cache requires an extended period of time to create. You can use DIFFzilla from this dialog to view differences between the current and any past version, or any two past versions, with each other (see [DIFFzilla®](#)) from this dialog.

Subversion Options

Options specific to Subversion are provided under the Version Control Providers options (**Tools** → **Options** → **Tools** → **Version Control** → **Version Control Providers** → **Subversion**). These options are pictured below.

Subversion	
Option	Value
Subversion executable	C:\Program Files\CollabNet\Subversion\bin\svn.exe
When updating, move to next file after diff	<input checked="" type="checkbox"/> OFF
When committing, restore last comment	<input checked="" type="checkbox"/> ON
Show branches in version history tree	<input checked="" type="checkbox"/> OFF
Traverse parent branch history	<input checked="" type="checkbox"/> ON
Show labels in version history tree	<input checked="" type="checkbox"/> ON
Use --stop-on-copy option	<input checked="" type="checkbox"/> OFF
Hide empty branches in version history tree	<input checked="" type="checkbox"/> OFF
Show file status	
Show status on file icons	<input checked="" type="checkbox"/> ON
Get updates from Subversion every (m)	1

The following options are available:

- **Subversion executable** - Specifies the path to the Subversion executable that you wish to use.
- **When updating, move to next file after diff** - When set to **True**, after the diff of one file is complete, the next file will be shown.
- **When committing, restore last comment** - Controls whether the last comment used to commit a file is restored for possible reuse.
- **Show branches in version history tree** - When set to **True**, the displayed history of the file will include activity in all branches. When set to **False**, then the history of branches other than the current one will not be displayed.
- **Traverse parent branch history** - When set to **On**, the version history tree will traverse parent branches. When set to **Off**, use Subversion's--stop-on-copy option.
- **Show labels in version history tree** - When set to **True**, then label names will be displayed in the history tree. Each label name will be included next to the revision to which it was applied. When set to **False**, then the labels are not included in the history tree.
- **Use --stop-on-copy option** - When set to **On**, when retrieving history information Subversion will stop when it gets to a commit on the parent branches
- **Hide empty branches in version history tree** - When set to **True**, only branches containing at least one revision for the current file will be displayed in the history tree.

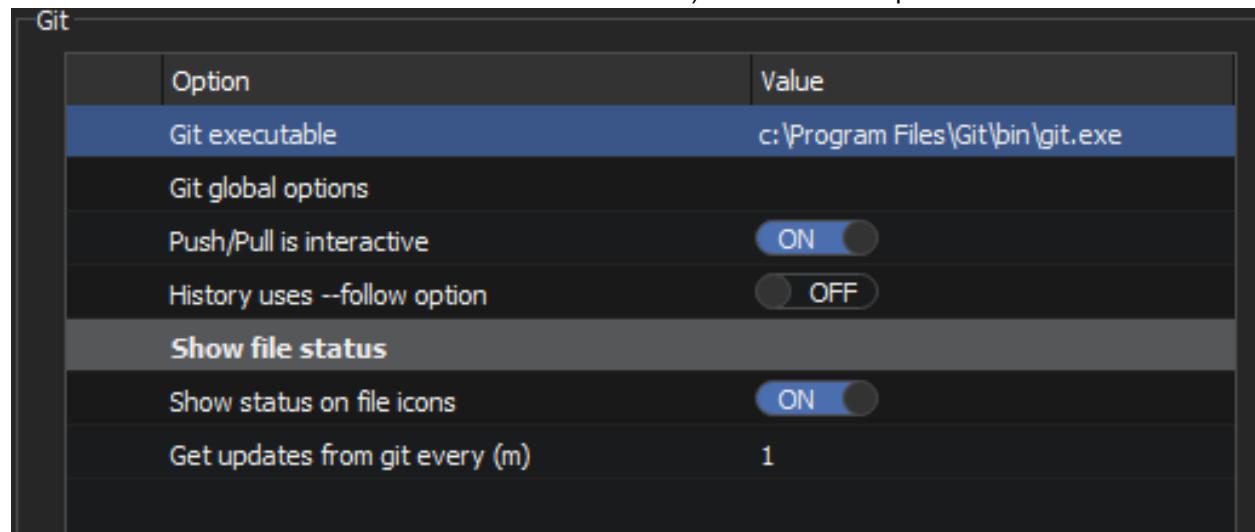
- **Show file status** - The following options relate to showing the Subversion status for a file in various tool windows and dialogs.
 - **Show status on file icons** - When set to **True**, dialogs that display file icons in the tree will show the file's Subversion status.
 - **Get updates from Subversion every (m)** - This option specifies, in minutes, how often the file status updates should be retrieved from Subversion.

Git

Git support provides a convenient front-end to working with Git within the editor. To use Git, go to **Tools** → **Options** → **Tools** → **Version Control** → **Version Control Setup** and select **Git** from the list of command line systems.

Git Options

Git-specific options are provided under the Version Control Providers options (**Tools** → **Options** → **Tools** → **Version Control** → **Version Control Providers** → **Git**). The available options are shown below.



Git	
Option	Value
Git executable	c:\Program Files\Git\bin\git.exe
Git global options	
Push/Pull is interactive	<input checked="" type="button"/> ON
History uses --follow option	<input type="button"/> OFF
Show file status	
Show status on file icons	<input checked="" type="button"/> ON
Get updates from git every (m)	1

The following options are available:

- **Git executable** - Specifies the path to the Git executable that you wish to use.
- **Git global options** - Global Git options that put on the command line before the Git command.
- **Push/Pull is interactive** - When set to **True**, the command window will be shown for the Push and Pull operations to allow entering a pass phrase.
- **Show status on file icons** - When set to **On**, dialogs that display file icons in the tree will show the file's Perforce status.
- **Get updates from Git every (m)** - This option specifies, in minutes, how often the file status updates should be retrieved from Git

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 environment variables, complete the following steps:

1. From the main menu, click **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 bitmap to the left of the file to display a tool tip which indicates what the bitmap means. The **File** bitmap with a blue star means that the file is not up-to-date. A **File** bitmap 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.

CVS Options

Options specific to CVS are provided under the Version Control Providers options (**Tools** → **Options** → **Tools** → **Version Control** → **Version Control Providers** → **CVS**). These options are pictured below.

Option	Value
CVS executable	C:\cygwin\bin\cvs.exe
Always pass CVSROOT to CVS commands (use cvs -d option)	<input type="checkbox"/> OFF
Use CVS status for file bitmaps on project tool window	<input type="checkbox"/> OFF
When updating, move to next file after diff	<input type="checkbox"/> OFF
When committing, restore last comment	<input checked="" type="checkbox"/> ON
When committing, restore last tag(s)	<input type="checkbox"/> OFF
Show labels in version history tree	<input type="checkbox"/> OFF
In GUI update hide folders missing in repository	<input type="checkbox"/> OFF

The following options are available:

- **CVS executable** - Specifies the path to the CVS executable that you wish to use.
- **Always pass CVSRoot to CVS commands (use cvs -d option)** - When set to **True**, the CVS -d is passed to all CVS commands.
- **Use CVS status for file bitmaps on project tool window** - When set to **True**, then the current CVS status is used to show icons on the project tool window.
- **When updating, move to next file after diff** - When set to **True**, after the diff of one file is complete, the next file will be shown.
- **When committing, restore last comment** - Controls whether the last comment used to commit a file is restored for possible reuse.
- **When committing, restore last tag(s)** - Controls whether the last tag used to commit a file is restored for possible reuse.
- **Show labels in version history tree** - Specifies whether the version history tree will show labels.
- **In GUI update hide folders missing in repository** - When this option is set to **True**, empty local directories will be hidden in the update tree.

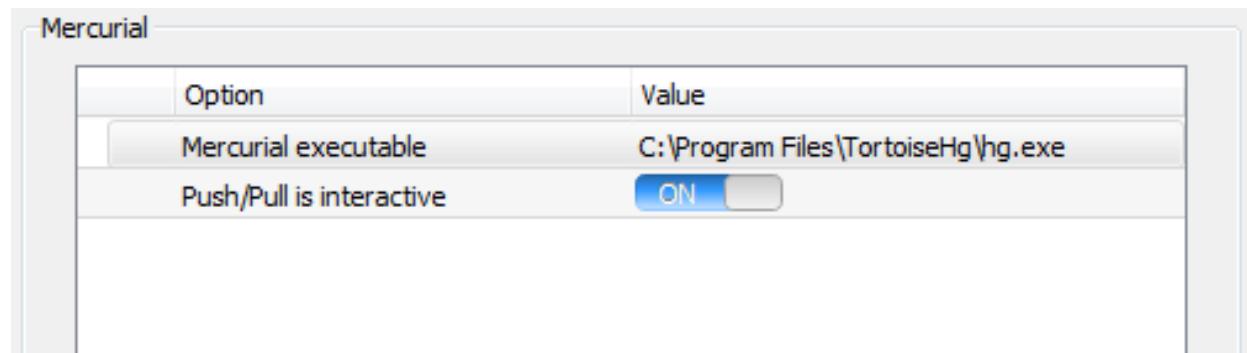
Mercurial

Mercurial support provides a way to use the editor as a front-end to Mercurial. To use Mercurial, go to **Tools → Options → Tools → Version Control → Version Control Setup** and select **Mercurial** from the list of command line systems.

Mercurial Options

Mercurial-specific options are provided under the Version Control Providers options (**Tools → Options → Tools → Version Control → Version Control Providers → Mercurial**). The available options are shown

below.



The following options are available:

- **Mercurial executable** - Specifies the path to the Mercurial executable that you wish to use.
- **Push/Pull is interactive** - When set to **True**, the command window will be shown for the Push and Pull operations to allow entering a pass phrase.

Shelving (Pro only)

Overview of Shelving

SlickEdit's shelving feature allows you to save modifications to a set of files that can then be reverted, and restore the modifications at a later date. It is designed for when an interruption like a bug fix requires you to put aside a feature you are working on.

Shelves are zip files that store the base version of the specified files, as well as the modified versions.

Comparing with Version Control

To compare all files in a directory, project, or workspace with version control and see the differences, use **Version Control → Compare Directory with current version control system**, **Version Control → Compare Workspace with your version control system**, or **Version Control → Compare Project with your version control system** to display the Version Control Update Directory dialog.

Files that are modified will appear with a red ball the upper left side of the file. Files with that need to be updated will have a blue ball will appear on the lower left.

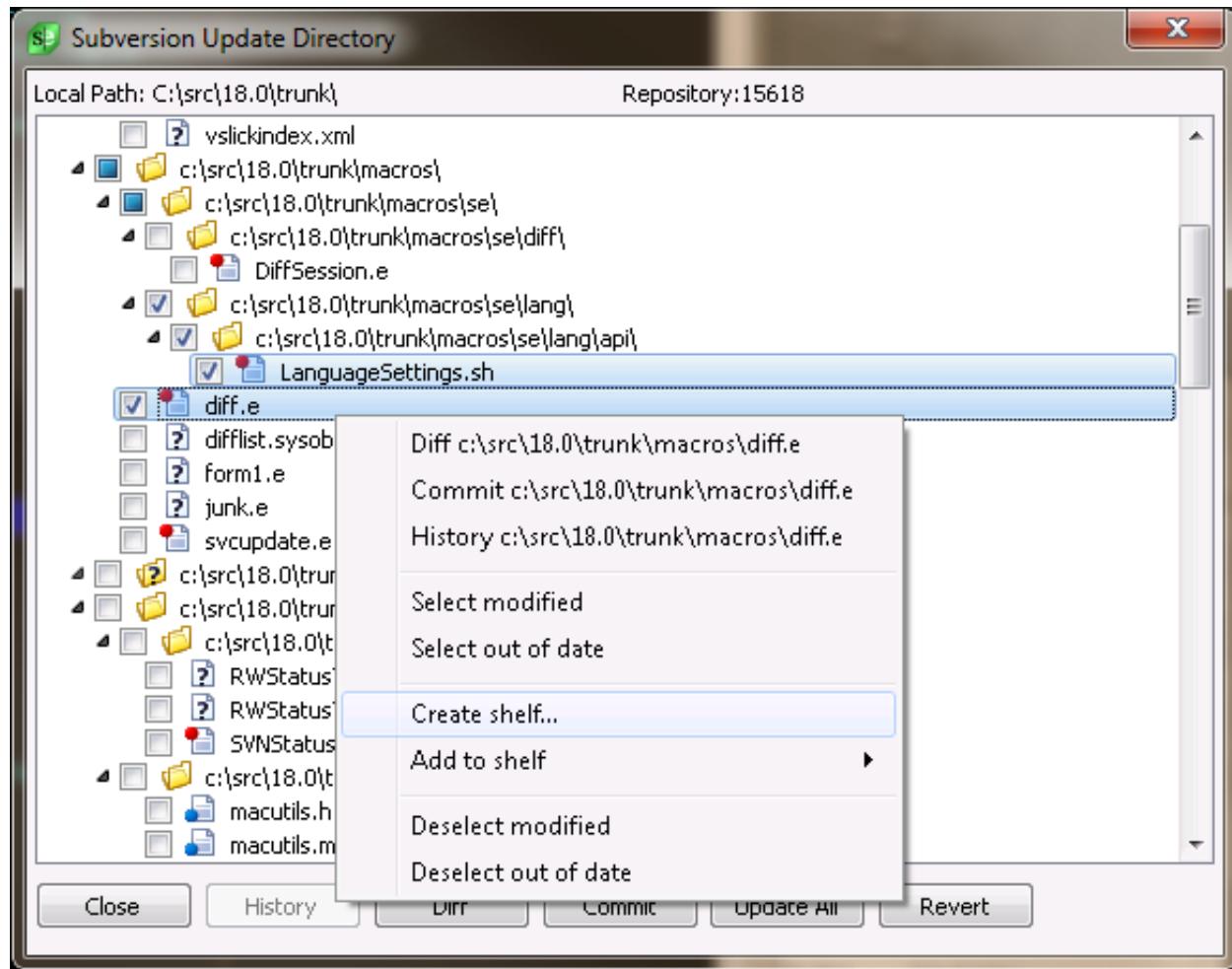
You can float over the files to see why they were listed.

In addition to the Diff button that launches DIFFzilla with the local version and the version in source control system, there is a Diff button the lower right that will expand the dialog with the diffs.

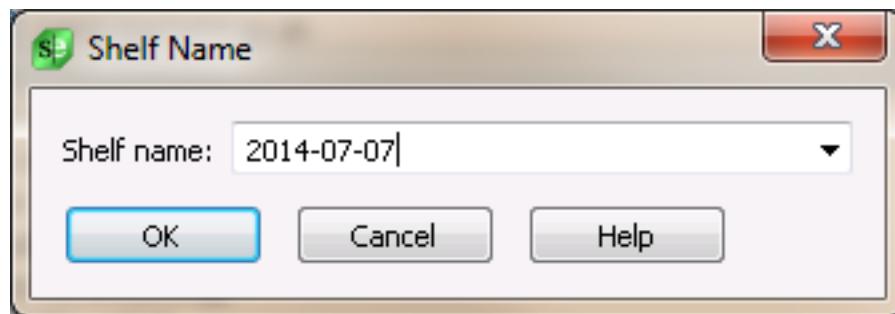
Creating a shelf

To create a shelf, invoke one of the Compare with version control menu items such as **Version Control → Compare Workspace with Subversion** to display the Version Control Update Directory dialog. Then select some files and right-click with the mouse and choose **Create shelf...** as shown below:

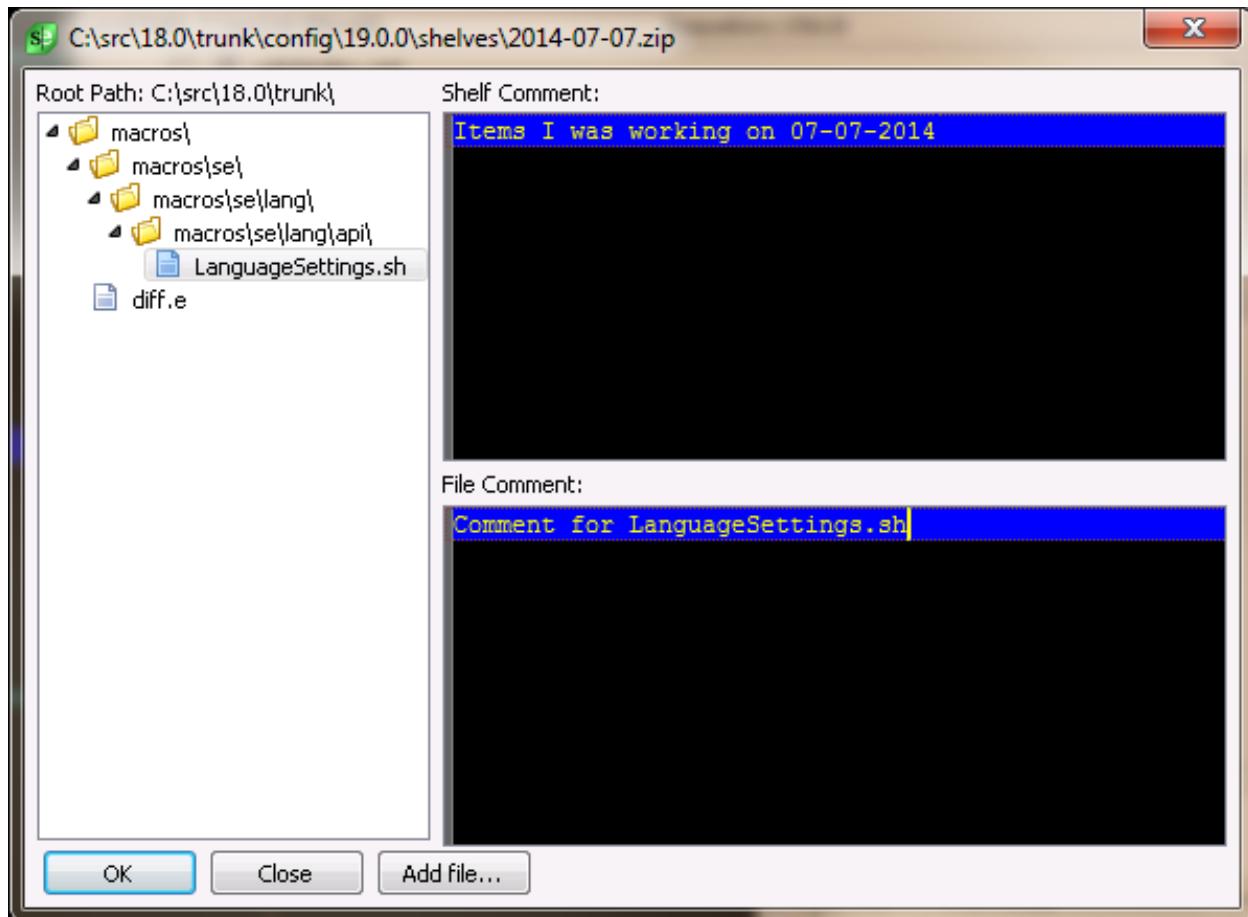
Shelving (Pro only)



Give the shelf a name



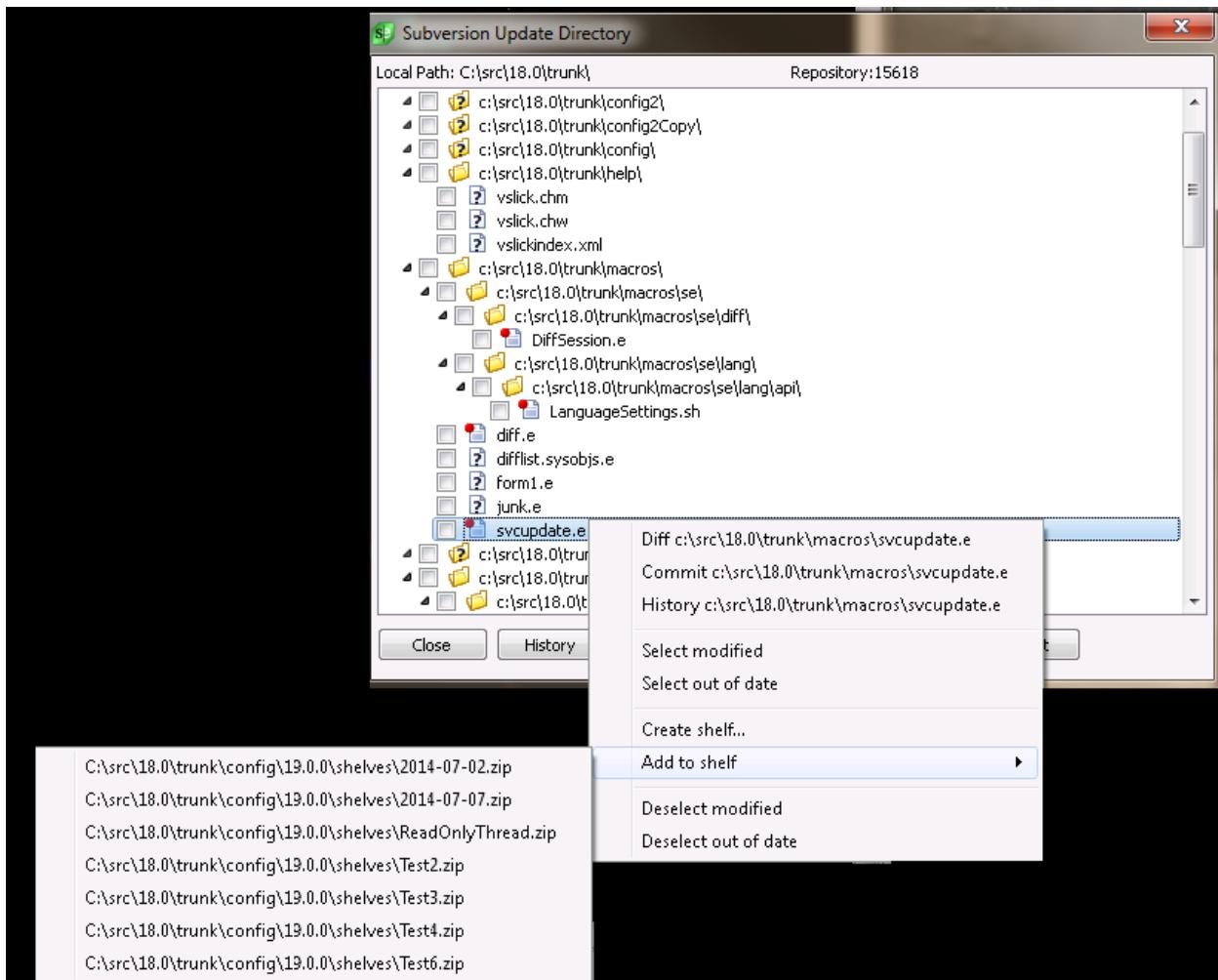
You will then be presented with the current shelf dialog. You can specify a comment for the shelf, as well as a comment for each file:



Adding to a shelf

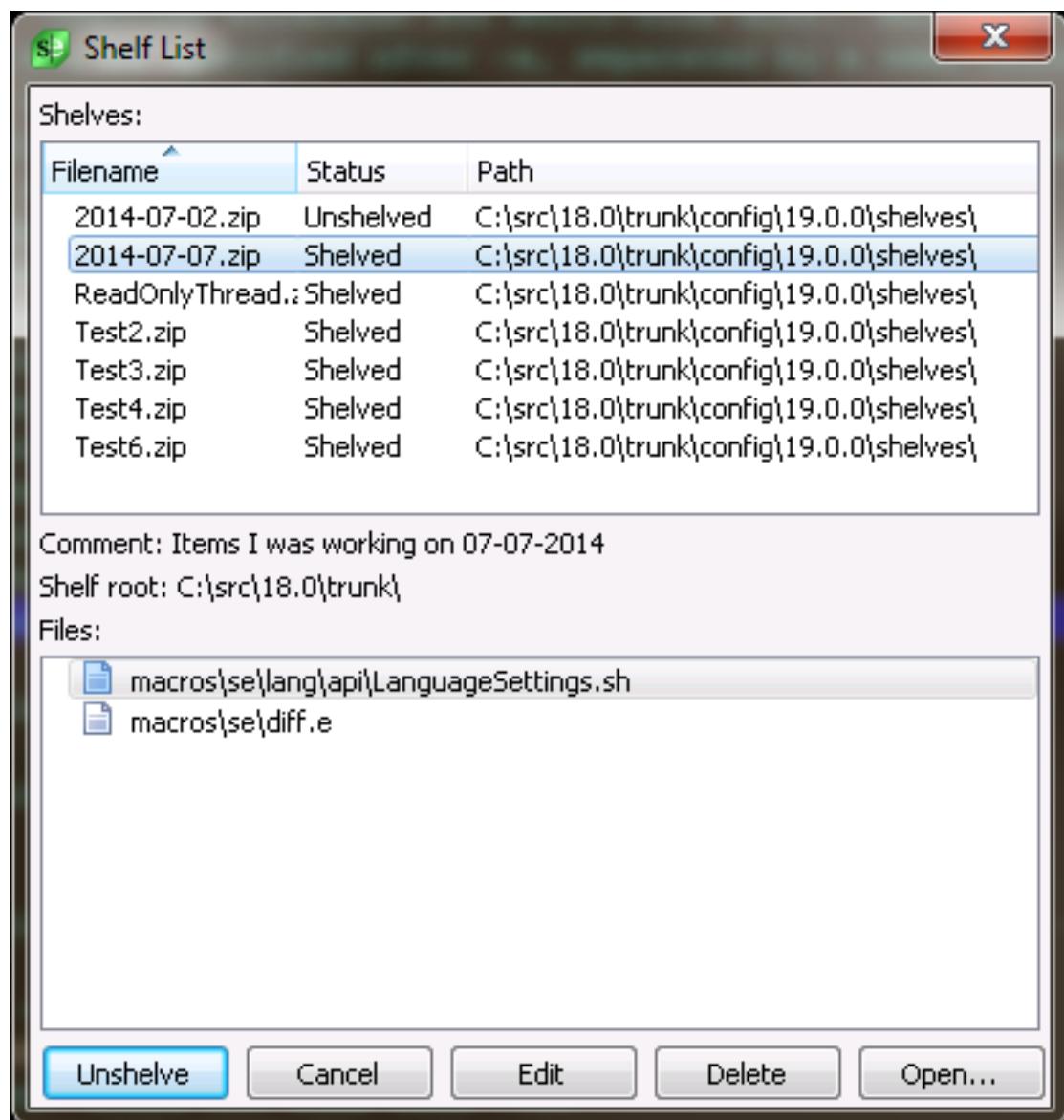
To add to an existing shelf, invoke one of the Compare with version control menu items such as **Version Control** → **Compare Workspace with Subversion** to display the Version Control Update Directory dialog. Then select some files and right-click with the mouse and choose **Add to shelf** as shown below:

Shelving (Pro only)

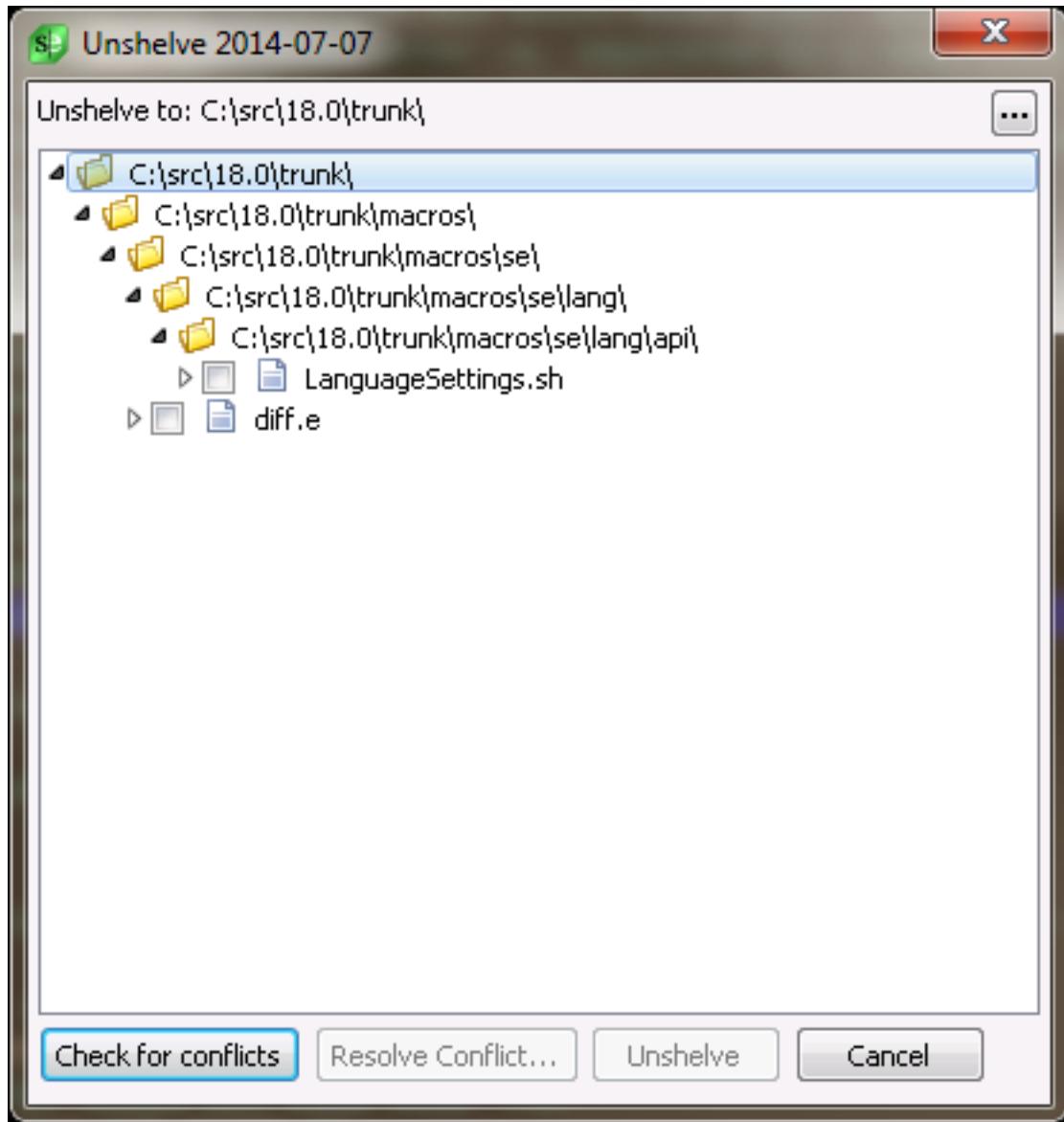


Listing Your Shelves

Use the **Version Control → List shelves...** menu item to see a list of your existing shelves:



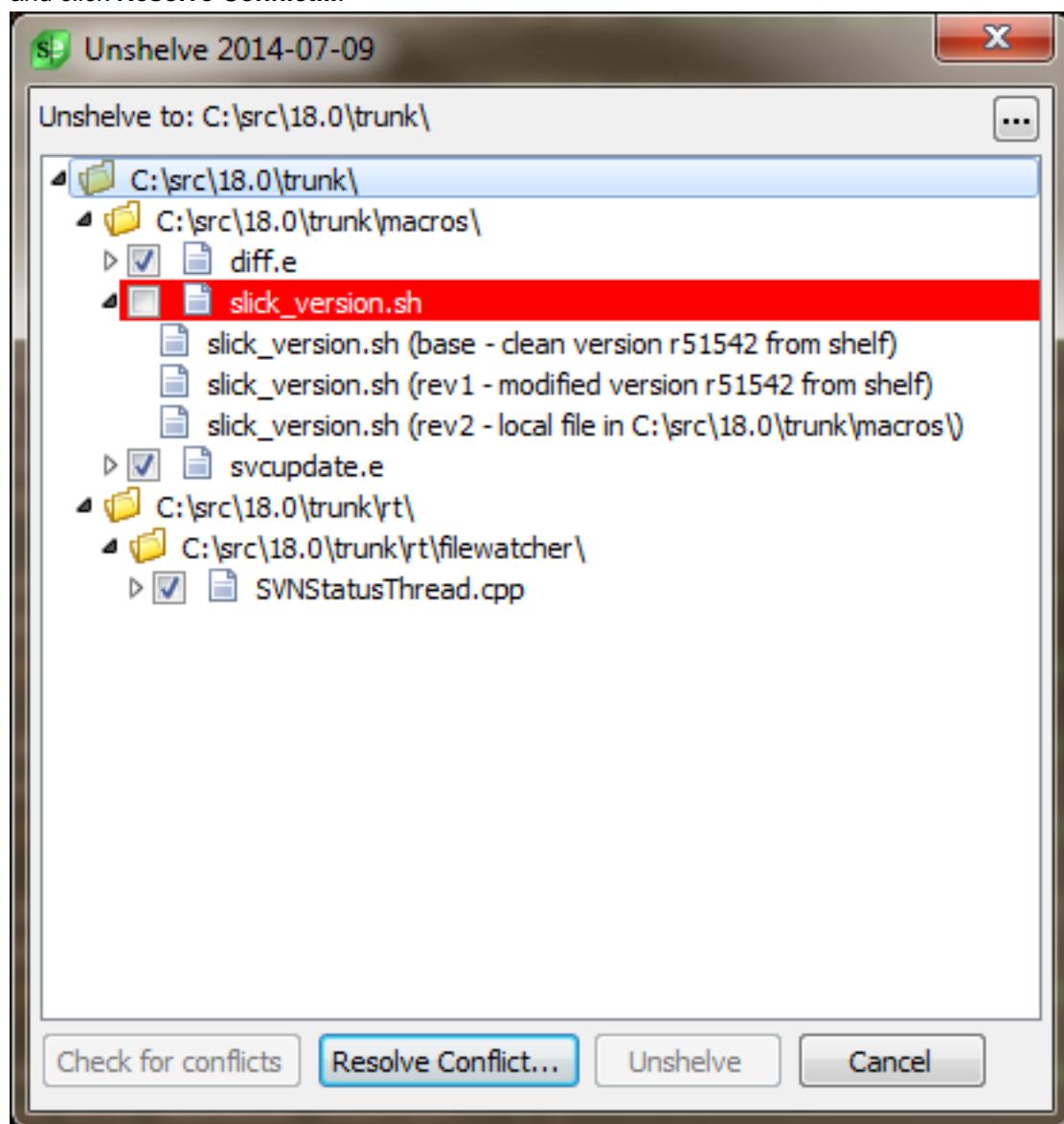
- **Unshelve** - Displays Unshelve dialog. Allows you to merge shelved changes back to your source tree.

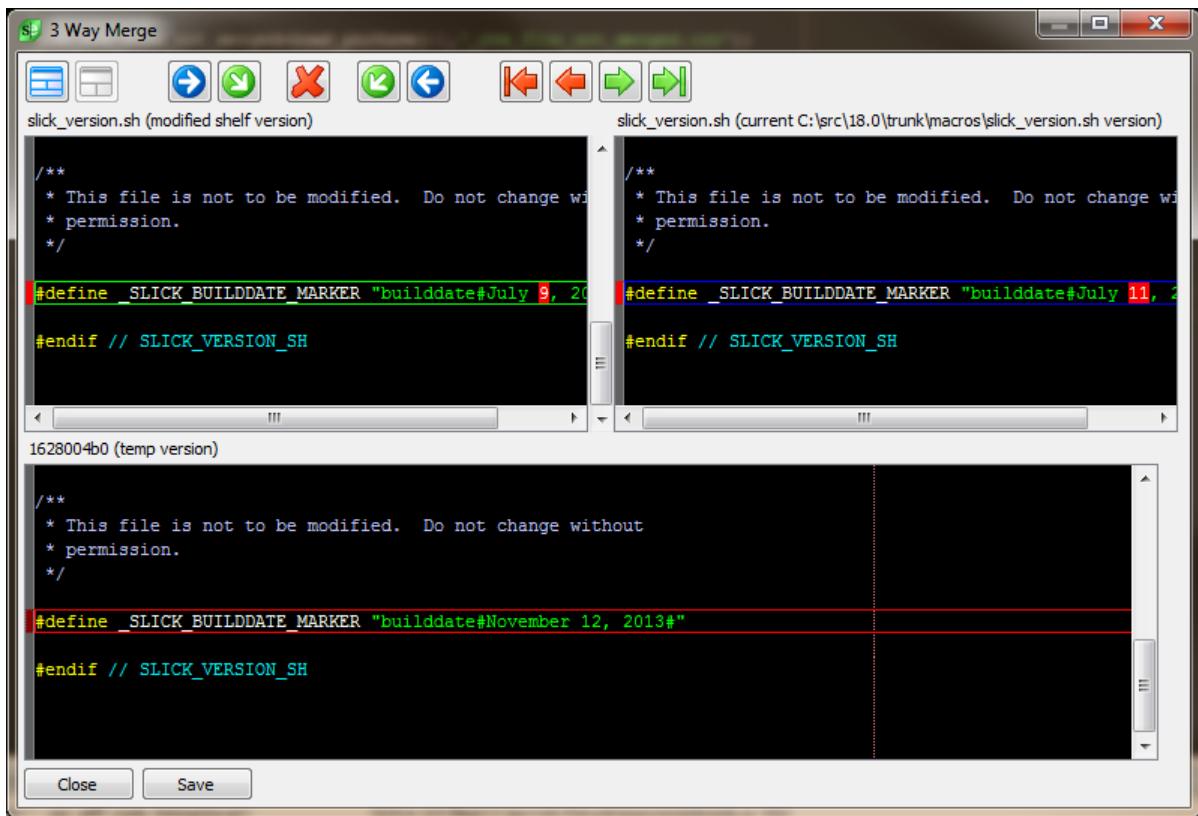


- **Check for Conflicts** - Before you can unshelve a set of shelved files, you first need to check for conflicts. This ensures that no new local changes will be overwritten by the files from the shelf. Clicking check for conflicts will pull the files from the zip file to temp files and use the 3-way merge engine to merge local files with shelved files and ensure no conflicts exist. This step may take a moment to complete.
- **Resolve Conflict...** - If there are conflicts, you will need to resolve them before you can unshelve a set of shelved files. It is possible to have 3-way conflicts and 2-way conflicts. 3-way conflicts occur when there is a collision between changes in the shelved file and the current local file. 2-way conflicts occur when a file in the shelf was new, but now a file by the same name exists locally. 3-way conflicts are resolved by using SlickEdit's GUI 3-way merge. The resulting file is saved to a temp file. 2-way conflicts are resolved by using SlickEdit's diff dialog. The resulting file is saved to a temp file.

After clicking the **Check for Conflicts** button, files with conflicts will appear in red and show the files

being merged as children in the Unshelve dialog. To resolve the conflict, select that item in the tree and click **Resolve Conflict**....



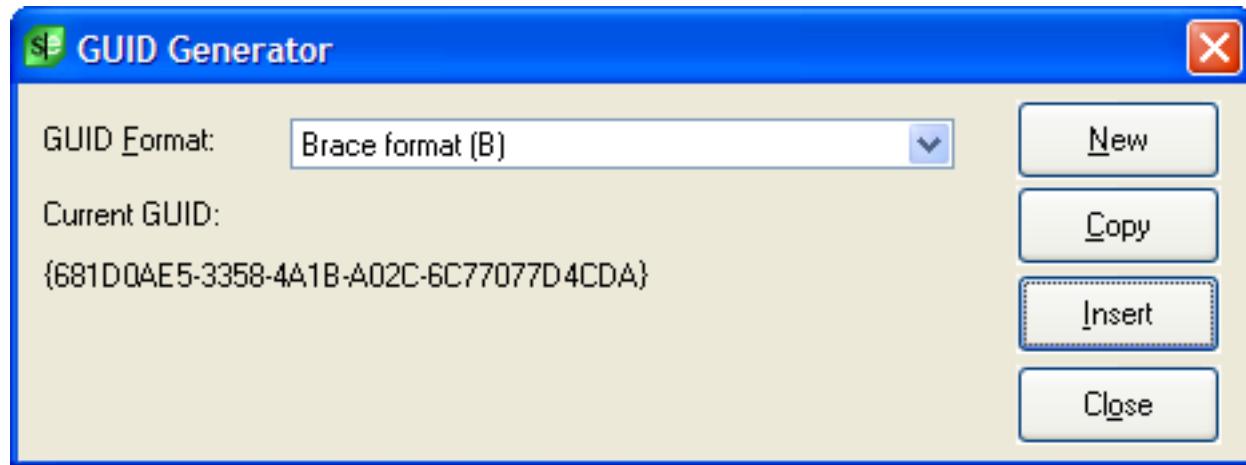


- **Edit** - Displays current shelf dialogs which allows to change comments for your shelf or add more files.
- **Delete** - Allows you to delete a shelf (shelves are .zip files).
- **Open...** - Allows you to choose a shelf not in the list.

GUID Generator

The GUID Generator creates a Globally Unique Identifier for use in your programs. While they are not guaranteed to be unique, the likelihood of generating the same identifier twice is very small.

To run the GUID Generator, select **Tools** → **Generate GUID** from the main menu.



Select the format for the new GUID using the **GUID Format** list. Click the **New** button to generate a new GUID. Click the **Copy** button to copy the current GUID to the clipboard. Click the **Insert** button to insert the current GUID into the current file at the cursor location.

Spell Checking

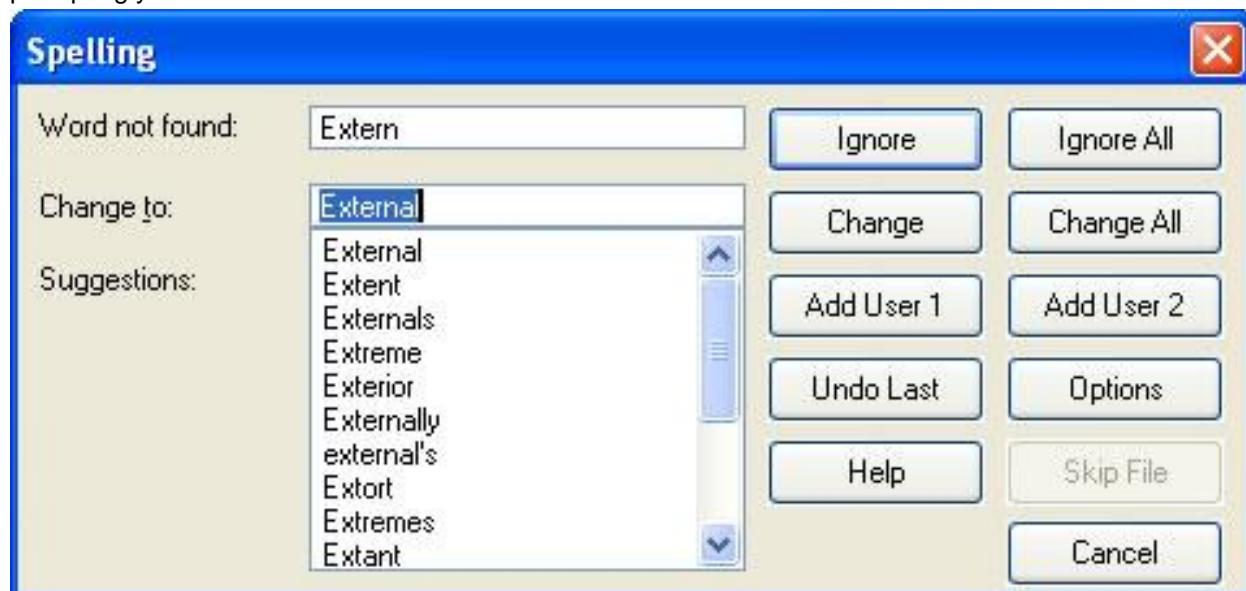
Spell Check Operations

You can access spell checking operations from the main menu by clicking **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.

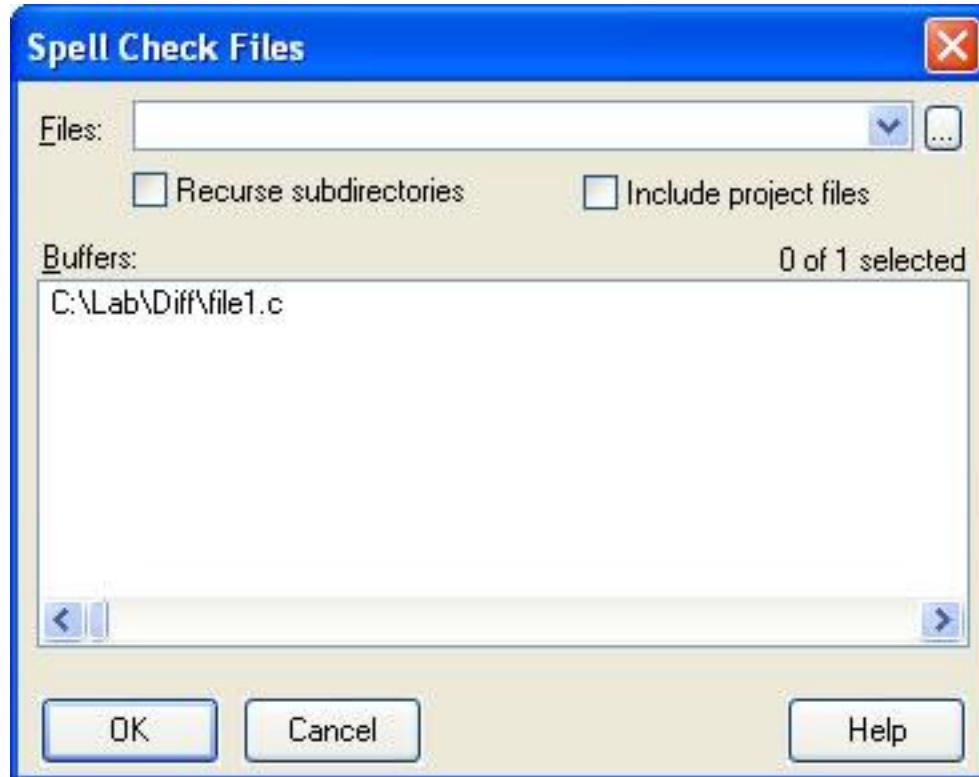


The dialog shows the word not found and gives suggestions for a word replacement. Use the buttons on the dialog to perform the following actions:

- **Ignore** - Disregard this word and continue spell checking.
- **Ignore all** - Disregard all instances of this word in the selected range and continue spell checking.
- **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 `userdct1.1st` and `userdct2.1st`. See [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 Options dialog open to the Spell Check Options node. Options include specifying the default dictionary, ignoring uppercase words, and detecting repeated words. You can also access these Spell Check Options from the menu item **Tools** → **Spell Check** → **Spell Options** or by using the **spell_options** command. See [Spell Check Options](#) for more information.
- **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 click **Tools** → **Spell Check** → **Check Files** (or use the **spell_check_files** command).



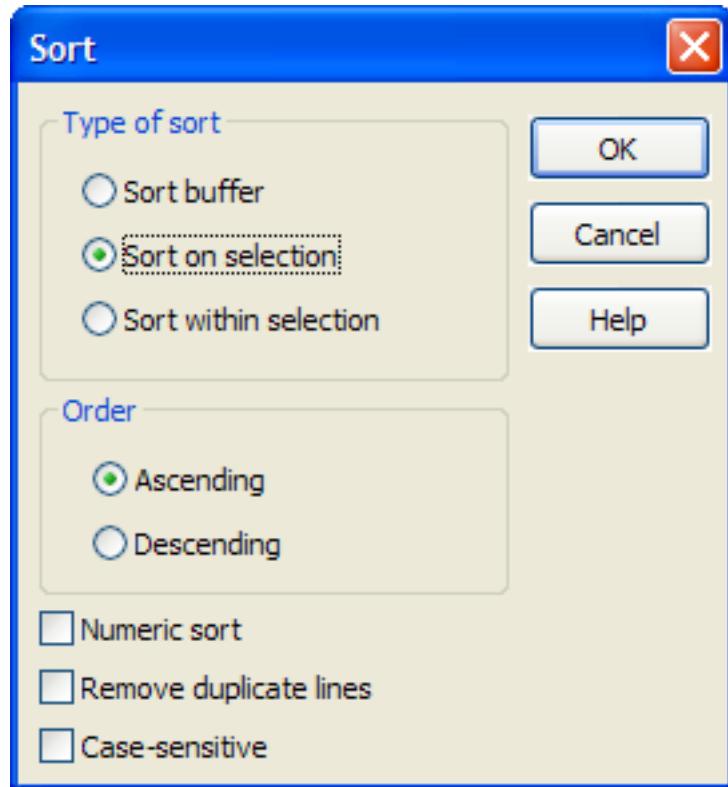
In the **Files** text box, enter one or more files separated by spaces. Wildcards may be used (for example, *.html or *.c). You can use the **Browse** button to the right of this text box to choose a directory. There are two file options available:

- **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.

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, click **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
A	Sort in ascending order.
D	Sort in descending order.
I	Case insensitive sort (ignore case).
E	Case sensitive sort (exact case which is the default).
-N	Numeric sort. C-style floating point numbers with up to 32-digit mantissa are supported.
-F	File name sort.

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 Window

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, click **View** → **Tool Windows** → **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, click **View** → **Tool Windows** → **FTP Client**. Right-click on files to display a menu of FTP operations.

FTP Profile Manager

To create a new FTP connection profile, complete the following steps:

1. From the main menu, click **File** → **FTP** → **Profile Manager(ftpprofilemanager** command). Alternatively, you can display the FTP tool window (**View** → **Tool Windows** → **FTP**) and click the button labeled **Start a New Session** . 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** button  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 the **Stoplight** button



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

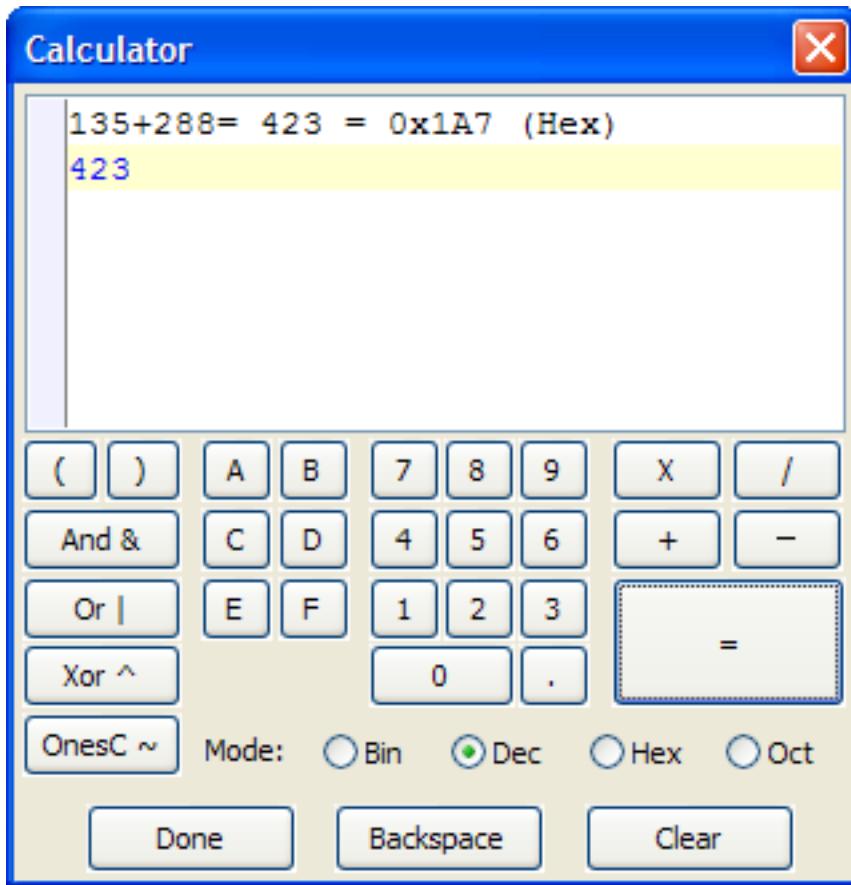
There are two types of settings available for working with FTP:

- **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, open the [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, from the main menu, click **Tools** → **Options**, expand **Network & Internet Options** and select **FTP Default Options**. See [FTP Default Options](#) for more information.

Using the Calculator and Math Commands

The Calculator

To access the calculator, click **Tools** → **Calculator**, or use the **calculator** command.



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 **Hex** then 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.

Operator	Meaning
&, 	bitwise AND, bitwise OR
^	xor
+, blank(s), -	addition, implied addition, subtraction

<code>*, /, %</code>	multiplication, division, remainder
<code>**</code>	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:

Example	Description
<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 **nextprime** 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 macOS, Konquerer on Linux KDE desktop, etc.

To display the OS file browser, click **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.

Interactive tool window (Pro only)

Overview

Interactive shells are great for learning a language or testing small pieces of code. Select some text and press Ctrl+Alt+Enter to load the selected code in an interactive shell. The interactive shell will be automatically started if necessary.

Interactive shells are preconfigured for many languages including Clojure, CoffeeScript, C#, Groovy, Haskell, Lua, PHP, Perl, PowerShell, Python, R, Ruby, and Scala.

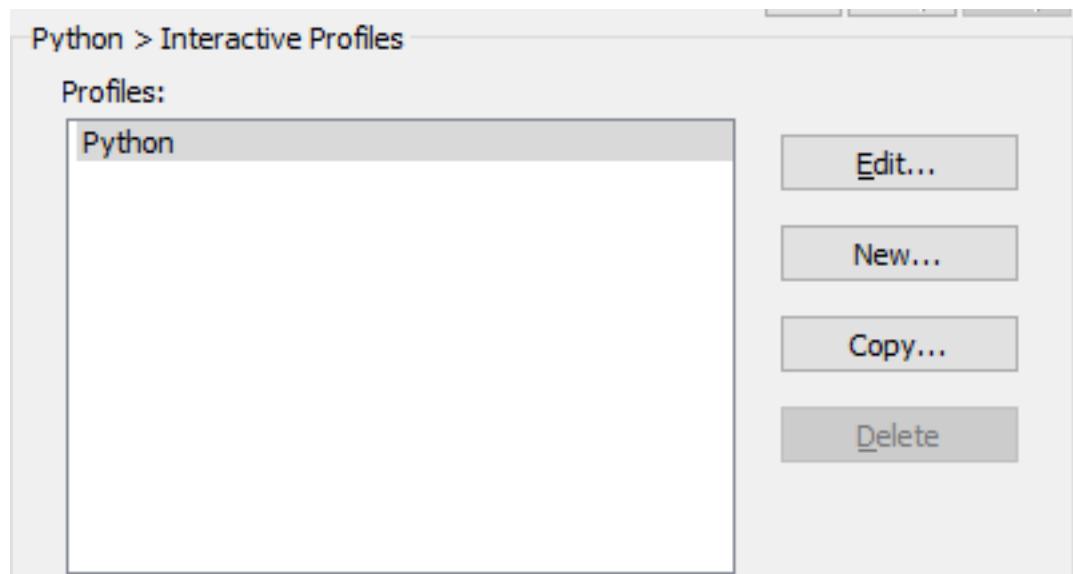
To open the Interactive tool window, select it from the tool window list **View → Tool Windows**. When the Interactive tool window is opened, it will display a list of interactive shell profiles for various languages. Double click on a profile to start that specific interactive shell. To open another interactive shell, right click on the interactive shell tab and select a different interactive shell profile.

Configuring Interactive Profiles

Go to **Document → [Language] Options → Interactive Profiles** to change or add interactive shell profiles for a specific language.

Note

Interactive profiles are stored in the configuration directory in `user.cfg.xml`.



- **Edit...** - Displays the Interactive Profile Settings dialog for the selected profile.

- **New...** - Creates a new profile.
- **Copy...** - Copies the settings from the selected profile.
- **Delete** - Deletes the selected profile. System profiles can't be deleted.

Regular Expressions

An Overview

searching

replacing

Regular Expressions

Regular expressions are patterns of text used to match and manipulate strings in your code. These patterns are expressed with combinations of characters defined by the regular expression syntax being used. A regular expression is sometimes referred to as a "regex".

Use regular expressions in your search and replace operations when you find normal search/replace too limiting. For example, with regular expressions, you can:

- Find quoted strings.
- Find blank lines.

- Find words starting at the beginning of lines.
- Find two words separated by any number of spaces or other text.

SlickEdit® supports several types of regular expression syntax:

- [Perl Regular Expressions](#)
- [SlickEdit® Regular Expressions](#)
- [Vim Regular Expressions](#)
- [Wildcards](#)

SlickEdit also provides a Regex Evaluator that you can use to interactively create, save, and re-use tests of regular expressions. See [The Regex Evaluator](#) for more information.

Unicode regular expression categories and character blocks are also supported. See [Unicode Categories and Character Blocks](#) for more information.

Note

- This documentation is not meant to be an exhaustive resource on regular expressions. Rather, we will present basic information, syntax charts, and examples. For novice users, there are many books and Web sites that go into more detail about this topic.

Using Regular Expressions in SlickEdit®

Specifying the Syntax to Use

All search and replace commands, the Find and Replace tool window, and incremental search support regular expressions. For search and replace commands and the tool window, you can specify the regular expression syntax to use through specific options. A global option is available to specify the default syntax to use when you invoke these features or when you use incremental search.

For example:

- **Search and replace commands** - When using the search commands `/` (slash) and `find`, or the replace commands `c` and `replace`, you can use the following options to specify regular expression syntax:
 - Use `R` to interpret the string as a SlickEdit regular expression.
 - Use the `L` option to interpret the string as a Perl regular expression.
 - Use the `~`option to interpret the string as a Vim regular expression.
- **Find and Replace tool window** - When using the tool window, select **Use** in the **Search options** box, and then pick the syntax to use from the drop-down list.
- **Incremental search** - When using incremental search, press **Ctrl+T** to toggle regular expression searching on and off. The syntax that will be used is based on the global syntax setting.

To set the global option, from the main menu, click **Tools** → **Options**, expand **Editing** and select **Search**. Set the **Regular expression** option to **True** and select the syntax you want to use from the **Expression type** drop-down list.

Minimal versus Maximal Matching

If you are using tagged expressions or regular expressions to perform a search and replace, it is important 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`.

Based on the syntax, the following regular expressions match the string `\path1\`:

Syntax	Expression
Perl	<code>^\\.*?\\</code>
SlickEdit	<code>^\\?*\\</code>
Vim	<code>^\\. {-}\\</code>

Minimal versus Maximal Matching

Syntax	Expression
--------	------------

The following regular expressions, which use the maximal operator, match the string `\path\path2\path3\`:

Syntax	Expression
Perl	<code>^\\.*\\</code>
SlickEdit	<code>^\\?@\\</code>
Vim	<code>^\\.*\\</code>

As a rule of thumb, the following minimal matching operators are generally used after a less-specific regular expression such as `?` in SlickEdit, `.` in Perl, or `.` in Vim

Syntax	Operators
Perl	<code>*?</code> and <code>+?</code>
SlickEdit	<code>*</code> and <code>+</code>
Vim	<code>\{-}</code> and <code>\{-1,}</code>

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 matched string with the character `$`, specify the following expressions:

Syntax	Expression
Perl	Search for: <code>([0-9]+)</code> Replace with: <code>\$1</code>
SlickEdit	Search for: <code>{[0-9]#}</code> Replace with: <code>\$#0</code>
Vim	Search for: <code>\([0-9]+\)</code> Replace with: <code>\$1</code>

If the minimal matching operator (`+?` in Perl, `+` in SlickEdit syntax) was used in the search string instead

of the maximal matching operator (+ in Perl, # in SlickEdit), the above search and replace would prefix each digit in the entire file with a \$ character.

Using Perl Tagged 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 expressions to insert parts of what is found into the replace string.

Tagged Expressions in Perl Search String

Perl Regex	Definition
(X)	Matches subexpression X and generates a tagged expression. The first tagged expression index is 1. No more tagged expressions are defined once an explicit tagged expression number is specified using (?dd).
(? dd X)	Matches subexpression X and adds tagged expression at the 1 or 2 digit index specified (01 is the same as 1). No more tagged expressions are automatically generated by the subexpression syntax (X) once this subexpression syntax is used. This is an extension to Perl syntax and will not work in a Perl script.
(?<name>X), (?'name'X), (?{name}X), (?P<name>X)	Matches subexpression X and specifies a tagged expression identified by name. A non-numeric tagged expression name must start with a letter [a-zA-Z] and be followed by the characters [a-zA-Z0-9_]. name may also be a numeric index [0-9]+ but this is an extension to Perl syntax and will not work in a Perl script. The first valid index is 1.
\k<name>, \k'name', \k{name}, \g<name>, \g'name', \g{name},	Matches tagged expression which was previously set. name may be a non-numeric tagged expression name or a numeric index. Using a numeric index for name will not work in a Perl script. \g<name>, and \g'name' are extensions to Perl syntax and will not work in a Perl script.
\gdigits	Matches tagged expression at index specified by digits.
\digits	Matches tagged expression at index specified by digits or specifies a 2-3 digit octal number. If the

Perl Regex	Definition
	tagged expression was not defined, \digits specifies a 2-3 digit octal number.
\g-digits \k<-digits>, \k'-digits', \k{-digits},	Matches relative tagged expression which was previously defined. -1 is the previous tagged expression, -2 is the previous before that, etc. \k<-digits>, \k'-digits', and \k{-digits} are extensions to Perl syntax and will not work in a Perl script.

Tagged Expressions in Perl Replace String

Perl Regex	Definition
\$digits	Specifies a tagged expression by index from search string. The first index is 1. Replaced with empty string if tagged expression was not set.
\digits	Specifies a tagged expression by index from search string. If the tagged expression was not defined, \digits specifies a 2-3 digit octal number.
\$+{name}	Specifies a tagged expression from search string. name may be a non-numeric tagged expression name or a numeric index. Using a numeric index for name will not work in a Perl script. Replaced with empty string if tagged expression was not set.
\k<name>, \k'name', \k{name}	Specifies a tagged expression from search string. name may be a non-numeric tagged expression name or a numeric index. Using a numeric index for name will not work in a Perl script. Replaced with empty string if tagged expression was not set. These are extensions to Perl syntax and will not work in a Perl script
\$&	Specifies the entire search string that was matched
\$(<exp1>X1 <exp2>X2 ... <expN>XN Xdefault)	Insert X corresponding to the first match group expression which evaluates to true. If no match group expression is true, insert Xdefault. The default clause is optional. exp specifies one or more tagged expression names or indexes separated with & (logical AND operator) or (logical OR operator). For example, <1&2> evaluates to true if

Perl Regex	Definition
	tagged expression index 1 and tagged expression index 2 matched something. The match group expression <1 2> evaluates to true if tagged expression index 1 or tagged expression index 2 matched something. There is no operator precedence. Instead, the operator farthest to the right has the highest precedence.

Tagged Expressions Perl Examples

Search For	Replace With	Description
(if while)	x\$1y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(if while)	x1y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(?4if while)	x\k<4>y	Replace occurrences of "if" and "while" with "xify" and "xwhiley". This is an extension to Perl syntax and will not work in a Perl script.
(?<n1>if while)	x\k<n1>y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(?<n1>if while)	x\${+{n1}}y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(?<n>if while)	x\$1y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(a) (b) (c)	\$(<1>x <2>y <3>z)	Replace occurrences of "a" with "x", "b" with "y", and "c" with "z".
(a) (b) c d	\$(<1>x <2>y z)	Replace occurrences of "a" with "x", "b" with "y", "c" with "z", and "d" with "z".
(?:a) (b))(?:c) (d))	\$(<1&3>AC <2&4>BD DD)	Replace occurrences of "ac" with "AC", "bd" with "BD", "ad" with "DD", and "bc" with "DD".

Using SlickEdit® Tagged Expressions

Search For	Replace With	Description
<code>^(.*?),(.*\$</code>	<code>\$2,\$1</code>	Reverse text on lines containing a coma. Lines with "abc,def" will be changed to "def,abc". Notice that the Perl regular expression search string uses a <code>.*?</code> minimal matching operator, so the comma matches the first comma in the line and not the last.
<code>(a)(b)(c)\g1</code>	<code>x\$1y</code>	Replace occurrences of "abca" with "xay".
<code>(a)(b)(c)\g-3</code>	<code>x\$1y</code>	Replace occurrences of "abca" with "xay".

Using SlickEdit® Tagged 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 expressions to insert parts of what is found into the replace string.

Tagged Expressions in SlickEdit® Search String

SlickEdit Regex	Definition
<code>{X}</code>	Matches subexpression X and generates a tagged expression. The first tagged expression index is 0. No more tagged expressions are defined once an explicit tagged expression number is specified using <code>(#dd)</code> or <code>{#ddX}</code> .
<code>(#ddX), {#ddX}</code>	Matches subexpression X and specifies a 1 or 2 digit tagged expression number index to use (01 is the same as 1). The expression <code>(#0if) (#0while)</code> is the same as <code>(# (if) (while))</code> .
<code>(#<name>X), (#'name'X), (#'{name}X), (#P<name>X)</code>	Matches subexpression X and specifies a tagged expression identified by name. name may be a non-numeric tagged expression name (start with a letter [a-zA-Z] and be followed by the characters [a-zA-Z0-9_]) or a numeric index ([0-9]+). The first valid index is 0.

Using SlickEdit® Tagged Expressions

SlickEdit Regex	Definition
<code>\k<name>, \k'name', \k{name}, \g<name>, \g'name', \g{name},</code>	Matches tagged expression which was previously set. <i>name</i> may be a non-numeric tagged expression name or a numeric index.
<code>\gdigits</code>	Matches tagged expression at index specified by <i>digits</i> .
<code>\g{-digits} \k<-digits>, \k'-digits', \k{-digits},</code>	Matches relative tagged expression which was previously defined. -1 is the previous tagged expression, -2 is the previous before that, etc.

Tagged Expressions in SlickEdit® Replace String

SlickEdit Regex	Definition
<code>#digits</code>	Specifies a tagged expression by index from search string. The first index is 0. Replaced with empty string if tagged expression was not set.
<code>\k<name>, \k'name', \k{name}</code>	Specifies a tagged expression from search string. <i>name</i> may be a non-numeric tagged expression name or a numeric index. Replaced with empty string if tagged expression was not set.
<code>#&</code>	Specifies the entire search string that was matched
<code>#(<exp1>X1 <exp2>X2 ... <expN>XN Xdefault)</code>	Insert <i>X</i> corresponding to the first match group expression which evaluates to true. If no match group expression is true, insert <i>Xdefault</i> . The default clause is optional. <i>exp</i> specifies one or more tagged expression names or indexes separated with & (logical AND operator) or (logical OR operator). For example, <1&2> evaluates to true if tagged expression index 1 and tagged expression index 2 matched something. The match group expression <1 2> evaluates to true if tagged expression index 1 or tagged expression index 2 matched something. There is no operator precedence. Instead, the operator farthest to the right has the highest precedence.

Tagged Expressions SlickEdit® Examples

Search For	Replace With	Description
{if while}	x#0y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(#4if while)	x\k<4>y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(#<n1>if while)	x\k< n1 >y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
(#<n>if while)	x#0y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
{a} {b} {c}	#(<0>x <1>y <2>z)	Replace occurrences of "a" with "x", "b" with "y", and "c" with "z".
((a) (b))({c) (d))	#(<0&2>AC <1&3>BD DD)	Replace occurrences of "ac" with "AC", "bd" with "BD", "ad" with "DD", and "bc" with "DD".
{a} {b} c d	#(<0>x <1>y z)	Replace occurrences of "a" with "x", "b" with "y", "c" with "z", and "d" with "z".
^{?*},{?*}\$	#1,#0	Reverse text on lines containing a coma. Lines with "abc,def" will be changed to "def,abc". Notice that the SlickEdit regular expression search string uses a * minimal matching operator, so the comma matches the first comma in the line and not the last.
{a}{b}{c}\g0	x#0y	Replace occurrences of "abca" with "xay".
{a}{b}{c}\g-3	x#0y	Replace occurrences of "abca" with "xay".

Using Vim Tagged 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 expressions to insert parts of what is found into the replace string.

Tagged Expressions in Vim Search String

Vim Regex	Definition
\(X\)	Matches subexpression X and generates a tagged expression. The first tagged expression index is 1. No more tagged expressions are defined once an explicit tagged expression number is specified using (?dd) .
\(1? dd X\)	Matches subexpression X and adds tagged expression at the 1 or 2 digit index specified (01 is the same as 1). No more tagged expressions are automatically generated by the subexpression syntax \(X\) once this subexpression syntax is used. This is an extension to Vim syntax and will not work in Vim
\(1?<name>X\), \('name'X\), \({name}X\), \(?P<name>X\)	Matches subexpression X and specifies a tagged expression identified by <i>name</i> . A non-numeric tagged expression name must start with a letter [a-zA-Z] and be followed by the characters [a-zA-Z0-9_] . <i>name</i> may also be a numeric index [0-9]+ . The first valid index is 1. These are extensions to Vim syntax and will not work in Vim.
\g<name>, \g'name',	Matches tagged expression which was previously set. <i>name</i> may be a non-numeric tagged expression name or a numeric index. These are extensions to Vim syntax and will not work in Vim.
\digit	Matches tagged expression at index specified by <i>digit</i> . \digit specifies a 1 digit number (1..9). Use \g to specify a larger index.
\g<-digits>, \g'-digits',	Matches relative tagged expression which was previously defined. -1 is the previous tagged expression, -2 is the previous before that, etc. These are extensions to Vim syntax and will not work in Vim.

Tagged Expressions in Vim Replace String

Vim Regex	Definition

Vim Regex	Definition
\digit	Specifies a tagged expression by index from search string. \digit specifies a 1 digit number (1..9). Use \g to specify a larger index.
\g<name>, \g'<name>'	Specifies a tagged expression from search string. name may be a non-numeric tagged expression name or a numeric index. Replaced with empty string if tagged expression was not set. These are extensions to Vim syntax and will not work in Vim
&, \0	Specifies the entire search string that was matched
~	Specifies previous replace string. This is only supported by the EX substitute command (i.e. s:/a/~/l)

Tagged Expressions Vim Examples

Search For	Replace With	Description
\(if\ while\)	x\1y	Replace occurrences of "if" and "while" with "xify" and "xwhiley".
\(if\ while\)	x\g<1>y	Replace occurrences of "if" and "while" with "xify" and "xwhiley". This is an extension to Vim syntax and will not work in Vim.
\(?4if\ while\)	x\g<4>y	Replace occurrences of "if" and "while" with "xify" and "xwhiley". This is an extension to Vim syntax and will not work in Vim.
\(?<n1>if\ while\)	x\g<n1>y	Replace occurrences of "if" and "while" with "xify" and "xwhiley". This is an extension to Vim syntax and will not work in Vim.
^(.*\?),(.*\)\$	\2,\1	Reverse text on lines containing a coma. Lines with "abc,def" will be changed to "def,abc". Notice that the Vim regular expression search

Replacing with Regular Expressions

Search For	Replace With	Description
		string uses a *? minimal matching operator, so the comma matches the first comma in the line and not the last.
(a)(b)(c)\g<1>	x\1y	Replace occurrences of "abca" with "xay".
(a)(b)(c)\g<-3>	x\1y	Replace occurrences of "abca" with "xay".

Replacing with Regular Expressions

When using regular expressions, some characters have a different meaning when used in the replace string, depending on the syntax:

- **Perl** - A backslash in the replace string has the same meaning as in the search string except for \g and a few others which make no sense (i.e. \oc, \u, \o:char etc.). A dollar sign (\$) must be escaped (\\$) when replacing a literal \$.
- **SlickEdit®** - The pound 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 \g and a few others which make no sense(\c, \u, :char etc.)
- **Vim** - A backslash in the replace string has the same meaning as in the search string except for those which make no sense (i.e. \<, \>, \k etc.).

See [Regular Expressions](#) for a complete list of regular expression syntax. See [Using Perl Tagged Expressions](#) or [Using SlickEdit® Tagged Expressions](#) for information on specifying tagged expressions in the replace string.

Case Modification in Replace

When used in a replace operation, the expressions in the following table can be used to modify the character casing of matched expressions. These work in Perl and SlickEdit syntaxes.

Expression	Description
\l	Convert next character to lowercase.
\u	Convert next character to uppercase.
\L	Convert all characters lowercase until \E.

Replacing with Regular Expressions

Expression	Description
\U	Convert all characters uppercase until \E.
\Q	Replace all characters literally until \E.
\E	End all case modification or \Q.

Examples of Replacing with Regular Expressions

The table below contains some examples of replace operations using regular expressions.

Operation	Expression
Search for occurrences "if" or "while" and replace with "IFIf" or "WHILEWhile".	Perl Search for: (if while) Replace with: \U\$1\E\U\$1 SlickEdit Search for: {if while} Replace with: \U#0\E\U#0
Search for occurrences of the string "hat" that occur at the end of a line and replace it with "cat".	Perl or SlickEdit Search for: hat\$ Replace with: cat
Delete blank lines.	Perl or SlickEdit Search for: ^\R Replace with: (leave blank) SlickEdit Search for: ^\n Replace with: (leave blank)
Replace occurrences of two consecutive blank lines with one.	Perl or SlickEdit: Search for: ^\R\R

The Regex Evaluator

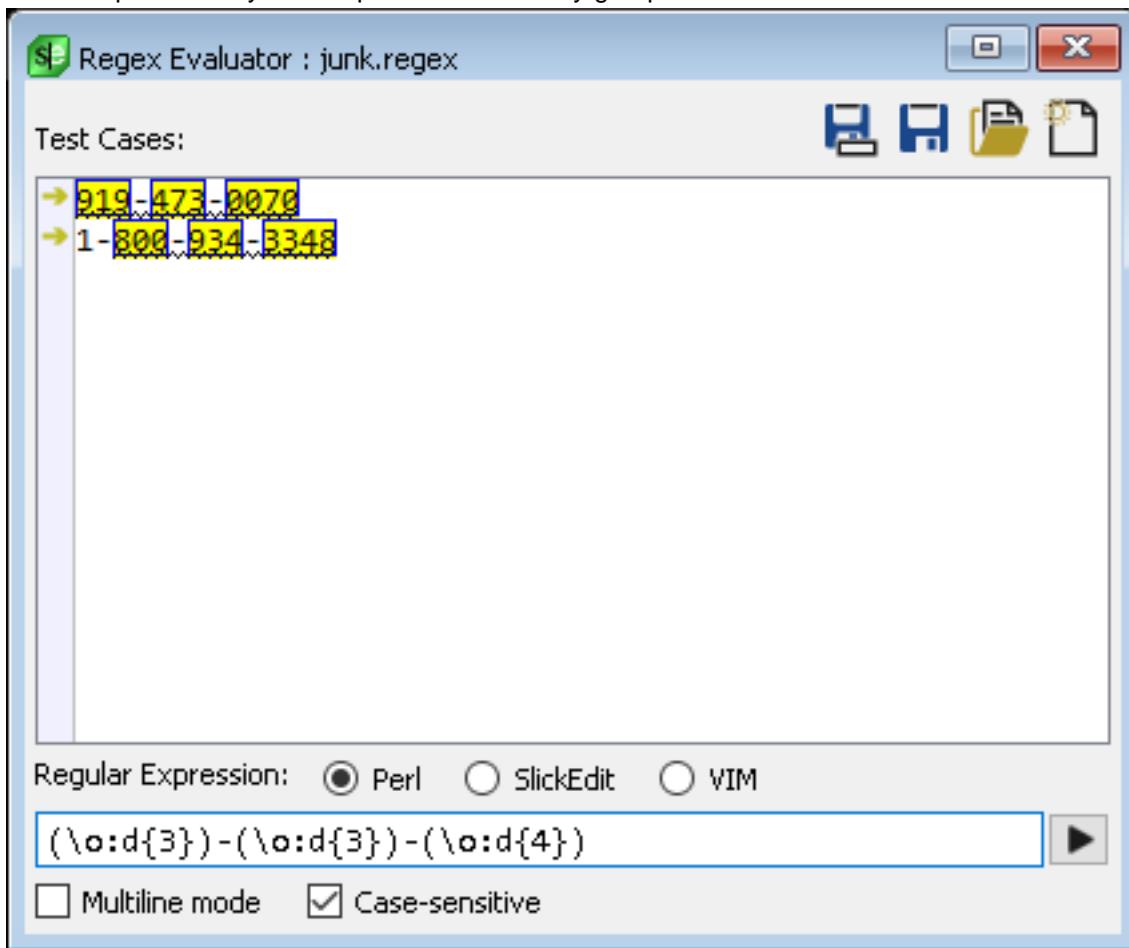
Operation	Expression
	<p>Replace with: \R</p> <p>SlickEdit:</p> <p>Search for: ^\n\n</p> <p>Replace with:\n</p>
Search for lines containing "a" and replace the "a" with a formfeed character.	<p>Perl:</p> <p>Search for:^a+\$</p> <p>Replace with:\o#{12}</p> <p>SlickEdit:</p> <p>Search for:^a+\$</p> <p>Replace with:\12</p>
Select occurrences of "Title:" at the beginning of a line and capitalize the text following "Title:".	<p>Perl</p> <p>Search for: ^Title: (.*)</p> <p>Replace with: Title: \U\1</p> <p>SlickEdit:</p> <p>Search for: ^Title\!: {?@}</p> <p>Replace with: Title: \U#0</p>

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, click **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.

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 tool tip 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: Perl, SlickEdit®, or Vim. 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 on by default.
- **New expression button** - To clear the tool window of all entries in order to start a new evaluation, click the button at the top of the tool window labeled **New expression**.
- **Open a saved expression button** - To open an expression that you have already saved, click the folder button at the top of the tool window labeled **Open a saved expression**.
- **Save the current expression button** - To save the current expression, click the diskette button 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 button** - To save the current expression with a different file name than what has previously been saved, click the button at the top of the tool window labeled **Save the current expression as**.

Perl Regular Expressions

Note

The intent of this Perl regular expression implementation is to be very compatible with the latest version of Perl. See [Compatibility Issues With Perl Regular Expressions](#) for known differences. There are a few extensions which are noted in the table below. See [Compatibility Issues Between Old and New Perl Regular Expressions](#) for a list of differences between the old and new implementation of Perl regular expressions.

Perl regular expressions are defined in the following table.

Perl Regular Expression	Definition
<code>^</code>	Matches beginning of line.
<code>\$</code>	Matches end of line.
<code>.</code>	Matches any character except newline unless in single line mode (also called match any character mode).
<code>X⁺</code>	Maximal match of one or more occurrences of X. See Minimal versus Maximal Matching .
<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 n_1 occurrences of X.
<code>X^{n1,}</code>	Maximal match of at least n_1 occurrences of X.
<code>X^{n1,n2}</code>	Maximal match of at least n_1 occurrences but not more than n_2 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 n_1 occurrences of X.
<code>X^{{n1,}?}</code>	Minimal match of at least n_1 occurrences of X.

Perl Regular Expression	Definition
X{,n2}?	Minimal match of at least zero occurrences but not more than <i>n2</i> occurrences of X.
X{n1,n2}?	Minimal match of at least <i>n1</i> occurrences but not more than <i>n2</i> 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)	Assert, positive look ahead. Searches for subexpression X, but X is not returned as part of the match. For example, to match words ending in "ed" while excluding "ed" as part of the match, use \b[a-z]+(?=ed\b) . See also (!X) .
(?<=X)	Assert, positive look behind. Matches subexpression X before the current position, but X is not returned as part of the match. For example, to match words starting with "ed" while excluding "ed" as part of the match, use (?<=\bed)[a-z]+\b . Variable length look behind is supported. For example, to match words that start with "a" or "ed" while excluding "a" or "ed" as part of the match, use (?<=\b(a ed))[a-z]+ . Variable length look behind will not work in a Perl script.
(?<!X)	Assert, negative look behind. Matches "not" subexpression X before the current position, but X is not returned as part of the match. For example, to match occurrences of "bar" not preceded by "foo", use (?<!foo)bar . Variable length look behind is supported. For example, to match occurrences of "bar" not preceded by "a" or "foo", use (?<!(a foo))bar . Variable length look behind will not work in a Perl script.
(?>X)	Matches expression X. Prohibit backtracking (give nothing back). It can be used to prevent the subexpression X from backtracking when using maximal (greedy) matching.
X*+	Maximal match of zero or more occurrences. Prohibit backtracking (give nothing back). It can be

Perl Regular Expression	Definition
	used to prevent the subexpression X from backtracking when using maximal (greedy) matching.
X++	Maximal match of one or more occurrences. Prohibit backtracking (give nothing back). It can be used to prevent the subexpression X from backtracking when using maximal (greedy) matching.
?+	Maximal match of zero or one occurrences. Prohibit backtracking (give nothing back).
X{n1}+	Match exactly <i>n1</i> occurrences of X. Prohibit backtracking (give nothing back).
X{n1,n2}+	Maximal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X. Prohibit backtracking (give nothing back).
(?(condition)yes-pattern no-pattern), (?(condition)yes-pattern)	<p>Matches <i>yes-pattern</i> if condition is true. Otherwise matches <i>no-pattern</i> if one was given. The <i>no-pattern</i> always matches if not given. Condition is one of the following:</p> <ul style="list-style-type: none"> • (digits), (<name>), ('name') Checks if tagged expression has matched something. For example, (?:this (that))(?(1)a b) matches "thisb" and "thata". • (?=X) True if X is a match. For example, (?(?=a)abc def) matches "abc" or "def". • (?!X) True if X is not a match • (?<=X) True if X matches before the current position.

Perl Regular Expression	Definition
	<ul style="list-style-type: none"> • (?<!X) True if X is not matched before the current position. • (R), (R<i>digits</i>), (R&<i>name</i>) Checks if tagged expression specified is in a recursive call. (R) specifies the whole search string. (R<i>digits</i>) specifies a tagged expression index. (R&<i>name</i>) specifies a named tagged expression. For example, (?(R)b a)(?R)?(?(R)c d) matches "abcd" and "abbccd". (?<foo>(?(R&foo)b a)(?&foo)?(?(R&foo)c d)) matches "abcd" and "abbccd".
(?(DEFINE)X)	Allows you to define subroutines in tagged expression syntax. No code is generated (i.e nothing is matched). This is useful for reusing a regular expression multiple times. For example, (?(DEFINE)(a))(?1)(?1) matches "aa". Since this subroutine was defined inside this construct, the tagged expression at index 1 is not set even though it was called. Since it's not set, it will return nothing when referenced as a backreference or in a replace string. Tagged expressions nested inside the subroutine do get set. Here's an example of how to give a subroutine a name using tagged expression syntax: (?(DEFINE)(?'s1'a))(?&s1)
(?R), (?0) (?<i>digits</i>), (?-<i>digits</i>), (?+<i>digits</i>), (?&<i>digits</i>), (?&-<i>digits</i>), (?&+<i>digits</i>), (?&<i>name</i>),	Calls tagged expression specified like it's a subroutine. (?R) or (?0) specifies the whole search string. (?<i>digits</i>) or (?&<i>digits</i>) specifies a tagged expression index. (?-1) or (?&-1) specifies the previous tagged expression, (?-2) or (?&-2) specifies the previous before that, etc. (?+1) or (?&+1) specifies the next tagged expression, (?+2) or (?&+2) specifies the next after that, etc. (?&<i>name</i>) specifies a named tagged expression. For example, a(?R)?b matches "ab" and "aaabbb". ([ab])(?1) matches "aa", "ab", "bb", and "ba". (?1)(a) matches "aa". (?<foo>a)(?&foo) matches "aa". (?&<i>digits</i>), (?&-<i>digits</i>), and (?&+<i>digits</i>) are extensions to Perl syntax and will not work in a Perl script.

Perl Regular Expression	Definition
(?#text)	Comment. No text is matched in this expression; it is used for comment and documentation only.
(? <i>OptionLetters</i> :X), (? <i>OptionLetters</i> - <i>OptionLetters</i>):X),	<p>Matches subexpression X using the options specified by <i>OptionLetters</i>. Option letters after the minus ('-') are turned off (or flipped). For example, (?-i:X) sets case sensitive matching. The form (?^<i>OptionLetters</i>:X) is only partially supported since the d option letter is not supported (see Perl documentation). <i>OptionLetters</i> is zero or more of the following option letters:</p> <ul style="list-style-type: none"> • i - Case insensitive matching • m - Multi line mode. When on, ^ an \$ match beginning and end of each line. When off, ^ and \$ match beginning and end of string/file. • n - Disabled automatic generation of numeric tagged expression with (X). • s - Single line mode. When on, . matches all characters including \x0d and 0x0a. Has no effect on \$, ^ or \N. • x - Skip whitespace mode. Used for commenting and/or making a regular expression more readable. When on, whitespace is ignored. • rl - Switch to Perl syntax regular expression within parenthesis. This is an extension to Perl syntax and will not work in a Perl script. • rr - Switch to SlickEdit syntax regular expression within parenthesis. For example (?rl:a?) matches 'a' followed by any character. This is an extension to Perl syntax and will not work in a Perl script.
(? <i>OptionLetters</i>), (? <i>OptionLetters</i> - <i>OptionLetters</i>)	Sets options specified by <i>OptionLetters</i> for outer subexpression. Option letters after the - are turned off (or flipped). For example ((?-i)A) matches A case sensitive. The form (?^ <i>OptionLetters</i>) is only partially supported since the d option letter is not supported (see Perl documentation). See meaning of <i>OptionLetters</i> above.

Perl Regular Expression	Definition
(X)	Matches subexpression X and specifies a numeric tagged expression. The first tagged expression is 1. Count left parenthesis from left to right to determine the the tagged expression number. No more tagged expressions are defined once an explicit tagged expression number is specified as shown below. For more information, see Using Perl Tagged Expressions .
(?:X)	Matches subexpression X but does not define a numeric tagged expression.
(? dd X)	Matches subexpression X and specifies a 1 or 2 digit tagged expression index to use (01 is the same as 1). No more numeric tagged expressions are automatically generated by the subexpression syntax (X) once this subexpression syntax is used. This is an extension to Perl syntax and will not work in a Perl script. The expression (?1if) (?1while) is the same as (? (if) (while)) . For more information, see Using Perl Tagged Expressions .
(? X)	Matches subexpression X and restores tagged expression numbering for each or branch (' '). For example, the tagged expressions inside the or branch expression (? (if) (while)) will both have the number 1.
(?<name>X), (?'name'X), (?{name}X), (?P<name>X)	Matches subexpression X and specifies a tagged expression identified by <i>name</i> . A non-numeric tagged expression name must start with a letter [a-zA-Z] and be followed by the characters [a-zA-Z0-9_] . <i>name</i> may also be a numeric index [0-9]+ but this is an extension to Perl syntax and will not work in a Perl script. For more information, see Using Perl Tagged Expressions .
\k<name>, \k'name', \k{name}, \g<name>, \g'name', \g{name},	Matches tagged expression which was previously set. \g<name> , and \g'name' are extensions to Perl syntax and will not work in a Perl script. For example, (?<n>a)\k<n> matches "aa". For more information, see Using Perl Tagged Expressions .
\gdigits	Matches tagged expression index specified by

Perl Regular Expression	Definition
	<i>digits</i> which was previously set. For example, (a)\g1 matches "aa". For more information, see Using Perl Tagged Expressions .
\d <i>digits</i>	Matches tagged expression index specified by <i>digits</i> or specifies a 2-3 digit octal number. If the tagged expression was not defined, \d <i>digits</i> specifies a 2-3 digit octal number.
\g{- <i>digits</i> } \k{<- <i>digits</i> >} , \k{'- <i>digits</i> '}, \k{<- <i>digits</i> >}, \k{'- <i>digits</i> '}, \k{<- <i>digits</i> >}, \k{'- <i>digits</i> '}	Matches relative tagged expression which was previously defined. -1 is the previous tagged expression, -2 is the previous before that, etc. For example, (a)(b)\g-2 matches "aba". \k{<- <i>digits</i> >} , \k{'- <i>digits</i> '}, and \k{<- <i>digits</i> >} are extensions to Perl syntax and will not work in a Perl script. For more information, see Using Perl Tagged Expressions .
X Y	Matches X or Y.
[<i>charset</i>]	Matches any one of the characters specified by <i>charset</i> . A dash (-) character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. Any backslash sequence which works outside brackets will work inside brackets if it specifies a character or set of characters. For example, \- specifies a literal dash character. The expression [x00-x1B] matches ASCII character codes 0..27. The expression \pL matches a unicode letter. The expression []] matches a right bracket. The expression [N] also matches a right bracket. The expression [^] matches a caret (^) character. [^] matches a caret (^) character.
[^ <i>charset</i>]	Matches any character not specified by <i>charset</i> . A dash (-) character may be used to specify ranges.
(?! [<i>extended-charset-expression</i>])	<p>BNF for <i>extended-charset-expression</i> is as follows:</p> <pre> <i>extended-charset-expression</i>: <i>unary-charset-expression</i> [<i>binary-operator</i> <i>extended-charset-expression</i>] <i>binary-operator</i>: + - & ^ --> Corresponds to OR(+) SUBTRACT(-) AND(&) XOR(^) </pre>

Perl Regular Expression	Definition
	<p><i>unary-charset-expression: escape</i> $\rightarrow \backslash d, \backslash xhh, \backslash x\{hhhh\}, \backslash N\{\dots\}$, etc. $[charset] \rightarrow$ Like regular character set but white space is ignored.</p> <p>$!unary-charset-expression \rightarrow$ Not the character set.</p> <p>$(extended-charset-expression)$</p> <p>$[:POSIX-character-class:]$</p> <p>White space is ignored before and after <i>unary-charset-expression</i>, unary not (!), <i>binary-operators</i>, and parenthesis around <i>unary-charset-expression</i>. Examples:</p> <ul style="list-style-type: none"> • $(?i [:digit:] + [a-z A-Z] - [0])$ - Match [A-Za-z1-9] (space not included here) • $(?i \x20 + [:digit:])$ - Match [0-9] (space included here) • $(?i [ab] & [ac])$ - Match [a] • $(?i ! ([ab] & [ac]))$ - Match [^a] • $(?i [ab] ^ [ac])$ - Match [bc] • $(?i [ab] - [ac])$ - Match [b] • $(?i ab)$ - Not valid (error!)
[:POSIX-character-class:]	<p>Matches <i>POSIX-character-class</i> specified. Only valid inside character set (i.e [[:alpha:]]). <i>POSIX-character-class</i> is one of the following:</p> <ul style="list-style-type: none"> • alpha - [A-Za-z] • alnum - [A-Za-z0-9] • ascii - Any alphabetical character [A-Za-z0-9] • blank - [\x20\t]

Perl Regular Expression	Definition
	<ul style="list-style-type: none"> • cntrl - [\x0-\1f\x7f] • digit - [0-9] • graph - [\x21-\x7e] • lower - [a-z] • print - [\x20-\x7e] • upper - [A-Z] • word - [A-Za-z0-9_] • xdigit - [0-9a-fA-F]
\char	Declares character after slash to be literal. For example, * represents the star character. You may not escape the letters [a-zA-Z].
\a	Matches bell character (ASCII 7).
\A	Matches beginning of string/file.
\b	Matches at word boundary. For example, \bre matches all occurrences of "re" that only occur at the beginning of a word.
\B	Matches all except at word boundary. For example, \Bre matches all occurrences of "re" as long as it is not at the start of a word.
\cx	Control character (ASCII values 0-31) '@' <=x<=' '_
\d	Equivalent to \p{Nd}. Can also be used inside a character class.
\D	Equivalent to \P{Nd}. Can also be used inside a character class.
\e	Matches escape character (ASCII 27).
\E	Terminates \Q. See \Q.
\f	Matches formfeed character (ASCII 12).

Perl Regular Expression	Definition
\h	Matches horizontal whitespace character. Same as [\t\x{20}\x{a0}\x{1680}\x{180e}\x{2000}\x{2001}\x{2002}\x{2003}\x{2004}\x{2005}\x{2006}\x{2007}\x{2008}\x{2009}\x{200a}\x{202f}\x{205f}\x{3000}].
\H	Matches non-horizontal whitespace character. Same as [^\t\x{20}\x{a0}\x{1680}\x{180e}\x{2000}\x{2001}\x{2002}\x{2003}\x{2004}\x{2005}\x{2006}\x{2007}\x{2008}\x{2009}\x{200a}\x{202f}\x{205f}\x{3000}].
\n	Matches newline character (ASCII 10). This escape will not match DOS <CR><LF>. Use \R to match newline sequence defined by current buffer.
\N	Matches any character except \n (ASCII 10). Not effected by single line mode.
\N{U+hhhh}	Matches up to 31-bit Unicode hexadecimal character specified by hhhh.
\N{UnicodeCharName}	Matches up <i>UnicodeCharName</i> specified. A complete list of character names is available on-line from the Unicode Consortium, http://www.unicode.org/charts/charindex.html
\o<	Match beginning of word.
\o>	Match end of word.
\oc	Case-sensitive match. Turns on case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside the current grouping level, after which case matching is restored to the previous case match setting. Note that this is equivalent to (?-i:X). See also \oi. This is an extension to Perl syntax and is not supported in a Perl script.
\od	Matches any 2-byte DBCS character. This escape is only valid in a match set ([... \od ...]). [^\od] matches any single byte character excluding end-

Perl Regular Expression	Definition
	of-line characters. When used to search Unicode text, this escape does nothing. This is an extension to Perl syntax and is not supported in a Perl script.
\oi	Ignore case. Turns off case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside the current grouping level, after which case matching is restored to the previous case match setting. Note that this is the equivalent to the Perl syntax (?i:X). See also \oc. This is an extension to Perl syntax and is not supported in a Perl script.
\ol	Turns off match any character mode (default). You can still use \R to create regular expressions which match one or more lines. However, expressions like .+ will not match multiple lines. This is much safer and usually faster than using the \om option. This is an extension to Perl syntax and is not supported in a Perl script. This is equivalent to (?-s:X)
\om	Turns on match any character mode. This enhances match any character (.) to support matching end-of-line characters. For example, \om.+ matches the rest of the buffer. This is an extension to Perl syntax and is not supported in a Perl script. This is equivalent to (?s:X)
\oz	Specifies cursor position if match is found. If the expression ab\ozc is found, the cursor is placed after the ab. This is an extension to Perl syntax and is not supported in a Perl script.
\o#\{ <i>digits</i> }	Matches up to 31-bit Unicode character specified by decimal <i>digits</i> . This is an extension to Perl syntax and is not supported in a Perl script.
\o{\ooo}	Matches up to 31-bit Unicode character specified by octal number <i>ooo</i> .
\o: <i>char</i>	This is an extension to Perl syntax and is not supported in a Perl script. Matches predefined expression corresponding to <i>char</i> . The pre-defined expressions are:

Perl Regular Expression	Definition
	<ul style="list-style-type: none"> • \o:a [A-Za-z0-9] - Matches an alphanumeric character. • \o:c [A-Za-z] - Matches an alphabetic character. • \o:b (?:[\t]+) - Matches one or more space or tab characters. • \o:d [0-9] - Matches a digit. • \o:f (?:[^\\\[\\]\> ;,\t"]+) - Windows: Matches a file name part. • \o:f (?:[^/ \t"]+) - UNIX: Matches a file name part. • \o:h (?:[0-9A-Fa-f]+) - Matches a hex number. • \o:i (?:[0-9]+) - Matches an integer. • \o:n (?:(?:[0-9]+(?:\.[0-9]+))\.[0-9]+)(?:[Ee](?:\+ -)[0-9]+)) - Matches a floating number. • \o:p (?:(?:[A-Za-z0-9]\:\ \ \)(?:\ \)(?:\o:f(\ \)*\o:f) - Windows: Matches a path. • \o:p (?:(?:\)?:(?:f(\))*\ f) - UNIX: Matches a path. • \o:q (?:\"[^\""]*\" [^\"]*) - Matches a quoted string. • \o:v (?:[A-Za-z_][A-Za-z0-9_]*\$) - Matches a C variable. • \o:w (?:[A-Za-z]+) - Matches a word. <p>Warning</p> <div style="background-color: #e0e0e0; padding: 10px; border: 1px solid blue; margin-top: 10px;"> <p>\o:f and \o:p</p> <p>Windows - this regular expression should not be used to validate an operating system filename. The intent with this predefined regular expression is to make it useful in</p> </div>

Perl Regular Expression	Definition
	<p>practice for handling filenames output from compilers and filenames in source files. For example, space characters in filenames are not allowed.</p> <p>Unix - this regular expression should not be used to validate an operating system filename. The intent with this predefined regular expression is to make it useful in practice for handling filenames output from compilers and filenames in source files. For example, space, :, ", and " characters in filenames are not allowed even though the OS allows them. In the future, we may add < and > to the list of characters not allowed in a filename.</p>
\p{UnicodeCategorySpec}	<p>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. The Perl documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.</p>
\P{UnicodeCategorySpec}	<p>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. The Perl documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.</p>
\p{UnicodelsBlockSpec}	<p>Matches characters in <i>UnicodelsBlockSpec</i>. Where <i>UnicodelsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, [p{isGreek}] matches Unicode characters in the Greek block. The Perl documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.</p>
\P{UnicodelsBlockSpec}	<p>Matches characters not in <i>UnicodelsBlockSpec</i>. For example, [P{isGreek}] matches all characters that</p>

Perl Regular Expression	Definition
	are not in the Unicode Greek block. This is equivalent to <code>[^\p{isGreek}]</code> . The Perl documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.
<code>\Q</code> and <code>\E</code>	<code>\Q</code> matches all characters as literals until <code>\E</code> . This is useful for longer sequences of characters without the need for the escape character. <code>\Q</code> does not require termination with <code>\E</code> , as it will continue to match characters literally until the end of the search string. <code>\E</code> returns to using special character tokens for matching.
<code>\r</code>	Matches carriage return (ASCII 13).
<code>\R</code>	Matches newline character sequence. For an edit buffer, what this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX/Mac (ASCII 10), Classic Mac (ASCII 13), or user-defined ASCII file. Use <code>\x0a</code> or if you want to match an ASCII 10 character. For a string, this matches <code>(\x0d\x0a \x0a \x0d)</code> .
<code>\s</code>	Equivalent to <code>[\f\n\r\t\v\x85\p{Z}]</code> . Can also be used inside a character class.
<code>\S</code>	Equivalent to <code>[\^{\f\n\r\t\v\x85\p{Z}}]</code> . Can also be used inside a character class.
<code>\t</code>	Matches tab character (ASCII 9).
<code>\v</code>	Matches vertical whitespace character. Same as <code>[\x{0a}\x{0b}\x{0c}\x{0d}\x{85}\x{2028}\x{2029}]</code> .
<code>\W</code>	Matches non-vertical whitespace character. Same as <code>[\^{\x{0a}\x{0b}\x{0c}\x{0d}\x{85}\x{2028}\x{2029}]</code> .
<code>\w</code>	Equivalent to <code>[a-zA-Z0-9_]</code> . Can also be used inside a character class.
<code>\W</code>	Equivalent to <code>[^a-zA-Z0-9_]</code> . Can also be used inside a character class.

Perl Regular Expression Examples

Perl Regular Expression	Definition
\x hh	Matches hexadecimal character <i>hh</i> where $0 \leq hh \leq \text{0xff}$.
\x{hhhh}	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhhh</i> .
\z	Matches end of string/file.
\Z	Matches end of string/file or last newline sequence. Even if the last line ends with a newline sequence, this will match end of string/file

The precedence of operators, from highest to lowest, is as follows:

- +, *, ?, {}, +?, *?, ??, {}? (These operators have the same precedence.)
- concatenation
- |

Perl Regular Expression Examples

The table below shows examples of Perl regular expressions.

Perl Regular Expression Example	Description
^defproc	Matches lines that begin with the word defproc .
^definit\$	Matches lines that only contain the word definit .
^*name	Matches lines that begin with the string *name . Notice that the backslash must prefix the special character *.
[t]	Matches tab and space characters.
[x9\x20]	Matches tab and space characters.
[o#{9}\o#{32}]	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

Perl Regular Expression Example	Description
	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 .
^\o:p	Matches lines beginning with a file name.
xy+z	Matches x followed by one or more occurrences of y followed by z .
[a-z-[qw]]	Character set subtraction. Matches all English lowercase letters except q and w .
[\p{isGreek}&[\p{L}]]	Character set intersection. Matches all Unicode letters in the Greek block.
\x{6587}	Matches Unicode character with hexadecimal value 6587 .
[\p{L}-[qw]]	Matches all Unicode letters except q and w .
[\p{L}]	Matches all Unicode letters.
[\p{Lul}]	Matches all Unicode uppercase and lowercase letters.
[\P{L}]	Matches all Unicode characters that are not letters.
[\p{isGreek}]	Matches all Unicode characters in the Greek block.
\x0d\x0a\x01\x02	Matches a sequence of hex binary characters.
\o#{13}\o#{10}\o#{1}\o#{2}	Matches a sequence of decimal binary characters.

SlickEdit Regular Expressions

Note

The SlickEdit regular expression engine has been completely rewritten and has a number of new features. In order to add the new features in a way that made sense, a small number of constructs were changed which could cause macros you've written to break. See [Compatibility Issues Between Old and New SlickEdit® Regular Expressions](#) for a list of known compatibility issues between the old and new syntax.

SlickEdit regular expressions are defined in the following table.

SlickEdit Regular Expression	Definition
<code>^</code>	Matches beginning of line.
<code>\$</code>	Matches end of line.
<code>?</code>	Matches any character except newline unless in single line mode (also called match any character mode).
<code>X+</code>	Minimal match of one or more occurrences of X. See Minimal versus Maximal Matching for more information.
<code>X#</code>	Maximal match of one or more occurrences of X.
<code>X*</code>	Minimal match of zero or more occurrences of X.
<code>X@</code>	Maximal match of zero or more occurrences of X.
<code>X:n1</code>	Matches exactly $n1$ occurrences of X. Use <code>()</code> to avoid ambiguous expressions. For example <code>a:9()1</code> searches for nine instances of the letter a followed by a 1.
<code>X:n1,</code>	Maximal match of at least $n1$ occurrences of X.
<code>X:n1,n2</code>	Maximal match of at least $n1$ occurrences but not more than $n2$ occurrences of X.
<code>X:*n1,</code>	Minimal match of at least $n1$ occurrences of X.
<code>X:*n1,n2</code>	Minimal match of at least $n1$ occurrences but not more than $n2$ occurrences of X.

SlickEdit Regular Expression	Definition
	more than n_2 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)	Assert, positive look ahead. Searches for subexpression X, but X is not returned as part of the match. For example, to match words ending in "ed" while excluding "ed" as part of the match, use $\backslash b[a-z]\#(\#=ed\backslash b)$. See also (#!X).
(#<=X)	Assert, positive look behind. Matches subexpression X before the current position, but X is not returned as part of the match. For example, to match words starting with "ed" while excluding "ed" as part of the match, use $(\#<=\backslash bed)[a-z]+\backslash b$. Variable length look behind is supported. For example, to match words that start with "a" or "ed" while excluding "a" or "ed" as part of the match, use $(\#<=\backslash b(a ed))[a-z]+\backslash b$.
(#<!X)	Assert, negative look behind. Matches "not" subexpression X before the current position, but X is not returned as part of the match. For example, to match occurrences of "bar" not preceded by "foo", use $(\#<!foo)bar$. Variable length look behind is supported. For example, to match occurrences of "bar" not preceded by "a" or "foo", use $(\#<!(a foo))bar$.
(#>X)	Matches expression X. Prohibit backtracking (give nothing back). It can be used to prevent the subexpression X from backtracking when using maximal (greedy) matching.
X@+	Maximal match of zero or more occurrences. Prohibit backtracking (give nothing back). It can be used to prevent the subexpression X from backtracking when using maximal (greedy) matching.
X#+	Maximal match of one or more occurrences. Prohibit backtracking (give nothing back). It can be used to prevent the subexpression X from

SlickEdit Regular Expression	Definition
	backtracking when using maximal (greedy) matching.
X:+n1	Match exactly <i>n1</i> occurrences of X. Prohibit backtracking (give nothing back).
X:+n1,	Maximal match of at least <i>n1</i> occurrences of X. Prohibit backtracking (give nothing back).
X:+n1,n2	Maximal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X. Prohibit backtracking (give nothing back).
(#(condition)yes-pattern no-pattern), (#(condition)yes-pattern)	<p>Matches <i>yes-pattern</i> if condition is true. Otherwise matches <i>no-pattern</i> if one was given. The <i>no-pattern</i> always matches if not given. Condition is one of the following:</p> <ul style="list-style-type: none"> • (<i>digits</i>), (<<i>name</i>>), ('<i>name</i>') Checks if tagged expression has matched something. For example, (#:this (that))#(0)a b matches "thisb" and "thata". • (#=X) True if X is a match. For example, (#(#=a)abc def) matches "abc" or "def". • (#!X) True if X is not a match • (#<=X) True if X matches before the current position. • (#<!X) True if X is not matched before the current position. • (R), (R<i>digits</i>), (R&<i>name</i>) Checks if tagged expression specified is in a recursive call. (R) specifies the whole search string. (R<i>digits</i>) specifies a tagged expression

SlickEdit Regular Expression	Definition
	<p>index. (R&name) specifies a named tagged expression. For example, (#(R)b a)((#R) (#(R)c d)) matches "abcd" and "abbccd".</p> <p>(#<foo>(#(R&foo)b a)((#&foo) (#(R&foo)c d)) matches "abcd" and "abbccd".</p>
(#(DEFINE)X)	Allows you to define subroutines in tagged expression syntax. No code is generated (i.e nothing is matched). This is useful for reusing a regular expression multiple times. For example, (#(DEFINE){a})(#&0)(#&0) matches "aa". Since this subroutine was defined inside this construct, the tagged expression at index 0 is not set even though it was called. Since it's not set, it will return nothing when referenced as a backreference or in a replace string. Tagged expressions nested inside the subroutine do get set. Here's an example of how to give a subroutine a name using tagged expression syntax: (#(DEFINE)(#s1'a))(#&s1)
(#R), (#&digits), (#&-digits), (#&+digits), (#&name),	Calls tagged expression specified like it's a subroutine. (#R) specifies the whole search string. (#&digits) specifies a tagged expression index. (#&-1) specifies the previous tagged expression, (#&-2) specifies the previous before that, etc. (#&+1) specifies the next tagged expression, (#&+2) specifies the next after that, etc. (#&name) specifies a named tagged expression. For example, a((#R))b matches "ab" and "aaabbb". {[ab]}(#&0) matches "aa", "ab", "bb", and "ba". (#&0){a} matches "aa". (#<foo>a)(#&foo) matches "aa".
(##text)	Comment. No text is matched in this expression; it is used for comment and documentation only.
~X	Search fails if expression X is matched. The expression ^~(if) matches the beginning of all lines that do not start with if .
(#OptionLetters :X), (#OptionLetters - OptionLetters :X),	Matches subexpression X using the options specified by <i>OptionLetters</i> . Option letters after the minus ('-') are turned off (or flipped). For example, (#-i:X) sets case sensitive matching. The form (#^OptionLetters:X) is only partially supported since

SlickEdit Regular Expression	Definition
	<p>the d option letter is not supported (see Perl documentation). <i>OptionLetters</i> is zero or more of the following option letters:</p> <ul style="list-style-type: none"> • i - Case insensitive matching • m - Multi line mode. When on, ^ an \$ match beginning and end of each line. When off, ^ and \$ match beginning and end of string/file. • n - Disabled automatic generation of numeric tagged expression with (X). • s - Single line mode. When on, ? matches all characters including \x0d and \x0a. Has no effect on \$, ^ or \N. • x - Skip whitespace mode. Used for commenting and/or making a regular expression more readable. When on, whitespace is ignored. • rl - Switch to Perl syntax regular expression within parenthesis. For example (#rl:a.) matches 'a' followed by any character. • rr - Switch to SlickEdit syntax regular expression within parenthesis.
(#OptionLetters), (# OptionLetters - OptionLetters)	Sets options specified by <i>OptionLetters</i> for outer subexpression. Option letters after the - are turned off (or flipped). For example ((#-i)A) matches A case sensitive. The form (#^ OptionLetters) is only partially supported since the d option letter is not supported (see Perl documentation). See meaning of <i>OptionLetters</i> above.
(X)	Matches subexpression X.
{X}	Matches subexpression X and specifies a numeric tagged expression. The first tagged expression is 0. Count left parenthesis from left to right to determine the the tagged expression number. No more tagged expressions are defined once an explicit tagged expression number is specified as shown below. For more information, see Using SlickEdit® Tagged Expressions .

SlickEdit Regular Expression	Definition
(#ddX), {#ddX}	Matches subexpression X and specifies a 1 or 2 digit tagged expression number index to use (01 is the same as 1). The expression (#0if) (#0while) is the same as (# (if) (while)). For more information, see Using SlickEdit® Tagged Expressions .
(# X)	Matches subexpression X and restores tagged expression numbering for each or branch (' '). For example, the tagged expressions inside the or branch expression (# (if) (while)) will both have the number 0.
(#<name>X), (#'name'X), (#{name}X), (#P<name>X)	Matches subexpression X and specifies a tagged expression identified by name. name may be a non-numeric tagged expression name (start with a letter [a-zA-Z] and be followed by the characters [a-zA-Z0-9_]) or a numeric index ([0-9]+). The first valid index is 0. For more information, see Using SlickEdit® Tagged Expressions .
\k<name>, \k'name', \k{name}, \g<name>, \g'name', \g{name},	Matches tagged expression which was previously set. name may be a non-numeric tagged expression name or a numeric index. For example, (?<n>a)\k<n> matches "aa". For more information, see Using SlickEdit® Tagged Expressions .
\gdigits	Matches numeric tagged expression which was previously set. For example, {a}\g0 matches "aa". For more information, see Using SlickEdit® Tagged Expressions .
\g-digits \k<-digits>, \k'-digits', \k{-digits},	Matches relative tagged expression which was previously defined. -1 is the previous tagged expression, -2 is the previous before that, etc. For example, {a}{b}\g-2 matches "aba". For more information, see Using SlickEdit® Tagged Expressions .
X Y	Matches X or Y.
[charset]	Matches any one of the characters specified by charset. A dash (-) character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. Any backslash sequence which

SlickEdit Regular Expression	Definition
	works outside brackets will work inside brackets if it specifies a character or set of characters. For example, \- specifies a literal dash character. The expression [10-127] matches ASCII character codes 0..27. The expression \pL matches a unicode letter. The expression [] matches no characters. The expression [\]] matches a right bracket. The expression [^] matches a caret (^) character.
[~charset], [^charset]	Matches any character not specified by charset. A dash (-) character may be used to specify ranges. The expression [~A-Z] matches all characters except uppercase letters.
(#[extended-charset-expression])	<p>BNF for <i>extended-charset-expression</i> is as follows:</p> <pre> <i>extended-charset-expression</i>: <i>unary-charset-expression</i> [iinary-operator <i>extended-charset-expression</i>] <i>binary-operator</i>: + - & ^ --> Corresponds to OR(+) OR() SUBSTRACT(-) AND(&) XOR(^) <i>unary-charset-expression</i>: <i>escape</i> --> \d, \xhh, \x{hhhh}, \N{...}, etc. [charset] --> Like regular character set but white space is ignored. !<i>unary-charset-expression</i> --> Not the character set. (<i>extended-charset-expression</i>) [:POSIX-character-class:]</pre> <p>White space is ignored before and after <i>unary-charset-expression</i>, unary not (!), <i>binary-operators</i>, and parenthesis around <i>unary-charset-expression</i>.</p> <p>Examples:</p> <ul style="list-style-type: none"> • (#[[:digit:] + [a-z A-Z] - [0]]) - Match [A-Za-z1-9] (space not included here)

SlickEdit Regular Expression	Definition
	<ul style="list-style-type: none"> • (#[\x20 + [:digit:]]) - Match [0-9] (space included here) • (#[[ab] & [ac]]) - Match [a] • (#[! ([ab] & [ac])]) - Match [^a] • (#[[ab] ^ [ac]]) - Match [bc] • (#[[ab] - [ac]]) - Match [b] • (#[ab]) - Not valid (error!)
[:POSIX-character-class:]	<p>Matches <i>POSIX-character-class</i> specified. Only valid inside character set (i.e [[:alpha:]]). POSIX-character-class is one of the following:</p> <ul style="list-style-type: none"> • alpha - [A-Za-z] • alnum - [A-Za-z0-9] • ascii - Any alphabetical character [A-Za-z0-9] • blank - [\x20\t] • cntrl - [\x0-\1f\x7f] • digit - [0-9] • graph - [\x21-\x7e] • lower - [a-z] • print - [\x20-\x7e] • upper - [A-Z] • word - [A-Za-z0-9_] • xdigit - [0-9a-fA-F]
\ddd	<p>Matches decimal character <i>ddd</i> where 0<=ddd<=255.</p>
\char	<p>Declares character after slash to be literal. For example, \: represents the colon character.</p>
\a	<p>Matches bell character (7).</p>

SlickEdit Regular Expression	Definition
\A	Matches beginning of string or file.
\b	Matches at word boundary. For example, \bre matches all occurrences of "re" that only occur at the beginning of a word.
\B	Matches all except at word boundary. For example, \Bre matches all occurrences of "re" as long as it is not at the start of a word.
\c	Specifies cursor position if match is found. If the expression xyz\c is found, the cursor is placed after the z.
\d	Equivalent to \p{Nd}. Can also be used inside a character class.
\D	Equivalent to \P{Nd}. Can also be used inside a character class.
\e	Matches escape character (ASCII 27).
\E	Terminates \Q. See \Q.
\f	Matches formfeed character (ASCII 12).
\h	Matches horizontal whitespace character. Same as [\t\x{20}\x{a0}\x{1680}\x{180e}\x{2000}\x{2001}\x{2002}\x{2003}\x{2004}\x{2005}\x{2006}\x{2007}\x{2008}\x{2009}\x{200a}\x{202f}\x{205f}\x{3000}].
\H	Matches non-horizontal whitespace character. Same as [^\t\x{20}\x{a0}\x{1680}\x{180e}\x{2000}\x{2001}\x{2002}\x{2003}\x{2004}\x{2005}\x{2006}\x{2007}\x{2008}\x{2009}\x{200a}\x{202f}\x{205f}\x{3000}].
\n	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.

SlickEdit Regular Expression	Definition
	Use \x0a if you want to match an ASCII 10 character.
\n	Matches newline character (ASCII 10). This escape will not match DOS <CR><LF>. Use \R to match newline sequence defined by current buffer.
\N	Matches any character except \n (ASCII 10). Not effected by single line mode.
\N{U+hhhh}	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhhh</i> .
\N{UnicodeCharName}	Matches up <i>UnicodeCharName</i> specified. A complete list of character names is available on-line from the Unicode Consortium, http://www.unicode.org/charts/charindex.html
\o<	Match beginning of word.
\o>	Match end of word.
\oc	Case-sensitive match. Turns on case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside the current grouping level, after which case matching is restored to the previous case match setting. Note that this is equivalent to (#i:X) . See also \oi
\od	Matches any 2-byte DBCS character. This escape is only valid in a match set ([... \od ...]). [^\od] matches any single byte character excluding end-of-line characters. When used to search Unicode text, this escape does nothing.
\oi	Ignore case. Turns off case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside the current grouping level, after which case matching is restored to the previous case match setting. This is the equivalent to (#i:X) . See also \oc
\ol	Turns off match any character mode (default). You can still use \R to create regular expressions which

SlickEdit Regular Expression	Definition
	match one or more lines. However, expressions like ?+ will not match multiple lines. This is much safer and usually faster than using the \om option. This is equivalent to (#s:X)
\om	Turns on match any character mode. This enhances match any character (?) to support matching end-of-line characters. Note that this is equivalent to (#s:X) For example, \om.+ matches the rest of the buffer.
\oz	Specifies cursor position if match is found. If the expression ab\ozc is found, the cursor is placed after the ab .
\o#{digits}	Matches up to 31-bit Unicode character specified by decimal <i>digits</i> .
\o{ooo}	Matches up to 31-bit Unicode character specified by octal number <i>ooo</i> .
\o: char	Same as :char . See :char below.
\p{UnicodeCategorySpec}	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. The Perl documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.
\P{UnicodeCategorySpec}	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. The Perl documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.
\p{UnicodelsBlockSpec}	Matches characters in <i>UnicodelsBlockSpec</i> . Where <i>UnicodelsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, [\p{isGreek}] matches Unicode characters in the Greek block. The Perl

SlickEdit Regular Expression	Definition
	documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.
\P{UnicodeIsBlockSpec}	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}]. The Perl documentation page http://perldoc.perl.org/perluniprops.html lists all supported properties.
\Q and \E	\Q matches all characters as literals until \E. This is useful for longer sequences of characters without the need for the escape character. \Q does not require termination with \E, as it will continue to match characters literally until the end of the search string. \E returns to using special character tokens for matching.
\r	Matches carriage return (ASCII 13).
\R	Matches newline character sequence. For an edit buffer, what this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX/Mac (ASCII 10), Classic Mac (ASCII 13), or user-defined ASCII file. Use \x0a or if you want to match an ASCII 10 character. For a string, this matches (\x0d\x0a \x0a \x0d).
\s	Equivalent to [\f\n\r\t\v\x85\p{Z}]. Can also be used inside a character class.
\S	Equivalent to [^\f\n\r\t\v\x85\p{Z}]. Can also be used inside a character class.
\t	Matches tab character (ASCII 9).
\v	Matches vertical whitespace character. Same as [\x{0a}\x{0b}\x{0c}\x{0d}\x{85}\x{2028}\x{2029}].
\V	Matches non-vertical whitespace character. Same as [^\x{0a}\x{0b}\x{0c}\x{0d}\x{85}\x{2028}\x{2029}].

SlickEdit Regular Expression	Definition
\w	Equivalent to [a-zA-Z0-9_]. Can also be used inside a character class.
\W	Equivalent to [^a-zA-Z0-9_]. Can also be used inside a character class.
\x hh	Matches hexadecimal character <i>hh</i> where 0<=hh<=0xff.
\x{hhhh}	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhhh</i> .
\z	Matches end of string/file.
\Z	Matches end of string/file or last newline sequence. Even if the last line ends with a newline sequence, this will match end of string/file
: char	<p>Matches predefined expression corresponding to <i>char</i>. The predefined expressions are:</p> <ul style="list-style-type: none"> • :a [A-Za-z0-9] - Matches an alphanumeric character. • :b ([\t#\s]) - Matches one or more space or tab characters - note that :b is not like the Perl/.NET \s. • :c [A-Za-z] - Matches an alphabetic character. • :d [0-9] - Matches a digit. • :f ([~\[\]:\<> ;,\t"]#) - Windows: Matches a file name part. • :f ([~/\t"]#) - UNIX: Matches a file name part. • :h ([0-9A-Fa-f]#) - Matches a hex number. • :i ([0-9]#) - Matches an integer. • :n (([0-9]#(.[0-9]#) .[0-9]#)([Ee](+ -)[0-9]#)) - Matches a floating number. • :p ([[A-Za-z0-9]!:\ \ \)(\ /)(:f(\ /))@:f) - Windows: Matches a path.

SlickEdit Regular Expression Examples

SlickEdit Regular Expression	Definition
	<ul style="list-style-type: none">• :p ((/)(:f(/))@:f) - UNIX: Matches a path.• :q ("[^\\"]@\\\" [~"]@") - Matches a quoted string.• :v ([A-Za-z_ \$][A-Za-z0-9_ \$]@) - Matches a C variable.• :w ([A-Za-z]#) - Matches a word. <p>Warning</p> <div style="border: 1px solid blue; padding: 10px;"><p>:f and :p</p><p>Windows - this regular expression should not be used to validate an operating system filename. The intent with this predefined regular expression is to make it useful in practice for handling filenames output from compilers and filenames in source files. For example, space characters in filenames are not allowed.</p><p>Unix - this regular expression should not be used to validate an operating system filename. The intent with this predefined regular expression is to make it useful in practice for handling filenames output from compilers and filenames in source files. For example, space, ;, “, and " characters in filenames are not allowed even though the OS allows them. In the future, we may add < and > to the list of characters not allowed in a filename.</p></div>

The precedence of operators, from highest to lowest, is as follows:

- +, #, *, @, :, :* (These operators have the same precedence.)
- concatenation
- |

SlickEdit Regular Expression Examples

The table below shows examples of SlickEdit regular expressions.

SlickEdit Regular Expression Example	Description
^defproc	Matches lines that begin with the word defproc .
^definit\$	Matches lines that only contain the word definit .
^\:name	Matches lines that begin with the string :name . Notice that the backslash must prefix the colon character (:).
[t]	Matches tab and space characters.
[9\32]	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 z .
\x0d\x0a\x01\x02	Matches a sequence of hex binary characters.
\13\10\1\2	Matches a sequence of decimal binary characters.

Vim Regular Expressions

Note

The intent of this Vim regular expression implementation is to be very compatible with the latest version of Vim. See [Differences Between SlickEdit Vim and gvim](#) for known differences. There are a few extensions which are noted in the table below. This implementation of Vim regular expressions does support the various syntax modes \v, \m, \M, and \V but this documentation is for \m mode. Due to the fact that SlickEdit converts Vim regular expression into Perl syntax, some error messages may be confusing.

Vim regular expressions are defined in the following table.

Vim Regular Expression	Definition
<code>^</code>	Matches beginning of line or literal character '^'. To always match beginning of line, use <code>_^</code> (or <code>\v^</code>). To always match the literal character '^', use <code>\^</code> .
<code>\$</code>	Matches end of line or literal character '\$'. To always match beginning of line, use <code>_\$</code> (or <code>\v\$</code>). To always match the literal character '\$', use <code>\\$</code> .
<code>.</code>	Matches any character except newline unless in single line mode (also called match any character mode).
<code>X\+</code>	Maximal match of one or more occurrences of X. See Minimal versus Maximal Matching .
<code>X*</code>	Maximal match of zero or more occurrences of X or the literal character '*'. To always maximal match zero or more occurrences, use <code>\{,\}</code> . To always match the literal character '*', use <code>*</code> .
<code>X\?, X\=</code>	Maximal match of zero or one occurrences of X.
<code>X\{n1\}</code>	Match exactly $n1$ occurrences of X.
<code>X\{n1,\}</code>	Maximal match of at least $n1$ occurrences of X.
<code>X\{,n2\}</code>	Maximal match of at most $n2$ occurrences of X.
<code>X\{,\}, X\{\}</code>	Maximal match of zero or more occurrences of X.

Vim Regular Expression	Definition
<code>X\{n1,n2}</code>	Maximal match of at least $n1$ occurrences but not more than $n2$ occurrences of X.
<code>X\{-n1}</code>	Matches exactly $n1$ occurrences of X.
<code>X\{-n1,}</code>	Minimal match of at least $n1$ occurrences of X.
<code>X\{-,n2}</code>	Minimal match of at least zero occurrences but not more than $n2$ occurrences of X.
<code>X\{-,n2}</code>	Minimal match of at most $n2$ occurrences of X.
<code>X\{-,}</code> , <code>X\{-,</code>	Minimal match of zero or more occurrences of X.
<code>X\{-n1,n2}</code>	Minimal match of at least $n1$ occurrences but not more than $n2$ 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 <code>if</code> .
<code>X\@=</code>	Assert, positive look ahead. Searches for subexpression X, but X is not returned as part of the match. For example, to match words ending in "ed" while excluding "ed" as part of the match, use <code>\<[a-z]\+\(ed\)\>@=</code> . See also <code>X\@!</code> .
<code>X\@<=</code>	Assert, positive look behind. Matches subexpression X before the current position, but X is not returned as part of the match. For example, to match words starting with "ed" while excluding "ed" as part of the match, use <code>\(\<\(ed\)\)\@<=[a-z]\+\></code> . Variable length look behind is supported. For example, to match words that start with "a" or "ed" while excluding "a" or "ed" as part of the match, use <code>\(\<\(a\ ed\)\)\@<=[a-z]\+\></code> . Variable length look behind will not work in Vim.
<code>X\@<!</code>	Assert, negative look behind. Matches "not" subexpression X before the current position, but X is not returned as part of the match. For example, to match occurrences of "bar" not preceded by "foo", use <code>\(foo\)\@<!bar</code> . Variable length look behind is supported. For example, to match occurrences of

Vim Regular Expression	Definition
	"bar" not preceded by "a" or "foo", use <code>\(\(a foo\)\)\@<!bar</code> . Variable length look behind will not work in Vim.
<code>X\@></code>	Matches expression X. Prohibit backtracking (give nothing back). It can be used to prevent the subexpression X from backtracking when using maximal (greedy) matching.
<code>X\{n1\}\+ X\{n1,\}\+</code>	Match exactly n_1 occurrences of X. Prohibit backtracking (give nothing back). This is an extension to Vim syntax and will not work in Vim.
<code>X\{n1,\}\+ X\{,n2\}\+</code>	Maximal match of at least n_1 occurrences of X. Prohibit backtracking (give nothing back). This is an extension to Vim syntax and will not work in Vim.
<code>X\{,n2\}\+ X\{,\}\+</code>	Maximal match of at most n_2 occurrences of X. Prohibit backtracking (give nothing back). This is an extension to Vim syntax and will not work in Vim.
<code>X\{n1,n2\}\+ \(?(\condition)yes-pattern\ no-pattern\), \?(\condition)yes-pattern\)</code>	Maximal match of at least n_1 occurrences but not more than n_2 occurrences of X. Prohibit backtracking (give nothing back). This is an extension to Vim syntax and will not work in Vim.
	<p>Matches <i>yes-pattern</i> if condition is true. Otherwise matches <i>no-pattern</i> if one was given. The <i>no-pattern</i> always matches if not given. Condition is one of the following:</p> <ul style="list-style-type: none"> • <code>\(digits\)</code>, <code>\(<name>\)</code>, <code>\('name'\)</code> <p>Checks if tagged expression has matched something. For example, <code>\(?::this\ \(that\)\)\(\?:(1)a\ b\)</code> matches "thisb" and "thata".</p> <ul style="list-style-type: none"> • <code>\(?=X\)</code> <p>True if X is a match. For example,</p>

Vim Regular Expression	Definition
	<p><code>\(?(\?=a)abc def)</code> matches "abc" or "def".</p> <ul style="list-style-type: none"> • <code>\(?!X\)</code> True if X is not a match • <code>\(?<=X\)</code> True if X matches before the current position. • <code>\(?<!X\)</code> True if X is not matched before the current position. • <code>\(R\)</code>, <code>\(R digits\)</code>, <code>\(R&name\)</code> Checks if tagged expression specified is in a recursive call. <code>\(R\)</code> specifies the whole search string. <code>\(R digits\)</code> specifies a tagged expression index. <code>\(R&name\)</code> specifies a named tagged expression. For example, <code>\(?(\(R)b a)\)(\?R)\?(\?(R)c d)</code> matches "abcd" and "abbccd". <code>\(?<foo>\(?(\(R&foo)b a)\)(\?&foo)\)\?(\?(R&foo)c d))</code> matches "abcd" and "abbccd". These are extensions to Vim syntax and will not work in Vim.
<code>\(?(\DEFINE)X\)</code>	<p>Allows you to define subroutines in tagged expression syntax. No code is generated (i.e nothing is matched). This is useful for reusing a regular expression multiple times. For example, <code>\(?(\DEFINE)(a)\)(\?1)\(\?1\)</code> matches "aa". Since this subroutine was defined inside this construct, the tagged expression at index 1 is not set even though it was called. Since it's not set, it will return nothing when referenced as a backreference or in a replace string. Tagged expressions nested inside the subroutine do get set. Here's an example of how to give a subroutine a name using tagged expression syntax: <code>\(?(\DEFINE)(?'s1'a)\)(\?&s1\)</code>This is an extension to Vim syntax and will not work in Vim.</p>
<code>\(R\)</code> , <code>\(0\)</code> , <code>\(?digits\)</code> , <code>\(?-digits\)</code> , <code>\(?+digits\)</code> , <code>\(?&digits\)</code> , <code>\(?&-digits\)</code> , <code>\(?&+digits\)</code> ,	<p>Calls tagged expression specified like it's a subroutine. <code>\(R\)</code> or <code>\(0\)</code> specifies the whole</p>

Vim Regular Expression	Definition
\()	<p>search string. <code>\(?digits)</code> or <code>\(?&digits)</code> specifies a tagged expression index. <code>\(?-1)</code> or <code>\(?&-1)</code> specifies the previous tagged expression, <code>\(?-2)</code> or <code>\(?&-2)</code> specifies the previous before that, etc. <code>\(?+1)</code> or <code>\(?&+1)</code> specifies the next tagged expression, <code>\(?+2)</code> or <code>\(?&+2)</code> specifies the next after that, etc. <code>\(?&name)</code> specifies a named tagged expression. For example, <code>\a\(?R)\?b</code> matches "ab" and "aaabbb". <code>\([ab]\)\(?1)</code> matches "aa", "ab", "bb", and "ba". <code>\(?1)\(a)</code> matches "aa". <code>\(?<foo>a)\(?&foo)</code> matches "aa". These are extensions to Vim syntax and will not work in Vim.</p>
\(?OptionLetters :X), \(?OptionLetters - OptionLetters :X),	<p>Matches subexpression X using the options specified by <i>OptionLetters</i>. Option letters after the minus ('-') are turned off (or flipped). For example, <code>\(?-i:X)</code> sets case sensitive matching. <i>OptionLetters</i> is zero or more of the following option letters:</p> <ul style="list-style-type: none"> • i - Case insensitive matching • m - Multi line mode. When on, ^ an \$ match beginning and end of each line. When off, ^ and \$ match beginning and end of string/file. • n - Disabled automatic generation of numeric tagged expression with <code>\(X)</code>. • s - Single line mode. When on, . matches all characters including <code>\x0d</code> and <code>0x0a</code>. Has no effect on \$, ^ or <code>\R</code>. • x - This Perl style regex option is not supported. <p>These are extensions to Vim syntax and will not work in Vim.</p>
\(?OptionLetters\), \(? OptionLetters - OptionLetters\)	<p>Sets options specified by <i>OptionLetters</i> for outer subexpression. Option letters after the - are turned off (or flipped). For example <code>\(?-i\)\A</code> matches A case sensitive. See meaning of <i>OptionLetters</i> above. These are extensions to Vim syntax and will not work in Vim.</p>
\(X)	<p>Matches subexpression X and specifies a numeric</p>

Vim Regular Expression	Definition
	tagged expression. The first tagged expression is 1. Count left parenthesis from left to right to determine the the tagged expression number. No more tagged expressions are defined once an explicit tagged expression number is specified as shown below. For more information, see Using Vim Tagged Expressions .
\%(X\)	Matches subexpression X but does not define a numeric tagged expression.
\(\? dd X\)	Matches subexpression X and specifies a 1 or 2 digit tagged expression index to use (01 is the same as 1). No more numeric tagged expressions are automatically generated by the subexpression syntax \ (X\) once this subexpression syntax is used. This is an extension to Vim syntax and will not work in Vim. The expression \(\?1if\ \(\?1while\) is the same as \(\? \(\if\ \(\while\)\). This is an extension to Vim syntax and will not work in Vim. For more information, see Using Vim Tagged Expressions .
\(\? X\)	Matches subexpression X and restores tagged expression numbering for each or branch (' '). For example, the tagged expressions inside the or branch expression \(\? \(\if\ \(\while\)\) will both have the number 1. This is an extension to Vim syntax and will not work in Vim.
\(\?<name>X\), \(\?'name'X\), \(\?P<name>X\)	Matches subexpression X and specifies a tagged expression identified by <i>name</i> . A non-numeric tagged expression name must start with a letter [a-zA-Z] and be followed by the characters [a-zA-Z0-9_]. <i>name</i> may also be a numeric index [0-9]+. These are extensions to Vim syntax and will not work in Vim. For more information, see Using Vim Tagged Expressions .
\g<name>, \g'<name>',	Matches tagged expression which was previously set. These are extensions to Vim syntax and will not work in Vim. For example, (?<n>a)\g<n> matches "aa". These are extensions to Vim syntax and will not work in Vim. For more information, see Using Vim Tagged Expressions .

Vim Regular Expression	Definition
\digit	Matches tagged expression index specified by <i>digit</i> (1..9).
\g<-digits>, \g'-digits'	Matches relative tagged expression which was previously defined. -1 is the previous tagged expression, -2 is the previous before that, etc. For example, (a)(b)\g'-2' matches "aba". These are extensions to Vim syntax and will not work in Vim. For more information, see Using Vim Tagged Expressions .
X Y	Matches X or Y.
[charset]	Matches any one of the characters specified by <i>charset</i> . A dash (-) character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. The following backslash sequences have the same meaning inside or outside of brackets: \b, \e, \t, \r, \n, \R (extension), \\, and \char. When inside brackets, \xHH, \dDDDD, \oOOO, \uHHHH, and \UHHHHHHHHH have the same mean respectively as \%xHH, \%dDDDD, \%oOOO, \%uHHHH, and \%UHHHHHHHH. For example, \- specifies a literal dash character. The expression [x00-\x1B] matches ASCII character codes 0..27 . The expression []] matches a right bracket. The expression [\\] also matches a right bracket. The expression [\\] matches the text string " \\ " and does not generate a character set match. The expression [^] matches the text string " ^ ". [^] matches a caret (^) character.
[^charset]	Matches any character not specified by <i>charset</i> . A dash (-) character may be used to specify ranges.
[:POSIX-character-class:]	Matches <i>POSIX-character-class</i> specified. Only valid inside character set (i.e [[:alpha:]])]. POSIX-character-class is one of the following:
	<ul style="list-style-type: none"> • alpha - [A-Za-z] • alnum - [A-Za-z0-9] • ascii - Any alphabetical character [A-Za-z0-9]

Vim Regular Expression	Definition
	<ul style="list-style-type: none"> • blank - [\\x20\\t] • cntrl - [\\x0-\\1f\\x7f] • digit - [0-9] • graph - [\\x21-\\x7e] • lower - [a-z] • print - [\\x20-\\x7e] • upper - [A-Z] • word - [A-Za-z0-9_] • xdigit - [0-9a-fA-F] • return - \\x0d • tab - \\x09 • escape - \\x1b • backspace - \\x08
\char	Declares character after slash to be literal except for \\+, \\{, \\?, \\), \\%, \\digit, or \\letter. For example, * represents the star character. You may not escape the letters [a-zA-Z].
\%^	Matches beginning of string/file.
\%\$	Matches end of string/file.
\%xHH	Matches 1-2 digit character specified by hexadecimal <i>digits</i> .
\%ddigits	Matches up to 31-bit Unicode character specified by decimal <i>digits</i> .
\%ooo	Matches 1-3 digit character specified by octal digits ooo.
\%uHHHH	Matches 1-4 digit Unicode character specified by hexadecimal digits HHHH.

Vim Regular Expression	Definition
\%UHHHHHHHHH	Matches 1-8 digit Unicode character specified by hexadecimal digits HHHHHHHH.
\^, \\$, ., \[These force a particular style of operation independent of syntax mode (magic mode). \^ always matches beginning of line. \\$ always matches end of line. . always matches any character, \[always matches the literal characters '['.
_char	Specifies the corresponding \char character set with end-of-line characters (\%x0a and \%x0d) added for the following 'i', 'I', 'k', 'K', 'f', 'F', 'p', 'P', 's', 'S', 'd', 'D', 'x', 'X', 'o', 'O', 'w', 'W', 'h', 'H', 'a', 'A', 'l', 'L', 'u', and 'U'. For example, \a matches [a-zA-Z\r\n] and \a only matches [a-zA-Z]
\a	Matches [a-zA-Z].
\A	Matches [^a-zA-Z].
\b	Matches \x08 byte.
\B: char	<p>These are extensions to Vim syntax and is not supported in Vim. Matches predefined expression corresponding to <i>char</i>. The pre-defined expressions are:</p> <ul style="list-style-type: none"> • \B:a [A-Za-z0-9] - Matches an alphanumeric character. • \B:c [A-Za-z] - Matches an alphabetic character. • \B:b \(\?:[\t]\+\) - Matches one or more space or tab characters. • \B:d [0-9] - Matches a digit. • \B:f \(\?:[^\V:\V<> ; \t"]\+\) - Windows: Matches a file name part. • \B:f \(\?:[^/ \r\n\t"]\+\) - UNIX: Matches a file name part. • \B:h \(\?:[0-9A-Fa-f]\+) - Matches a hex number.

Vim Regular Expression	Definition
	<ul style="list-style-type: none"> • \B:i <code>\(\?:[0-9]\+\)</code> - Matches an integer. • \B:n <code>\(\?:(\(\?:[0-9]\+\)(\(\?:\.\[0-9]\+\)\)\ \.[0-9]\+\))(\(\?:[Ee]\(\?:\+\-\)\)[0-9]\+\)\)</code> - Matches a floating number. • \B:p <code>\(\?:(\(\?:[A-Za-z0-9]\:\ \ \ \ \)\ (\?\:\ \ \ \ \)\ (\?\:\B:f\(\ \ \ \ \)\)*\B:f)</code> - Windows: Matches a path. • \B:p <code>\(\?:(\(\?:\A\)\ ?\:(\?\:\f\(\A\))\ ^:\f)</code> - UNIX: Matches a path. • \B:q <code>\(\?:"[^r\n]"*\\"[^r\n]"*\")</code> - Matches a quoted string. • \B:v <code>\(\?:[A-Za-z_\\$][A-Za-z0-9_\\$]^*\")</code> - Matches a C variable. • \B:w <code>\(\?:[A-Za-z]\+\)</code> - Matches a word.

Vim Regular Expression	Definition
\B<	Match beginning of word. This is an extension to Vim syntax and will not work in Vim.
\B>	Match end of word. This is an extension to Vim syntax and will not work in Vim.
\Bc	Case-sensitive match. Turns on case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside the current grouping level, after which case matching is restored to the previous case match setting. See also \Bi. This is an extension to Vim syntax and is not supported in Vim.
\Bd	Matches any 2-byte DBCS character. This escape is only valid in a match set ([...]\Bd...]). [^\Bd] matches any single byte character excluding end-of-line characters. When used to search Unicode text, this escape does nothing. This is an extension to Vim syntax and is not supported in Vim.
\Bi	Ignore case. Turns off case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside the current grouping level, after which case matching is restored to the previous case match setting. Note that this is the equivalent to the \(\?i:X\). See also \Bc. This is an extension to Vim syntax and is not supported in Vim.
\BI	Turns off match any character mode (default). You can still use \R to create regular expressions which match one or more lines. However, expressions like .\+ will not match multiple lines. This is much safer and usually faster than using the \Bm option. This is equivalent to \(\?s:X\). This is an extension to Vim syntax and will not work in Vim.
\Bm	Turns on match any character mode. This enhances match any character (.) to support matching end-of-line characters. For example, \Bm.\+ matches the rest of the buffer. This is equivalent to \(\?s:X\). This is an extension to Vim syntax and will not work in Vim.

Vim Regular Expression	Definition
\Bz	Specifies cursor position if match is found (same as \zs). If the expression ab\Bzc is found, the cursor is placed after the ab . This is an extension to Vim syntax and is not supported in Vim.
\B#{ <i>digits</i> }	Matches up to 31-bit Unicode character specified by decimal <i>digits</i> . This is an extension to Vim syntax and is not supported in Vim.
\B{ <i>ooo</i> }	Matches up to 31-bit Unicode character specified by octal number <i>ooo</i> . This is an extension to Vim syntax and will not work in Vim.
\c	Set case sensitivity to ignore case for the entire search string.
\C	Set case sensitivity to match case for the entire search string.
\d	Equivalent to [0-9].
\D	Equivalent to [^0-9].
\e	Matches escape character (ASCII 27).
\f	Equivalent to [\\$\%+,\\-./0-9=A-Z_a-z~].
\F	Equivalent to [\\$\%+,\\-./=A-Z_a-z~].
\h	Equivalent to [a-zA-Z_].
\H	Equivalent to [^a-zA-Z_].
\i	Equivalent to [0-9A-Z_a-z].
\l	Equivalent to [A-Z_a-z].
\k	Equivalent to [0-9A-Z_a-z].
\K	Equivalent to [a-zA-Z_].
\l	Equivalent to [a-z].

Vim Regular Expression	Definition
	Equivalent to [^a-z].
\m	Set syntax mode to "magic" which is the default setting.
\M	Set syntax mode to "no magic" which makes more characters like '.' and '*' require escaping to get a regular expression-like effect.
\n	Matches newline character sequence. For an edit buffer, what this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX/Mac (ASCII 10), Classic Mac (ASCII 13), or user-defined ASCII file. Use \%\x0a if you want to match an ASCII 10 character. For a string, this matches \(\%\x0d\%\x0a\ \%\x0a\ \%\x0d\). Note that when \n is used in the replace string, it specifies an ASCII 0. Use \r in the replace string to insert the appropriate new-line sequence for your edit buffer.
\o	Equivalent to [0-9].
\O	Equivalent to [^0-9].
\p	Equivalent to [\x20-\x7e].
\P	Equivalent to [\x20-\x2F\x3a-\x7e].
\r	Matches carriage return (ASCII 13). Note that when \r is used in the replace string, the appropriate new-line sequence for the edit buffer is inserted.
\R	Matches newline character sequence. For an edit buffer, what this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX/Mac (ASCII 10), Classic Mac (ASCII 13), or user-defined ASCII file. Use \%\x0a if you want to match an ASCII 10 character. For a string, this matches \(\%\x0d\%\x0a\ \%\x0a\ \%\x0d\). This is an extension to Vim syntax and is not supported in Vim.
\s	Equivalent to [\t].

Vim Regular Expression Examples

Vim Regular Expression	Definition
\s	Equivalent to [^ \t]. Can also be used inside a character class.
\t	Matches tab character (ASCII 9).
\u	Equivalent to [A-Z].
\U	Equivalent to [^A-Z].
\v	Set syntax mode to "very magic" which makes characters like '+', '?', '=', '<', '>', and more no longer require escaping to get their regular expression meaning. For example .+ matches 1 or more characters.
\W	Matches non-vertical whitespace character. Same as [^\x{0a}\x{0b}\x{0c}\x{0d}\x{85}\x{2028}\x{2029}].
\V	Set syntax mode to "very no magic" which makes more characters like '.', '*', '\$', and '^' require escaping to get a regular expression-like effect. In this mode, the only character with special meaning is backslash. All other characters are considered literal.
\w	Equivalent to [0-9a-zA-Z].
\W	Equivalent to [^0-9a-zA-Z].
\x	Equivalent to [0-9A-Fa-f].
\X	Equivalent to [^0-9A-Fa-f].
\zs	Specifies cursor position if match is found (same as \zs). If the expression ab\Bzc is found, the cursor is placed after the ab . This is an extension to Vim syntax and is not supported in Vim.

Vim Regular Expression Examples

The table below shows examples of Vim regular expressions.

Vim Regular Expression Example	Description
<code>^defproc</code>	Matches lines that begin with the word defproc .
<code>^definit\$</code>	Matches lines that only contain the word definit .
<code>^*name</code>	Matches lines that begin with the string *name . Notice that the backslash must prefix the special character * .
<code>[t]</code>	Matches tab and space characters.
<code>[x9\x20]</code>	Matches tab and space characters.
<code>p.t</code>	Matches any three-letter string starting with the letter p and ending with the letter t . Two possible matches are pot and pat .
<code>s.\{-}t</code>	Matches the letter s followed by any number of characters followed by the nearest letter t . Two possible matches are seat and st .
<code>for while</code>	Matches the strings for or while .
<code>^B:p</code>	Matches lines beginning with a file name. This is an extension to Vim syntax and will not work in Vim.
<code>xy\+z</code>	Matches x followed by one or more occurrences of y followed by z .
<code>\%U00006587</code>	Matches Unicode character with hexadecimal value 6587 .
<code>\%x0d\%x0a\%x01\%x02</code>	Matches a sequence of hex binary characters.

Wildcard Expressions

SlickEdit® supports *, ?, and # wildcards:

- The asterisk (*) matches zero or more characters. For example, search for **a*b** to find any string that contains a lowercase letter "a" followed by a lowercase letter "b" allowing for text in between.
- The question mark (?) matches any single character. Use multiple question marks in succession to represent that number of characters. For example, search for **a???b** to find any string that contains a lowercase letter "a" followed by any three characters, followed by a lowercase letter "b".
- The pound sign (#) matches any single digit, 0-9. Use multiple pound signs in succession to represent that number of digits. For example, use **##:##** to search for four-digit time-of-day values.

Brief Regular Expressions

Support for Brief regular expressions has been removed. Use [Perl Regular Expressions](#), [Vim Regular Expressions](#), [SlickEdit® Regular Expressions](#), or [Wildcards](#))

UNIX Regular Expressions

Support for UNIX regular expressions has been removed. Use [Perl Regular Expressions](#), [Vim Regular Expressions](#), [SlickEdit® Regular Expressions](#), or [Wildcards](#))

Compatibility Issues With Perl Regular Expressions

The following is a list of compatibility issues between this implementation of Perl regular expressions and Perl

- <CR><LF> is handled differently than perl for better DOS line ending support.
 - . - Difference in multi-line mode only. Does not match <CR> for DOS <CR><LF> files or string if <LF> follows
 - \$ - Difference in multi-line mode only. Does not match <LF> for DOS <CR><LF> files or strings if <CR> precedes the <LF>.
 - \R - Does not match <LF> if the previous character is <CR> for DOS <CR><LF> files or string. In other words \R will only match a complete new-line sequence and not a partial new-line sequence.
- There are some extensions to Perl syntax like (**?dd X**), **\o#{ddd}**, **\o:char**, **\o<**, and **\o>**.
- Case insensitive matching of Unicode characters is not the same as Perl or done according to the Unicode consortium standards
- **\X, \K, \C, \G, \F** - Not supported.
- **c, a, d, l, u** - These option letters are not supported in option letter expressions like (**?imsxn-imsxn:X**).
- **(?{code}), (??{code})** - Not supported.

Compatibility Issues Between Old and New Perl Regular Expressions

The following is a list of compatibility issues between the old and new Perl Regular Expressions

- **\n, \N** - Previously matched new-line sequence defined for the file/buffer (for a string search with pos() there's no difference). Now matches a single character \n (ASCII 10). Use \R to match the new-line sequence defined for the file/buffer.
- **\z** - Previously place cursor. Now use \oz. \z now matches end of file/string - Previously place cursor. Now use \oz. \z now mathces end of file/string
- **\#ddd** - Previously matched decimal character. Now use **\o#{ddd}**.
- **\R** - Previously matched \r (ASCII 13). Now use \r instead. All escape are case sensitive.
- **\q, \e** - Previously these were case insensitive. Now these escapes are case sensitive so these lowercase versions will not be interpreted the same. Use **\Q** and **\E** instead.
- **[charset1-[charset2]], [charset1&[charset2]]** - Removed for better and more complete expression syntax. Use **(#[[charset1]-[charset2]])** and **(#[[charset1]&[charset2]])** respectively. See **(? [extended-charset-expression])** for more information.
- **[^charset]** - This will match \r or \n (ASCII 13 or ASCII 10) like Perl. Previously this did not match \r and \n. Change your expression to **[^charset\r\n]**
- Tagged expression in replace string can be two digits.

```
Search For:(1)234567890ab
Replace With: $11bar
(New Perl Regex) Translates "1234567890ab" to "bar"
(Old Perl Regex) Translates "1234567890ab" to "11bar"
```

```
Search For:(1)(2)(3)(4)(5)(6)(7)(8)(9)(0)(a)(b)
Replace With: $11bar
(New Perl Regex) Translates "1234567890ab" to "abar"
(Old Perl Regex) Translated "1234567890ab" to "11bar"
```

Use **\k<1>** to work around this issue. This is an extension to Perl syntax and will not work in a Perl script.

Compatibility Issues Between Old and New SlickEdit® Regular Expressions

The following is a list of compatibility issues between the old and new SlickEdit® Regular Expressions

- **:char** - Previously was case insensitive. Now *char* is case sensitive. **:l** is an error and **:i** matches an integer.
- **\C, \F, \Gd, \Xhh, \R, \N, \T** - Previously were case insensitive. Now all escape are case sensitive so these uppercase versions will not be interpreted the same.
- **\q, \e** - Previously these were case insensitive. Now these escapes are case sensitive so these lowercase versions will not be interpreted the same.
- **[charset1-[charset2]], [charset1&[charset2]]** - Removed for better and more complete expression syntax. Use **(#[[charset1]-[charset2]])** and **(#[[charset1]&[charset2]])** respectively. See **(#[extended-charset-expression])** for more information.
- Tagged expression in replace string can be two digits.

```
Search For:{1}234567890ab
Replace With: #11bar
(New SlickEdit Regex) Translates "1234567890ab" to "bar"
(Old SlickEdit Regex) Translates "1234567890ab" to "1bar"
```

```
Search For:{1}{2}{3}{4}{5}{6}{7}{8}{9}{0}{a}{b}
Replace With: #11bar
(New SlickEdit Regex) Translates "1234567890ab" to "bbar"
(Old SlickEdit Regex) Translated "1234567890ab" to "21bar"
```

Use **\k<1>**; to work around this issue.

Unicode Categories and Character Blocks

Unicode Category Specifications for Regular Expressions

The Unicode consortium standard regular expression categories are supported. The syntax for specifying categories is:

```
\p{MainCategoryLetter Subcategories}
```

The above syntax matches the categories specified. The following syntax matches all characters *not* in the categories specified:

```
\P{MainCategoryLetter Subcategories}
```

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 file `UnicodeData-3.1.0.txt` found on the Unicode Consortium Web site (<http://unicode.org>).

Subcategory	Description
Lu	Letter, Uppercase
Li	Letter, Lowercase
Lt	Letter, Titlecase
Lo	Letter, Other
Mn	Mark, Non-Spacing
Mc	Mark, Spacing Combining
Me	Mark, Enclosing
Nd	Number, Decimal Digit

**Unicode Category Specifications
for Regular Expressions**

Subcategory	Description
NI	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
ZI	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

**Unicode Category Specifications
for Regular Expressions**

Subcategory	Description
	this property)

Macros and Macro Programming

This chapter describes how to use macros with SlickEdit.

There are two types of macros in SlickEdit®:

- [Recorded Macros](#) - A recorded macro is a series of SlickEdit operations that you can save and run as one operation any time. Recorded macros are useful for automating repetitive tasks. Because recording a macro generates Slick-C® code for the action being recorded, recorded macros provide a good starting point for learning the Slick-C macro programming language.
- [Programmable Macros](#) - Slick-C lets you take editor customization to the next level. With Slick-C, you have more control over modifying or adding new functionality. "Programmable macros" is a term we use to describe Slick-C modules, Slick-C batch macros, and Slick-C variables. In other words, Slick-C code that you write yourself.

Recorded 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 the behavior of SlickEdit.

Topics in this section are:

- [Recorded Macros](#)
- [Recording a Macro](#)
- [Binding Recorded Macros to Keys](#)
- [Running a Recorded Macro](#)
- [Saving and Editing Recorded Macros](#)
- [Deleting Recorded Macros](#)
- [Using Macros to Discover and Control Options](#)

Common 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 Macros](#) 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. When you have finished recording the macro, end recording by clicking **Macro** → **Stop Recording Macro** (or the same toggle you used in Step 1). 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.
- If you're recording the macro to discover Slick-C® code (see [Using Macros to Discover and Control Options](#)), click **Edit** (or press **Alt+E**) at this time to view the source code. If you choose to view/edit the source code now, you will need to save it by using **Macro** → **Save last-macro** prior to recording a new macro or exiting the editor. See [Saving and Editing Recorded Macros](#) for more details.

4. Specify the name for the macro in the **Macro Name** text box.
5. Select any desired options. Leave the default settings if you aren't sure what to select. See [Save Macro Dialog](#) for more information on these advanced options.
6. If you plan to use the macro frequently, we recommend creating a key binding for it. To create the key binding, click **Save and Bind to Key**, then see [Binding Recorded Macros to Keys](#) for more information.

Alternately, click **Save**. The [List Macros Dialog](#) is displayed, from which you can run the macro, edit the source, delete it, or bind it to a key sequence.

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 option screen (see [Binding Macros Using the Key Bindings Option Screen](#)), 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 Option Screen

To create a key binding for a recorded macro, you can either click the **Save and Bind to Key** button on the Save Macro dialog that appears automatically after you end recording, or at any time you can use the **Bind to Key** button on the [List Macros Dialog](#) (**Macro** → **List Macros** or **list_macros** command). Clicking either button displays the Key Bindings option screen with the macro selected, so you can add a key binding.

Note

You can also display the Key Bindings option screen by clicking **Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**, or by using the `gui_keybindings` command. However, if you display the screen 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 on the Key Bindings option screen.

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 that 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** through **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 dialog or Key Bindings option screen).

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 `vusrmacs.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 `vusrmacs.e` file opens in the editor. Save the file when you're done making edits.

If you are using recorded macros to discover Slick-C® code (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 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 clicking **Macros** → **Save last-macro**. The Save Macro dialog is displayed again so you can name the macro and then click **Save**, which appends the new code to the user macros file (`vusrmacs.e`). Or click **Save and Bind to Key** to save and create a key binding for the macro (see [Binding Macros Using the Key Bindings Option Screen](#)).

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/clear 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. The **Line insert style** option is located on the **General Editing** options page. Instead of every time clicking **Tools → Options → Editing → General**, 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. See [Saving and Editing Recorded Macros](#) for more information.

Programmable Macros

Note

Many of the Slick-C macro programming features described here are only available in the Pro edition and not the Standard or Community editions. The Standard and Community editions have very limited macro programming capabilities. These editions have Limited macro recording and some configuration batch macros (like vusrdefs.e (UNIX: vunxdefs.e)) but not much else. If you want to do significant macro programming such as writing dialog boxes or sophisticated macros you can share with others, you need the Pro edition. Dialog boxes written in Slick-C will run on all platforms SlickEdit supports.

The Slick-C® macro programming language is behind most of the actions performed in SlickEdit®. Slick-C functions are mapped to menus, buttons, and keys, and perform the action behind an event. You can use Slick-C to extend the editor's functionality. With Slick-C, you can manipulate buffers; parse strings; navigate buffers and source code; and create and modify menus, dialogs, toolbars, and tool windows.

For documentation purposes, "programmable macros" is a term used to encompass Slick-C modules, variables, and batch macros. These items are described in detail below.

This section contains the following topics:

- [Slick-C® Modules](#)
- [Slick-C® Variables \(Config Variables\)](#)
- [Slick-C® Batch Macros](#)
- [State File](#)
- [Slick-C® Header Files and More Resources](#)

Slick-C® Modules

A Slick-C module is a file with the extension .e that contains Slick-C code. Slick-C modules are the most typical use of Slick-C, used to define functionality that you want to keep loaded, such as user-defined commands. You must compile and load a Slick-C module into the [State File](#) before it can be utilized. When a module is compiled, the Slick-C translator converts the file into byte code, which is saved in a corresponding file with the extension .ex. To compile and load the module, use the menu item **Macro → Load Module**. See [Loading and Unloading Slick-C Modules](#) for more information.

Tip

Slick-C modules that are included with SlickEdit are located in the [SlickEditInstallDir]/macros subdirectory. You can store the macros you write in any

directory you like. It is best not to store your macros in the `macros` subdirectory, however.

Loading and Unloading Slick-C® Modules

To compile and load a Slick-C module into the [State File](#), from the main menu, click **Macro → Load Module**. You can also press **F12** or use the **gui_load** command on the SlickEdit® command line. The Open dialog is displayed, prompting you for the file to load.

Note

The dialog used to open a file depends on whether you are using Smart Open. With Smart Open, you use the Open tool window to browse files and open them. With that off, a standard file browser dialog is displayed. For more information see [Opening Files](#).

To unload a Slick-C module from the [State File](#), from the main menu, click **Macro → Unload Module**, or use the **gui_unload** command.

Caution

- Use caution when unloading modules that are shipped with SlickEdit. Unloading a base module could cause the editor to behave unpredictably.
- Base modules are identified by file name. If you load a module with a name that matches a base SlickEdit module, it will replace the module, which (like unloading a base module), could potentially cause problems.

Slick-C® Variables (Config Variables)

Slick-C variables are global variables that are persistently stored in the [State File](#). Because these typically contain user configuration settings, Slick-C variables are also called "configuration variables". Config variables start with the prefix **def_**. See [Configuration Variables](#) for more information.

Slick-C® Batch Macros

A Slick-C batch macro is a `.e` file that contains a **defmain()** function. This file cannot be loaded - you must compile and run it from the SlickEdit® command line. Slick-C batch macros are useful for infrequent tasks that do not involve a persistent state. They are different from recorded macros in that they usually perform specific tasks and cannot be bound to a key. An example of a Slick-C batch macro is a file called `autotag.e`. This batch macro launches the Create Tag Files for Run-Time Libraries dialog that appears when you run SlickEdit for the first time. See [Creating Tag Files for Compiler-Specific Libraries](#) for more information.

State File

SlickEdit® ships with a system state file that contains default settings. The system state file is only changed or updated when you upgrade. As you make changes to the configuration of the editor or apply customizations, the changes are saved to a user state file, which is a copy of the system state file with user customizations.

The user state file (`vslick.sta`) is located in your [User Configuration Directory](#).

The state file is a binary file that stores the following information:

- Loadable [Slick-C® Modules](#).
- Slick-C settings such as global options, language-specific options, etc.
- Slick-C resources such as event tables, dialogs, toolbars, tool windows, menus, bitmaps, and icons.
- DLL-exported function linkage.

The state file does not include DLLs themselves or [Slick-C® Batch Macros](#).

Slick-C® Header Files and More Resources

Slick-C header files use the `.sh` extension. All Slick-C source files `#include slick.sh`.

To learn more about Slick-C functions, from the main menu, click **Help → Macro Functions by Category**. This will display the Help dialog, with a list of all macro functions organized into categories.

For information about writing Slick-C, see the [Slick-C Macro Programming Guide](#), which is included in the SlickEdit® Help system and also available as a stand-alone PDF in the `docs` installation subdirectory.

Menus, Dialogs, and Tool Windows

This chapter contains a comprehensive description of the menus and dialogs available in SlickEdit. For a general overview, see [User Interface](#).

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 table below describes items on the **File** menu.

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 File Dialogs and Tool Windows .	<code>new</code>
New Item from Template	Displays the Add New Item dialog, which allows you to create a new file from a template. See Code Templates .	<code>add_item</code>
Open	Displays the Open dialog, which allows you to open a file for editing. See Standard Open Dialog .	<code>gui_open</code>
Open URL	Displays the Open URL dialog, allowing you to open an HTTP file. See Open URL Dialog .	<code>open_url</code>
Close	Closes the current file.	<code>quit</code>
Close All	Closes all files.	<code>close_all</code>
Save	Saves the current file.	<code>save</code>
Save As	Displays the Save As dialog, which allows you to save the current file under a different name. See Save As Dialog .	<code>gui_save_as</code>
Save Copy As	Displays the Save Copy As dialog, which allows you to save a	<code>gui_save_copy</code>

File Menu

File Menu Item	Description	Command
	copy the current file under a different name without changing the name of the current file in the editor. See Save Copy As Dialog .	
Save All	Saves all modified files.	<code>save_all</code>
Refresh/Revert	Refreshes or reverts the current file based on its modify status. If the buffer has been modified, then it is reverted. If the file has not been modified, but has an earlier time stamp than the file on disk, then the file is refreshed.	<code>revert_or_refresh</code>
Reload with Encoding...	Displays a 'Reload with..' dialog, allowing you to reload the current document with a different encoding. The user-selected encoding will be remembered the next time the file is opened. See Reload With Encoding Dialog .	<code>reload-with-encoding</code>
Change Directory	Displays the Change Directory dialog, which lets you change the current working directory. See Change Directory Dialog .	<code>gui_cd</code>
Backup History for	Displays the Backup History dialog, which allows you to quickly diff against previous versions. See Backup History .	<code>activate_deltasave</code>
Backup History Browser...	Displays the Backup History Browser dialog, which displays your save history. Deleted files can be restored here. See Backup History .	<code>backup_history_browser</code>
FTP	Displays menu of FTP commands. See File Menu .	N/A
Print	Displays the Print dialog which contains options to print the current file or selection and	<code>gui_print</code>

File Menu

File Menu Item	Description	Command
	provides setup options. See Print Dialog .	
Insert a File	Displays the Insert File dialog, which lets you insert a selected file at the cursor location. See Inserting Files .	<code>gui_insert_file</code>
Write Selection	Displays the Write Selection dialog, which lets you write or append selected text to a file you choose. See Using Write Selection .	<code>gui_write_selection</code>
Template Manager	Create, edit, and delete your templates. See Code Templates .	<code>template_manager</code>
Export to HTML	Write file to HTML format.	<code>export_html</code>
File Manager	Displays menu of file manager commands. See File Manager Menu .	N/A
More Files	Displays menu of files that were recently opened, saved, or closed. The number of files displayed is configurable. See History Options .	N/A
Exit	Prompts you to save files if necessary and exits the editor. See Exiting the Program .	<code>safe_exit</code>

File FTP Menu

The **File → FTP** menu is available for performing FTP operations and changing FTP options. See [FTP](#) for more information about working with these features.

FTP Menu Item	Description	Command
Start New Connection	Activates FTP tool window and starts a new connection.	<code>ftpOpen 1</code>

File Menu

FTP Menu Item	Description	Command
Activate FTP	Activates FTP tool window.	<code>activate_ftp</code>
Upload	Uploads the current FTP file.	<code>ftpUpload</code>
Client	Activates FTP Client toolbar.	<code>ftpClient</code>
Profile Manager	Display FTP Profile Manager dialog box. See Add/Edit FTP Profile Dialog .	<code>ftpProfileManager</code>
Default Options	Displays the default FTP options. See FTP Default Options .	<code>show_ftpOptions_form</code>

File Manager Menu

The **File → File Manager** menu contains options for working with the SlickEdit® File Manager, which offers a rich set of file listing, selecting, and operating capabilities. See [The SlickEdit® File Manager](#) for detailed information about this feature.

File Manager Menu Item	Description	Command
New File List	Displays a directory of files you choose.	<code>fileman</code>
Append File List	Appends files to current list.	<code>fileman append</code>
Sort	Sorts file list.	<code>fsort</code>
Backup	Copies selected files and preserves directory structure.	<code>fileman_backup</code>
Copy	Copies selected files to a directory you choose.	<code>fileman_copy</code>
Move	Moves selected files to a directory you choose.	<code>fileman_move</code>
Delete	Delete selected files.	<code>fileman_delete</code>
Edit	Edits selected files.	<code>fileman_edit</code>
Select	Displays menu of file manager select commands. See File	N/A

File Menu

File Manager Menu Item	Description	Command
	Manager Select Menu.	
Files	Displays menu of file manager listing commands. See File Manager Files Menu .	N/A
Attribute	Sets the Read Only, Hidden, System, and Archive attributes of the selected files.	<code>fileman_attr</code>
Repeat Command	Runs internal or external command on selected files.	<code>for_select</code>
Global Replace	Performs search and replace on selected files.	<code>fileman_replace</code>
Global Find	Performs search on selected files.	<code>fileman_find</code>

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.	<code>fileman_select_all</code>
Deselect All	Deselects all files.	<code>deselect_all</code>
Invert Select	Selects files which are not selected and deselects files which are selected.	<code>select_reverse</code>
Attribute	Selects files based on file attribute.	<code>select_attr</code>
Extension	Selects files based on file extension.	<code>gui_select_ext</code>
Highlight	Selects files which are highlighted.	<code>select_mark</code>
Deselect Highlight	Deselects files which are highlighted.	<code>deselect_mark</code>

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.	unlist_all
Unlist Select	Removes selected files from the list.	unlist_select
Unlist Extension	Removes files with a specific extension from the list.	gui_unlist_ext
Unlist Attribute	Removes files with a specific attribute from the list.	unlist_attr
Unlist Search	Removes lines which contain a particular search string.	unlist_search
Read List	Appends a list of files contained in a file.	read_list
Write List	Writes a file containing the currently selected files.	write_list

File Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with **File** menu items.

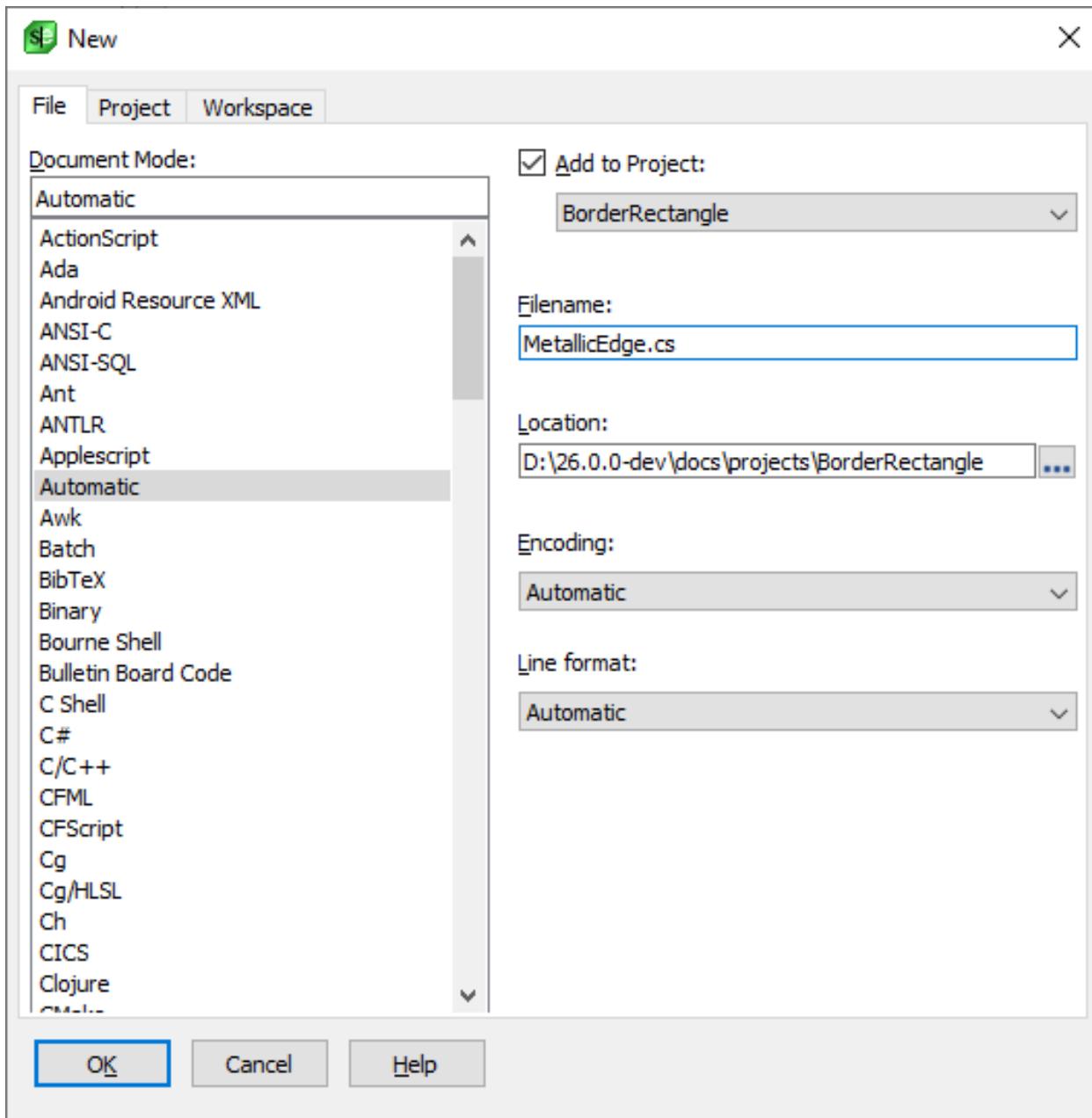
New Dialog

The New dialog is used to create new files, projects, and workspaces. The dialog consists of three tabs:

- [New Dialog](#)
- [Project Tab](#)
- [Workspace Tab](#)

File Tab

This tab on the [File Dialogs and Tool Windows](#) is used to create new files. It is displayed when you click **File → New** or use the **new** command. See [Creating Files](#) for more information.



The fields and options on the File tab are described as follows:

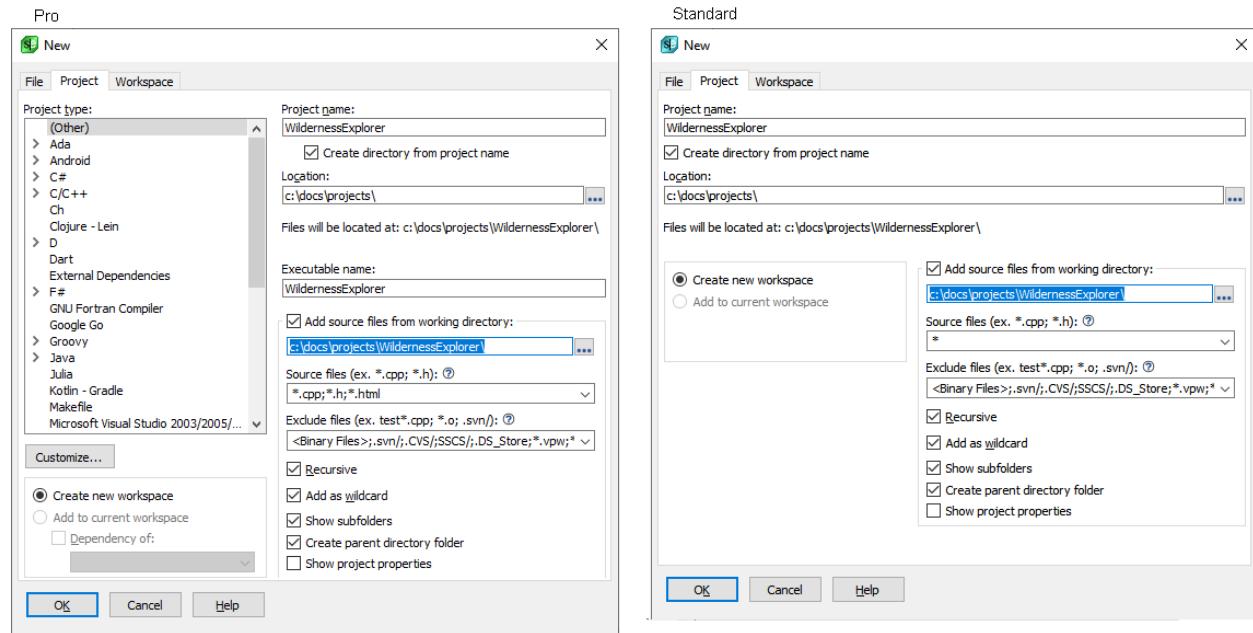
- **Document Mode** - Specifies the language editing mode for the new file. The language editing mode determines the options and features that will be available for this file (see [Language Editing Mode](#)). Double-click on a mode to instantly create an empty, untitled buffer set to that mode. Recently used language modes appear at the top of the list. You can control the number of recently used items that appear with the option **Number of recent modes to store** on **Tools → Options → File Options → History**.

When you select the **Automatic** mode, the language mode is picked based on the file extension of the file you name. If your extension doesn't match a mode, you'll be asked if you want to create it as plain text. Double-clicking **Automatic** creates an untitled plain text file.

- **Add to Project** - When selected, the new file will be added to the selected project in the current workspace. Use the drop-down list to select the project. This option is set automatically if a project is open. This option is disabled if you have not typed in a file name yet.
- **Filename** - Specifies the name (without path) of the file to create. If you type a path in the **Filename** field, the **Location** field will be filled in automatically. Leave this field blank and hit **OK** to create an new, untitled buffer.
- **Location** - Specifies the directory in which the file should be created. By default, this is set to the current working directory (see [The Working Directory](#)). Click the **Browse** button to the right of this field to browse for a location. You can also type a path and if the directory does not exist, you will be prompted to create it after you click **OK**.
- **Encoding** - Specifies the encoding you want when the file is saved. See [Encoding](#) for more information.
- **Line format** - Specifies the line endings for the file. **Automatic** specifies that the line ending be determined by options specified elsewhere.

Project Tab

The Project tab on the **New** dialog is used to create new projects. It is displayed when you click **Project** → **New** or use the **workspace_new** command. The fields and options on the Project tab are described below. See [Creating Projects](#) for step-by-step instructions and [Managing Projects](#) more information about choosing the correct project type.



The fields and options on the Project tab are described as follows:

- **Project type** - (Pro only) Specifies the type of project that you want to create (see [Managing Projects](#) for more information). Expand your language in the tree to see all types for that language. It is very important to select the correct type of project at this stage, because it is not possible to change it after

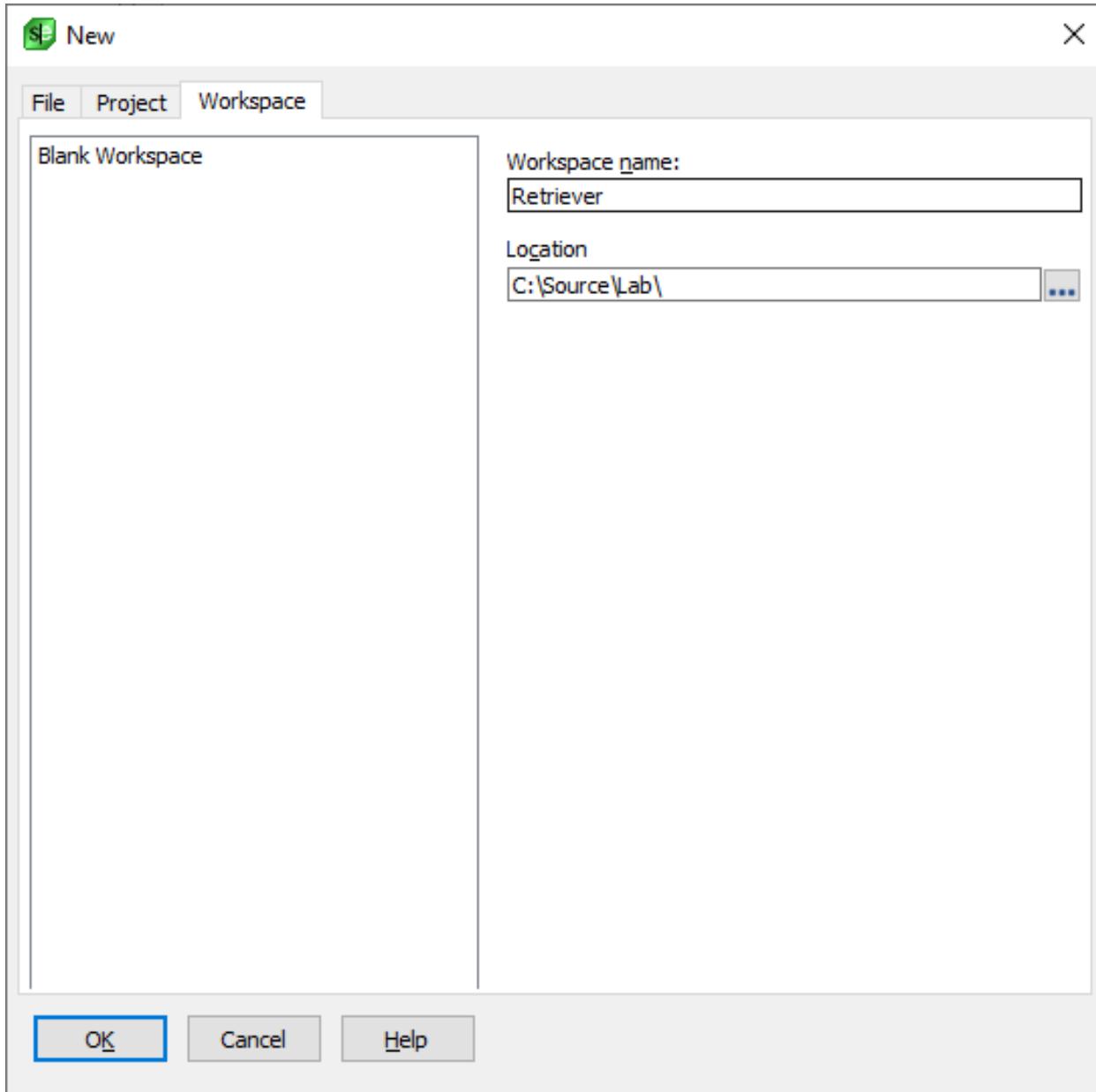
the project has been created. Recently used project types appear at the top of the list. You can control the number of recently used items that appear with the option **Number of recent types to store** found on **Tools** → **Options** → **File Options** → **History**

- **Customize** - (Pro only) Displays the Customize Project Types dialog, which allows you to create, edit, and delete project packages from the **Project type** list. System-defined project types are stored in `prjtemplates.vpt`. User-modified or additional project types are stored in `usrprjtemplates.vpt`. See [Creating Custom Project Types](#) for more information.
- **Project name** - Specifies the name of the new project.
- **Create project directory from project name** - When selected, the name you specify for the project is used as the name for the new directory in which the project will be created. The new directory will be created as a subdirectory under the specified **Location**. When this option is not selected (cleared), the project will be created inside the **Location** directory. The area below the **Location** field shows where the project files will reside based on this option and other Project tab settings. For example, if the project is named "TestProject", this option is selected, and the location is set to `C:\dev\`, files for this project will reside at `C:\dev\TestProject`. If the option is not selected, files for the project will reside at `C:\dev`.
- **Location** - Specifies the directory in which the project should be created. By default, this is set to the current working directory (see [The Working Directory](#)). Click the **Browse** button to the right of this field to select a different location. You can also type a path and if the directory does not exist, you will be prompted to create it after you click **OK**. The area below this field shows where the project files will reside based on this field and other Project tab settings.
- **Files will be located at** - Shows the path where the project files will reside, based on the information you have specified on the tab.
- **Executable name** - Specifies the name of the executable or output file.
- **Create new workspace** - Specifies that the project should be created in a new workspace. This option and **Add to current workspace** are mutually exclusive.
- **Add to current workspace** - Specifies that the project should be created inside the current workspace. For example, if the current workspace is named "OrderChart" located in `C:\dev\OrderChart`, the name of the new project is "TestProject", and this option is selected, files for the new project will reside at `C:\dev\OrderChart\TestProject`. This option and **Create new workspace** are mutually exclusive.
 - **Dependency of** - (Pro only) When **Add to current workspace** is selected, this option specifies that the new project should be a dependency of an existing project in the current workspace. Use the drop-down to select the existing project on which you want the new project to depend. See [Defining Project Dependencies](#) for more information.
- **Add source files from working directory** - When selected, immediately add the source files under the source tree to the project. When not selected, the **Files** tab of the **Project** → **Project Properties** dialog will normally be brought up after the project is created so you can add source files.
- **Working directory:** - Specifies both the directory containing the source files to be added to the project, as well as the directory to be used as the working directory for the project.

- **Working directory:** - Specifies both the directory containing the source files to be added to the project, as well as the directory to be used as the working directory for the project.
- **Source files** - Sets the default Add Tree **Include filespecs** (see [Add Tree](#)).
- **Exclude files** - Sets the default Add Tree **Exclude filespecs** (see [Add Tree](#)).
- **Recursive** - Sets the default Add Tree option (see [Add Tree](#)) to **Recursive**.
- **Add as wildcard** - Sets the default Add Tree option (see [Add Tree](#)) to **Add as wildcard**.
- **Show subfolders** - Sets the default Add Tree option (see [Add Tree](#)) to **Create subfolders**. When this option is used, filter folders (i.e. Source Files, Headers Files) in the project are removed.
- **Create parent directory folder** - Sets the default Add Tree (see [Add Tree](#)) option to **Create parent directory folder**. When this option is used, filter folders (i.e. Source Files, Headers Files) in the project are removed.
- **Show project properties** - When adding source files immediately, it is not always necessary to show the project properties dialog after creating a new project. When selected, the **Files** tab of the **Project → Project Properties** dialog will be brought up after the project is created. Note that the project properties dialog is always brought up after creating a new project if not adding source files immediately.

Workspace Tab

This tab on the [File Dialogs and Tool Windows](#) is used to create a new workspace. To access it, click **Project → New** or use the **workspace_new** command. See [Creating Workspaces](#) for more information.



- **Blank Workspace** - Select this only if you want to create an empty workspace to which you can add projects.
- **Workspace name** - Specify the name of the new workspace.
- **Location** - Specify the directory location of the new workspace. If the directory does not exist, you will be prompted to create it when you click **OK**.

The data for each workspace, solution, or project is stored in a text file with the .vpw extension.

Open Tool Window

The Open tool window is for browsing and opening files. Like other tool windows, it can be docked in

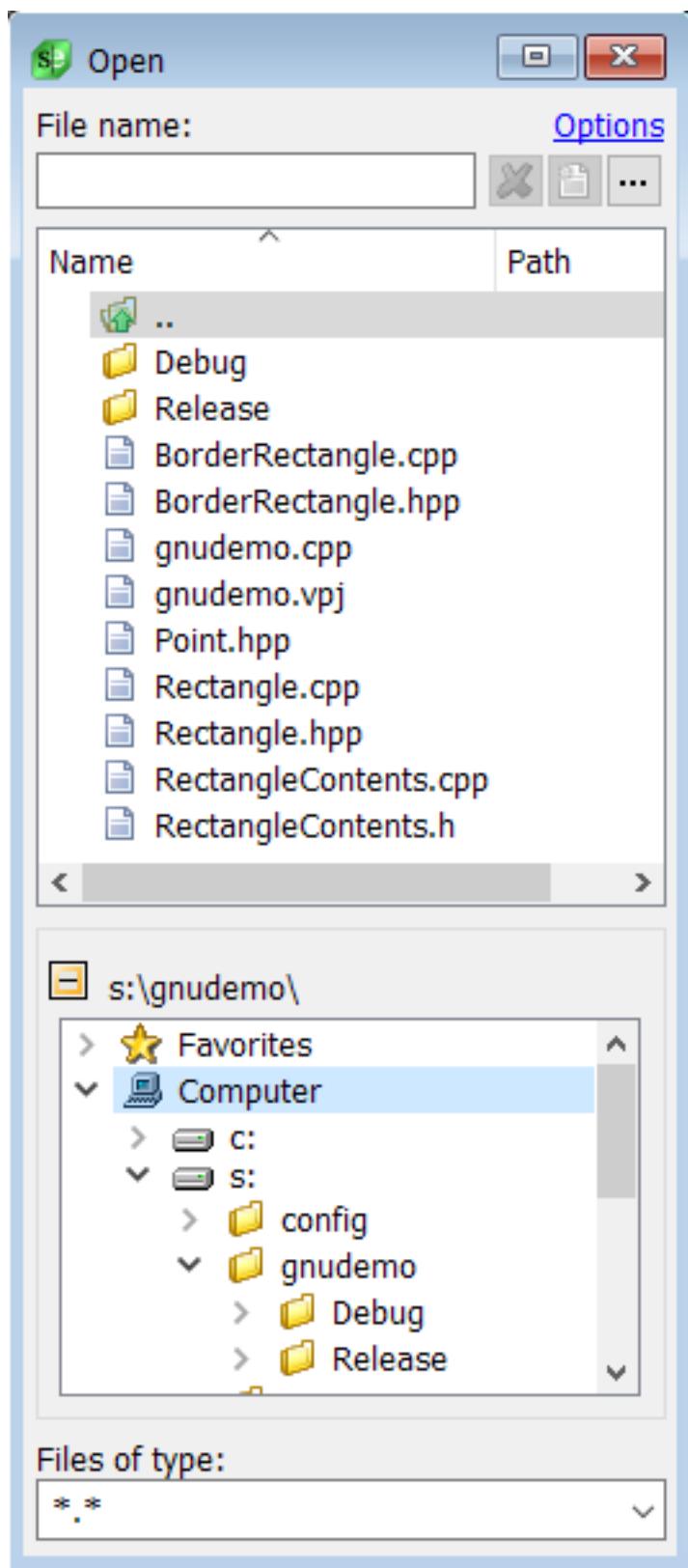
any location within SlickEdit or left floating. You can type part of a file name to filter the list. Once a pattern is typed, this window also will display files from the current workspace, list of open files, and file history, depending on your settings.

Note

The Community edition does not support Smart Open or features which search for filename patterns in the workspace or project.

By default, patterns typed into the **File name** field will be used to find matching files in the current workspace, the list of open files, and the file history. This makes it faster to locate files by name with less concern about where they come from. See [Open Tool Window Options \[1060\]](#) for information on how to change this behavior.

The **File name** field box supports ant-like wildcards (wildcards in path parts and use of **). Depending on what you've typed and the options you've specified (click **Options** link), an ant-like wildcard specification is generated and used to match files. For example, if you specify **path-part/name-part**, the generated wildcard is ****/path-part**/*name-part***. There are **Options** for how to handle certain **name-part** specifications. For example, if **name-part** is ***.ext**, no leading ***** is prepended to **name-part**, and no trailing ***** is appended to **name-part..**. There are options for this and other **name-part** specifications.



At the top of window, the **File name** field is used to filter the list of files and directories displayed in the pane, below it. By default, the string is matched anywhere in the file name. If you prefer, you can set your

options to do a prefix match. See [Open Tool Window Options \[106\]](#) for more information.

You can also filter the list to show only files of certain types, using the **Files of type** field at the bottom of the tool window.

Next to the **File name** field are three icons. Use the first button to quickly clear the **File name** filter field. The next button is for creating a new file using the name in the **File name** field. Newly created files are not saved but are opened for editing. The ellipsis icon brings up the standard file browser to open a file.

Below the **File name** field is the list of files and directories. The file list is divided into two columns: **Name** and **Path**. The Path column is useful to distinguish files pulled in outside of the selected directory. This list is not sortable. If you right-click in the file list, a number of operations are available, including Open, Execute, Print, and Refresh. You can also open a file by double-clicking it or by selecting it and pressing **Enter**.

Below the file list is a tree that displays the folder hierarchy for the current directory. It also displays systems drives, a Network entry, and a Favorites node. You can use this to navigate to a different directory. If you don't want this action to change the current directory in SlickEdit, change the options to set **Sync current directory** to **False**.

You can add folders to your list of favorites by selecting one in the file list and picking **Add to Favorites** from the context menu. Remove a favorite by selecting it and pressing the **Del** key. On Windows, the list of Favorites is automatically populated with **Documents** and **Desktop**. On Linux/UNIX we added an entry for **Home**.

Filtering and Matching

The **File Name** field is used both to filter the list of files and directories and to match files and directories from the current workspace, file history, or open files. Patterns can be matched at the beginning of a name or anywhere in the name, depending on your setting for **Prefix match** in the **Open Tool Window** options. You can use a "*" to match 0 or more characters.

You can use a slash to match against the final directory in the path along with a part of the file name. The pattern "foo\bar" matches files with "foo" somewhere in the final directory name and "bar" somewhere in the file name, controlled by your setting for **Prefix match**. This is useful to match files in a particular directory. Patterns without a slash will match against any files and directories displayed in the list below the **File name** field, but they will only be used to match file names in your workspace, open files, or file history.

Here are some sample patterns and what they will match:

Pattern	Matches
foo	Files that begin with or contain "foo".
foo*	This produces the same result as typing "foo". So, you never need to add a star at the end of a pattern.
*foo	This produces the same result as typing "foo" if you

Pattern	Matches
	have Prefix match set to False . If you have Prefix match set to True , then a star at the front will allow you to match items with "foo" anywhere in the name.
foo*bar	Items that contain "foo", followed by zero or more characters, followed by "bar".
docs\	Matches files that have "docs" in the final directory name, but not directory names earlier in the path. This will match "C:\src\docs\foo.txt" and "C:\src\MyDocs\foo.txt" depending on your setting for Prefix match . This will not match "C:\docs\current\foo.txt".
*docs\	This produces the same result as typing "docs\" if you have Prefix match set to False . If you have Prefix match set to True , then a star at the front will allow you to match items with "docs" anywhere in the name.
docs\foo	Matches files with "docs" in the final directory of the path and "foo" somewhere in the file name: "C:\src\docs\food.txt". Will not match "C:\docs\src\food.txt".

Completions

Completions can be used to speed the entry of file names and paths in the **File name** field. After typing a few characters of a name, press the spacebar to fill in the rest of the name. If no matches are available, then a space will be entered in this field.

Hotkeys for Open Tool Window

The following sequences can be used when the cursor is in the **File name** field:

- **Shift +Enter** - creates a new file with the name in the filter.
- **Ctrl +Enter** - if the text in the **File name** field is a filespec (i.e. *.ext), then that filespec is added to the **Files of type** combo box at the bottom of the screen and selects that filter.

The following key sequences are supported when the focus is anywhere in the Open tool window:

- **Alt +E** - changes the focus to the **File name** field.

- **Alt +N** - creates a new file using the name in the **File name** field.
- **Alt +D** - changes the focus to the explorer tree at the bottom of the tool window.
- **Alt +F** - changes the focus to the list of files and directories under the **File name** field.
- **Alt +T** - changes the focus to the **Files of type** combo box at the bottom of the tool window.

Open Tool Window Options

You can set options for the Open tool window in either of two ways:

- Right-click in the file list portion of the tool window and select Options.
- Using the SlickEdit Options dialog, select **Tools** → **Options** → **File Options** → **Open**.

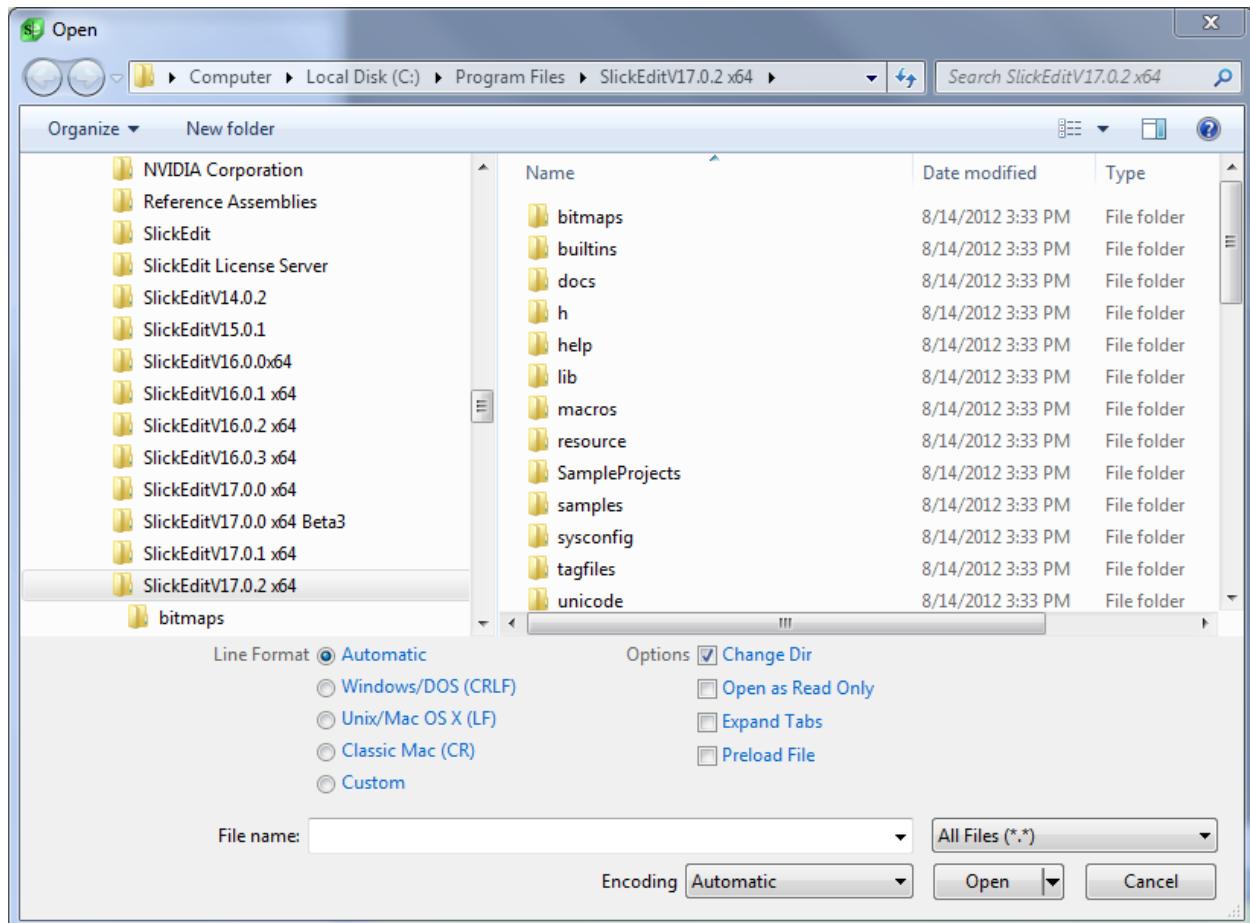
See [Open Tool Window Options \[100\]](#) for information on the Open tool window options.

Standard Open Dialog

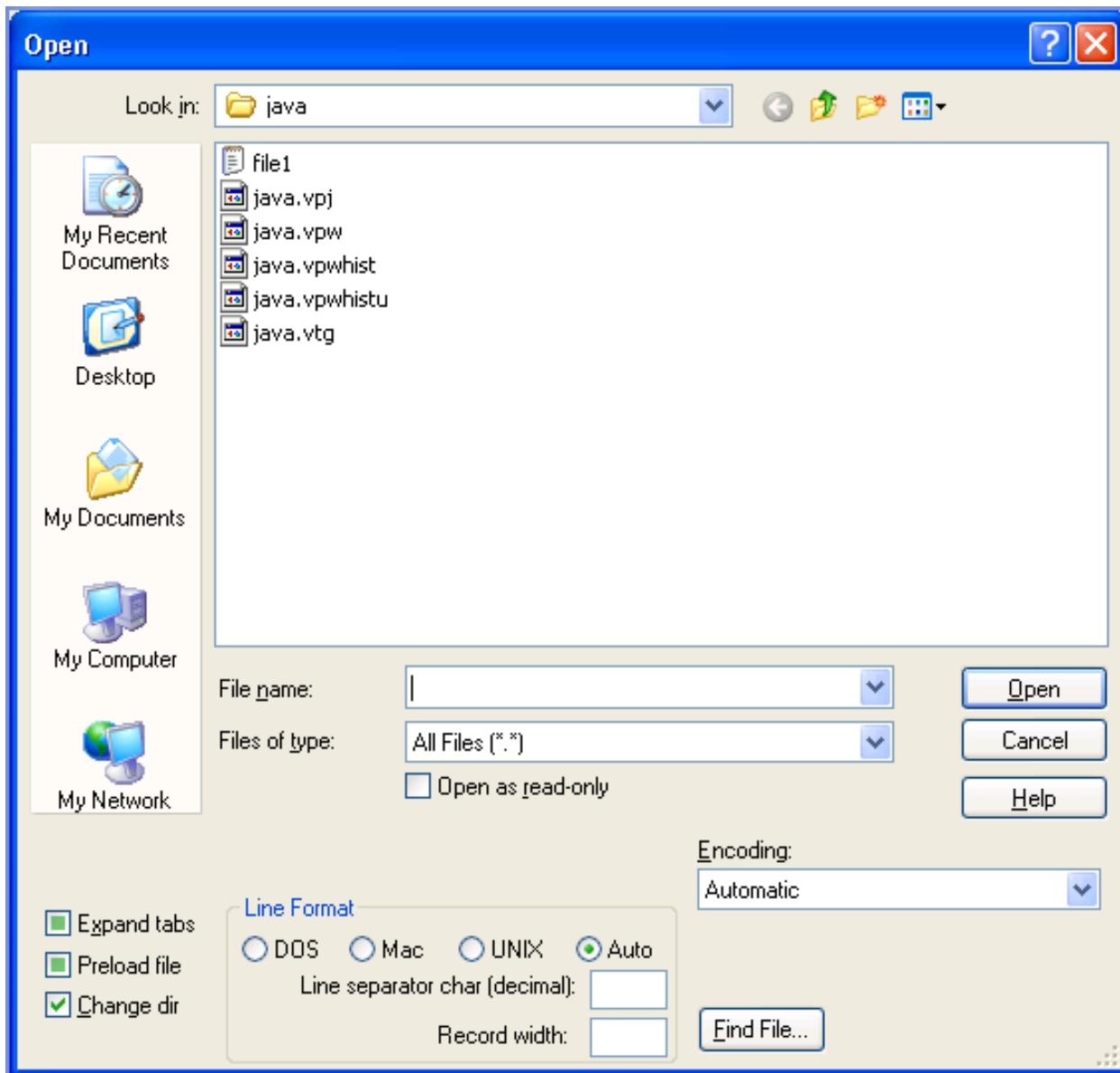
On Windows, this is the standard dialog used to browse and open files. It is displayed when you click **File** → **Open** or use the **gui_open** command if you have turned Smart Open off (see [Opening Files](#)).

Windows 7 and Windows Vista Open Dialog:

File Dialogs and Tool Windows



Windows XP Open Dialog:



- **Files list** - 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.

You can change the file specifications that appear in the **Files of type** combo box at the bottom left of the Open, Save Copy As, and Save As dialog boxes. From the main menu, click **Tools** → **Options** → **File Options**, then select **File Filters**. The first wildcard pattern(s) specified 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. See [Files of Type Filter Options](#) for more information.

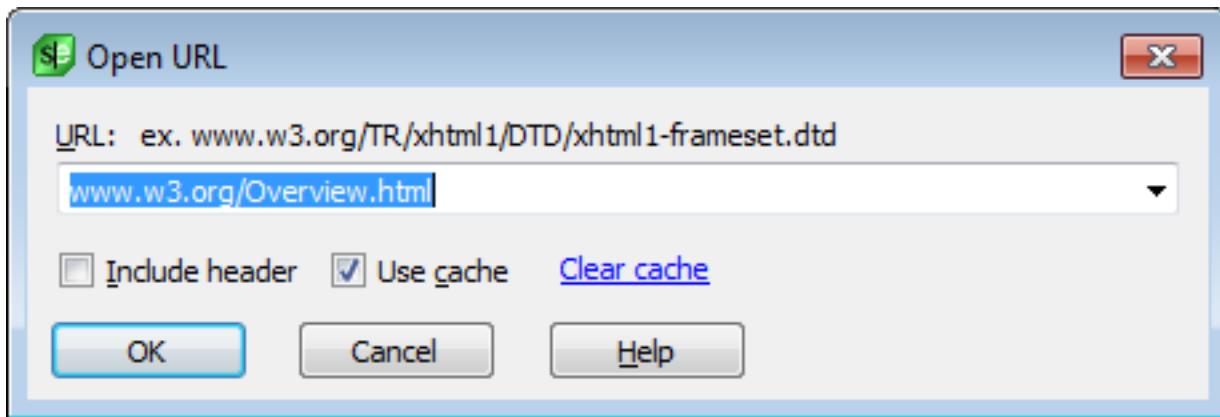
- **Open as read-only** - When opening a file, check this box if you wish to open a file but do not want to accidentally modify the file. Files are automatically opened with the read-only attribute set as read-only, regardless of the setting of this check box.
- **Encoding** - The encoding specifies 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 to always enabled, from the main menu click **Tools** → **Options** → **Appearance** → **General**, and set the **Change directory** option to **True**.
- **Line format** - This option specifies the type of line ending to be used. Select **Windows/DOS(CRLF)**, **Unix/macOS(LF)**, or **Classic Mac(CR)** line endings, or select **Automatic** 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.

Open URL Dialog

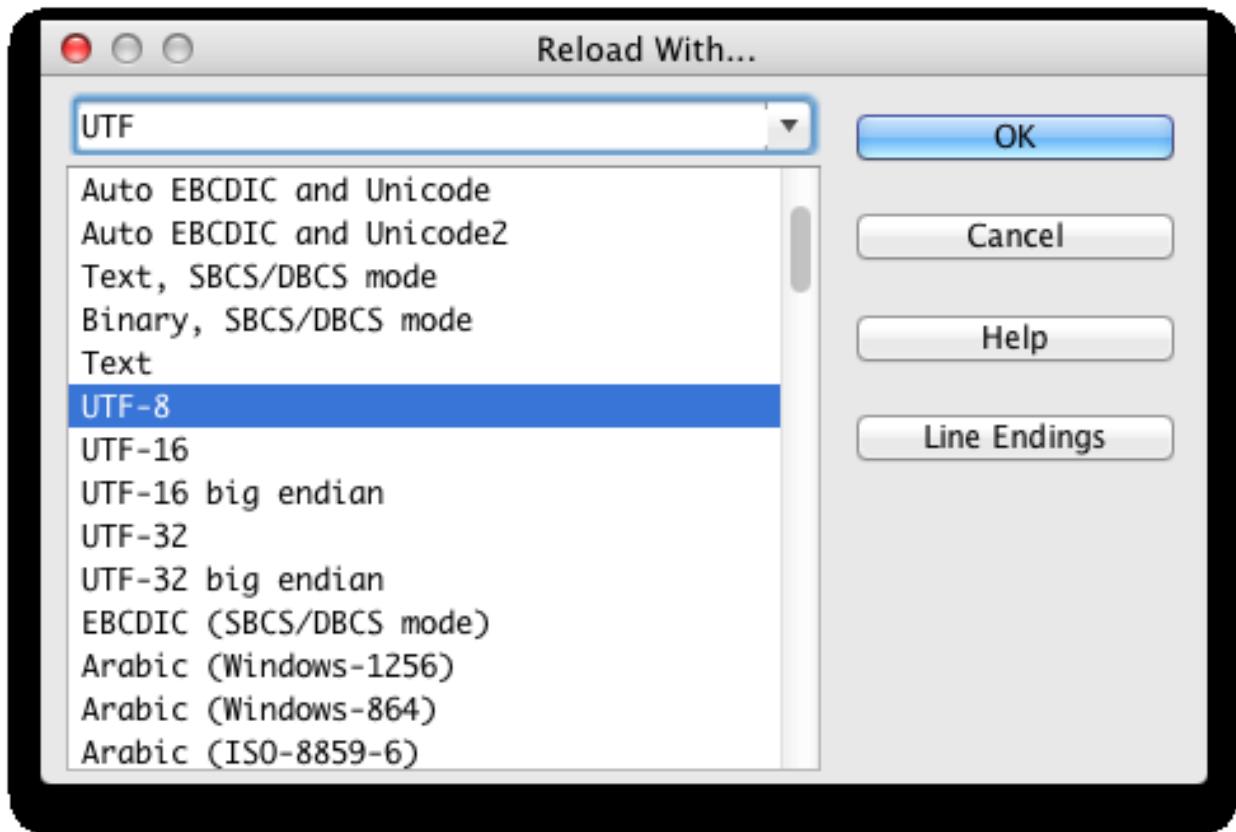
This dialog is used to specify an HTTP file to open. It is displayed when you click **File** > **Open URL** or use the **open_url** command.



- **URL** - Enter a URL to open in this field. You may use forward or backward slashes. The prefix "http://" is not required.
- **Include header** - If this option is selected, all of the URL header data is displayed. If the URL specified is already open, this option has no effect.
- **Use cache** - If this option is selected, 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.
- **Clear cache** - Click on this link to clear some or all of the files from the URL cache.

Reload With Encoding Dialog

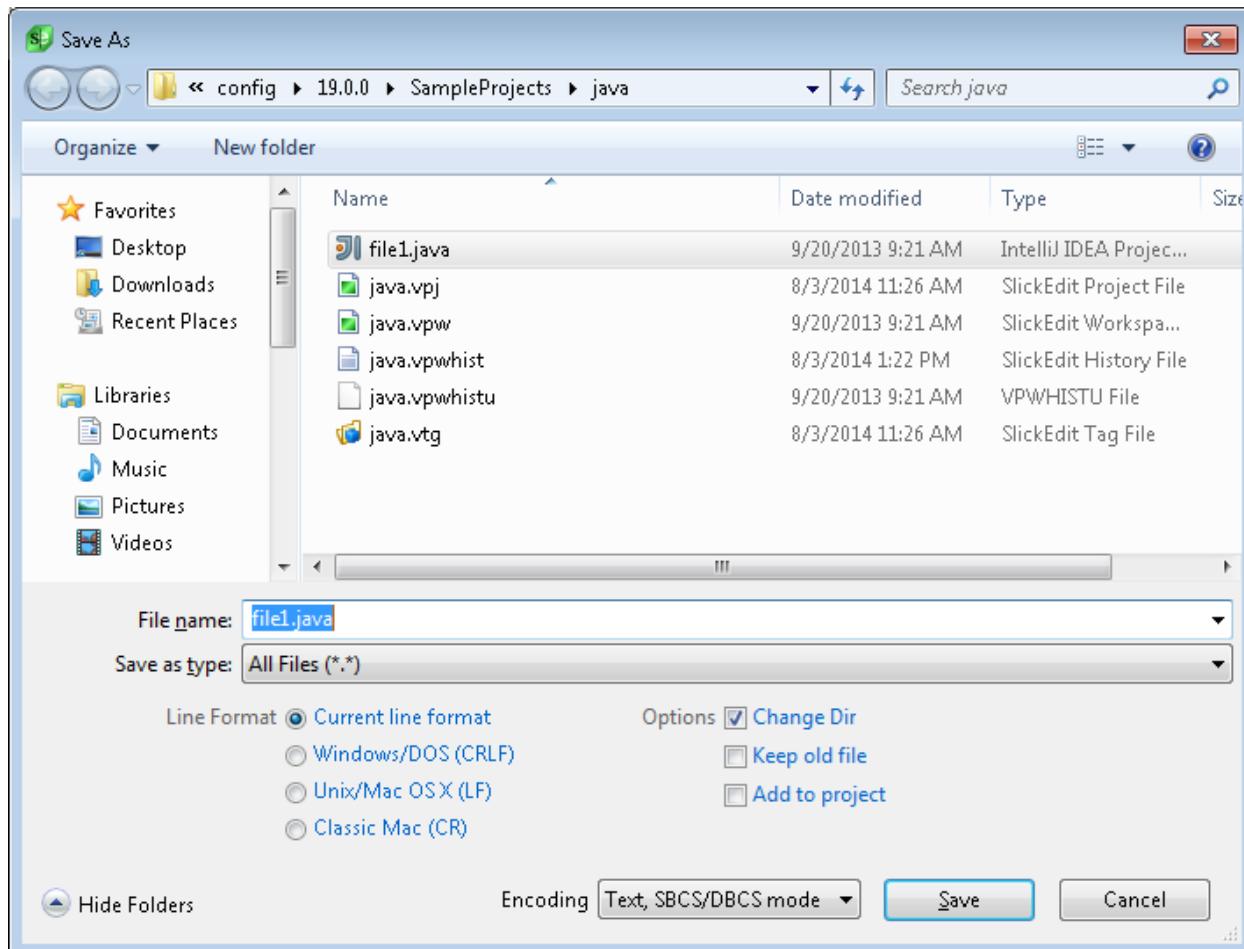
This dialog is displayed when selecting the **File > Reload with Encoding...** menu item or invoking the **reload-with-encoding** or **rwe** commands. The combo box at the top can be used to search within the list of available encodings.



- **Line Endings** - This button will prompt you to specify a line ending format. The default is 'Automatic'.

Save As Dialog

This dialog is used to save the current file under a different name. It is displayed when you click **File** → **Save As** or use the **gui_save_as** command. By default, the standard Save As dialog is displayed. This dialog is described below.



- **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 **Windows/DOS(CRLF)**, **Unix/macOS(LF)**, or **Classic Mac(CR)** 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, click **Tools** → **Options** → **Appearance** → **General**, and set 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** (**Tools** → **Options** → **Appearance** → **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 unavailable. To set the default value of this check box, from the main menu click **Tools** → **Options** → **File Options** → **Save**, and change the option **Add file to project upon Save As**.

Tip

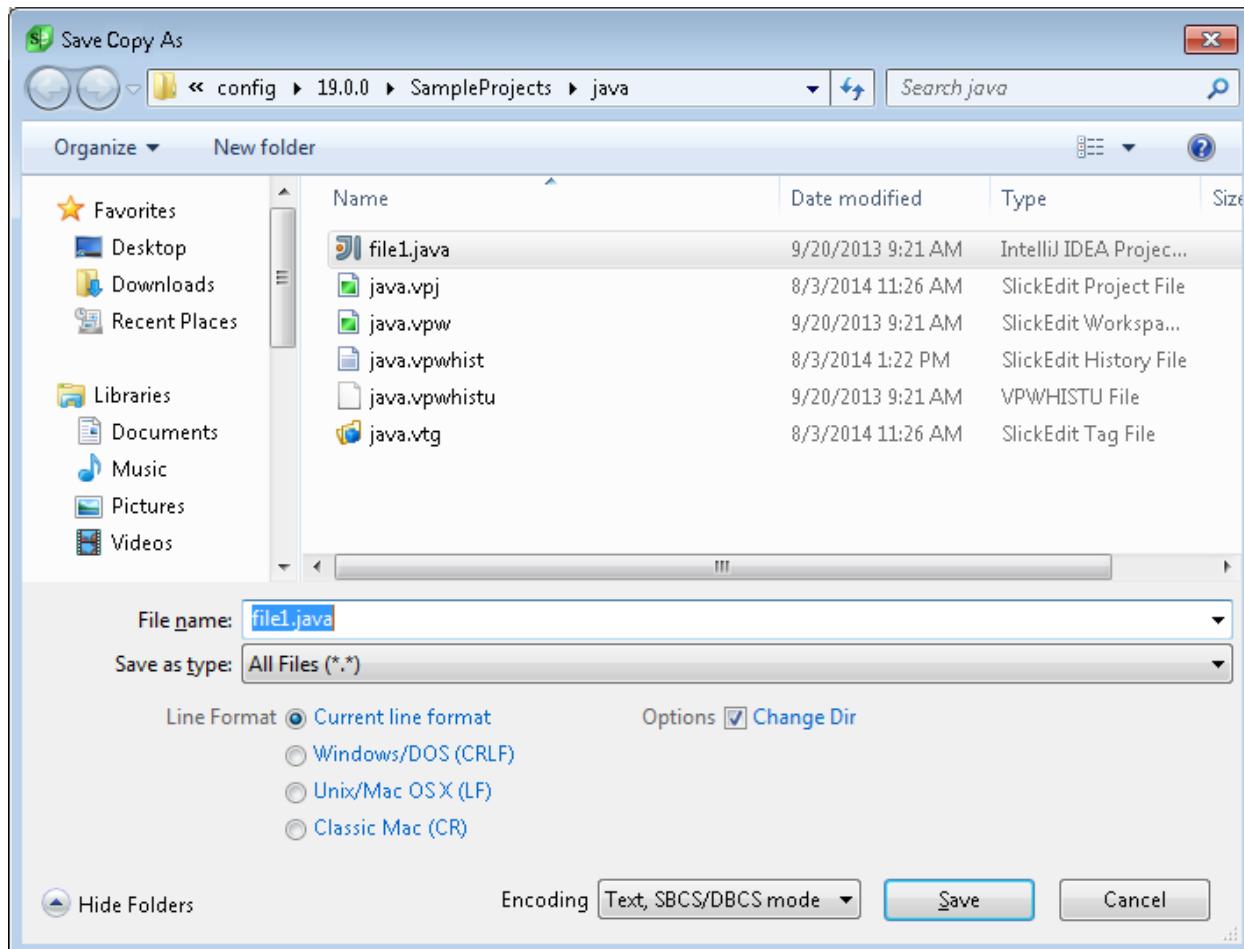
This option is also available from the command line by using the option letter **+P** with the **save_as** command. For example, **save_as +P /path/to/file.cpp** will add **file.cpp** to the current project. Similarly, **save +P** will save the current file and add it to the current project.

Save Copy As Dialog

This dialog is used to save a copy of the current file under a different name without changing the name of the current file in the editor. It is displayed when you click **File** → **Save Copy As** or use the **gui_save_copy** command. By default, the standard Save Copy As dialog is displayed. This dialog is described below.

Note

The typical use-case for Save Copy As is to make a backup copy of a file. Thus, unlike the Save As dialog, the Save Copy As dialog does not have the option to add the saved file to the project.



- **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 **Windows/DOS(CRLF)**, **Unix/macOS(LF)**, or **Classic Mac(CR)** 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, click **Tools** → **Options** → **Appearance** → **General**, and set the **Change directory** option.
- **Show hidden files** - (UNIX only) Dot files (files with names beginning with a dot character) are hidden in the Save Copy 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** (**Tools** → **Options** → **Appearance** → **General**). **Show files beginning with a dot** can be also set using the **def_filelist_show_dotfiles** configuration variable.

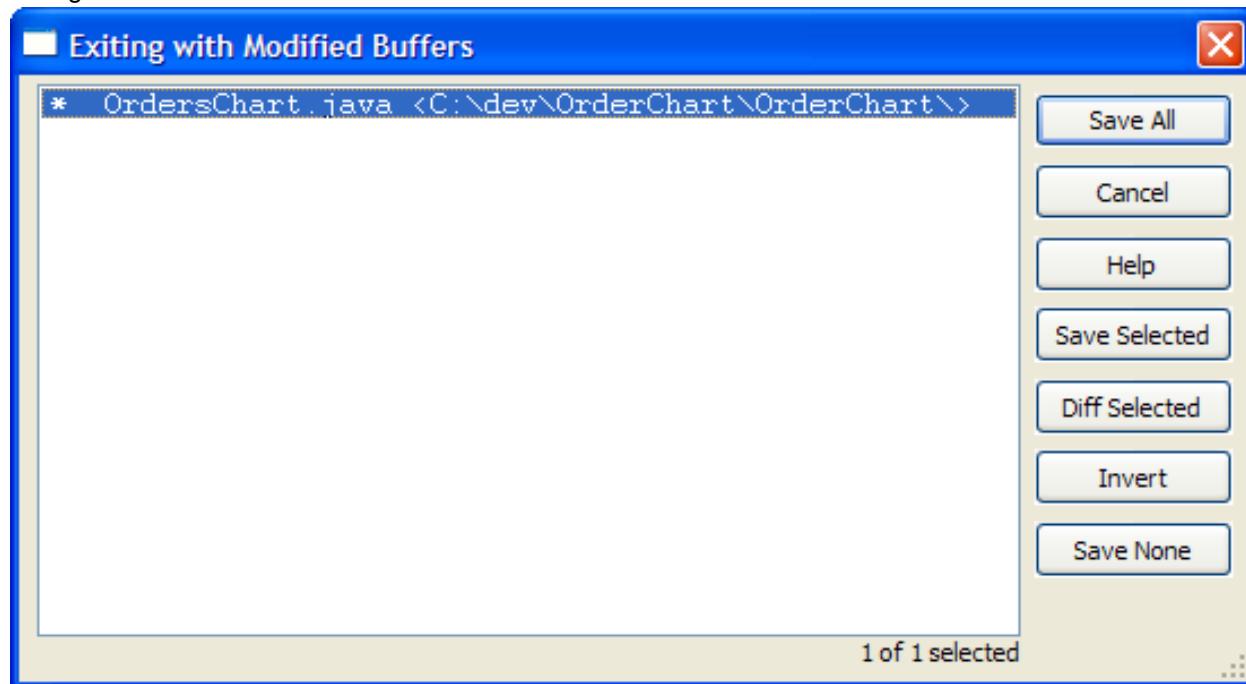
Save Failed Dialog

This dialog appears automatically when a Save operation has failed. The following options are available:

- **Save as read only** - (UNIX only) This check box is automatically selected if your file does not have any write permissions (no "w" letter). Turn this option 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 automatically selected 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, this dialog appears so you can save or discard any changes.



The buffer names in the list box are buffers which have not been saved.

- **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.

File Tabs

If you want to see a document tab per buffer (probably because you are using "Multiple files share window"), you might want to use the **File Tabs** tool window. You can toggle display of this tool window, by selecting **View → Tool Windows → File Tabs** from the main menu, or by using the **toggle_filetabs** command. The maximum number of file tabs that can be displayed is 255. By default, the file tabs are sorted alphabetically. For information on changing the sort order, see [File Tab Sort Order \[755\]](#)

The File Tabs tool window is shown below:



The **File Tabs** tool window displays a single row of file tabs. If there isn't enough space for all of the file tabs to be displayed, a left and right arrow icon is drawn in the tool window, allowing you to scroll the tabs. A down-arrow icon is always visible, allowing you to select a file from a list of the open files. This is a convenient way to select a file when you have a lot of files open and some tabs aren't visible.

Tip

You may also find that using the **Files** tool window provides a convenient way to view a list of open buffers and select one for editing (see [Document Dialogs and Tool Windows](#)).

When a buffer is modified (changed and not yet saved), the text of the file tab turns red by default. You can change the color of modified file tabs with the **Modified file tab(s)** screen element (**Tools → Options → Appearance → Colors**).

Some file tabs, like those for search results buffers, build output, and File Manager operations, display a picture by default. You can further save space in the file tabs area by turning pictures off. To turn pictures on and off for all file tabs, right-click in the File Tabs tool window and select **Show pictures** from the context menu.

When you create a new, unnamed "scratchpad" buffer, it is indicated by the text "Untitled" in the file tab along with a number that indicates the internal ID. You can create a scratchpad buffer by using the menu item **File → New** and not naming the file.

If you prefer to keep your hands on the keyboard for buffer/file navigation, two commands are available: **next_buff_tab** and **prev_buff_tab**. Use these commands to navigate through the file tabs in the order they are displayed. Both circle around to the other end when you are on the last item. These commands are not bound to keys by default. To create key bindings for these commands, see [Creating Bindings](#).

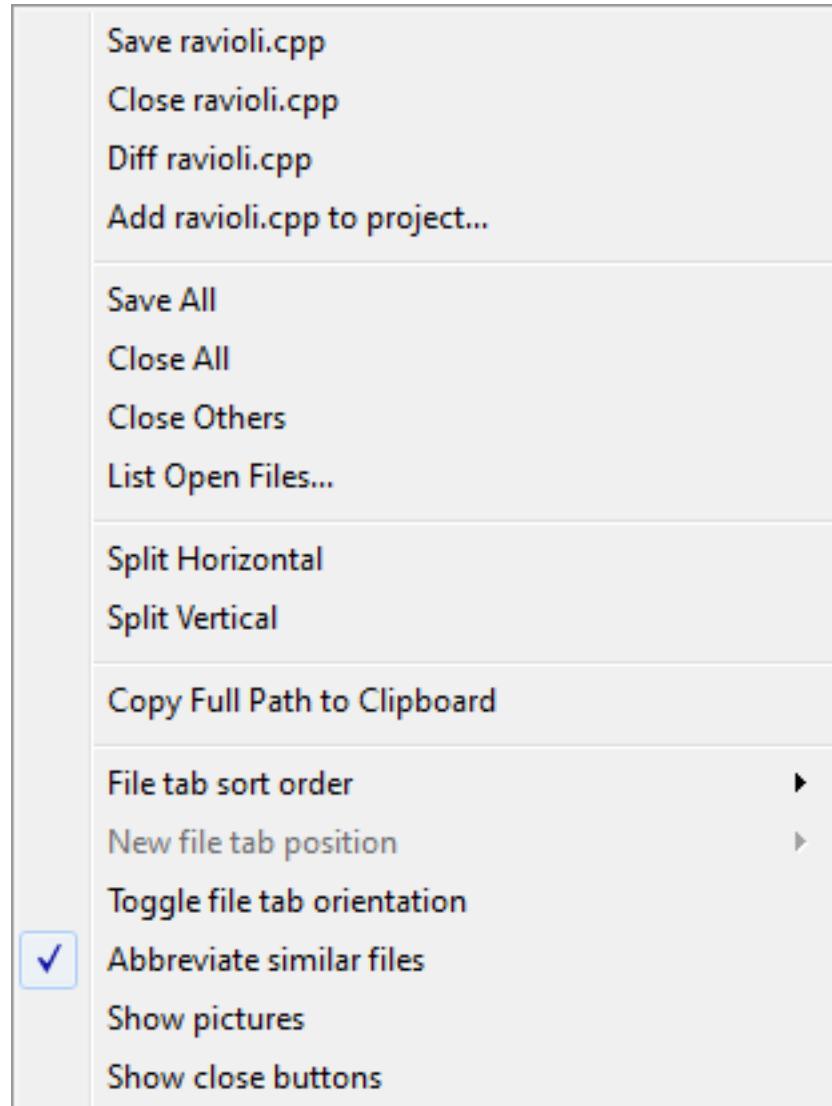
File Tab Context Menu

The right-click context menu in the **File Tabs** tool window provides operations for saving files, closing files, splitting windows, and controlling the appearance of the file tabs.

Note

You get a different menu if you right-click on the background of the **File Tabs** tool window than if you right-click on a file tab. Right-clicking on the background lists the set of available tool windows and toolbars to display. Right-clicking on a file tab lists operations specific to that file or the tabs.

The right-click context menu for the file tabs tool window is shown below.



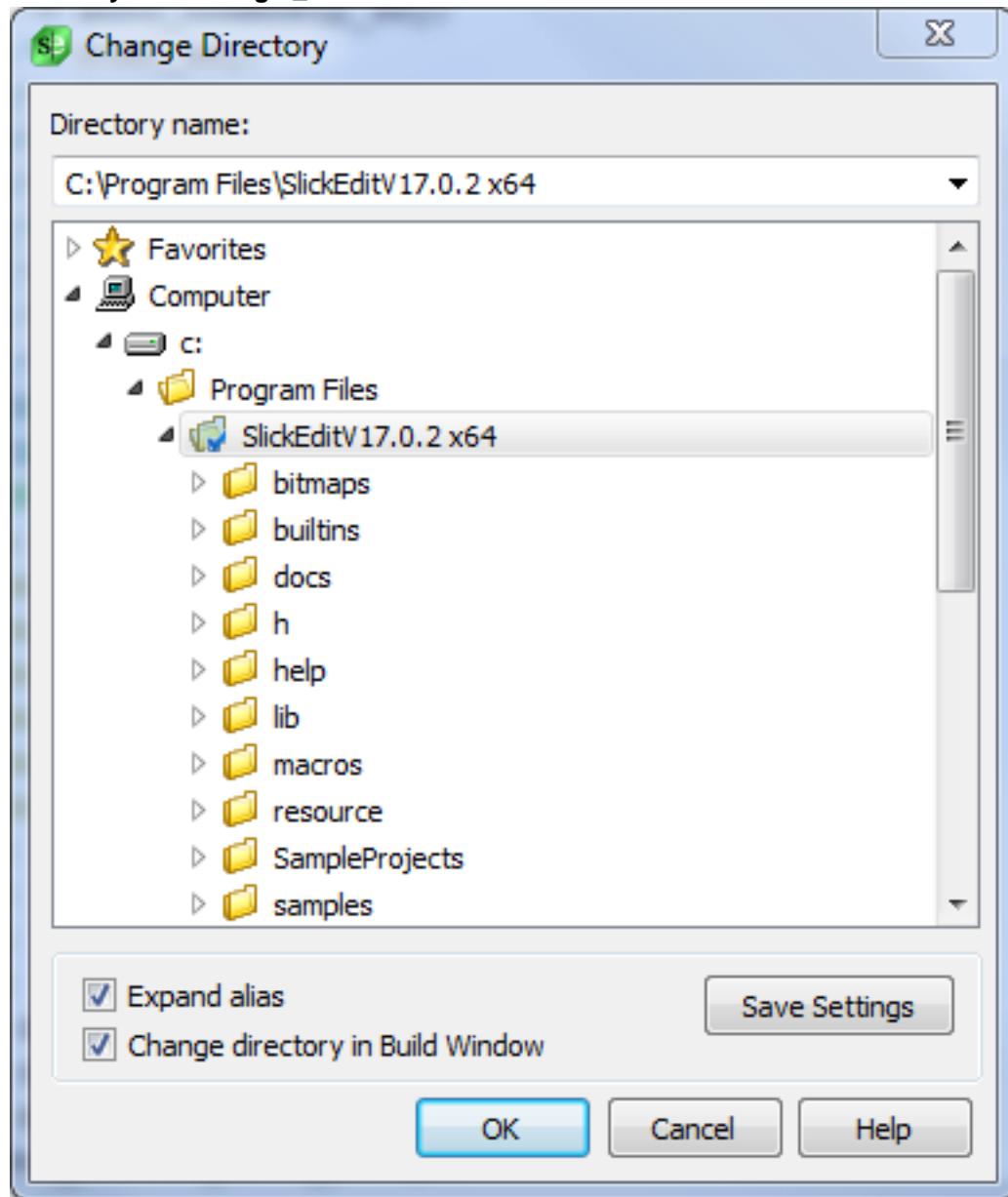
The items available in the right-click context menu are outlined below. Some items apply to the file specified by the file tab underneath the mouse. Others apply to the file tabs tool window as a whole.

- **Save <file>** - Saves the file specified by the file tab under the mouse.
- **Close <file>** - Closes the file specified by the file tab under the mouse.
- **Diff <file>** - Diffs the file specified by the file tab under the mouse with the version in source control. Only available when the file is modified.
- **Add <file> to project...** - Adds the file to a project in the workspace. You are prompted to choose which project.
- **Save All** - Saves all modified files.
- **Close All** - Closes all open files.
- **Close Others** - Closes all open files except for the one specified by the file tab under the mouse.
- **List Open Files...** - Shows the **Files** tool window See [Files Tool Window](#) for more information.
- **Split Horizontal** - If the file tab under the mouse refers to the current file in the editor, then you are given the option to split that window horizontally. If the file is not the current file, then this creates a horizontal split, with the current file on the top and the file under the mouse on the bottom.
- **Split Vertical** - If the file tab under the mouse refers to the current file in the editor, then you are given the option to split that window vertically. If the file is not the current file, then this creates a horizontal split, with the current file on the left and the file under the mouse on the right.
- **Copy Full Path to Clipboard** - Copies the full path of the file on the file tab under the mouse to the clipboard.
- **File tab sort order** - The default order for the file or document tabs is alphabetic. This makes it easy to predict where a file tab will be based on the file name. You can also change this value by going to **Tools → Options → Editing → Editor Windows**. This option has several possible values: **Alphabetical**, **Most recently opened**, **Most recently viewed**, or **Manual**. For more information see [File Tab Sort Order Options](#).
- **New file tab position** - Sets whether new tabs are inserted at the right or left end of the file tabs. This option is only available when **File tab sort order** is set to **Most recently opened** or **Manual**.
- **Toggle file tab orientation** - This item changes whether the file tabs are coming down from the top of the tool window or the bottom. This does not affect where the tool window is positioned.
- **Abbreviate similar files** - When the file or document tabs are sorted alphabetically, by default, the tabs do not show the complete name of the file when adjacent files differ only by file extension. This saves space and provides better visibility for associated files. For file names to be abbreviated in this style, their paths and base file name must match exactly. For example, `C:\rectangles\BorderRectangle.cpp` would not abbreviate with `C:\src\include\BorderRectangle.h`. You can also set this option by going to **Tools → Options → Editing → Editor Windows**.
- **Show pictures** - Controls whether or not icons are shown for special windows, such as Build Window, Search Results, or File Manager windows.

- **Show close buttons** - Sets whether to show a close button on each individual tab to make it easier to close tabs. You can also close individual files by clicking on them with the middle mouse button.

Change Directory Dialog

This dialog is used to change the current working directory. It is displayed when you click **File → Change Directory** or use the **gui_cd** command.



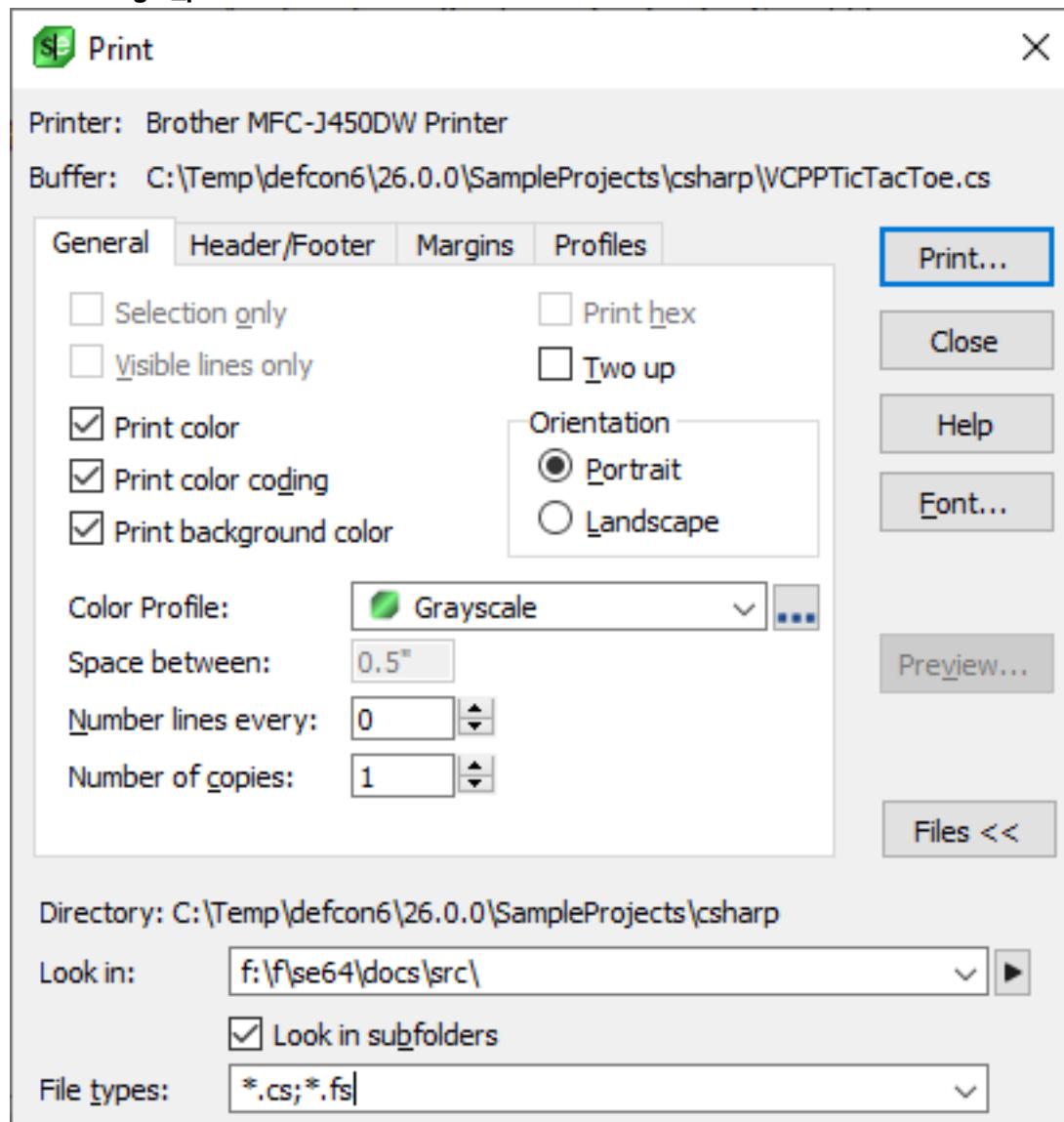
- **Directory name** - Name of directory to change to make active.
- **Expand alias** - Check this box if you want to specify directory aliases in the **Directory name** text box. See [Directory Aliases](#) for more information.
- **Change directory in Build Window** - Check this box if you want the current directory in the Build tool

window to be changed in addition to the editor's current directory.

- **Save Settings** - Save the settings of the **Expand alias** and **Change directory in Build Window** 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

This dialog is used to configure print options and print text files. It is displayed when you click **File** → **Print** or use the **gui_print** command.



The general settings on the Print dialog for Windows are described below.

- **Print** - Sends the selection or active buffer to the printer specified in the Print Setup dialog (**print** command).

- **Font** - Displays the **F**onts option screen, which allows you to specify the font for the printed text. See [Font Options](#) 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). See [Print Preview Dialog](#) for more information.
- **Files** - Expands the 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:

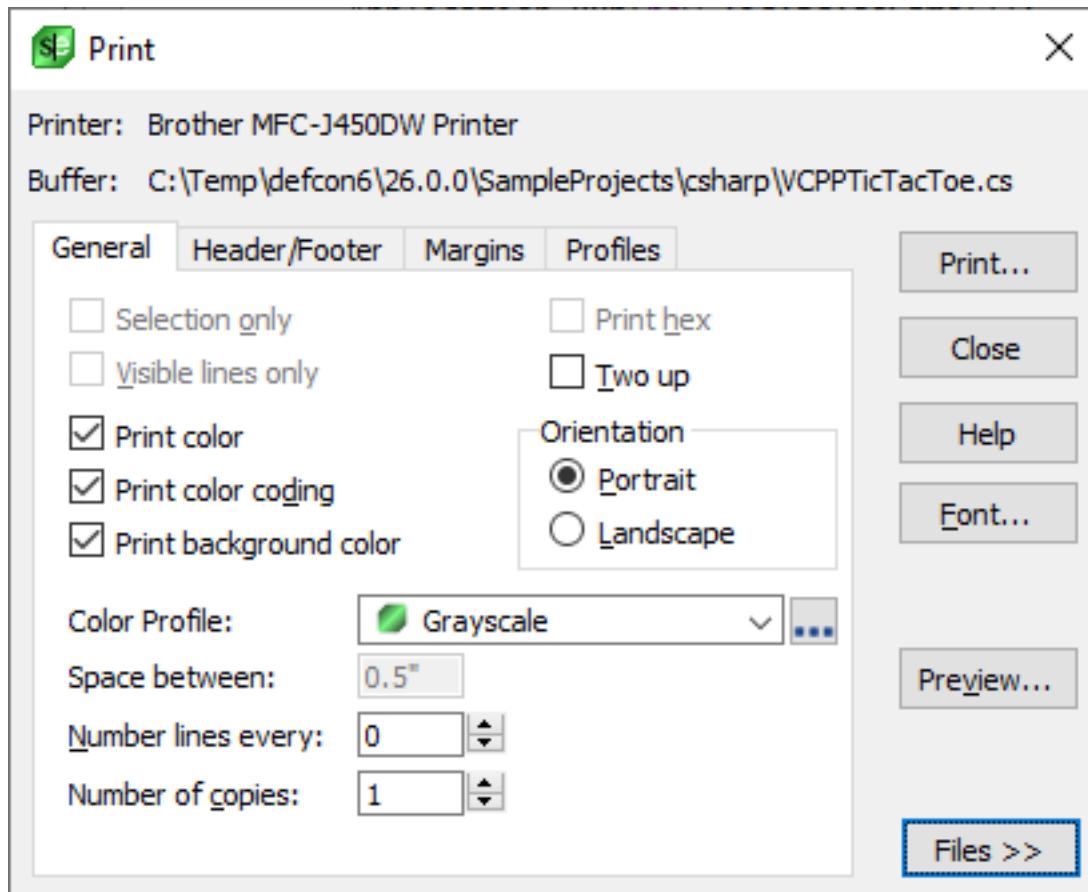
- [General Tab](#)
- [Header/Footer Tab](#)
- [Margins Tab](#)
- [Profiles Tab](#)

Note

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](#)).

General Tab

This tab on the [Print Dialog](#) is used to set general print options.

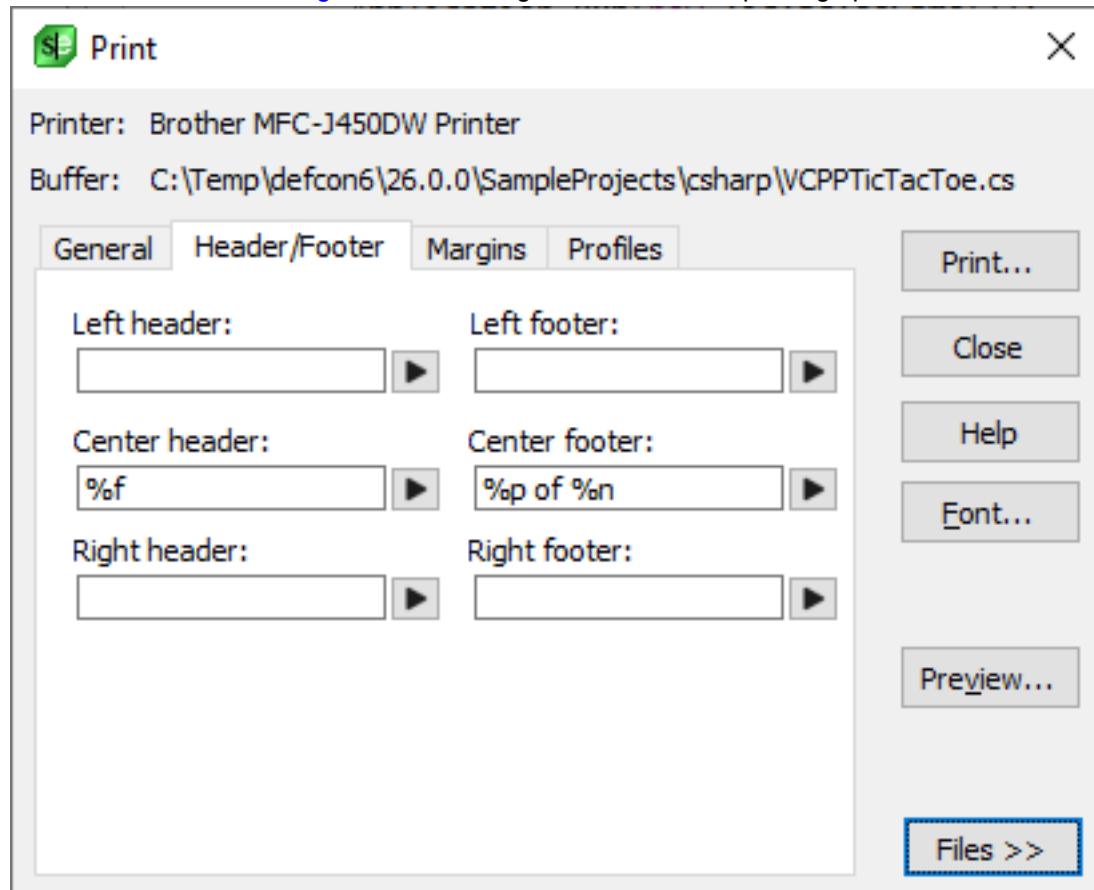


- **Selection only** - When this option is selected, only the selection is printed. To print the selection immediately using the default configuration, use the **print_selection** command.
- **Visible lines only** - When this option is selected, only visible lines are printed. This option allows you to print selective display.
- **Print color** - When this option is selected, language-specific color coding is printed using the same colors.
- **Print color coding** - When this option is selected, language-specific color coding is printed using the same font attributes (bold, italic, underline).
- **Print background color** - When this option is selected, language-specific color coding is printed using the same colors, including background colors. When not enabled, all background colors are replaced with white. When printing using a color profile with a dark background, enabling this feature will not always produce good results and will consume a lot of ink, and disabling this option may result in printing light-colored text against a white background. It is better to select an alternate color profile for printing, such as the **Default** or **Grayscale** profile. See [Color Profile](#) for more information.
- **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).
- **Color Profile** - This lets you select an alternate color profile for printing color coding. By default printing will use the same color profile you use for editing. However, if you use a color profile with a dark background, it will not always be optimal for printing, so it is better to select a color profile with a light background, or the **Grayscale** color profile if you want to avoid using color ink.
- **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

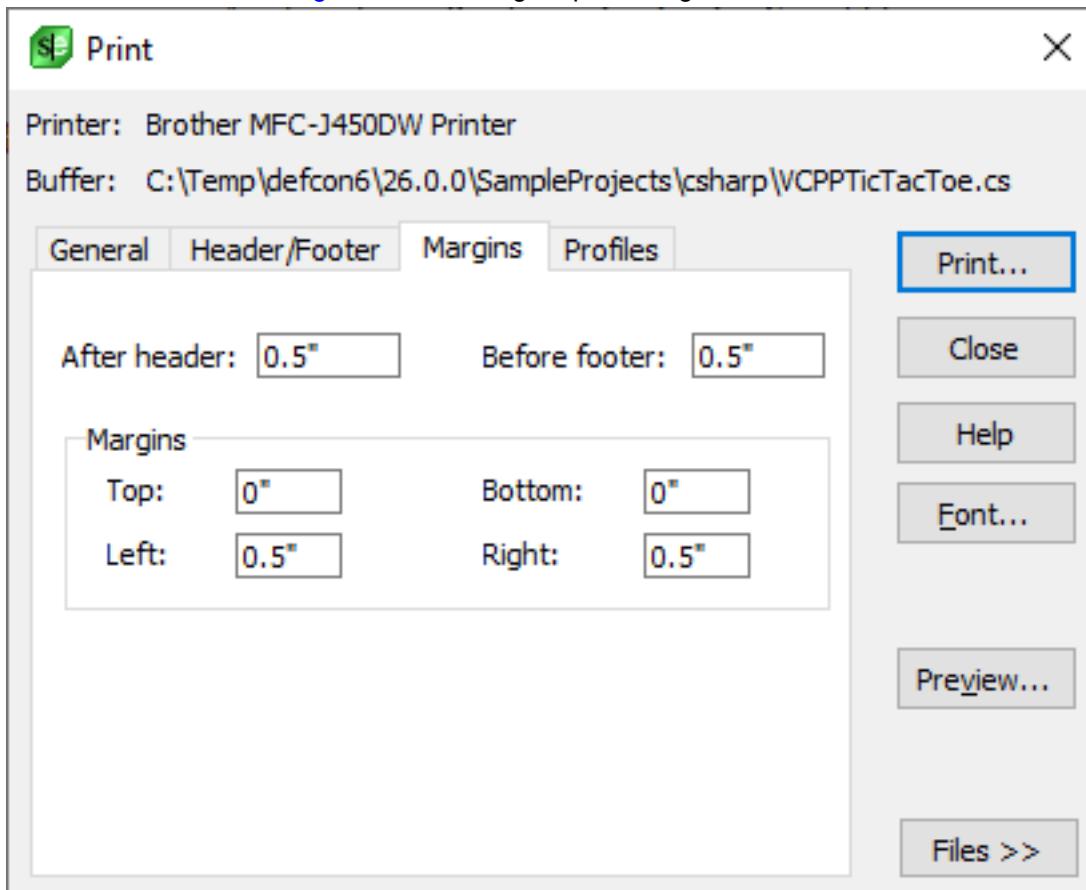
This tab on the [Print Dialog](#) is used to configure header/footer printing options.



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

This tab on the [Print Dialog](#) is used to configure print margins.



- **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.

Profiles Tab

This tab on the [Print Dialog](#) lets you save the current printer settings as a profile

- **Save...** - Allows you to specify a profile name to save the print settings to.
- **Delete...** - Allows you to delete a profile
- **Rename...** - Allows you to rename a profile

Print Preview Dialog

Displays a preview of what the printed file will look like (`print_preview` command).

File Dialogs and Tool Windows

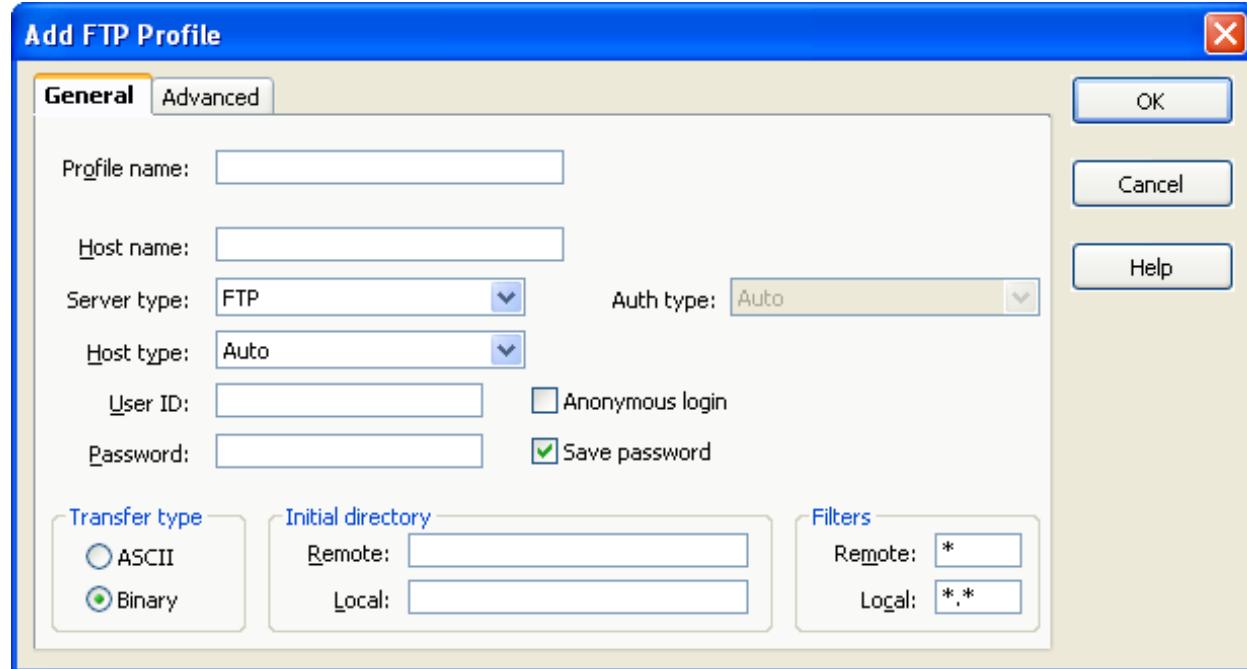


Add/Edit FTP Profile Dialog

This dialog is used to create or edit profiles for FTP connections. It is displayed when you click **File** → **FTP** → **Profile Manager**, then click **Add** or **Edit**. The title of the dialog box changes based on whether you are adding a profile or editing an existing profile, but the interface is the same. The dialog consists of two tabs: [General Tab](#) and [Advanced Tab](#).

General Tab

This tab on the [Add/Edit FTP Profile Dialog](#) is used to configure general FTP profile settings.

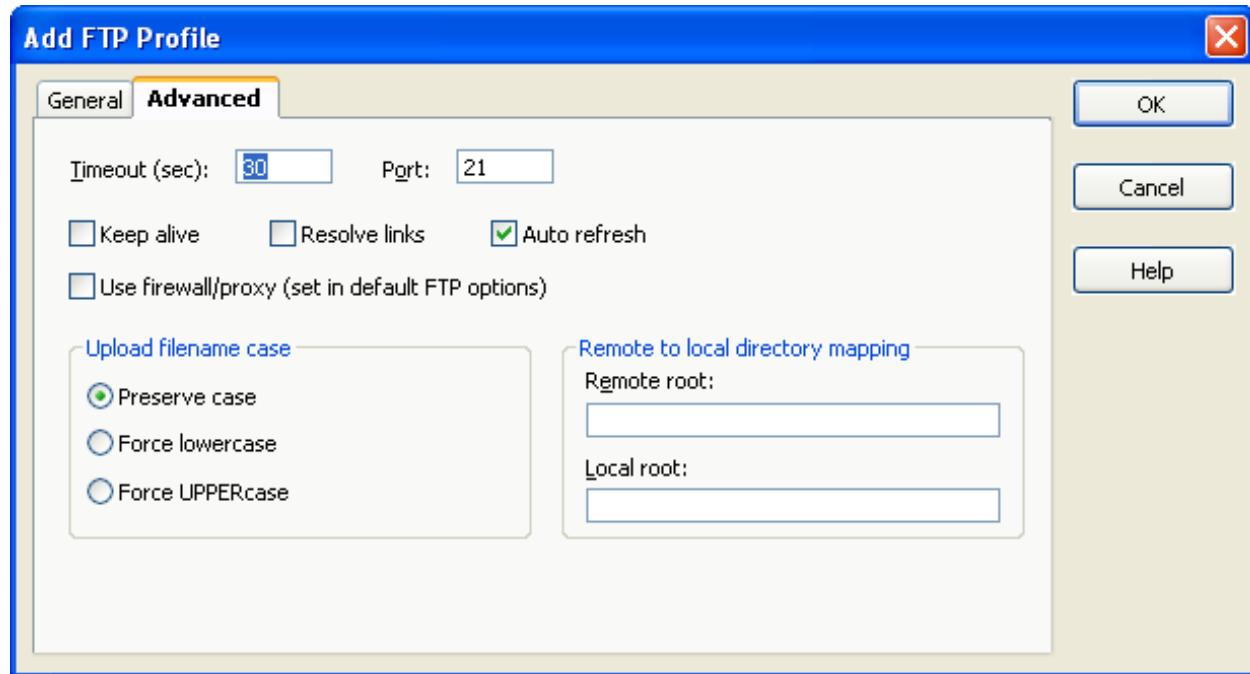


- **Profile name** - Name displayed in connection combo box.
- **Host name** - Host name of the FTP server.
- **Server type** - If you are connecting to an FTP server, select FTP. If you are connecting to a Secure Shell (SSH) server that supports the SFTP subsystem, select SFTP/SSH.
- **Host type** - 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

This tab on the [Add/Edit FTP Profile Dialog](#) is used to configure advanced FTP profile settings.



- **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. Unavailable for SFTP/SSH server type.
- **Resolve links** - Resolves symbolic links on the remote host. Unavailable 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 activated.
- **Upload filename case** - Indicates what file case should be used for the remote file name based on the local file name.
- **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.

Edit

This section describes items on the **Edit** menu and associated dialogs and tool windows. See [Basic Editing](#) for more details about editing operations.

Edit Menu

The table below describes items on the **Edit** menu.

Edit Menu Item	Description	Command
Undo	Undoes the last edit operation.	undo
Redo	Undoes an undo operation.	redo
Multi-file Undo	Undoes the last multi-file operation (i.e.: Find and Replace in all files).	mfundo
Multi-file Redo	Undoes an multi-file undo operation.	mfredo
Cut	Deletes the selected text and copies it to the clipboard.	cut
Copy	Copies the selected text to the clipboard.	copy_to_clipboard
Paste	Inserts the clipboard into the current file.	paste
List Clipboards	Displays the Clipboards tool window, which allows you to view and insert the selected clipboard. See Clipboards and also Edit Dialogs and Tool Windows .	list_clipboards
Copy Word	Copies the current word to the clipboard.	copy_word
Append to Clipboard	Appends the selected text to the clipboard.	append_to_clipboard
Append Cut	Deletes the selected text and appends it to the clipboard.	append_cut

Edit Menu

Edit Menu Item	Description	Command
Insert Literal	Displays the Insert Literal dialog, which allows you to insert a specified character code. See Inserting Literal Characters .	<code>insert_literal</code>
Select	Displays menu for selecting and deselecting text. See Edit Select Menu .	N/A
Delete	Displays menu for deleting text. See Edit Delete Menu .	N/A
Complete Previous Word	Retrieves previous word or variable matching word prefix at cursor.	<code>complete_prev</code>
Complete Next Word	Retrieves next word or variable matching word prefix at cursor.	<code>complete_next</code>
Fill	Displays the Fill Selection dialog, which lets you fill the selected text with a character you choose.	<code>gui_fill_selection</code>
Indent	Indents the selected text based on the tabs or indent for each level.	<code>indent_selection</code>
Unindent	Unindents the selected text based on the tabs or indent for each level.	<code>unindent_selection</code>
Other	Displays menu containing more edit-related commands. See Edit Other Menu .	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.	<code>select_char</code>

Edit Menu

Edit Select Menu Item	Description	Command
Block	Starts or ends a block/column selection.	<code>select_block</code>
Line	Starts or ends a line selection.	<code>select_line</code>
Word	Selects the word under cursor.	<code>select_whole_word</code>
Code Block	Selects text in current code block (if/loop/switch block etc.).	<code>select_code_block</code>
Procedure	Selects text in current function including function heading.	<code>select_proc</code>
Deselect	Unhighlights selected text.	<code>deselect</code>
All	Selects all text in current buffer.	<code>select_all</code>

Edit Delete Menu

The **Edit → Delete** menu contains text deletion operations. See [Cutting and Deleting Text](#) for more information.

Edit Delete Menu Item	Description	Command
Word	Deletes text from the cursor to the end of the current word and copies it to the clipboard.	<code>cut_word</code>
Line	Deletes the current line and copies it to the clipboard.	<code>cut_line</code>
To End of Line	Deletes text from the cursor to the end of the line and copies it to the clipboard.	<code>cut_end_line</code>
Selection	Deletes the selected text.	<code>delete_selection</code>
All	Delete all text in current buffer.	<code>delete_all</code>

Edit Other Menu

Edit Menu

The **Edit → Other** menu contains miscellaneous editing operations.

Edit Other Menu Item	Description	Command
Lowcase	Translates the characters in the selected text to lowercase.	lowcase_selection
Upcase	Translates the characters in the selected text to uppercase.	upcase_selection
Capitalize	Capitalizes the first character of the current word.	cap_selection
Shift Left	Deletes the first column of text in each line of the selected text.	shift_selection_left
Shift Right	Inserts a space at the first column of each line of the selected text.	shift_selection_right
Overlay Block	Overwrites selected block/column of text at the cursor.	overlay_block_selection
Adjust Block	Overlays the selected text at the cursor and fills the original selected text with spaces.	adjust_block_selection
Copy to Cursor	Copies the selection to the cursor without using the clipboard	copy_to_cursor
Enumerate	Displays the Enumerate dialog, which allows you to add incrementing numbers to a selection. See Enumerate Dialog .	gui_enumerate
Filter Selection	Displays a Command dialog, which allows you to filter selected text through an external command. See Filter Selection: Command Dialog .	filter_selection
Copy UCN As Unicode	Copies selected UCN to the clipboard as Unicode. See Using Unicode .	copy_ucn_as_unicode
Copy Unicode As	Displays menu containing commands for copying Unicode as UCN. See Copy Unicode As	N/A

Edit Menu

Edit Other Menu Item	Description	Command
	Menu.	
Tabs to Spaces	Converts tabs to appropriate number of spaces. If there is no selection the entire buffer is converted.	convert_tabs2spaces
Spaces to Tabs	Converts leading spaces to tabs. If there is no selection the entire buffer is converted.	convert_spaces2tabs
Remove Trailing Whitespace	Removes trailing whitespace characters from the ends of lines.	remove_trailing_spaces
Check Line Endings	Check for inconsistent line endings and optionally change line endings.	check_line_endings
Check Line Endings	Check for inconsistent line endings and optionally change line endings.	check_line_endings
Block Insert Mode	Allows you to insert/delete text for an entire block/column selection.	block_insert_mode

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.	copy_unicode_as_c
Regex (UTF-32 \x{HHHH})	Copies Unicode characters in selection as Regex UTF-32 \x{HHHH} notation.	copy_unicode_as_regex
Java/C# (UTF-16 \uHHHH)	Copies Unicode characters in selection as Java/C# UTF-16 \uHHHH notation.	copy_unicode_as_java

Copy Unicode As Menu Item	Description	Command
UCN (UTF-32 \uHHHH and \UHHHHHHHHH)	Copies Unicode characters in selection as UCN UTF-32 \uHHHH and \UHHHHHHHHH notation.	copy_unicode_as_ucn
SGML/XML hexadecimal (UTF-32 &xHHHH;)	Copies Unicode characters in selection as SGML/XML hexadecimal UTF-32 &xHHHH; notation.	copy_unicode_as_xml
SGML/XML decimal (UTF-32 &xDDDD;)	Copies Unicode characters in selection as SGML/XML decimal UTF-32 &xDDDD; notation.	copy_unicode_as_xmldec

Edit Dialogs and Tool Windows

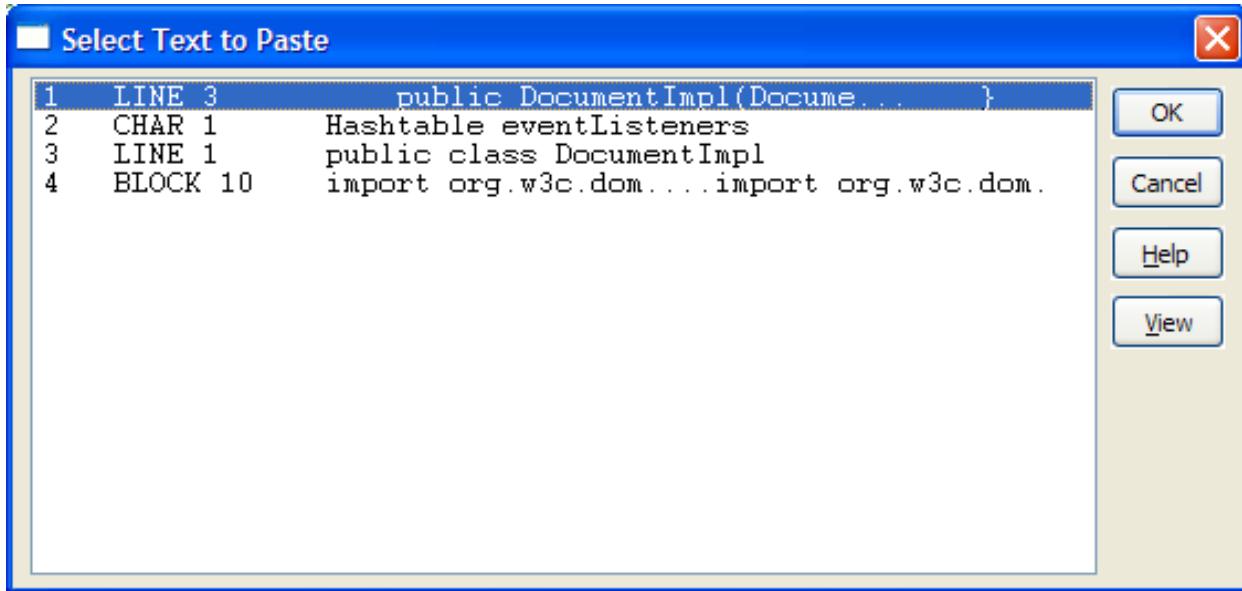
This section describes the dialogs and tool windows that are associated with text editing.

Select Text to Paste Dialog

This is a modal dialog available for viewing and inserting recently used SlickEdit® clipboards (not the same as the operating system clipboard). To display the dialog, you can use either the **old_list_clipboards** command or the **list_clipboards_modal** command. These commands are identical. If there are no clipboards, the message line states **No clipboards**. See [Clipboards](#) for more information.

Tip

You can also use the Clipboards tool window to view and insert clipboards (**Edit → List Clipboards** or **list_clipboards** command). It provides the same information as the Select Text to Paste dialog, except it includes a Preview area to view color-coded clipboard contents and provides some additional functionality. See [Clipboards](#) for more information.



The dialog shows a list of clipboards. To insert a clipboard at the current cursor location, double-click on the clipboard to insert, or, select the clipboard to insert and press **Enter** or click **OK**.

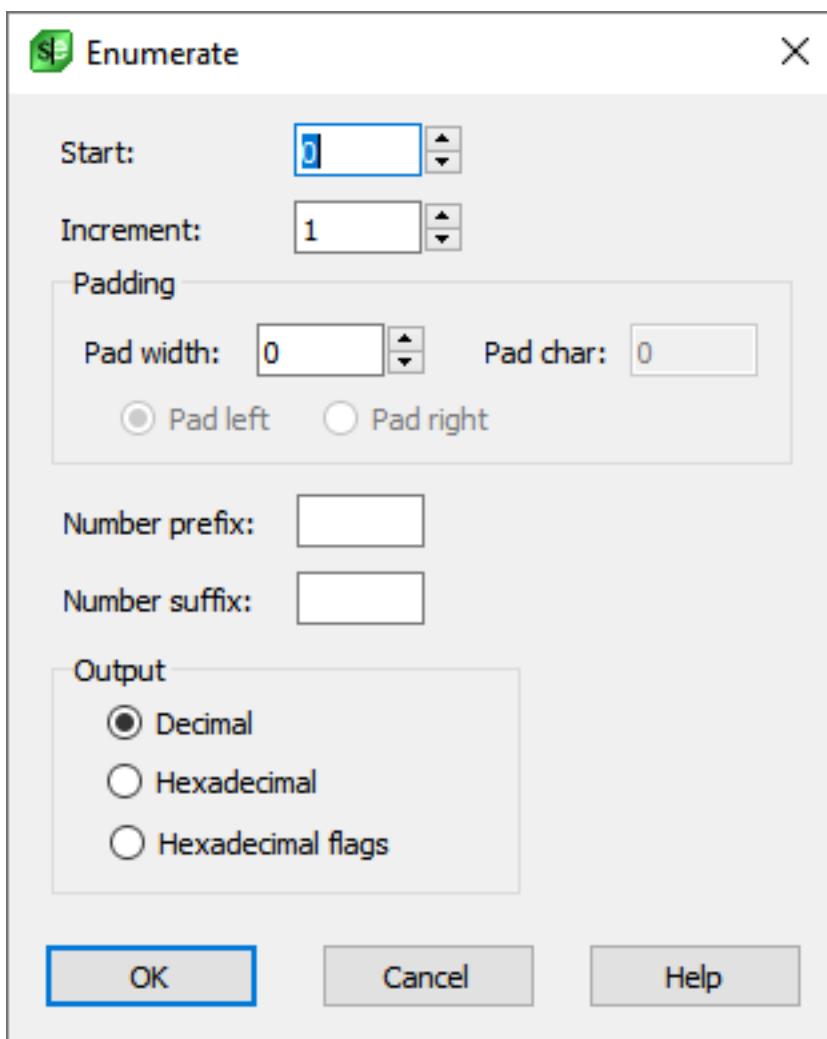
To see the entire contents of a condensed clipboard, click the **View** button. The View Clipboard dialog opens showing the color-coded contents in an edit window. From here, you can copy all or part of the contents to the operating system clipboard.

The dialog shows the following information:

- **Clipboard name/number** - This is the number of the clipboard or the name, if using [Named Clipboards](#). Clipboards are numbered with the most recent clipboard first, which always appears at the top of the list. You can use this value with the paste command to insert the specified clipboard. For example, type **paste 2** on the command line to insert clipboard 2 at the cursor location.
- **Clipboard type** - The clipboard type can be CHAR, LINE, or BLOCK. A CHAR type clipboard is inserted before the current character. 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** → **Editing** → **General**). A BLOCK type clipboard is inserted before the current character and pushes over all text intersecting with the block. No lines are inserted.
- **Line count** - The number following the clipboard type indicates the number of complete or partial lines of text in the clipboard.
- **Clipboard contents/summary** - This area shows all or a portion of the clipboard contents. If the contents exceed the viewing area, they are condensed.

Enumerate Dialog

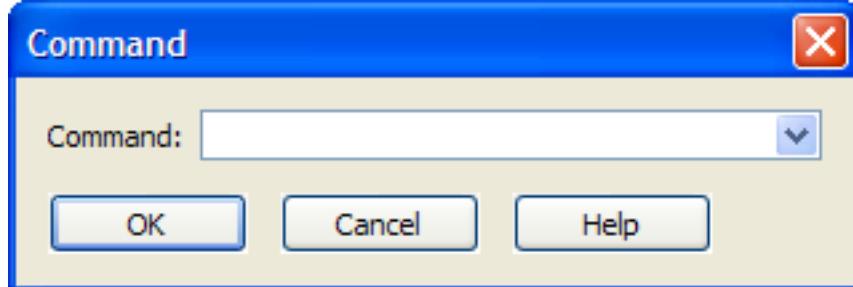
This dialog contains options for adding incrementing numbers to a selection. It is displayed when you click **Edit** → **Other** → **Enumerate** or use the **gui_enumerate** command. Alternatively, you can add incrementing numbers to a selection using the **enumerate** command with options on the command line. See the Help system for command syntax.



- **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.
- **Padding - Pad width** specifies the width for each number. Number is padded with the **Pad char** specified either on the left or right depending on the **Pad left/Pad right** radio button setting.
- **Number prefix** - Additional text to place before the optionally padded number.
- **Number suffix** - Additional text to place after the optionally padded number.
- **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), **hhhhH** (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 is used to specify a command to run against the selected text. It is displayed when you click **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.

Search

This section describes items associated with searching and replacing. 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.

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 Search Dialogs and Tool Windows .	<code>gui_find</code>
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 Find in Files Tab .	<code>find_in_files</code>
Next Occurrence	Searches for the next occurrence of the last string you searched for.	<code>find_next</code>
Previous Occurrence	Searches for the previous occurrence of the last string you searched for.	<code>find_prev</code>
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 Replace Tab .	<code>gui_replace</code>
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 Replace in Files Tab .	<code>replace_in_files</code>
Incremental Search	Searches for match incrementally. See Incremental Searching .	<code>i_search</code>

Search Menu

Search Menu Item	Description	Command
Find File	Displays the Find File dialog, which lets you search for files on disk.	<code>find_file</code>
Find Symbol	(Pro only) Searches Context Tagging® databases for a symbol you specify. See Find Symbol Tool Window .	<code>activate_find_symbol</code>
Go to Line	Places the cursor on a line you specify. See Navigating to a Specific Line .	<code>gui_goto_line</code>
Go to Column	Places the cursor in a column you specify on the current line.	<code>gui_goto_col</code>
Go to Offset	Places the cursor on a byte/character offset in the current file. See Navigating to an Offset .	<code>gui_seek</code>
Go to Matching Parenthesis	Finds the matching parenthesis or begin/end structure pair. See Begin/End Structure Matching .	<code>find_matching_paren</code>
Go to Definition of	(Pro only) Pushes a bookmark at the cursor and navigates to the definition of the current symbol. See Symbol Navigation .	<code>gui_push_tag</code>
Go to Reference of	(Pro only) Searches for references to the symbol under the cursor. See Symbol Navigation .	<code>push_ref</code>
Bookmarks	Displays bookmarks menu. See Search Bookmarks Menu .	N/A
Last Find/Grep List	Displays list of Files/Buffers generated by Find commands.	<code>grep_last</code>

Search Bookmarks Menu

Search Dialogs and Tool Windows

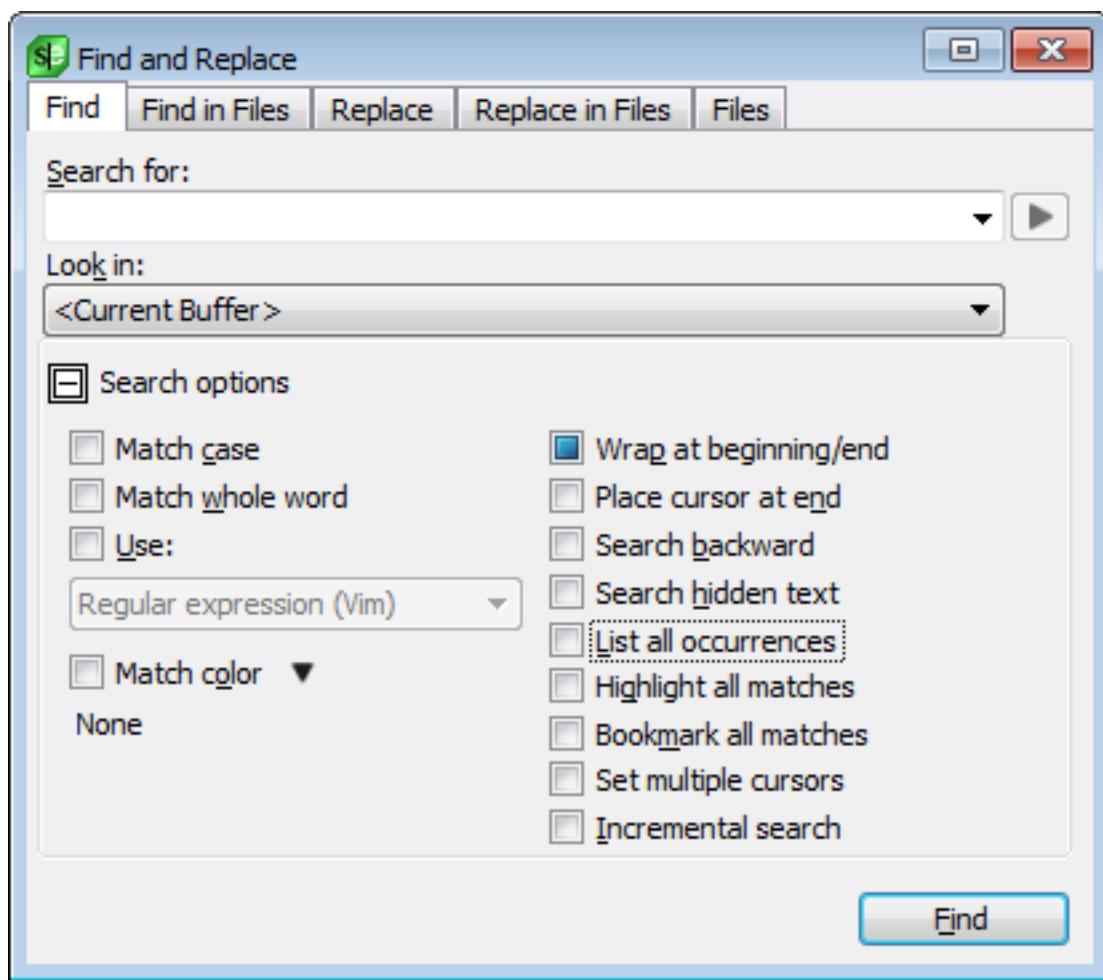
Bookmarks Menu Item	Description	Command
Push Bookmark	Pushes a bookmark at the cursor. See Bookmarks .	<code>push_bookmark</code>
Pop Bookmark	Pops the last bookmark. See Bookmarks .	<code>pop_bookmark</code>
Bookmark Stack	Displays the Bookmark Stack tool window. See Bookmark Stack Dialog .	<code>pop_bookmark</code>
Set Bookmarks	Sets a persistent bookmark on the current line. See Bookmarks .	<code>set_bookmark</code>
Go to Bookmark	Displays the Go to Bookmark dialog from which you can select a bookmark to navigate to. See Navigating Named Bookmarks .	<code>goto_bookmark</code>
Toggle Bookmark	Toggles setting a bookmark on the current line. See Bookmarks .	<code>toggle_bookmark</code>
Bookmarks Tool Window	Lists bookmarks and allows you to add and delete bookmarks. See Bookmarks .	<code>activate_bookmarks</code>
Next Bookmark	Go to next bookmark. See Bookmarks .	<code>next_bookmark</code>
Previous Bookmark	Go to previous bookmark. See Bookmarks .	<code>prev_bookmark</code>

Search Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with searching and replacing. Note that default search options are also available (**Tools** → **Options** → **Editing** → **Search**). See [Search Options](#) for a description of these settings.

Find and Replace Tool Window

This tool window is displayed when you click one of the find or replace items on the **Search** menu, or when you click **Search** → **Find**. See [Find and Replace](#) for information about searching and replacing in SlickEdit®.



The Find and Replace tool window contains a right-click context menu and five tabs: the [Find Tab](#), the [Find in Files Tab](#), the [Replace Tab](#), the [Replace in Files Tab](#), and the [Files Tab](#).

Find and Replace Tool Window: Context Menu

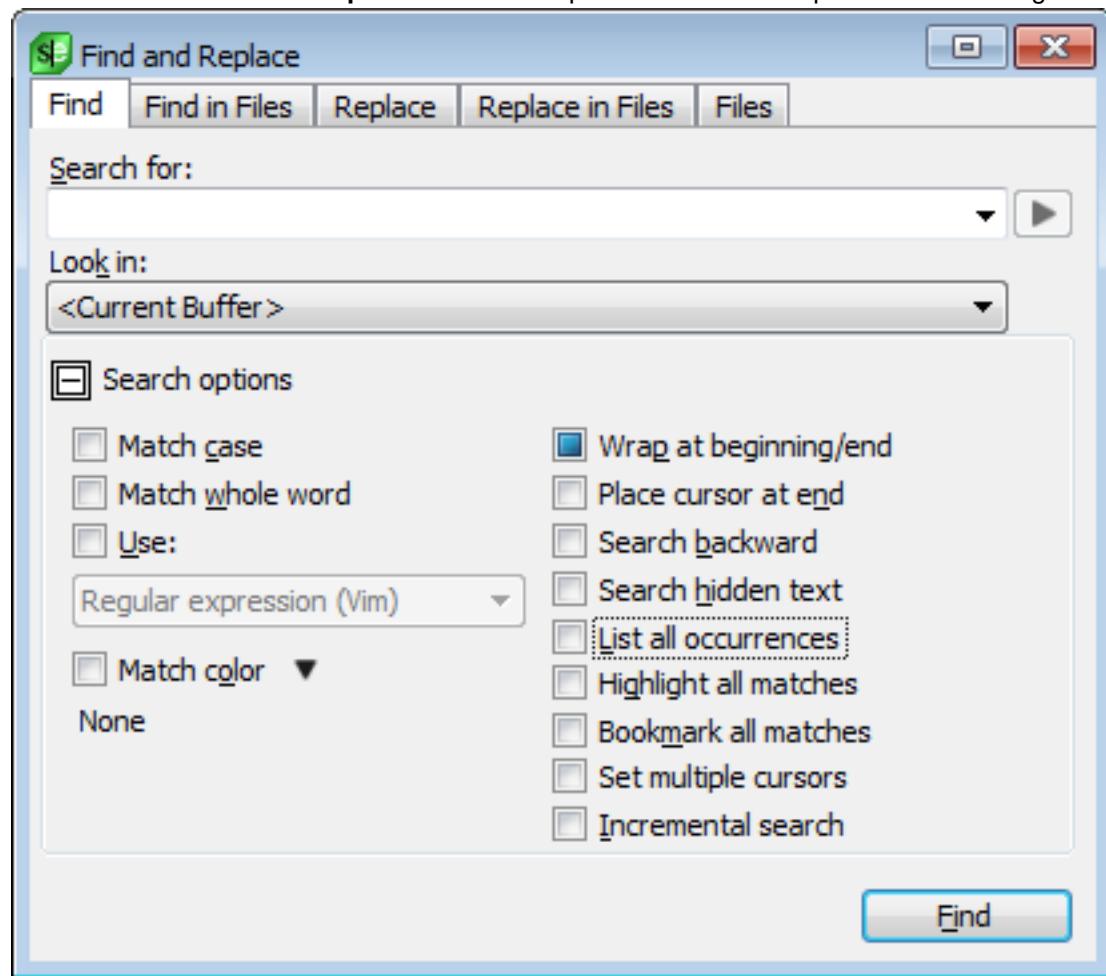
Right-click in the background of the **Find and Replace** tool window to access the following items:

- **Saved Search Expressions** - See [Saving Search and Replace Values](#).
- **Current Word at Cursor** - Sets Find and Replace tool window Search for textbox with current word at cursor.
- **Current Selection** - Sets Find and Replace tool window Search for textbox with current selection.
- **Configure Options** - Displays the [Search Options](#) screen, from which you can set the default search options that the tool window should use.
- **Use Default Options** - If selected, the options specified in the [Search Options](#) are used instead of the options selected in the Find and Replace tool window. See [Default Search Options](#).
- **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 Options](#).
- **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.
- **Switch to Mini Find** - Change to using Mini Find with current options.

Find Tab

This tab on the **Find and Replace** tool window provides fields and options for searching and finding text.



- **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 selected 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 click **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **General**. 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 (SlickEdit®)**
 - **Regular expression (Perl)**
 - **Regular expression (Vim)**
 - **Wildcards (*,?)**

See [Using Regular Expressions in SlickEdit®](#) for more information.

- **Match color** - Enables color coding search. To configure color coding search, use the button next to the option to display Color Coding Search Options. 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 cleared or until one or more other elements are selected. Putting a check in any check box essentially clears 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 clear 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.
 - **Cleared** - 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.).

- **Wrap at beginning/end** - If selected, the search will always be performed on the entire buffer, starting from the cursor when the range is the current buffer.
- **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 set Selective Display options, from the main menu click **View** → **Selective Display**. See [Selective Display](#) for more information.
- **List all occurrences** - Select this option to see a list of all instances of the search string in the file. The Find tab expands to show the **Results options**, where you can specify the output destination. These options are similar to the Results options on the Find in Files tab. See [Find in Files Tab](#) 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, from the main menu, click **Tools** → **Options** → **Appearance** → **Colors** and select **Highlight** from the **Screen element** list. Choose your desired color settings and click **OK**. See [Colors](#) for more information.

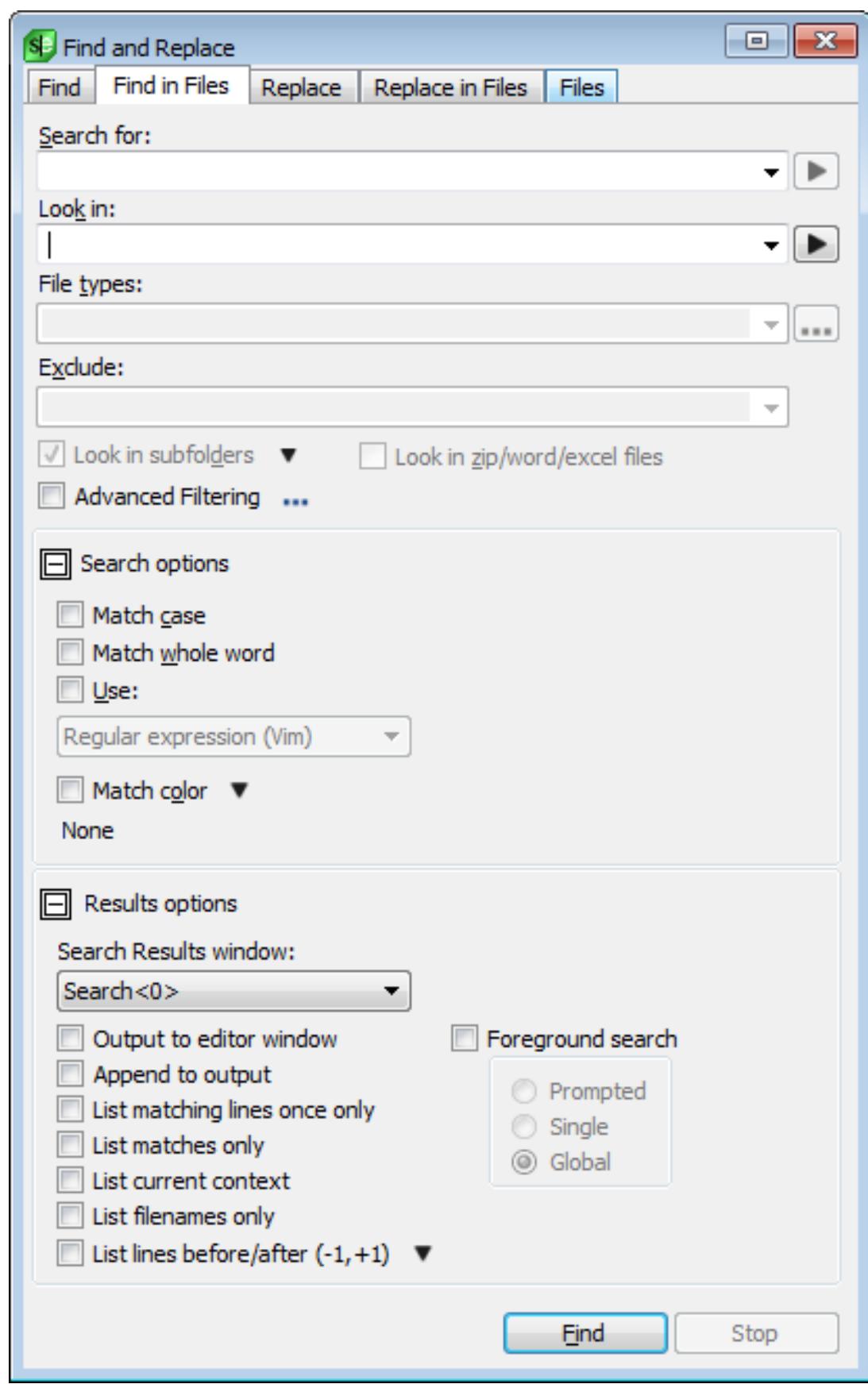
To clear all highlighted text in all buffers, clear the **Highlight matches** option or simply close the Find and Replace tool window.

- **Bookmark matches** - Select this option to bookmark lines with matching patterns and display the Bookmarks tool window when a match is bookmarked.
- **Set multiple cursors** - Select this option to add a cursor to all matched patterns in current search range. See [Multiple Cursors and Selections](#) for more information on multiple cursors.
- **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 in Files Tab

This tab on the [Find and Replace Tool Window](#) provides the same functionality as the [Find Tab](#), with the added ability to conduct multi-file searches. Additional options are described below.

Search Dialogs and Tool Windows



- **Look in** - This field allows you to specify one or more wildcard filespecs to search separated with semicolons.

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.

In addition to directories, a file list can be specified as the source for searches. A file list is a simple text file with a filename on each line. Additionally, a file list can be auto generated from an existing Search Results tab, that only searches files with matches in the current set of Search Results.

In all SlickEdit internal dialogs, SlickEdit can treat a .zip or .jar file like a file system. To search through a .zip file append a trailing file separator (ex. "filename.zip\").

- **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. Click the ... button to the right of the **File types** field to edit the list of file type filters that appear in this list.
- **Exclude** - Paths, files, file types, or language modes can be excluded from a multi-file search by specifying ant-like wildcards. To specify multiple patterns, separate them with semicolons. No files are searched in a path that is excluded, including any files in sub-directories beneath. The Excludes combo box list allows you to choose <Default Excludes> or <Binary Files>. When <Binary Files> is chosen (or <Default Excludes> which contains <Binary Files> by default), files that appear to have binary data are excluded from the search. This is useful when searching files with a * or *.* wildcard. For example, search Unix shell scripts with no file extensions while excluding executables. You can configure the excludes specified by <Default Excludes> at **Tools** → **Options** → **Editing** → **Search** → **Default Excludes**. See examples of exclude patterns, below.

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.
*.png; *.ico; *.jpg	Exclude any file with extension .png, .ico, or .jpg.
.svn\	Exclude any files in paths named ".svn".
C*\	Exclude any files in paths that start with "C".
/b*/debug//backup/	Exclude all files in this path name.
demo	Exclude any file (not directory) with "demo" in the name.

Example	Description
<Binary Files>	Exclude Binary files. All extensions defined by the Binary language are excluded. Also, unknown files which appear to contain binary data use the Binary language.

- **Look in subfolders** - Select this option to expand the search to sub-directories of the folder specified in the **Look in** field.
- **Look in zip/word/excel files** - Select this option to search the files contained in .zip, .jar, .xlsx, .docx, .jmod, .tar, .gz, .Z, .xz, .bz2, .cpio, .cpgz, and .rpm files. Keep in mind that a .xlsx and .docx file (word or excel file) is not a single file. These are zip files containing many XML files. When searching for text in a word document you probably want to specify "document.xml" (or "*.xml") for **File types**:. When searching for text in a excel document you probably want to specify "sharedStrings.xml" (or "*.xml") for **File types**:. This option only supports recursive subdirectory specifications and not project files which happen to be zip, tar, gz, or other compressed files.
- **Advanced Filtering** - Enable and configure advanced file stats filtering options. Additional filtering options include setting a maximum file size for searches and selecting files by last modified time. Use the button next to the option to configure advanced file stats. Optionally specify the maximum searchable file size in kilobytes (Max File Size). Optionally specify a last modified date range (File Modified Time). Date range options include:
 - **Date Only** - Only search files modified on selected date.
 - **Before** - Only search files modified before selected date and time.
 - **After** - Only search files modified after selected date and time.
 - **Range** - Only search files modified in selected starting and ending date/time.
 - **Not in Range** - Only search files modified either before selected starting date/time or after date/time.

Date and time options use local file time. Date setting uses YYYY-MM-DD (Year-Month-Date) format. Time uses hh:mm:ss format (Hour:Minutes:Seconds), seconds optional. Ctrl+Space or Shift+Space can be used to quickly set to current Date or current Time. Menu option button next to the date range drop down include actions to preset the dates to: Today, Last Week, Last Month, Last 3 Months, Last Year.

- **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:

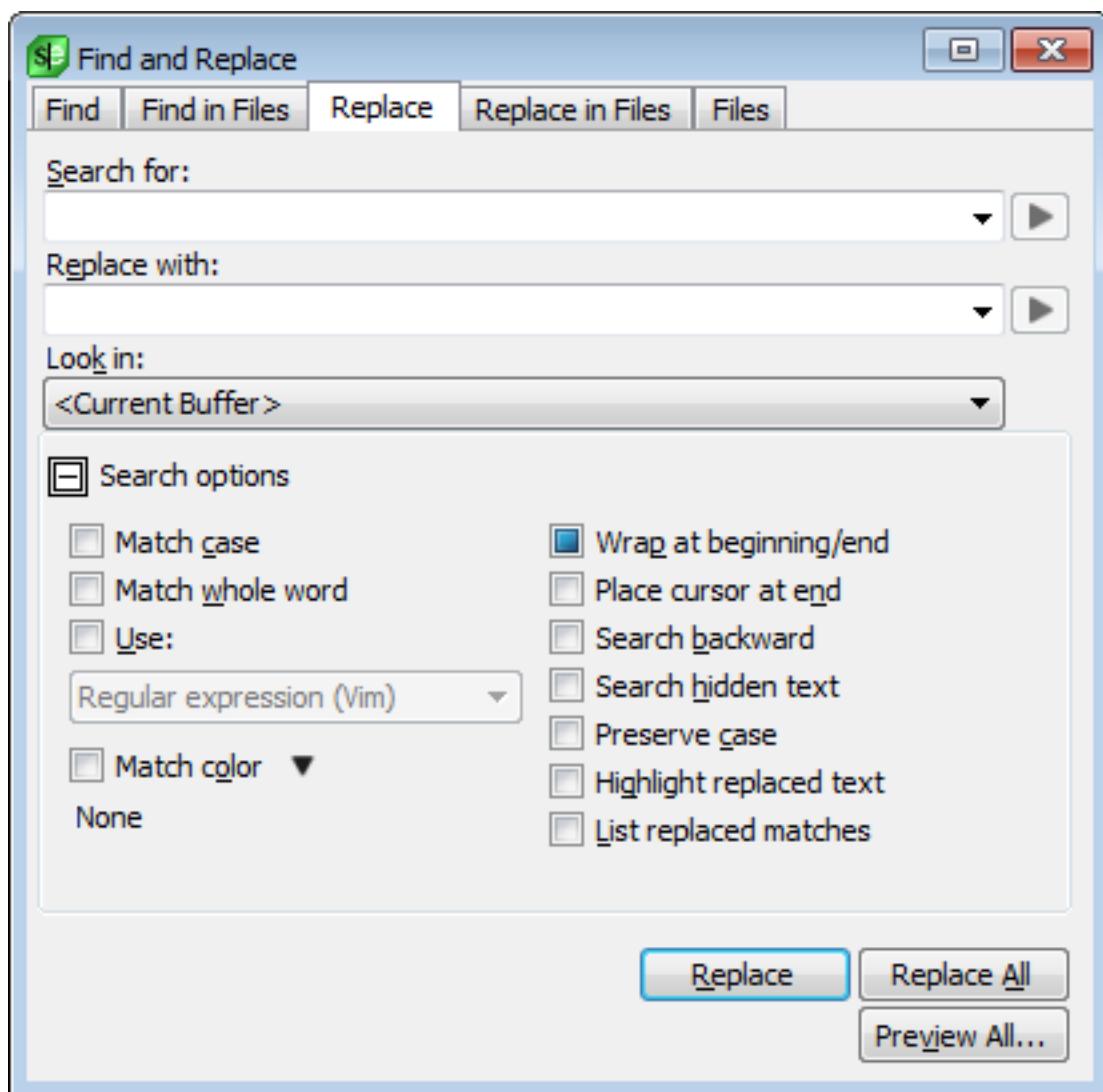
- **Quick Search** - Finds the next occurrence of the text selected.
- **Filter Search Results** - Select this option to display the Filter Search Results dialog. There are two filter tabs: Filter Text and Filter Files. With Filter Text, you can search for other text in the search results. If a match 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). With Filter Files, you can additionally filter by filename. The Include and Exclude options allow you to specify with files to include or exclude using ant-like wildcard syntax. Behaves identically to how File Types/Exclude options are used in Find in Files options.
- **Find in Files using Search Results** - Configure Find in Files to search file list of only files found in the current Search Results 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.
- **Generate File List** - Create new file with list of filenames of matching files in the current Search Results tab.
- **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 matching lines once 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.
- **List matches only** - If selected, only the matching expression is displayed in search output, instead of the line the match occurs.
- **List current context** - If selected and context tagging is supported for the file, include current context with matches.
- **List filenames only** - If selected, only file names and not occurrences are listed in the search output.
- **List lines before/after** - If selected, include lines before and/or after a matching line in Search Results. Use button next to options to configure how many lines to include before/after matching line.

- **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

This tab on the [Find and Replace Tool Window](#) provides options for searching and replacing text. The same search options from the [Find Tab](#) are provided, as well as the additional replace options described below.

Search Dialogs and Tool Windows



- **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 Perl Tagged Expressions](#) for more information.
- **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.
- **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.

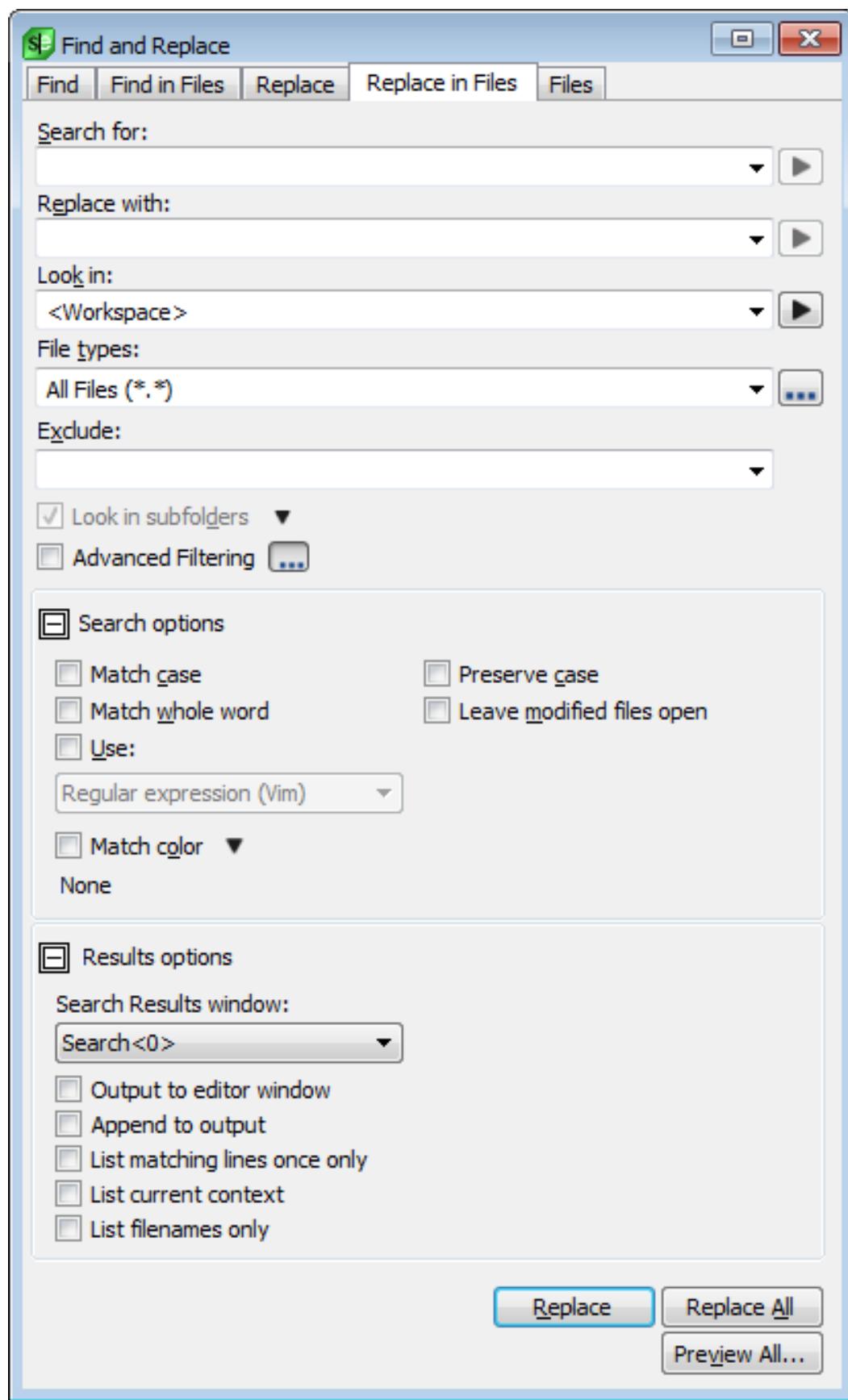
Tip

You can use the menu items **Edit → Undo** and **Edit → Redo** to undo/redo replacements.

Replace in Files Tab

This tab on the [Find and Replace Tool Window](#) provides the same functionality as the [Replace Tab](#), with the added ability to conduct multi-file replacements. It contains one additional option, described below.

Search Dialogs and Tool Windows



- **Leave modified files open** - Select this option to open all of the files on which a replace has been performed.

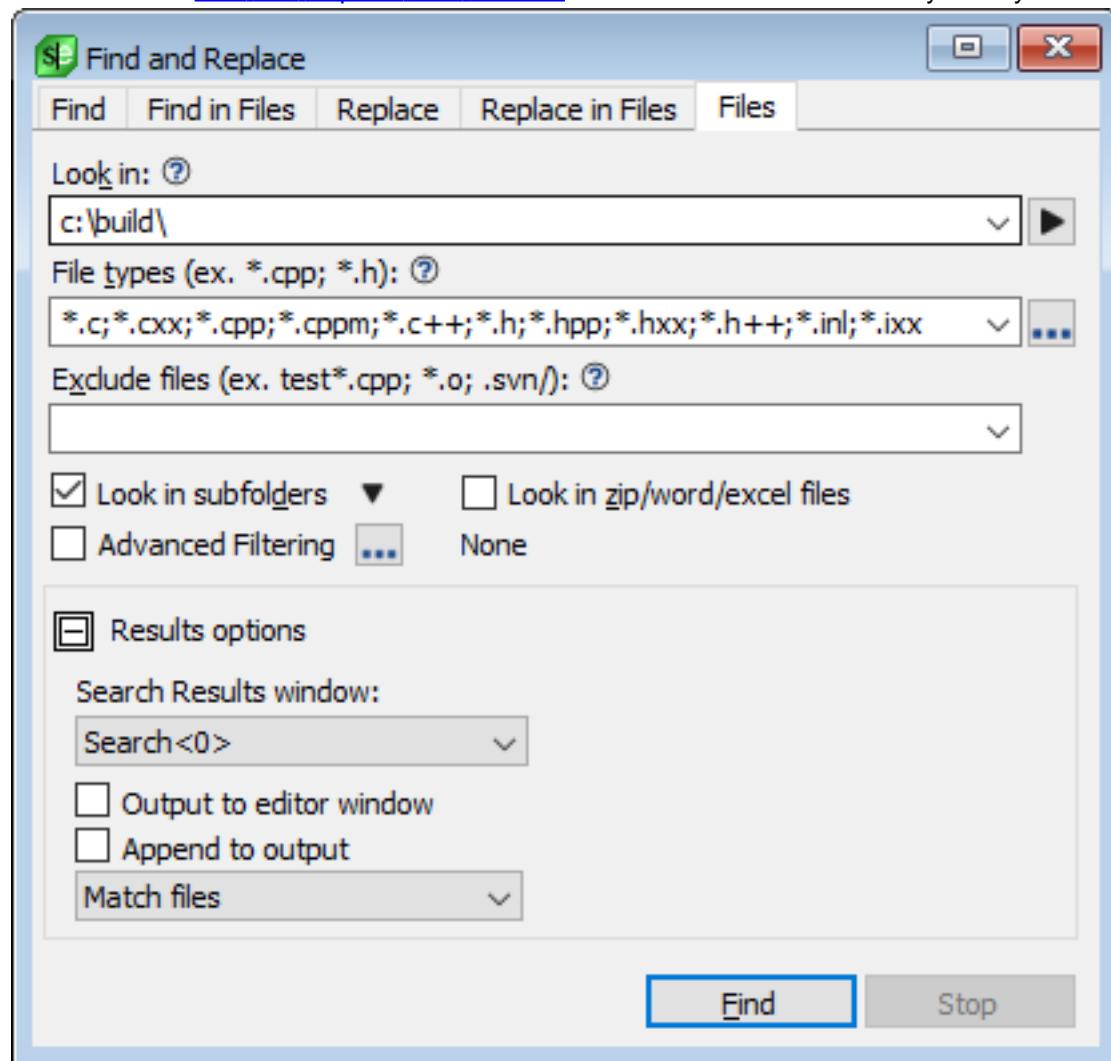
The **Results options** are the same as those on the [Find in Files Tab](#).

Tip

You can use the menu items **Edit → Multi-File Undo** and **Edit → Multi-File Redo** to undo/redo replacements in multiple files.

Files Tab

This tab on the [Find and Replace Tool Window](#) is used to search for files that you may wish to open.



- **Look in** - This field allows you to specify one or more wildcard filespecs to search separated with semicolons.

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.

In all SlickEdit internal dialogs, SlickEdit can treat a .zip or .jar file like a file system. To search through a .zip file append a trailing file separator (ex. "filename.zip/").

- **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. Use the combo box, near the bottom of the dialog to choose whether to match files, directories, or files and directories. Click the ... button to the right of the **File types** field to edit the list of file type filters that appear in this list.
- **Exclude** - Paths, files, file types, or language modes can be excluded from a multi-file search by specifying ant-like wildcards. To specify multiple patterns, separate them with semicolons. No files are searched in a path that is excluded, including any files in sub-directories beneath. The Excludes combo box list allows you to choose <Default Excludes> or <Binary Files>. When <Binary Files> is chosen (or <Default Excludes> which contains <Binary Files> by default), files that appear to have binary data are excluded from the search. This is useful when searching files with a * or *.* wildcard. For example, search Unix shell scripts with no file extensions while excluding executables. You can configure the excludes specified by <Default Excludes> at **Tools** → **Options** → **Editing** → **Search** → **Default Excludes**. See examples of exclude patterns, below.

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.
*.png; *.ico; *.jpg	Exclude any file with extension .png, .ico, or .jpg.
.svn\	Exclude any files in paths named ".svn".
C*\	Exclude any files in paths that start with "C".
/b*/debug//backup/	Exclude all files in this path name.
demo	Exclude any file (not directory) with "demo" in the name.

- **Look in subfolders** - Select this option to expand the search to sub-directories of the folder specified in the **Look in** field.
- **Look in zip/word/excel files** - Select this option to search the files contained in .zip, .jar, .xlsx, and

.docx files. Keep in mind that a .xlsx and .docx file (word or excel file) is not a single file. These are zip files containing many XML files. When searching for text in a word document you probably want to specify "document.xml" (or "*.xml") for **File types**:. When searching for text in a excel document you probably want to specify "sharedStrings.xml" (or "*.xml") for **File types**:. This option only supports recursive subdirectory specifications and not project files which happen to contain some zip files.

- **Advanced Filtering** - Enable and configure advanced file stats filtering options. Additional filtering options include setting a maximum file size for searches and selecting files by last modified time. Use the button next to the option to configure advanced file stats. With Max File Size enabled, allows set the maximum searchable file size in kilobytes. With File Modified Time enabled, configure what files are searchable by last modified time by date (time optional). Options include:
 - **Date Only** - Only search files modified on selected date only.
 - **Before** - Only search files modified before selected date and time.
 - **After** - Only search files modified after selected date and time.
 - **Range** - Only search files modified in selected starting and ending date/time.
 - **Not in Range** - Only search files modified either before selected starting date/time or after date/time.
- **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. There are two filter tabs: Filter Text and Filter Files. With Filter Text, you can search for other text in the search results. 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). With Filter Files, you can additionally filter by filename. The Include and Exclude options allow you to specify with files to include or exclude using ant-like wildcard syntax. Behaves identically to how File Types/Exclude options are used in Find in Files options.
- **Find in Files using Search Results** - Configure Find in Files to search file list of only files found in the current Search Results 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.
- **Generate File List** - Create new file with list of filenames of matching files in the current Search Results tab.
- **Send to References** - Send the current search results to the References tool window.
- **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.

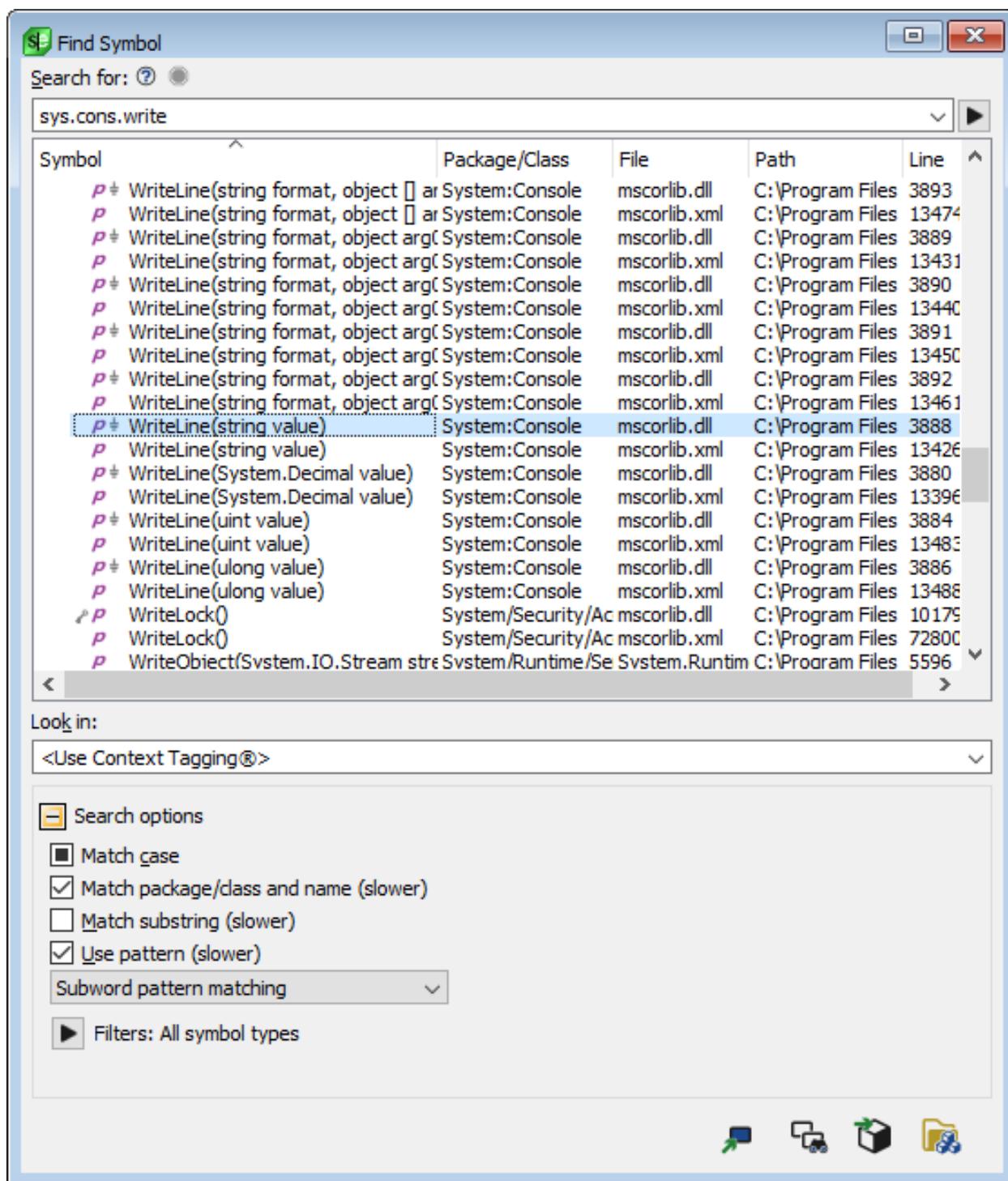
Find Symbol Tool Window (Pro only)

The Find Symbol tool window is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, symbol matching pattern, substring, or fast prefix match. The tool window is displayed when you click **Search → Find Symbol** or **View → Tool Windows → Find Symbol**, or when you use the `gui_push_tag` command.

See [Find Symbol Tool Window](#) under the [Symbol Browsing](#) topic for more information.

Search Dialogs and Tool

Windows



- **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.

Clicking on the red stop button will stop a long-running search. Changing the **Search for:** expression

will restart the search, as will changing any other **Find Symbol** search options. Also, clicking away from the **Find Symbol** tool window will cancel the search.

The last search can be started again by hitting the refresh button.

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, its package/class scope, the file that contains it, and the line number. You can sort by any of the five 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 based on your current workspace and current language mode.
- **<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.
- **<Projects Containing Current File>** - Select this setting to only search in files that are in projects that contain the current file.
- **<Current Workspace>** - Select this setting to only search in files that are in the current workspace.
- **<Current Workspace and Language Tag Files>** - Select this setting to only search in files that are in the current workspace and language-specific tag files for the current file.
- **<Language Tag Files>** - Select this setting to search all language-specific tag files for the indicated file type. 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 package/class and name (slower)** - When selected, instead of just matching the search expression against the symbol name, the search expression is matched against the symbol name prefixed with the name of the class scope that it belongs to. You can use either "." or ":" to represent package/class name separators. Note that this feature is not generally compatible with searching

using regular expression patterns, because the package/class name separators can overlap with special characters needed in regular expressions. Selecting this option causes the search to execute more slowly.

When used in combination with **Match substring** enabled, both the symbol's class name and the symbol's name is expected to participate in the pattern match. When used with **Match substring** disabled, at least one of the symbol name or the package/class name must be a prefix match for that part of the pattern.

- **Match substring (slower than prefix match)** - 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. When not selected, results are limited to symbols whose prefix matches the pattern.
- **Use pattern (slower, enables Match substring)** - When selected, SlickEdit interprets the search string as a regular expression, wildcard expression, or symbol pattern matching expression. This can result in slower search times, since SlickEdit must test every symbol in the tag file against the given expression. See [Regular Expressions](#) for more information about regular expressions. See [Wildcard Expressions](#) for more information about wildcard expressions. See [Subword matching](#) for more information about symbol pattern matching.

When this option is enabled, the **Match substring** search option is turned on for convenience because normally when searching for a symbol matching a pattern, you do not want to restrict the match to a prefix match. However, **Match substring** can be disabled and used in combination with pattern matching to force a prefix match in those special cases.

- **Filters** - Use filters to restrict the search to certain types of symbols. The filters are the same the ones available on the Definitions tool window. See [Defs Tool Window](#) for more information.
- **Buttons** - The following buttons 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 language-specific Context Tagging® options screen (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Context Tagging**): 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. Click **Tools** → **Options** → **Editing** → **Context Tagging** and uncheck the option **Jump to first item when finding 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.

Right-click on a symbol or file in the left pane of the Find Symbols window to display the following options:

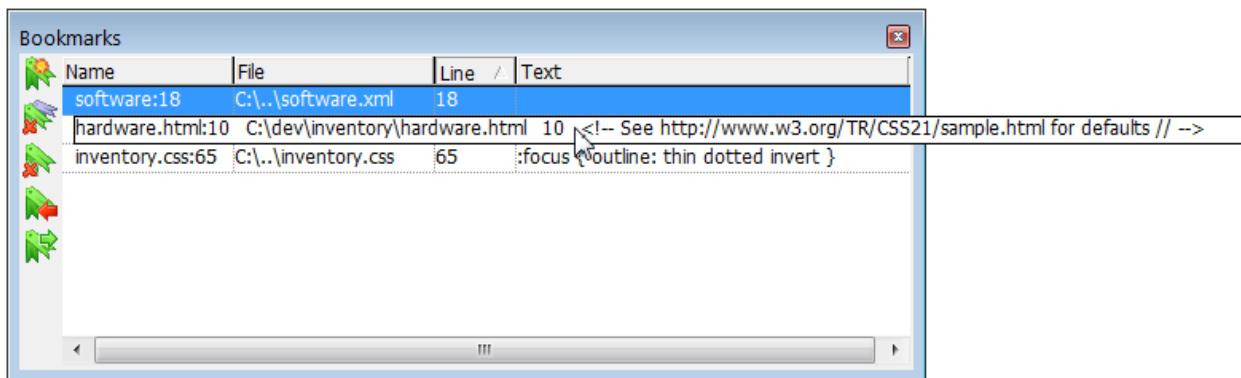
- **Go to reference to ...** - Displays a list of references for the selected symbol in the [References Tool Window](#) and, optionally, navigates to the first reference. See [Code Navigation](#) for more information.
- **Show in symbol browser** - Displays the selected symbol in the [Symbols Tool Window](#). Note that this feature does not work for local variables or symbols from the current file that are not in a tag file.
- **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.
 - **Send to Search Results** - Send the list of symbols found to the Search Results tool window.
 - **Send to References** - Send the list of symbols found to the References tool window.
- **Quick filters, Scope, Functions, Variables, Data Types, Statements, and Others** - All of these items are for filtering the data displayed in the References tool window.

Bookmarks Tool Window

The Bookmarks tool window is used to create and manage [Named Bookmarks](#). To display the tool window, from the main menu, click **Search** → **Bookmarks** → **Bookmarks Tool Window**. You can also press **Ctrl+Shift+N** or use the **activate_bookmarks** command to display the window.

Note

The Bookmarks tool window does not work with [Pushed Bookmarks](#). To view a list of pushed bookmarks, use the [Bookmark Stack Dialog](#).



See [Toolbars and Tool Windows](#) for information about working with tool windows. See [Tabular Lists](#) for information about resizing columns and other layout information.

Tip

- Global named bookmarks are shown in the Bookmarks tool window, unless the option **Use workspace bookmarks** is enabled (**Tools** → **Options** → **Editing** → **Bookmarks**). In this case, only bookmarks for the current workspace are shown. See [Using Workspace Bookmarks](#) for more information.
- If you activate the Bookmarks tool window and start typing the name of a bookmark, SlickEdit attempts to match it. This provides a fast and easy way to jump to a named bookmark. Using the preceding screen shot as an example, you could press **Ctrl+Shift+N** to activate the tool window, type "i" to select the bookmark named "inventory.css:65", then press **Enter** to quickly go to the bookmark's location in your code.

The Bookmarks tool window can be used to perform the following operations:

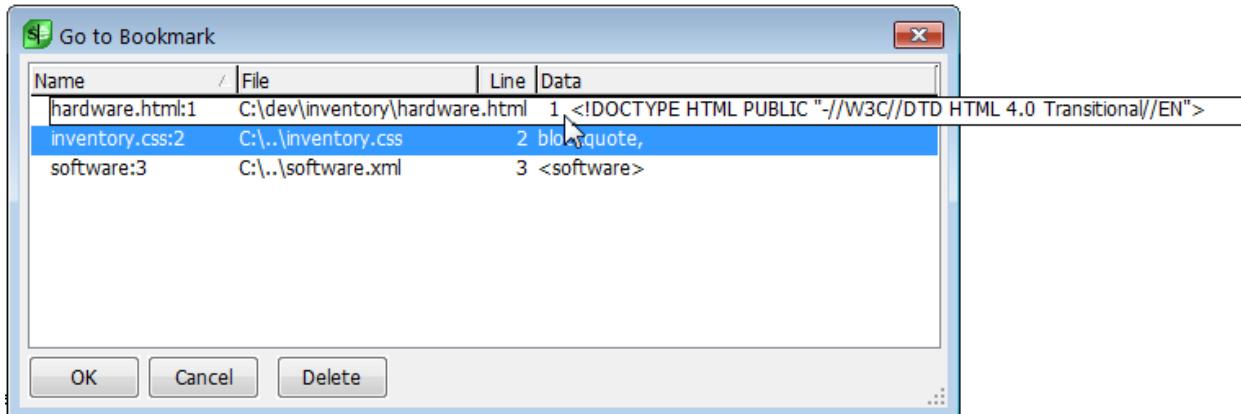
- **Jump to bookmark** - To go to the line in your source code that contains a bookmark, double-click on a bookmark in the tool window. Alternately, select the bookmark to jump to, then select **Go to Bookmark** from the right-click context menu or press the **Enter** key. See [Navigating Named Bookmarks](#) for more navigation methods.
- **Create new bookmark** - To create a new, named bookmark on the current line, click the **Create New Bookmark** button, or select this operation from the window's right-click context menu. You can also press the **Insert** key (with the focus in the tool window). A dialog is displayed where you can specify a name for the bookmark or allow automatic naming (see [Setting a Bookmark With an Automatic Name](#)). Each new bookmark is placed at the top of the list in the Bookmarks tool window.
- **Delete all bookmarks** - To delete all named bookmarks, click the **Delete All Bookmarks** button, or select this operation from the window's right-click context menu. You can also press **Shift+Delete** to delete all bookmarks. A confirmation prompt is displayed prior to deletion. See also [Deleting Named Bookmarks](#).
- **Delete selected bookmark** - This operation is also available as a button on the tool window and on the window's right-click context menu. You can also press the **Delete** key to delete the selected bookmark. See also [Deleting Named Bookmarks](#).
- **Go to previous bookmark** and **Go to next bookmark** - Use these two buttons to navigate to the previous and next bookmark in your source code, respectively. The order of navigation matches the order in which the bookmarks were created, regardless of any selection in the tool window. See [Navigating Named Bookmarks](#) for more information.

Go to Bookmark Dialog

The Go to Bookmark dialog appears automatically when you use the **gb** or **goto_bookmark** command without arguments. It can be used to view, navigate to, and delete [Named Bookmarks](#).

Note

The Go to Bookmark dialog does not work with [Pushed Bookmarks](#). To view a list of pushed bookmarks, use the [Bookmark Stack Dialog](#).



See [Tabular Lists](#) for information about resizing columns and other layout information.

The Go to Bookmark dialog displays the same information as the [Bookmarks Tool Window](#), except it does not provide a way to create bookmarks.

Tip

Global named bookmarks are shown in the Go to Bookmark dialog, unless the option **Use workspace bookmarks** is enabled ([Tools](#) → [Options](#) → [Editing](#) → [Bookmarks](#)). In this case, only bookmarks for the current workspace are shown. See [Using Workspace Bookmarks](#) for more information.

The following operations are available:

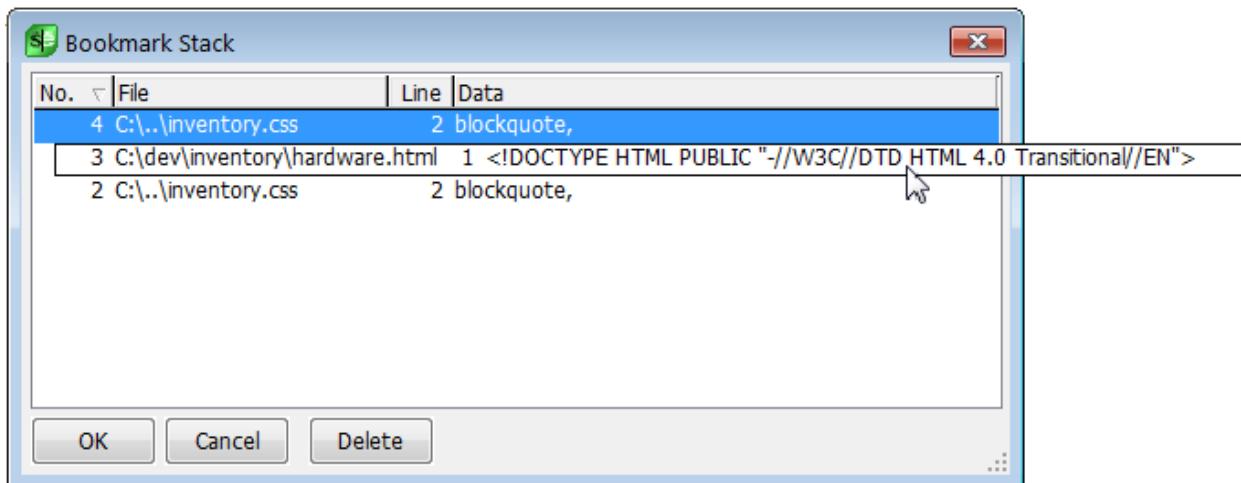
- To go to the line in your source code that contains a bookmark, double-click on the bookmark in the dialog. Or, select the bookmark to jump to and press the **Enter** key or click **OK**. The Go to Bookmark dialog is dismissed after this operation.

Prior to jumping to the bookmark, SlickEdit® automatically creates a pushed bookmark at the current location in your source code, so you can return to the current location easily by popping the bookmark with **Ctrl+Comma**. See [Pushed Bookmarks](#) for more information.

- To delete a bookmark, select it and press the **Delete** key or click **Delete**. See also [Deleting Named Bookmarks](#).

Bookmark Stack Dialog

The Bookmark Stack dialog can be used to view, navigate to, and delete [Pushed Bookmarks](#). To display it, from the main menu, click **Search** → **Bookmarks** → **Bookmark Stack**, or use the **bookmark_stack** command.



See [Tabular Lists](#) for information about resizing columns and other layout information.

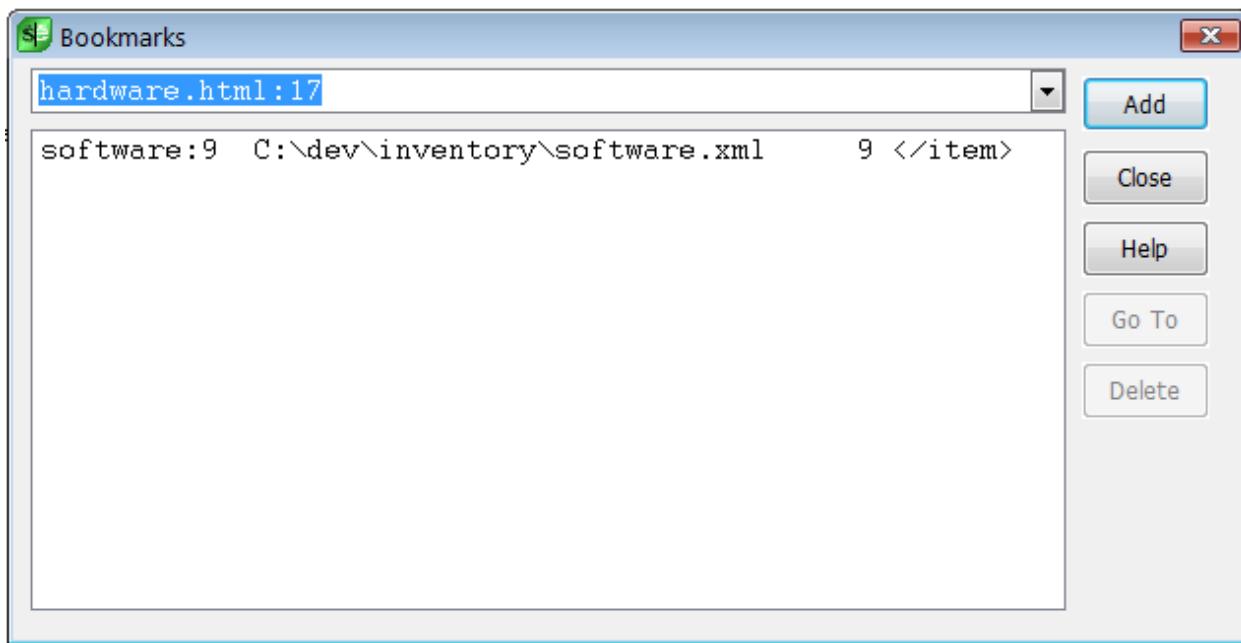
The first column in the Bookmark Stack dialog (labeled **No.**), indicates the numerical order of each bookmark in the stack, according to when the bookmark was pushed. New pushed bookmarks are always placed at the top of the stack, so you can pop them off one by one with **Ctrl+Comma** as you make your way back through the code. See [Pushing and Popping Bookmarks](#) for more information.

Popping bookmarks is the best way to navigate through your pushed bookmarks, deleting them in the process. However, you can also use the Bookmark Stack dialog to navigate to pushed bookmarks or delete a pushed bookmark:

- To go to the line in your source code that contains a pushed bookmark, double-click on the bookmark in the dialog. Or, select the bookmark to jump to and press the **Enter** key or click **OK**. The Bookmark Stack dialog is dismissed after this operation. Prior to jumping to the bookmark, SlickEdit® automatically creates a new pushed bookmark at the current location and places it on top of the stack.
- To delete a pushed bookmark, select it and press the **Delete** key or click **Delete**. The order of the stack is still maintained, indicated by the numbers in the **No.** column.

Bookmarks Dialog

The Bookmarks dialog can be used to set [Named Bookmarks](#). It appears when you click **Search** → **Bookmarks** → **Set Bookmark** from the main menu, or use the **set_bookmark** (or **sb**) command without arguments. See [Setting Named Bookmarks](#) for more information.



Tip

Global named bookmarks are shown in the Bookmarks dialog, unless the option **Use workspace bookmarks** is enabled (**Tools** → **Options** → **Editing** → **Bookmarks**). In this case, only bookmarks for the current workspace are shown. See [Using Workspace Bookmarks](#) for more information.

When the dialog appears, the name field is prepopulated with an automatic name (see [Setting a Bookmark With an Automatic Name](#)). You can use the automatic name, or type over it to name the bookmark yourself. Use the drop-down list to select a previously used name. Click **Add** to set the new bookmark.

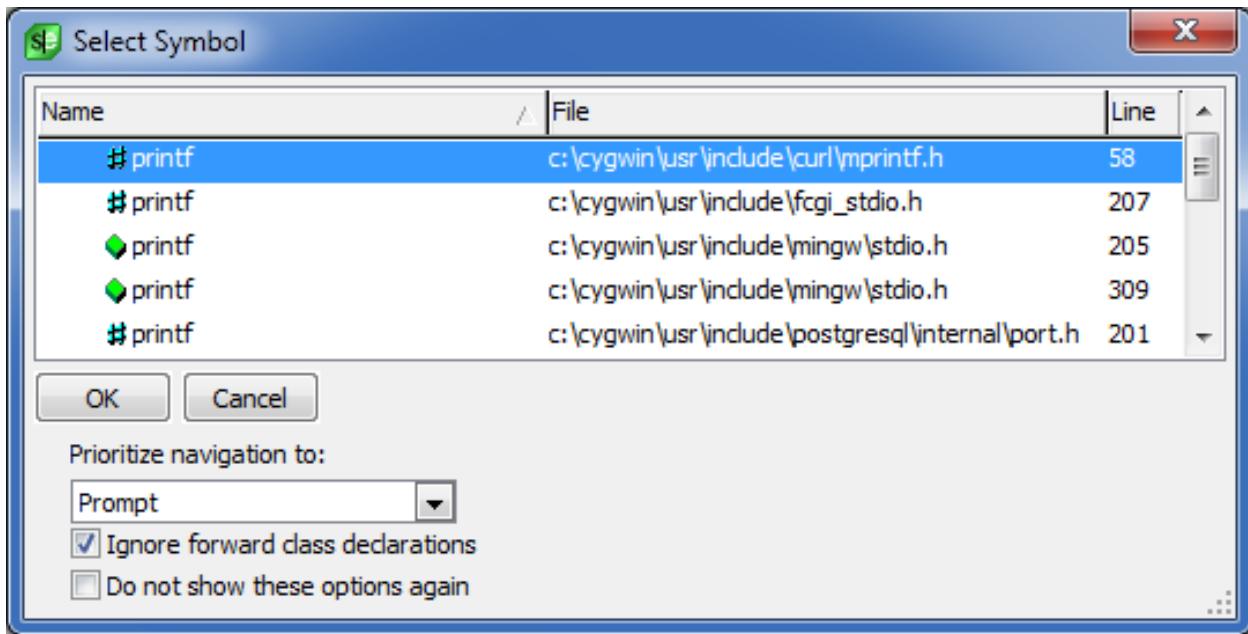
The box below the bookmark name field shows a list of your current named bookmarks. It shows the name, path and file name, line number, and the text of the bookmarked line. Use your keyboard navigation keys to move up and down in the bookmark list. Press **Enter** to jump to the selected bookmark.

The buttons on the Bookmarks dialog are described as follows:

- **Add** - Used to set a new bookmark. The **Add** button changes to a **Replace** button if you select a bookmark in the list. When you click **Replace**, a new bookmark is set with the name of the selected bookmark, so you can have multiple bookmarks with the same name but at different locations in your code.
- **Close** - Cancels the operation and closes the dialog.
- **Go To** - Navigates to the selected bookmark in the source code.
- **Delete** - Deletes the selected bookmark.

Select Symbol Dialog (Pro only)

The Select Symbol dialog is automatically displayed when you use the Go to Definition feature (by pressing **Ctrl+Dot**, clicking **Search → Go to Definition** from the main menu, or using the **push_tag** command), when multiple code locations match the symbol under the cursor. See [Symbol Navigation](#) for more information about this feature.



The dialog displays all tagged instances of the symbol in your project, including definitions and declarations. The name of the symbol and the file path and line number are shown.

To go to a symbol, select it and click **OK** or double-click on it. Like other selection dialogs, you can also start to type the name of the symbol and as you type, SlickEdit auto-selects the matched item in the list.

The options on this dialog set the behavior of Go to Definition going forward. The first three options match the Go to Definition options found at **Tools → Options → Languages → [Language Category] → [Language] → Context Tagging** (see [Language-Specific Context Tagging® Options](#)). When you make a setting on the Select Symbol dialog, the settings on the Options dialog are updated to match.

When the option **Do not show these options again** is selected, the Select Symbol dialog does not show these options when/if it is displayed in the future. This option is automatically checked when you select one of the **Prioritize navigation to...** options. Note that if you have navigation priority set to prompt with both definitions and declarations, then the **Go to Declaration** command, invoked by pressing **Ctrl+Alt+Dot**, will continue to display these options on the Select Symbol dialog. This is intentional because to fully utilize Go to Declaration, it is best if symbol navigation prioritizes definitions or declarations.

To reset the default behavior so this dialog with options appears again when applicable, go to the language-specific Go to Definition options and uncheck both of the **Prioritize navigation to...** options.

The item selected in the Select Symbol dialog will be shown in the **Preview** tool window. If you wish to scroll the previewed text use the following keyboard shortcuts.

- **Ctrl+Down** -- Scroll down one line.
- **Ctrl+Up** -- Scroll up one line.
- **Ctrl+Page Down** -- Scroll down one page.
- **Ctrl+Page Up** -- Scroll down one page.

View

This section describes items related to viewing and displaying within the editor. For more information, see [Viewing and Displaying](#).

View Menu

The **View** menu contains options that pertain to viewing and displaying special characters, code, and comments. It also allows you to control the visibility of tool windows and toolbars.

View Menu Item	Description	Command
Hex	Toggles hex/ASCII display. See Hex Mode Editing .	<code>hex</code>
Line Hex	Toggles line hex/ASCII display. See Hex Mode Editing .	<code>linehex</code>
Special Chars	Toggles viewing of tabs, spaces, and new line character(s) on/off. See Viewing Special Characters .	<code>view_specialchars_toggle</code>
New Line Chars	Toggles viewing of new line character(s) on/off. See Viewing Special Characters .	<code>view_nlchars_toggle</code>
Tab Chars	Toggles viewing of tab character(s) on/off. See Viewing Special Characters .	<code>view_tabs_toggle</code>
Spaces	Toggles viewing of space character(s) on/off. See Viewing Special Characters .	<code>view_spaces_toggle</code>
Other Ctrl Characters	Toggles viewing of control character(s) on/off. See Viewing Special Characters .	<code>view_other_ctrl_chars_toggle</code>
Line Numbers	Toggles the display of line numbers on/off for the current document. See Viewing Line Numbers .	<code>view_line_numbers_toggle</code>
Soft Wrap	Toggles wrapping of long lines to window width.	<code>softwrap_toggle</code>

View Menu

View Menu Item	Description	Command
Symbol Coloring	(Pro only) Displays Symbol Coloring menu. See Symbol Coloring Menu .	N/A
Language View Options	Displays View options for the language in the current buffer. This is the same as if you had selected Tools → Options → Languages → Application Languages → C/C++ → View from the main menu. See Language-Specific View Options .	setupext -view
Toolbars	Show, hide, or customize a toolbar . See Customizing Toolbars .	toolbars
Tool Windows	Show, hide, or customize a tool window. See Customizing Tool Windows .	customize_tool_windows
Fullscreen	Toggles full screen editing mode. See Full Screen Mode .	fullscreen
Selective Display	Displays the Selective Display dialog, which allows you to hide lines and create an outline. See Selective Display .	selective_display
Hide All Comments	Hides all lines that only contain a comment.	hide_all_comments
Hide Code Block	Hides lines inside current code block. See Expanding/Collapsing Code Blocks .	hide_code_block
Hide Selection	Hides selected lines.	hide_selection
Hide #region Blocks	Hides .NET #region blocks.	hide_dotnet_regions
Function Headings	Collapses all function code blocks in the current file. See Selective Display .	show_procs

View Menu Item	Description	Command
Expand/Collapse Block	Toggles between hiding and showing the code block under the cursor. See Expanding/Collapsing Code Blocks .	<code>plusminus</code>
Copy Visible	Copies text not hidden by Selective Display. See Selective Display .	<code>copy_selective_display</code>
Show All	Ends selective display. All lines are displayed and outline bitmaps are removed. See Selective Display .	<code>show_all</code>

Symbol Coloring Menu (Pro only)

The table, below, describes each item on the **View → Symbol Coloring** menu and its corresponding command. For more information see [Symbol Coloring](#).

Symbol Coloring Menu Item	Description	Command
Customize	Opens the Symbol Coloring options screen.	<code>config Symbol Coloring</code>
Enable Symbol Coloring	Turns Symbol Coloring on/off for this file.	<code>symbol_coloring_toggle</code>
Highlight Unidentified Symbols	Turns on/off highlighting of symbols that are unidentified by the Symbol Coloring engine.	<code>symbol_coloring_errors_toggle</code>
Override Library Symbols	Turns on/off highlighting of symbols that are configured in color coding to use Library Symbol or User Defined Keyword color.	<code>symbol_coloring_overrides_toggle</code>
All symbols - Default	Selects the default, All symbols profile.	<code>symbol_coloring_set_scheme All symbols - Default</code>
Global Variables	Selects the Global Variables profile.	<code>symbol_coloring_set_scheme Global Variables</code>

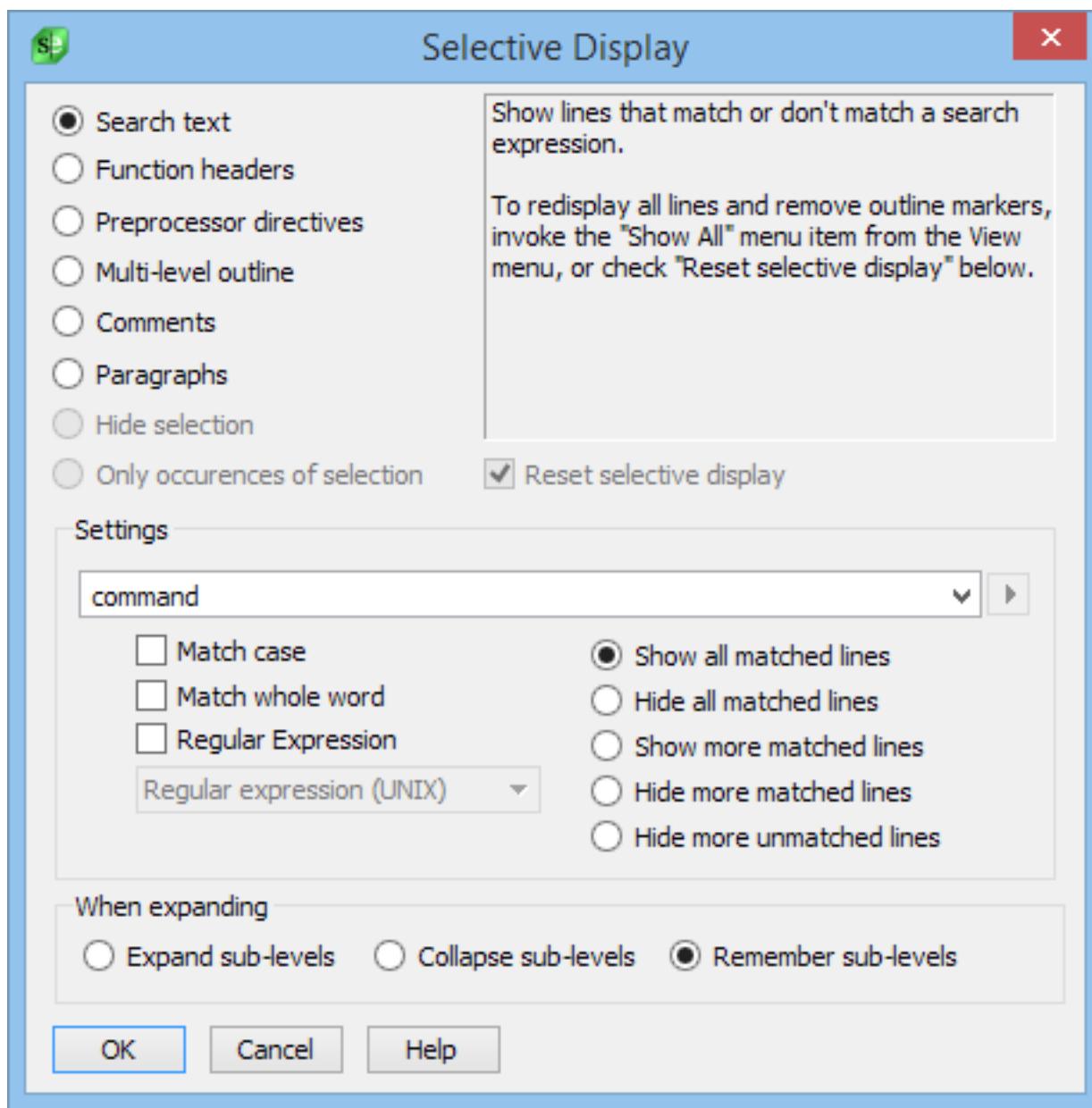
Symbol Coloring Menu Item	Description	Command
Protected and Private	Selects the Protected and Private profile.	<code>symbol_coloring_set_scheme Protected and Private</code>
Unidentified Symbols Only	Selects the Unidentified Symbols Only profile.	<code>symbol_coloring_set_scheme Unidentified Symbols Only</code>

View 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 activate Selective Display and choose the regions in your code that you want to display or hide. 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 using the **Word chars** field on the language-specific **General** options screen (see [Language-Specific](#))

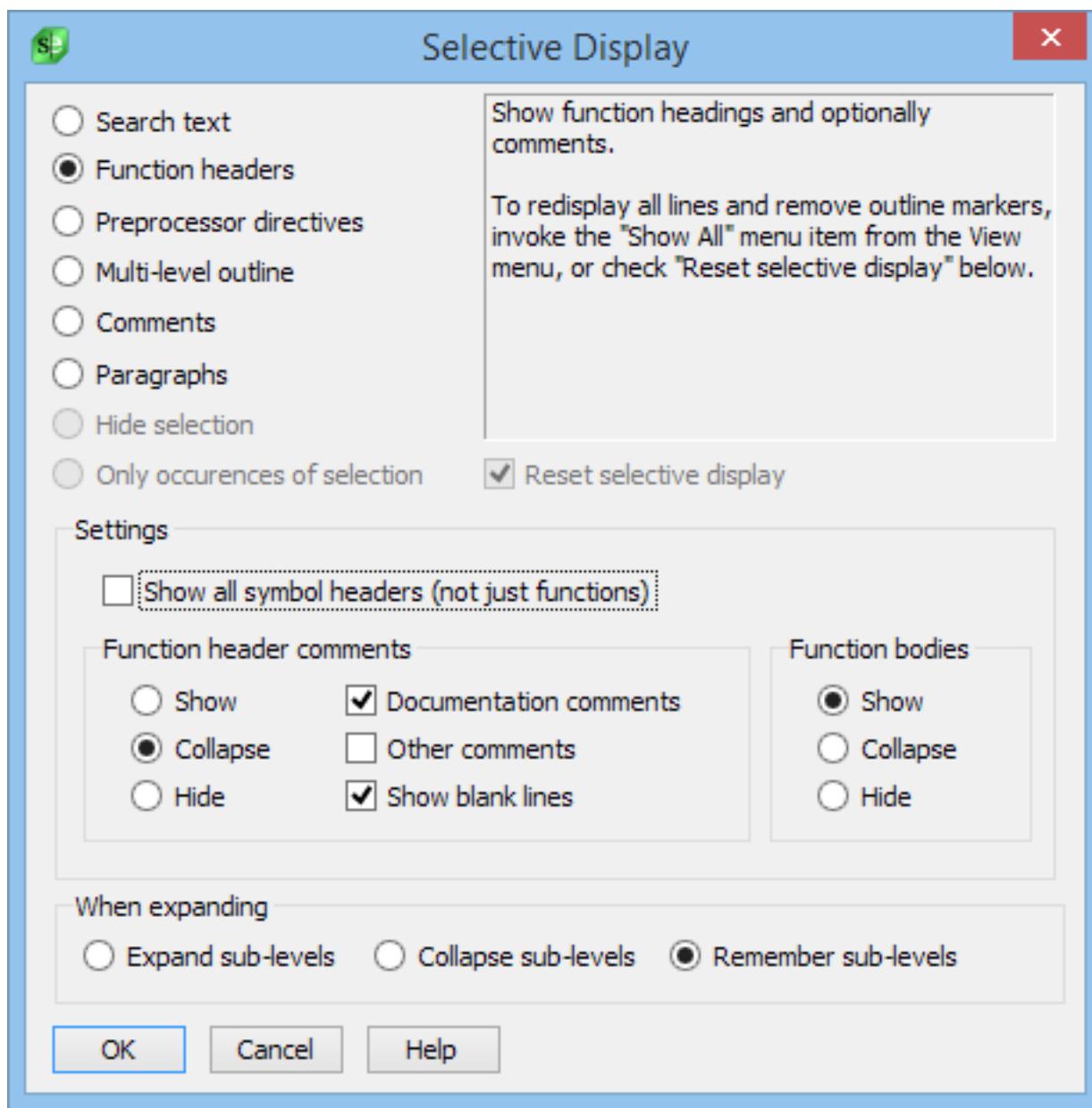
[General Options](#)).

- **Regular expression** - When checked, a regular expression search is performed. See [Find and Replace with Regular Expressions](#) for more information.
- **Show all matched lines** - When checked, all lines are made visible and **Plus** and/or **Minus** bitmaps are removed before the search is performed, then all lines that do not contain the search expression are hidden.
- **Hide all matched lines** - When checked, all lines are made visible and **Plus** and/or **Minus** bitmaps are removed before the search is performed, then all lines that contain the search expression are hidden.
- **Show more matched lines** - When selected, all hidden lines that contain the search expression are un-hidden. Any lines that were previously shown, remain shown.
- **Hide more matched lines** - When selected, all lines that contain the search expression are hidden. Any lines that were previously hidden, remain hidden.
- **Hide more unmatched lines** - When selected, all lines that do not contain the search expression are hidden. Any lines that were previously hidden, remain hidden.

Function Headers

Select **Function headers** to display only function headings and optional function heading comments. Check **Show all symbol headers (not just functions)** to display heading for all symbols in the file, not just functions.

The following settings affect how comments before function headings are handled:



- **Show** - When checked, comments above function headings are displayed as if they were part of the function heading.
- **Collapse** - When checked, comments above function headings are visible but multi-line comments will require that you expand them to see the entire comment.
- **Hide** - When checked, comments above function headings are not visible.

Note: using the **Hide** setting, selected comments will not be visible at all, making it difficult to copy or move functions and comments.

- **Documentation comments** - When checked, apply the **Show**, **Collapse**, or **Hide** option to documentation comments, such as JavaDoc, XMLDoc, or Doxygen formatted comments.

- **Other comments** - When checked, apply the **Show**, **Collapse**, or **Hide** option to all other comments, excluding documentation comments.
- **Show blank lines** - When checked, blank lines between function headings will not be hidden.

The following settings affect how function bodies are handled:

- **Show** - When checked, function bodies are displayed as an already expanded selective display region, as if they were part of the function heading.
- **Collapse** - When checked, function bodies are displayed as a collapsed selective display region.
- **Hide** - When checked, function bodies are not visible. This is the default behavior.

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]
```

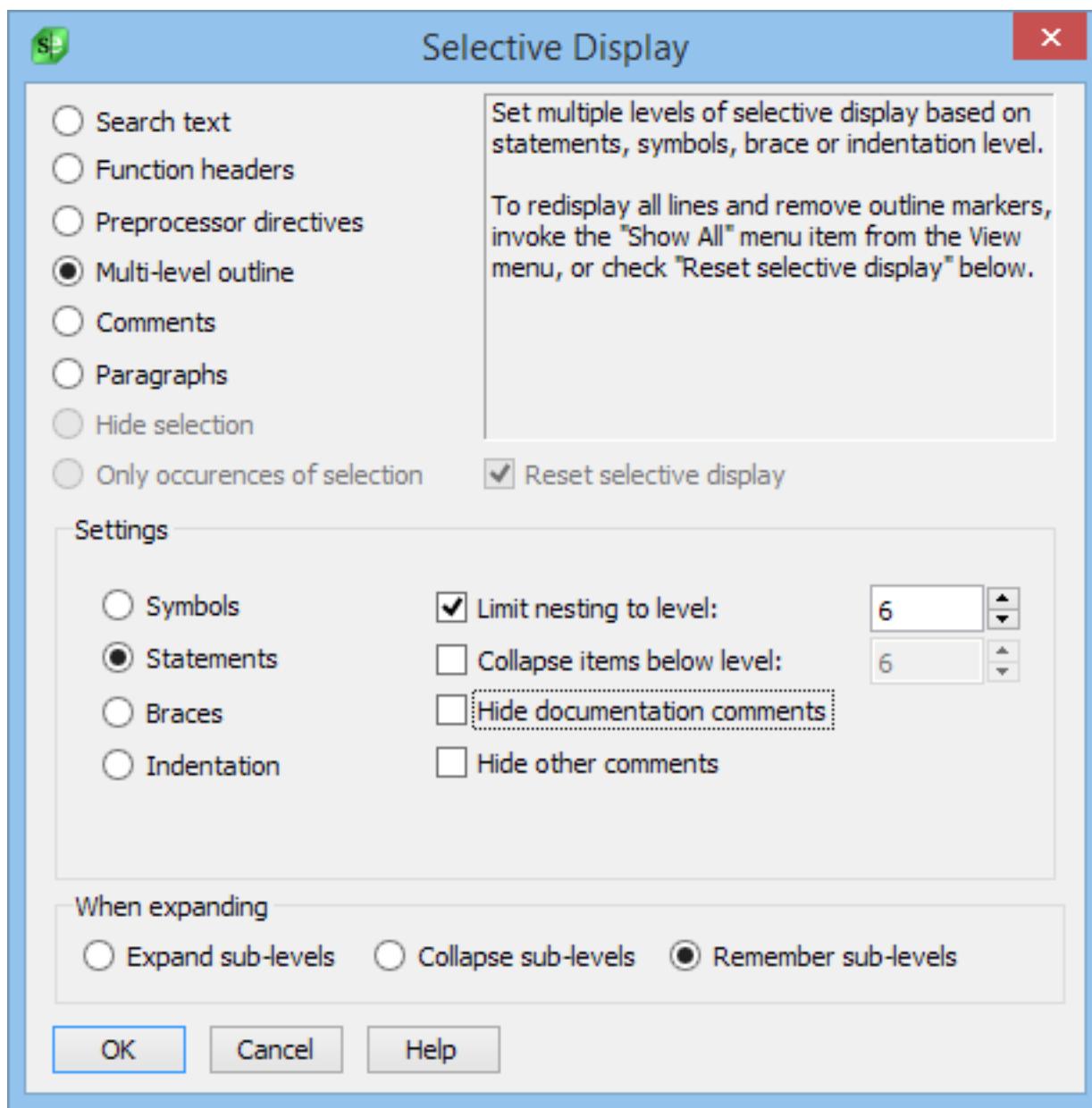
For example:

```
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 outline** to set multiple levels of selective display based on braces or indent. The following settings are available:



- **Symbols** - When on, multiple levels of selective display are set to correspond to all symbol definitions and declarations found in the file. This option is disabled for languages which do not support tagging.
- **Statements** - When on, multiple levels of selective display are set to correspond to all symbol definitions, declarations, and statements in the file. This option is disabled for languages which do not support statement level tagging.
- **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 line indentation levels.
- **Limit nesting to level:** - When too many nested levels of selective display get confusing, place a limit

on the maximum number of nested levels. Nesting deeper than this specified level is ignored.

- **Collapse items below level:** - Specifies that items deeper than the specified level of nesting should be collapsed. All items at higher levels will remain expanded. This is useful, for example, in Java, to specify that classes (level 1) should be expanded, and methods (level 2) should be collapsed.
- **Hide documentation comments:** - In addition to the multi-level outlining, collapse documentation comments, such as JavaDoc, XMLDoc, or Doxygen formatted comments.
- **Hide other comments:** - In addition to the multi-level outlining, hide all other comments (except for documentation comments).

Comments

Select **Comments** to collapse multi-line comment blocks to just display the first line of each comment. The following settings are available:

- **Hide documentation comments** - When on, documentation comments, such as JavaDoc, XMLDoc, or Doxygen formatted comments will be collapsed to display only the first line of the comment.
- **Hide other comments** - When on, all other comments, excluding JavaDoc, XMLDoc, or Doxygen formatted comments will be collapsed to display only the first line of the comment.

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.

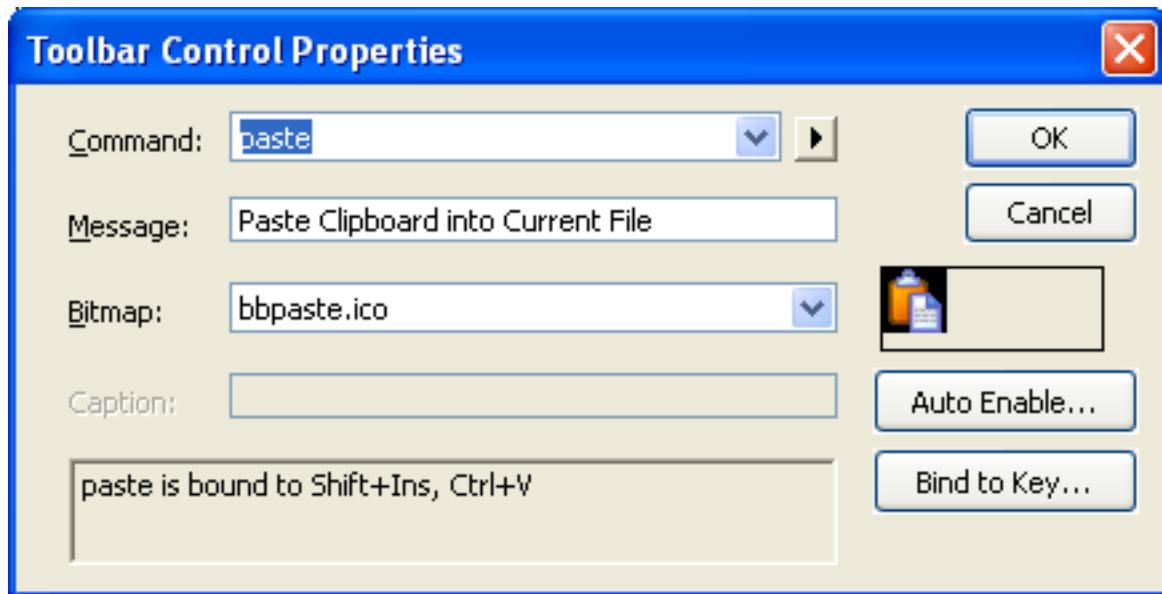
Expansion Options

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 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 tool tip 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 Binding Options](#) 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.	<code>workspace_new</code>
Open Workspace	Opens a workspace.	<code>workspace_open</code>
Open Other Workspace	Displays menu for open projects, workspaces, or makefiles from other tools. See Open Other Workspace Menu .	N/A
Close Workspace	Closes the current workspace.	<code>workspace_close</code>
Organize All Workspaces	Allows you to organize your workspaces which appear in the All Workspaces menu. See Organizing Workspaces .	<code>workspace_organize</code>
Workspace Properties	Displays the Workspace Properties dialog, which allows you to add/remove projects from the current workspace. See Project Dialogs and Tool Windows .	<code>workspace_properties</code>
Retag Workspace	(Pro only) Updates the tag file for the current workspace.	<code>workspace_retag</code>
Retag Project	(Pro only) Updates the tag file for the current project.	<code>project_retag</code>
Refreshes current workspace, project files, and tag files(Pro only)		<code>workspace_refresh</code>

Project Menu

Project Menu Item	Description	Command
Add New Item from Template	Adds new template file to the existing project.	<code>project_add_item</code>
Open Files from Project	Allows you to open files from the current project. See Document Dialogs and Tool Windows .	<code>project_load -p</code>
Open Files from Workspace	Allows you to open files from the current workspace. See Document Dialogs and Tool Windows .	<code>project_load</code>
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.	<code>workspace_insert</code>
Dependencies	(Pro only) Displays the Project Properties dialog open to the Dependencies tab, which lets you set the dependencies for the active project. See Dependencies Tab .	<code>workspace_dependencies</code>
Set Active Project	Allows you to set the active project	<code>projecttbSetCurProject</code>
Project Properties	Displays the Project Properties dialog, which is used to edit settings for the current project. See Project Properties Dialog	<code>project_edit</code>
All Workspaces	Displays menu of recently opened workspaces and (optionally) recent active projects. The number of items displayed is configurable. See History Options . The most recently used workspaces are displayed under the Project menu. The items under the All Workspaces submenu are sorted by workspace name, directory, and	<code>project_edit</code>

Project Menu Item	Description	Command
	project name.	

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.

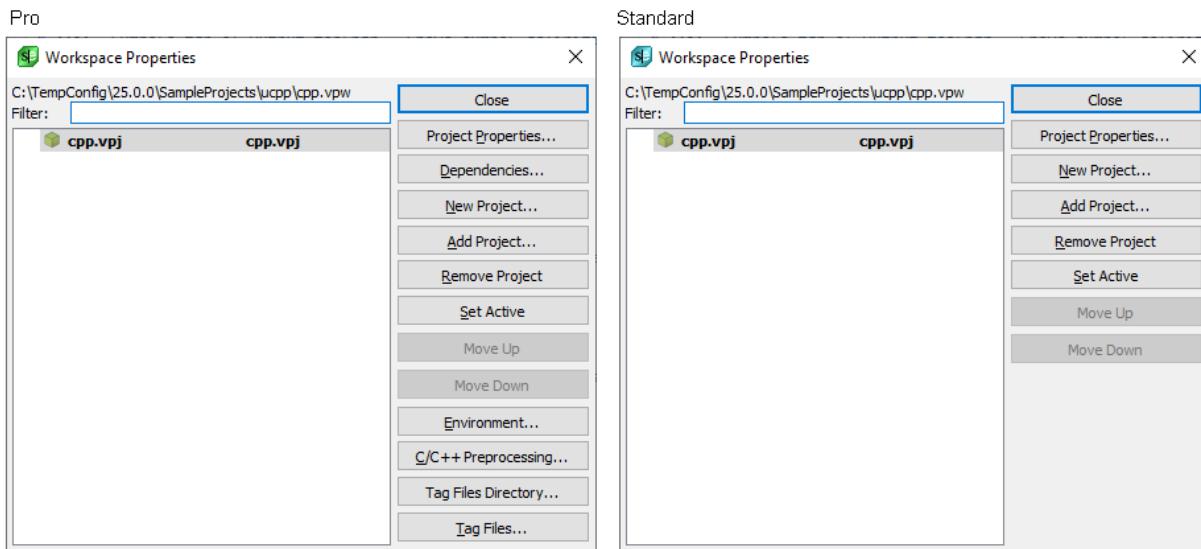
Open Other Workspace Menu Item	Description	Command
Visual Studio .NET Solution	Open a Visual Studio .NET Solution.	workspace_open_visualstudio
Visual C++ Workspace	Open a Visual C++ Workspace.	workspace_open_visualcpp
Visual C++ Embedded Workspace	Open a Visual C++ Embedded Workspace.	workspace_open_visualcppembed
Tornado Workspace	Open a Tornado Workspace.	workspace_open_tornado
Ant XML Build File	Open an Ant XML Build File.	workspace_open_ant
Maven Project File	Open a Maven project file.	workspace_open_maven
Makefile	Open a Makefile. See Open Makefile as Workspace Dialog .	workspace_open_makefile
NAnt .build file	Open a NAnt .build file.	workspace_open_nant
JBuilder Project	Open a JBuilder® Project.	workspace_open_jbuilder
Xcode Project	Open a Xcode Project.	workspace_open_xcode
Flash Project	Open a Flash Project.	workspace_open_flash
Workspace from CVS	Checkout and open a workspace from CVS.	cvs_open_workspace
Convert CodeWright Workspace	Convert a CodeWright workspace and projects to SlickEdit workspace and projects.	cwprojconv.e

Project 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 clicking **Project → Workspace Properties**.

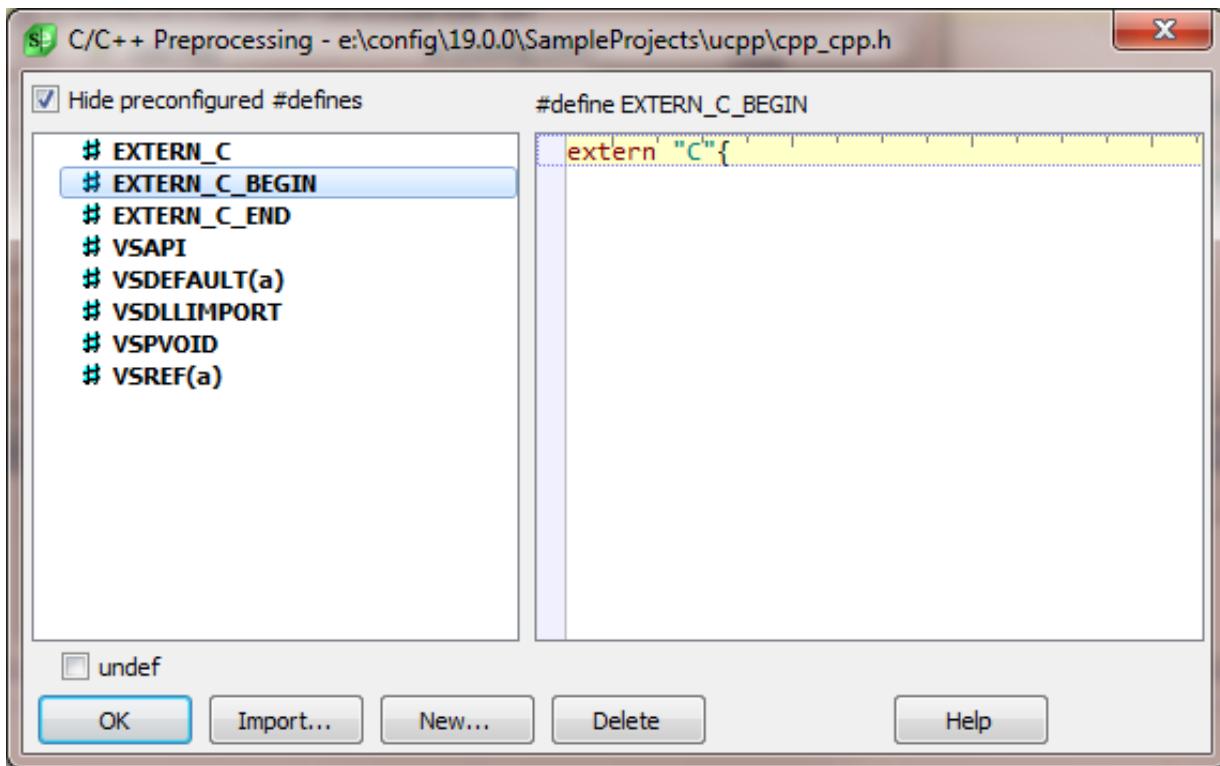


The following options are available:

- **Project Properties** - Displays project properties for the selected project.
- **Dependencies** - (Pro only) Displays project dependencies for the selected project.
- **New Project** - Allows you to add a new project to the current workspace.
- **Add Project** - Allows you to add an existing project to the current workspace.
- **Remove Project** - Removes the selected project from the workspace.
- **Set Active** - Sets the selected project active.
- **Move Up** - Moves the selected project up. By default, this button is always disabled because project files are sorted. Uncheck "Sort Projects" from the Projects tool window workspace context menu.
- **Move Down** - Moves the selected project down. By default, this button is always disabled because project files are sorted. Uncheck "Sort Projects" from the Projects tool window workspace context menu.
- **Environment** - (Pro only) Displays the Workspace Environment Options dialog box, allowing you to set environment variables. For more information on setting environment variables, see [Environment Variables](#).

Project Dialogs and Tool

Windows



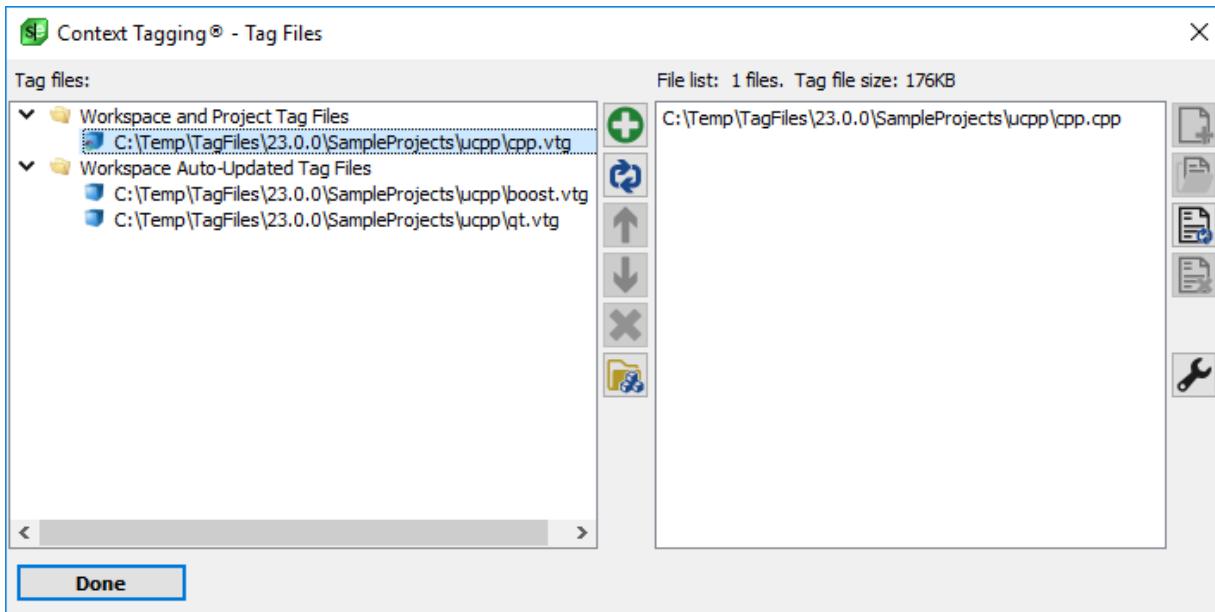
- **C/C++ Preprocessing** - (Pro only) Displays the Workspace C/C++ Preprocessing options dialog box, allowing you to define preprocessing symbols specific to the corresponding workspace. For more information about the C/C++ Preprocessing dialog, see [C/C++ Preprocessing](#).
- **Tag Files Directory** - (Pro only) Displays a directory chooser dialog where you can select a directory for workspace tag files (and project tag files) to be placed. This option is useful when you have workspaces that are on network drives or if you have a high-speed drive that you prefer to store your workspace and project tag files on for performance.

This option is also useful to avoid cluttering your workspace directory with tag files (in the case where you have several project-specific tag files or auto-updated tag files in your workspace).

In addition, this option can be useful in order to avoid conflicts with other users when working with a workspace that is in a shared directory. By setting the workspace tagging directory to a location under your home directory or your SlickEdit configuration directory using an environment variable such as **%HOME** or **%(SLICKEDITCONFIG)**, you can insure that each user has a private copy of all the workspace tag files and the workspace history file.

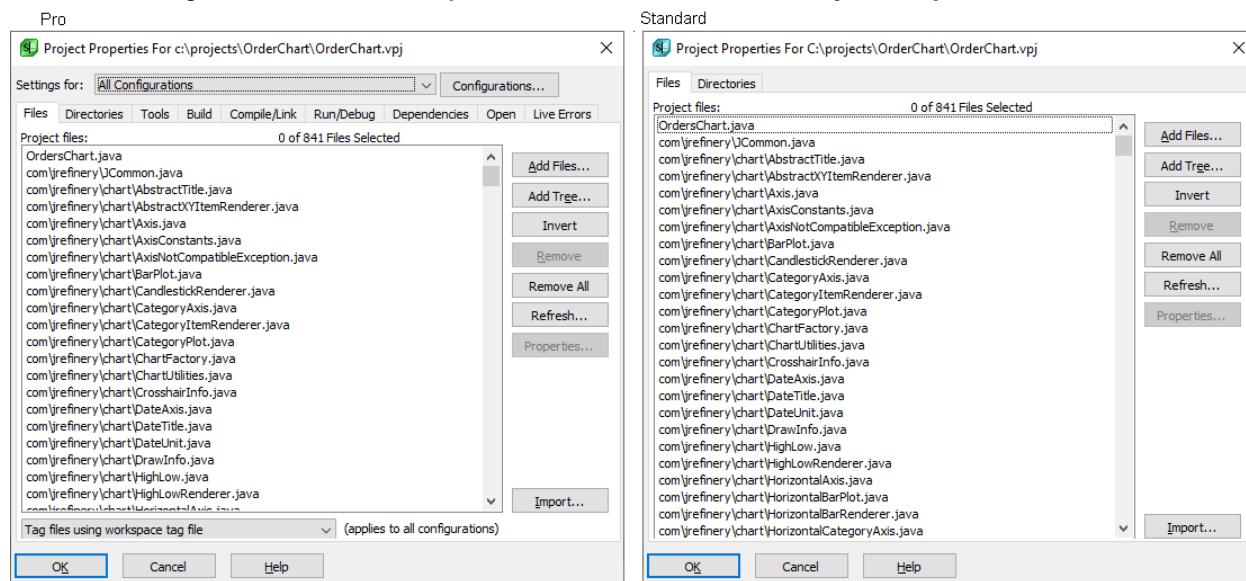
- **Tag Files** - (Pro only) Displays the Context Tagging® - Tag Files dialog, shown below, to manage all your workspace and project-specific tag files. For more information, see [Context Tagging - Tag Files Dialog](#). For more information about tag files, see [Building and Managing Tag Files](#).

Project Dialogs and Tool Windows



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, click **Project → Project Properties** or use the **project_edit** command. You can also right-click within the Projects tool window and select **Project Properties**.



Note

By default SlickEdit displays the **Project Properties** dialog with **All Configurations** selected. To invoke the **Project Properties** dialog with the current active configuration, use **Project Properties for Config** found on the **Build** menu.

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](#)
- [Live Errors Tab](#)

Project Properties Dialog - General Options

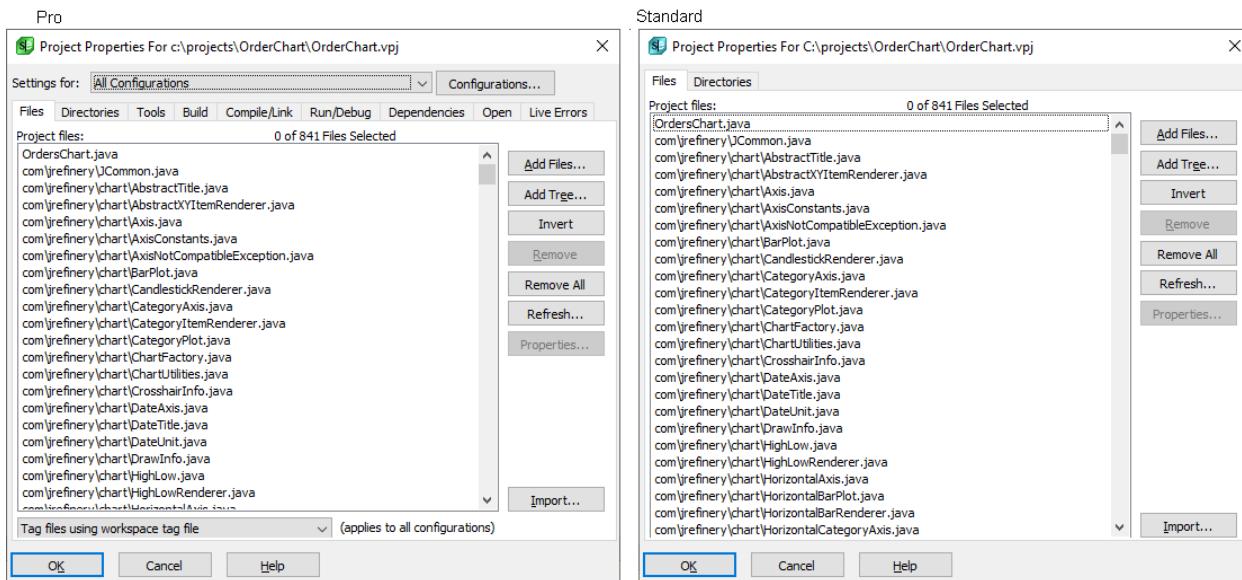
The following options are available at the top of the dialog:

- **Settings for** - (Pro only) 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** - (Pro only) 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.

Project Dialogs and Tool Windows



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 and searches through a directory tree adding the files that are found to the project. To search directories recursively, select the **Recursive** option. You can also specify to add the pattern as a wildcard, meaning that the tree will be repeatedly traversed to pick up any new files that were added. For more information, see [Add Tree Dialog](#).

Note

(Close the any modal dialog like the Project Properties dialog first) You can drag/drop a directory from your operating system file explorer onto SlickEdit to perform an Add Tree to the active project. If no workspace/project is open, you will be prompted to create one.

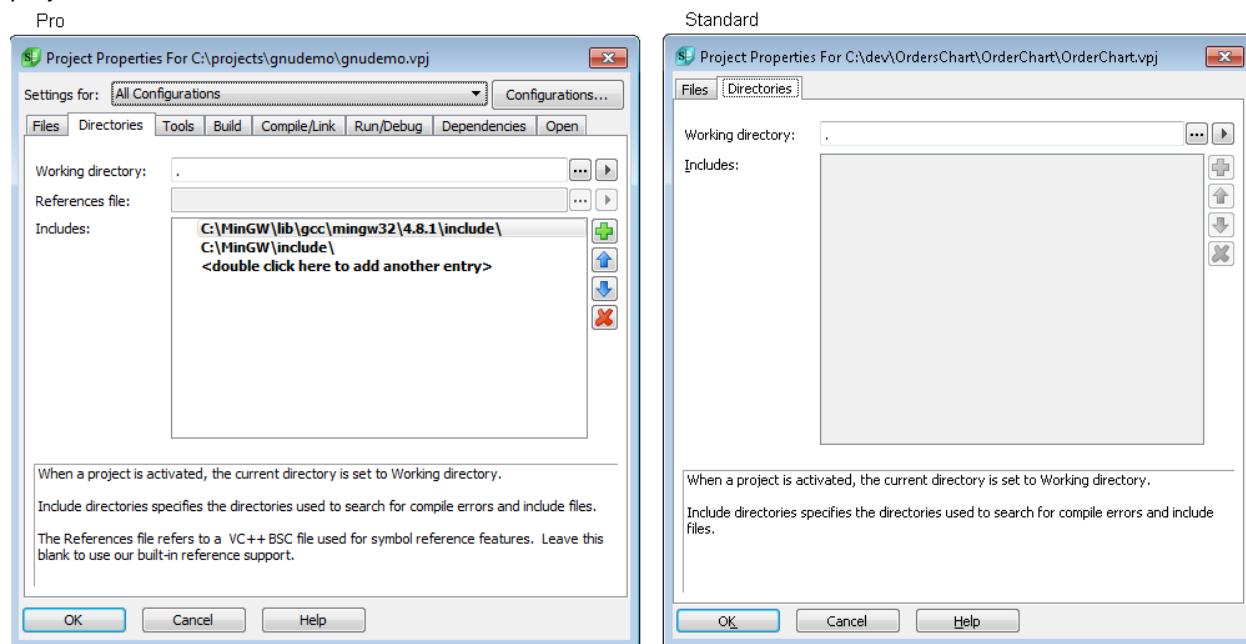
- **Invert** - Inverts the selected items in the **Project files** list box.
- **Remove** - Removes selected items from the project.
- **Remove All** - Removes all files from the project.
- **Refresh** - Provides an easy way to remove files that do not exist from the project.
- **Properties** - Only available when the current file has wildcard characters. Displays various supported options for a wildcard specification.
- **Import** - Allows you to specify a file which contains files or directories to be added to the current project. For more information, see [Importing Files](#).

(Pro only) To specify how the files in the project are to be tagged, the combo box below the file list has the following options:

- **Tag files using workspace tag file** - Specifies that the files should be part of the workspace tag file. For more information, see [Tag File Categories](#).
- **Tag files with project-specific tag file** - Specifies that this project should have its own dedicated tag file. For more information, see [Tag File Categories](#).
- **Tag files with project-specific tag file, without references** - Specifies that this project should have its own dedicated tag file. References are not generated for this tag file, so types and functions from the project will be visible, but reference searches will not include other symbols from the project when doing a references search. This is useful for tagging libraries you depend on where you are mainly interested in the API for the library, and not its implementation details. For more information, see [Tag File Categories](#).
- **Do not tag files** - specifies that the files should not be tagged at all. This is useful if you have a project which contains XML or other data files, test programs, or other non-essential files that you do not need to have tagged. Using this option can help reduce the time required to build (and access) the workspace tag file, as well as avoid cluttering the workspace with unwanted symbols.

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** - (Pro only) 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.

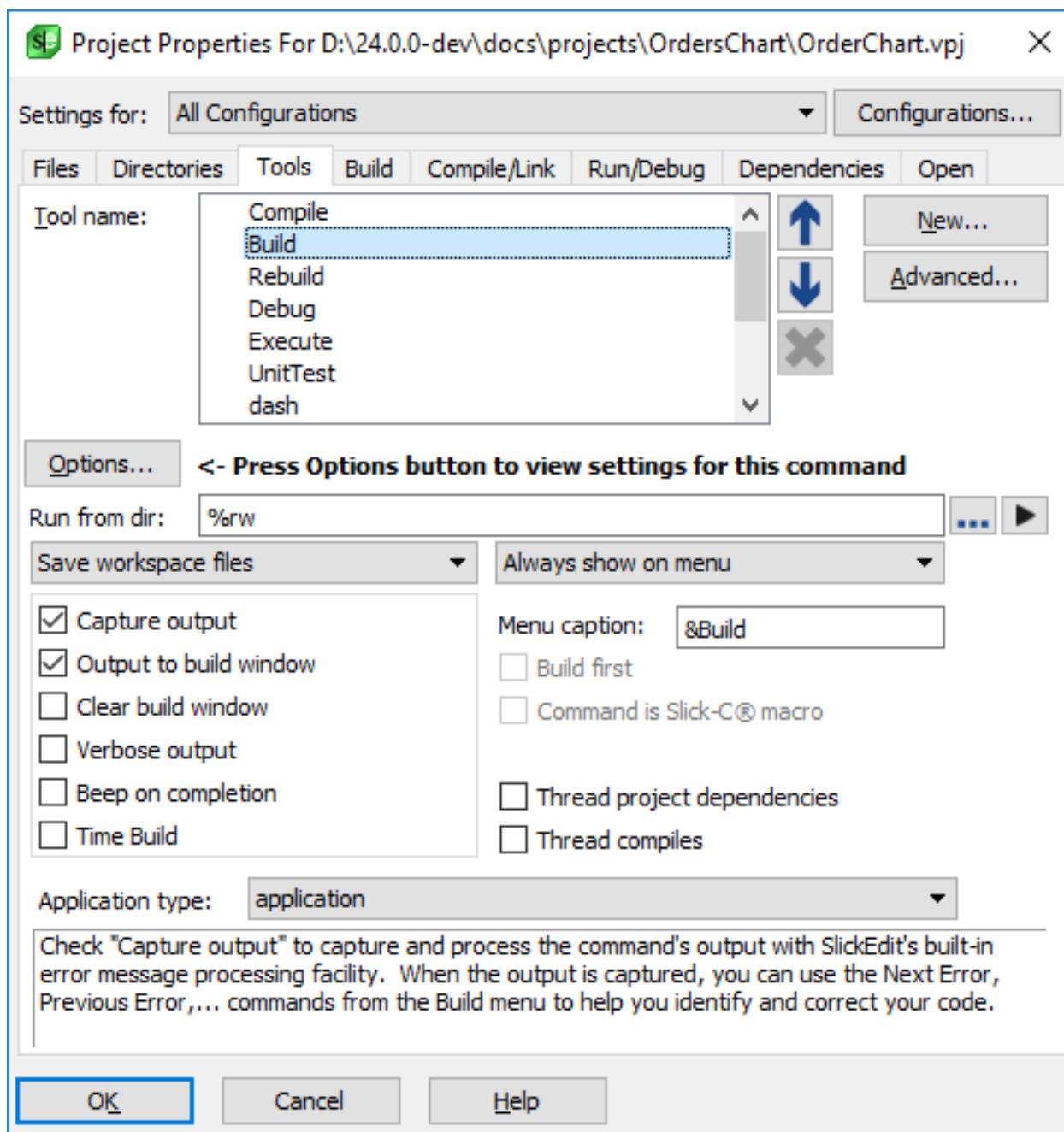
- **Includes** - Specifies the directories the **cursor_error** (**Alt+1**) and **next_error** (**Ctrl+Shift+Down**) commands will search when trying to open a file. For COBOL and High Level Assembler, this list of directories is used to find copy books or macros. You might want to add some of your own 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 (Pro only)

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

Project Dialogs and Tool

Windows



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** button (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. This will launch the [New Project Tool Wizard](#).

- **Advanced** - Click the **Advanced** button to change environment variables (see [Environment Variables](#)).
- **Options** - Displays an Options dialog box specific to the language with which you are currently working. This button is only available for selected tools that support the language-specific options. For more information, see [Language Options](#).
- **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. See [Escape Sequences for Build Commands](#) for a full list of available escape sequences.
- **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.
 - For Windows 2000, 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.
- **Verbose output** - Specifies that the **vsbuild** utility is used to build the projects (see [Using Build and Compile Operations](#)). Detailed information about the commands that are executed will be output during the build.

- **Time build** - Specifies that the **vsbuild** utility is used to build the projects (see [Using Build and Compile Operations](#)). The time required to perform the build operation will be reported after the build is complete.
- **Save combo box** - Specifies whether to save any files before running the current tool. Choose from the following options:
 - **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.
- **Show combo box** - This setting determines when your command is shown on the **Build** menu. The following options are available:
 - **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 Build and Compile Operations](#)) 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.

Note

- **Windows:** 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**).

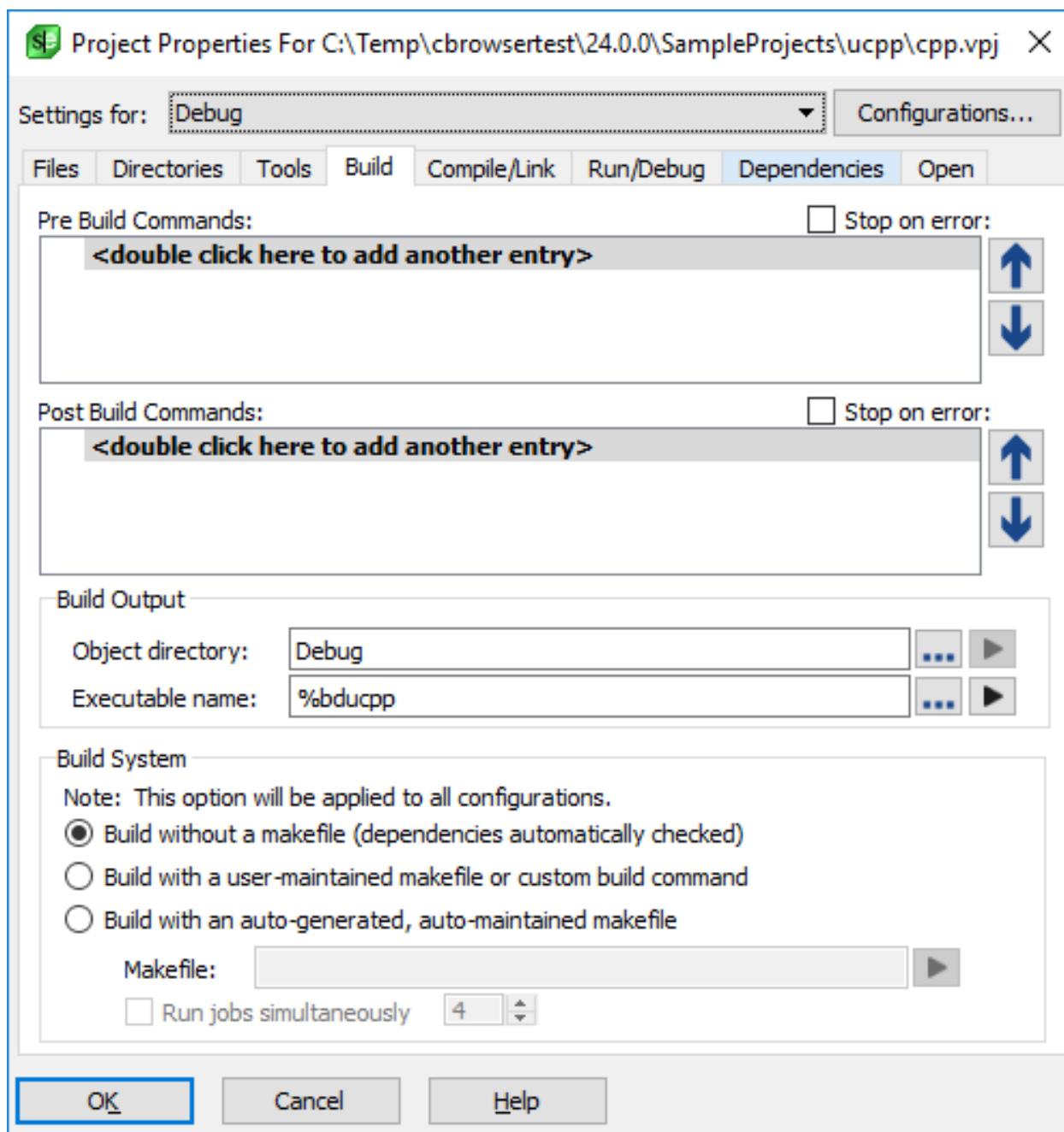
- **Command is Slick-C macro** - Specifies that the command line is a Slick-C® macro as opposed to an external program.
- **Beep on completion** - Specifies that the **vsbuild** utility is used to build the projects and to sound a status beep upon completion (see [Using Build and Compile Operations](#)). 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.
- **Thread project dependencies** - Enabled only if the **vsbuild** utility is used to build the project (see [Using Build and Compile Operations](#)). Specifies that **vsbuild** should build dependent projects using multiple threads.
- **Thread compiles** - Enabled only if the **vsbuild** utility is used to build the project (see [Using Build and Compile Operations](#)). Specifies that **vsbuild** should use multiple threads to invoke commands to compile source files.

Build Tab (Pro only)

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 unavailable for extension-based projects.

Project Dialogs and Tool

Windows



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 Build and Compile Operations](#)) 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) **vsbuild** cannot detect error codes returned from a batch program.

- **Build Output Options** - These options allow you to configure where object files are placed by the build, as well as the name of the output file being created.
 - **Object directory** - This is the directory where object files and the executable are placed by the build process. When building using a user-defined command or a custom makefile, set this to the directory where you expect the build process to place the object files and executable. When building using SlickEdit's build system or an auto-generated Makefile, this is the directory where object files will be placed. This setting is referenced elsewhere in the Project Properties dialog using the **%bd** project escape sequence.

If the **Object directory** is not specified, the default is to use a subdirectory matching the configuration name under the project directory.

- **Executable name** - This is the name of the item created by this build process. For a program, this is typically the name of the executable. For a library, this is the name of the library, including the library suffix (for example, .dll, .so, .a, .lib, .dylib).

This setting can be either an absolute path or just a file name or path relative to the **Object directory**. This setting is referenced elsewhere in the Project Properties dialog using the **%o** (output file name) project escape sequence, as well as the related escape sequences **%on** (output file name only), **%oe** (output file extension), and **%op** (output file path).

- **Build System Options** - These build methods apply to 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 Build and Compile Operations](#)) 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 Build and Compile Operations](#)). Make sure the

project include directories are set up correctly (**Project → Project Properties**, select the [Directories Tab](#)) so include files may be found.

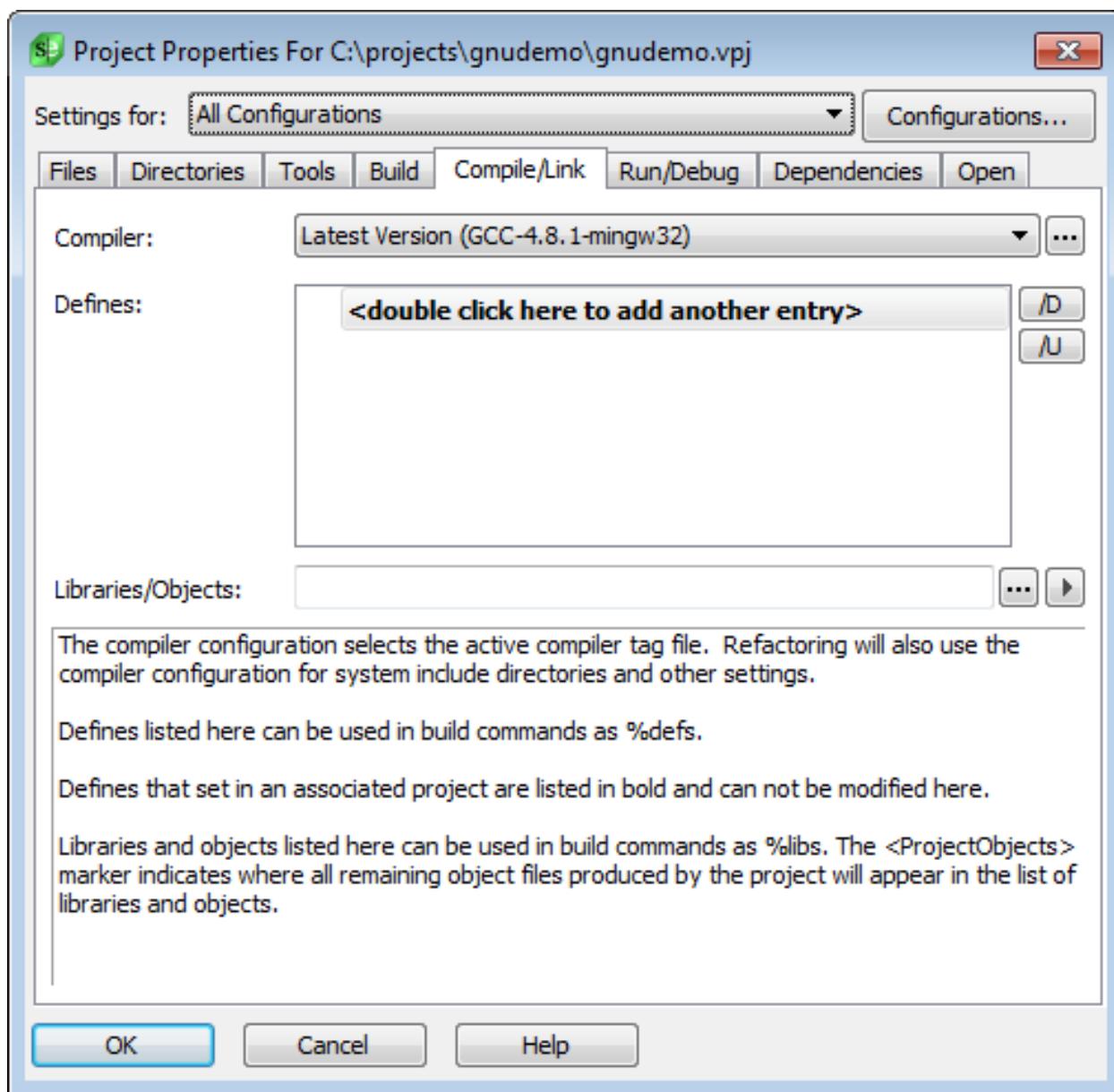
- **Makefile** - Specify the path to the makefile in the **Makefile** field. Make sure the project include directories are set up correctly (**Project → Project Properties**, select the [Directories Tab](#)) so include files may be found.
- **Run jobs simultaneously** - For projects using GNU makefiles, you can enable this in order to invoke **make** with the **-j** argument in order to specify how many threads to use for building makefile targets in parallel.

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 (Pro only)

The **Compile/Link** tab of the Project Properties dialog (**Project → Project Properties**), shown below, is used to specify project compilation and linking options.

Project Dialogs and Tool Windows



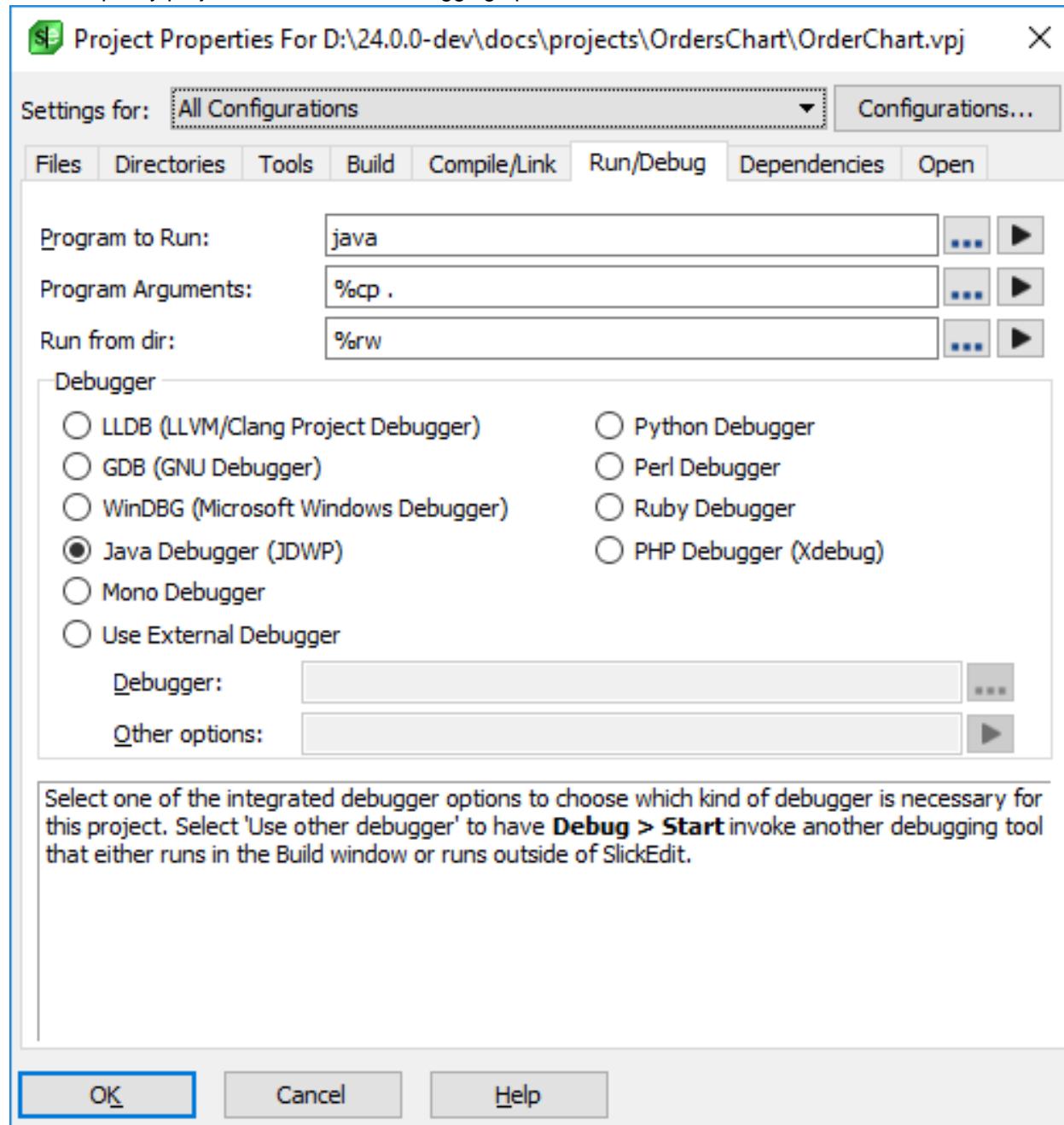
The following settings are available:

- **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. Select **None** to specify that a project should not use a compiler tag file. 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** button to enter a macro. Click the **/U** button to undefine a macro. Defines that are 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 button to remove a library or object.

Run/Debug Tab

The **Run/Debug** tab of the Project Properties dialog (**Project → Project Properties**), shown below, is used to specify project execution and debugging options.



The following settings are available:

- **Program to Run** - Specifies the full path to the program to execute when **Execute** is invoked from the

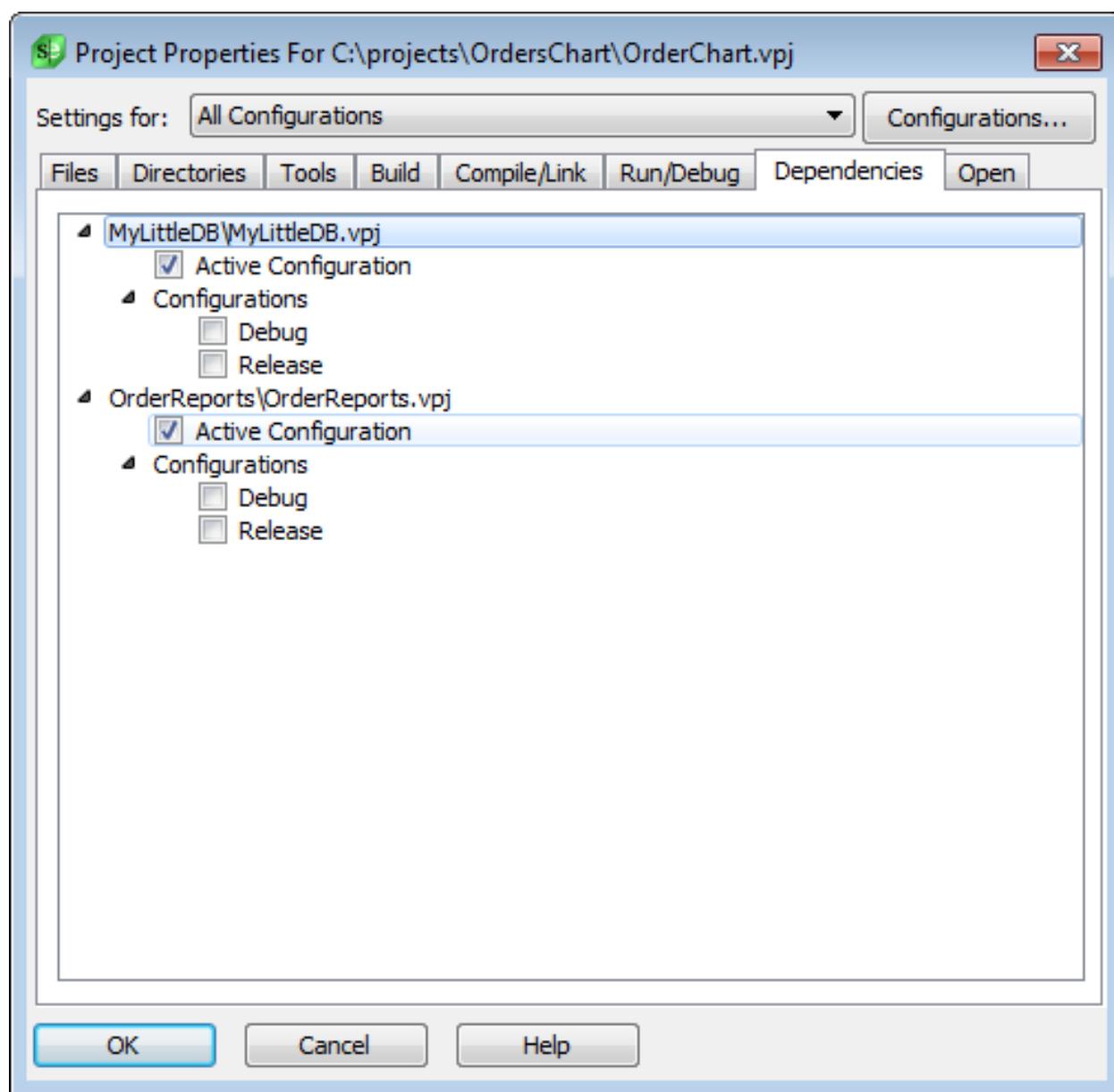
Build menu, as well as the program to be debugged when **Start** is invoked from the **Debug** menu.

- **Program Arguments** - Specifies the command line arguments to be passed to the program when it is executed.
- **Run from dir** - Specifies the directory to run the program from. Typically, this will be set to "%rw" to indicate the project directory.
- **Debugger** - Specifies which integrated debugging tool is appropriate to use for this project. Select **Use External Debugger** to select an external or command line debugging tool to run. Depending on the tool specified, you may need to fine tune the settings for the **Debug** target on the **Tools** tab of the **Project Properties** dialog.
- **Other options** - Specifies additional command line options to pass to the debugger. For example, to pass the path to the program you wish to debug here.

Dependencies Tab (Pro only)

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.

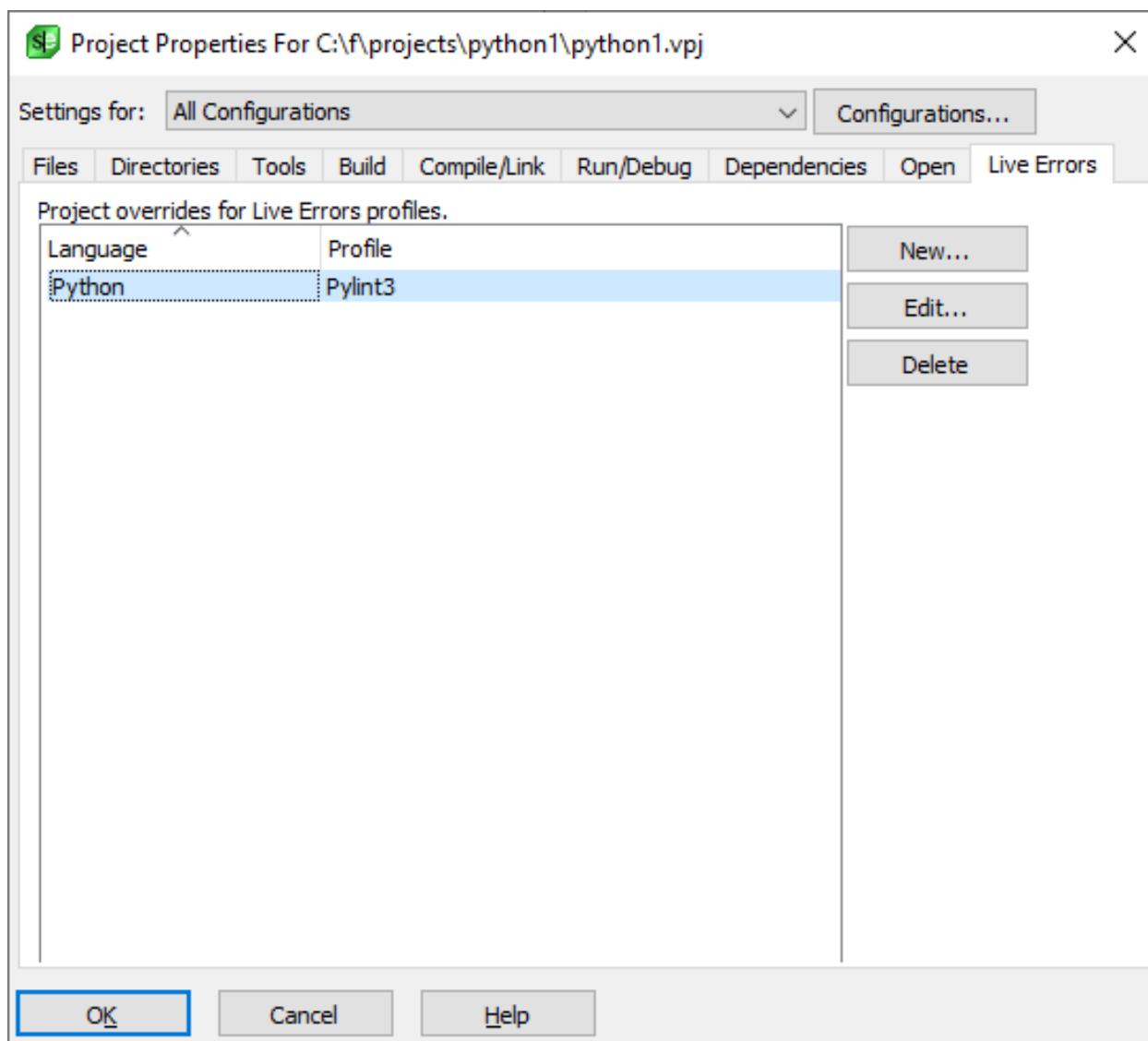
Project Dialogs and Tool Windows



Live Errors Tab (Pro only)

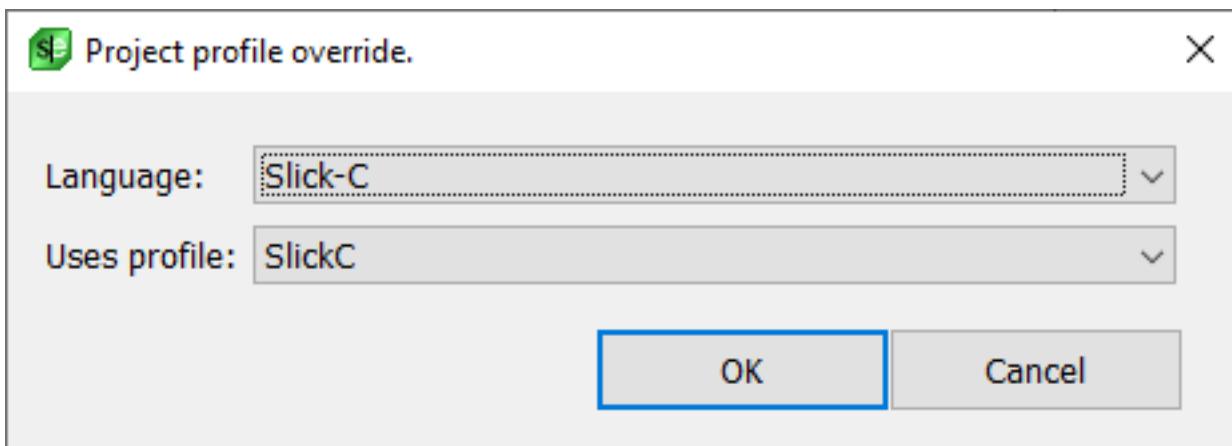
The **Live Errors** tab on the Project Properties Dialog (**Tools** → **Project Properties**) allows you to override the active Live Errors profiles that are used when this project is active. See [Live Errors](#) for more details on the Live Errors system.

Project Dialogs and Tool Windows



- The main list shows any existing overrides for the project. In the example image above, Live Errors will use the "Pylint3" profile for any Python files while the project is active.
- **New...** - Creates a new override.
- **Edit...** - Edits the currently selected profile override.
- **Delete** - Deletes the currently selected profile override.

New/Edit Profile Override Dialog



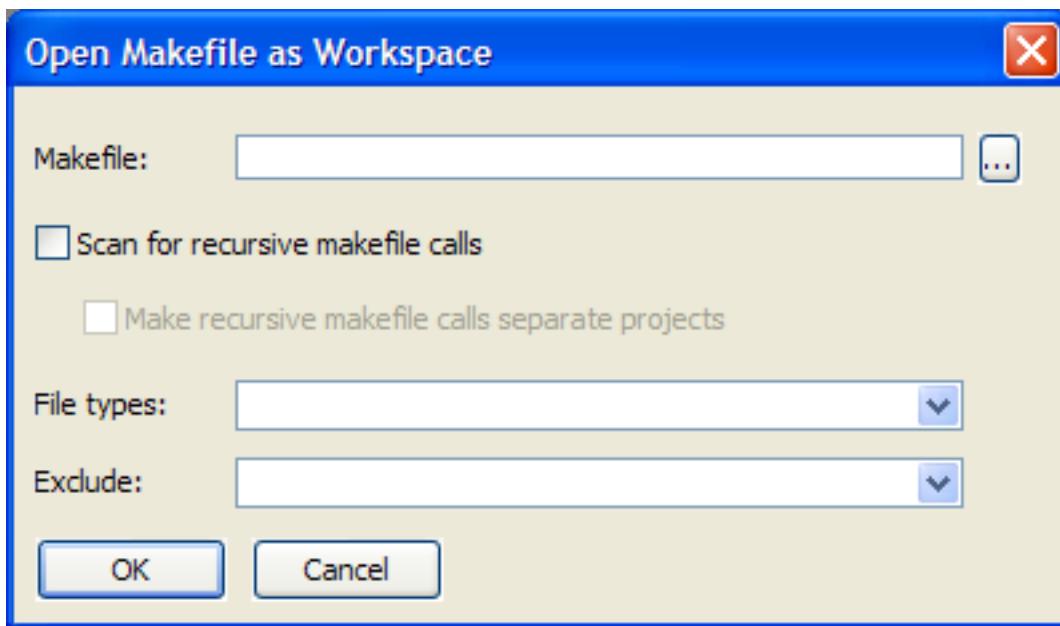
- **Language** - Selects the language mode the override will be active for. Only languages that have Live Errors profiles that aren't already overridden will be available for selection.
- **Uses profile** - Live Errors profile that will be used for files with the associated **Language** document mode.

Open Tab (Pro only)

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 unavailable for extension-based projects. For instructions on entering commands on this tab, see [Specifying Open Commands](#).

Open Makefile as Workspace Dialog

The Open Makefile as Workspace dialog is used to import makefiles. To display it, from the main menu, click **Project** → **Open Other Workspace** → **Makefile** (or use the `workspace_open_makefile` command). See [Importing Makefiles](#) for more information about using this feature.



The dialog contains the following:

- **Makefile** - Specifies the makefile. Use the **Browse** button to browse for the makefile.
- **Scan for recursive makefile calls** - When selected, SlickEdit scans the makefile for invocations of make on other makefiles and includes the referenced files in the new project.
 - **Make recursive makefile calls separate projects** - When this option is selected, if **Scan for recursive makefile calls** is enabled, and if the makefile contains invocations of make on other makefiles, a new, separate project is created for each of the referenced makefiles. When this option is cleared, SlickEdit creates only one project that includes all of the files found in all of referenced makefiles.
- **File types** - Specifies the file types to include. Use the drop-down arrow to populate this field or type a list of file types separated with semicolons.
- **Exclude** - Specifies the file types to exclude. Use the drop-down arrow to populate this field or type a list of file types separated with semicolons.

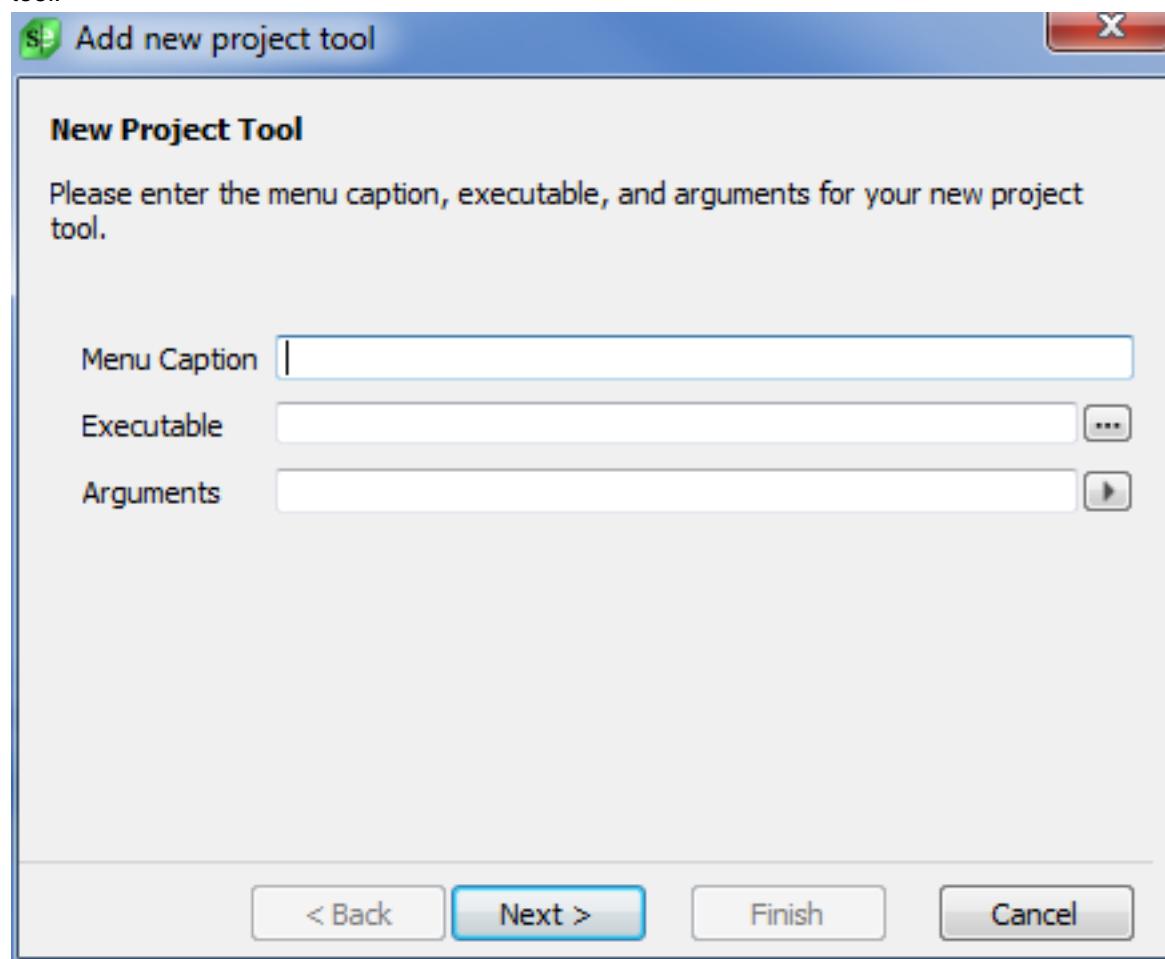
New Project Tool Wizard

The New Project Tool Wizard is used to set up new build tools for a project. Build tools appear in the **Build** menu when a project is active. For information about editing existing build tools, see [Tools Tab](#) of the Project Properties dialog.

The New Project Tool Wizard can be accessed in two ways: by clicking the **New** button on the Tools tab of the Project Properties dialog or by going to **Build → Add new build tool...**. In addition to your own source code projects, you can add build tools to Project Templates (see [Project Types](#)) and extension-specific projects (see [Defining Language-Specific Projects](#)).

New Project Tool

The first page of the New Project Tool Wizard allows you to fill in basic information about your new build tool.



The following fields are available:

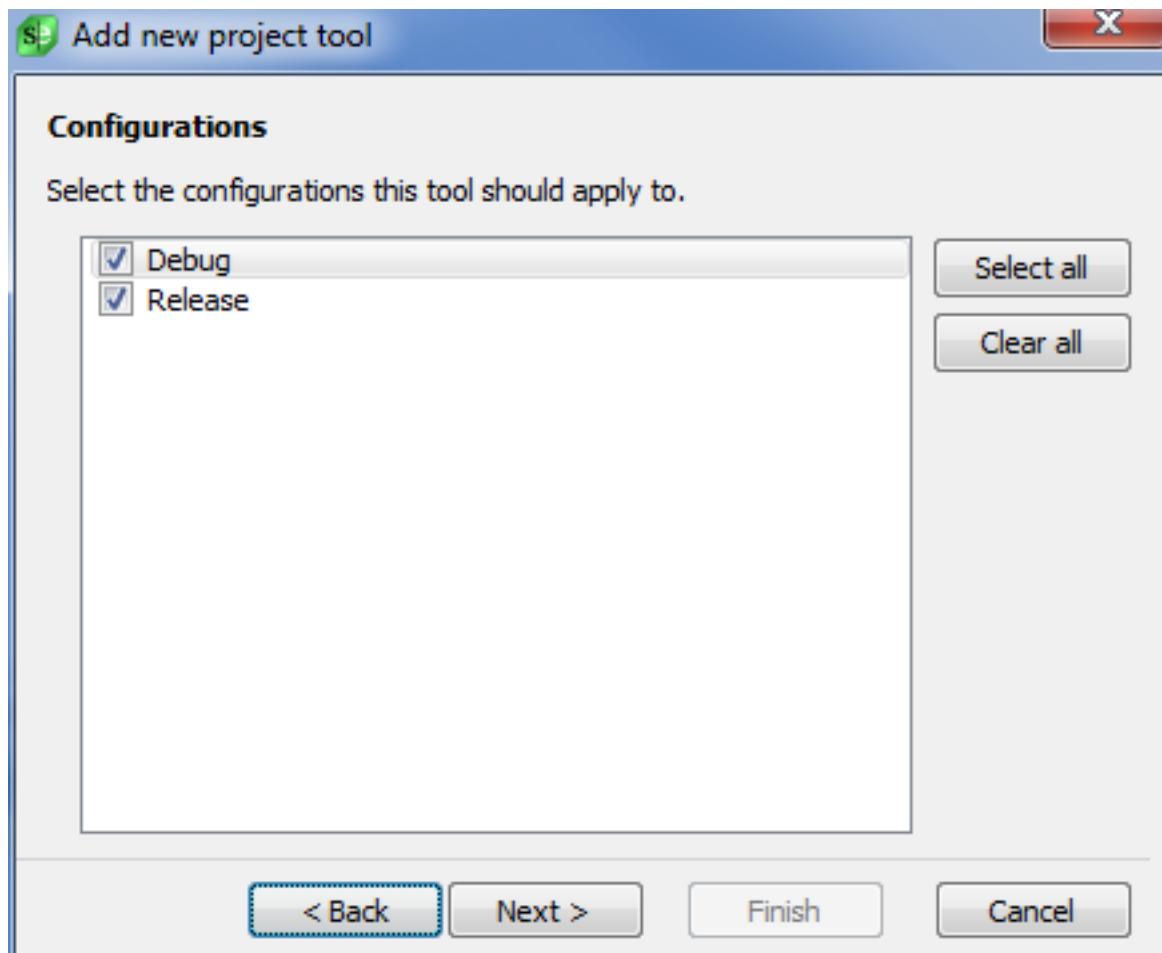
- **Menu Caption** - the caption that will appear on the **Build** menu and can be used to launch your new tool.
- **Executable** - the executable to be launched when this tool is selected. Use the button to the right of this field to browse to and select the specific file.
- **Arguments** - the arguments sent to the executable. Use the menu launched from the button to the right of this field to send common project-related arguments.

Configurations

The second part of the New Project Tool Wizard is the Configurations page, where you select which configurations of your project will allow access to the new tool. You must select at least one configuration for your tool. For more information, see [Project Configurations](#).

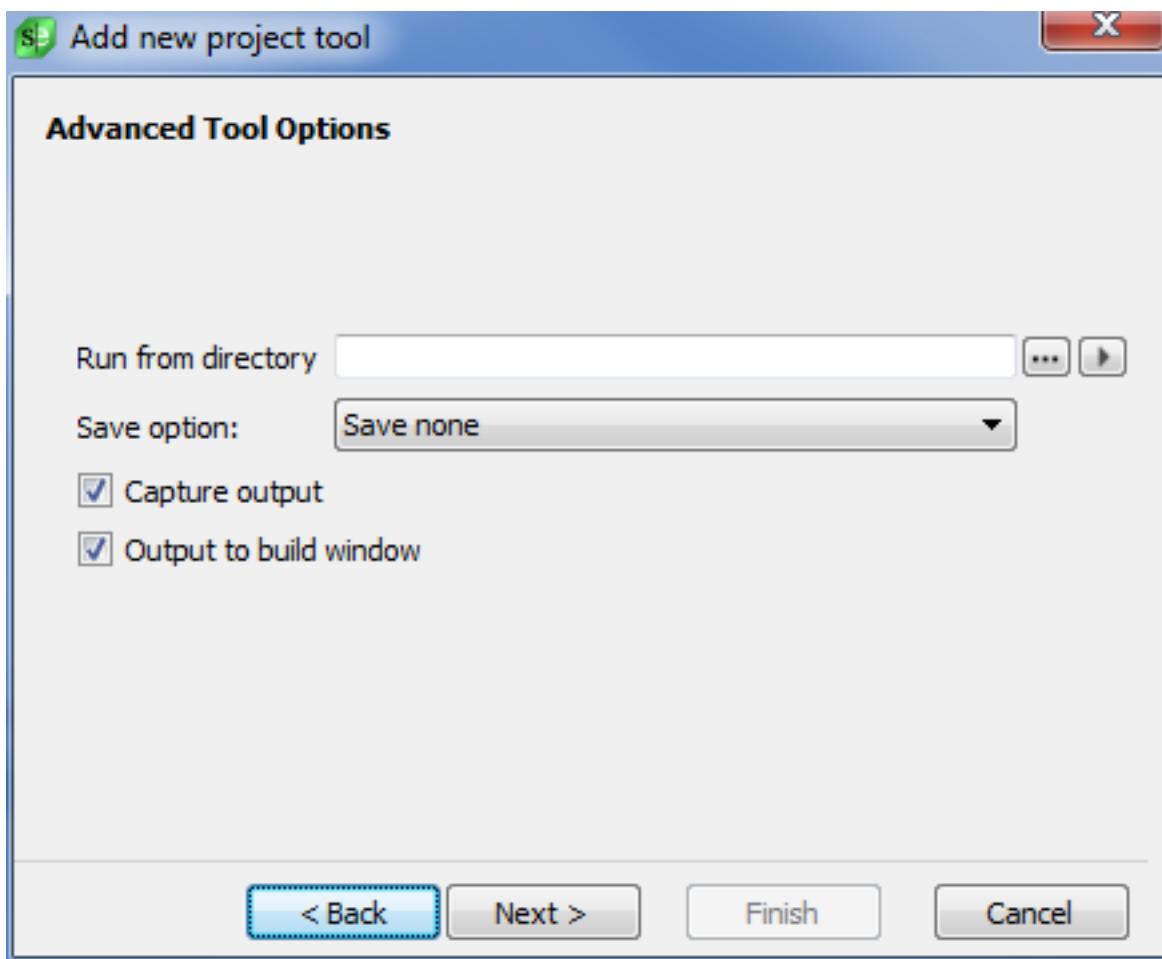
Note

When creating a tool for a language-specific project, this step is skipped.



Advanced Tool Options

You can set more advanced options for your new project tool on the third page.



The following fields are available on this page:

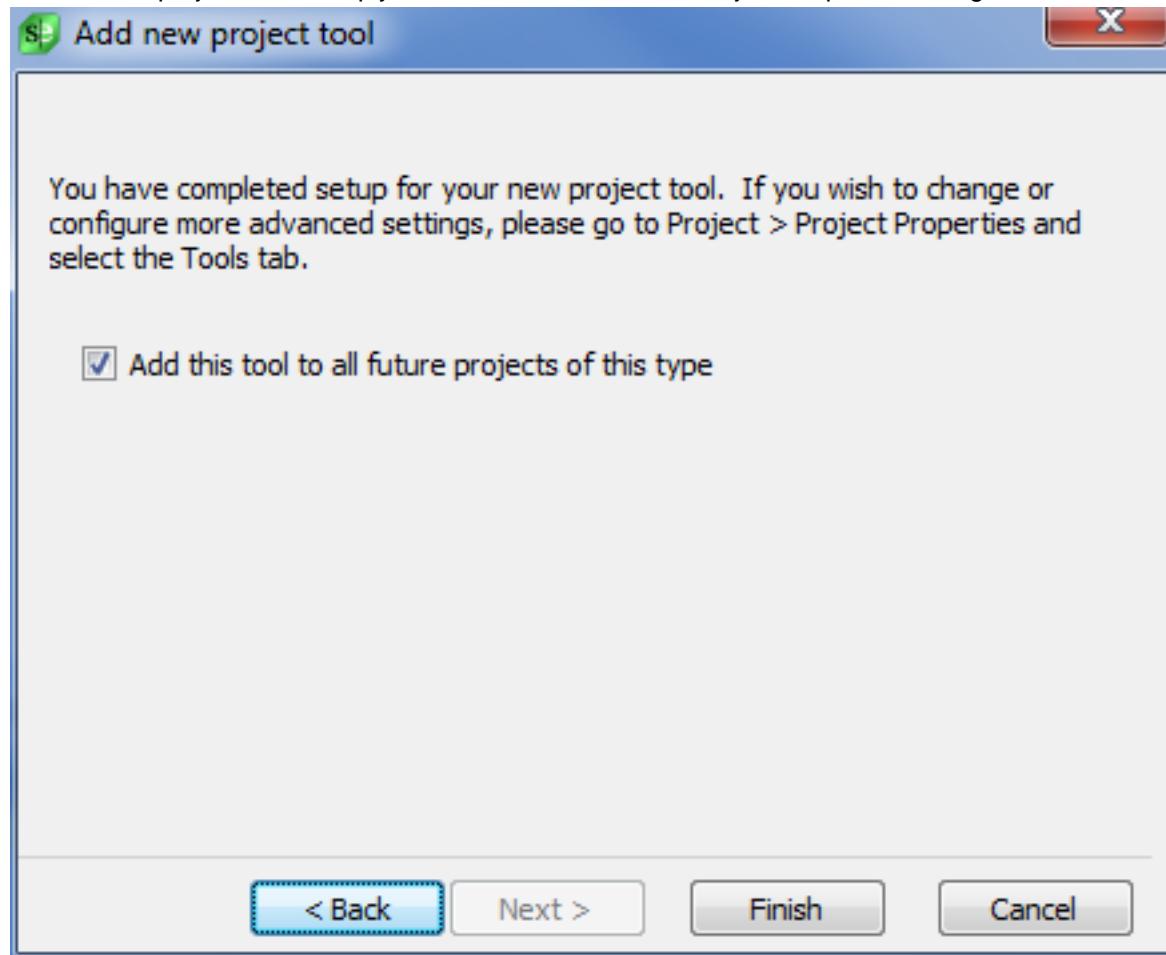
- **Run from directory** - The directory that the command will be run from. Use the buttons to the right of this field to browse to a directory or to specify an escape sequence symbolizing a project-related directory.
- **Save option** - Specifies which, if any, files to save before launching the tool. Choose from the following choices:
 - **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.
- **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**)

) are used to go to the next and previous compilation error positions respectively.

- **Output to build window** - Specifies that the output of the command be run in the concurrent build window.

Finish Wizard

The final page of the wizard lets you know that your new build tool is completed and ready to use. To edit this or other project tools, simply visit the **Tools** tab of the Project Properties dialog.



If you are adding a tool to a source code project, you will have the opportunity to add this same tool to all future projects of that type. Existing projects of the same type will not have the tool.

Build (Pro only)

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 (Pro only)

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
Compile	Compiles the current file.	<code>project_compile</code>
Build	Builds the active project, typically compiling only the files that have changed.	<code>project_build</code>
Rebuild	Rebuilds the active project, typically compiling all files in the project.	<code>project_rebuild</code>
Execute	Executes the built program associated with the active project.	<code>project_execute</code>
Add new build tool...	Launches the New Project Tool Wizard .	<code>project_tool_wizard</code>
Next Error	Processes the next compiler error message.	<code>next_error</code>
Previous Error	Processes the previous compiler error message.	<code>prev_error</code>
Go to Error or Include	Parses the error message or file name at the cursor and places cursor in file.	<code>cursor_error</code>
Clear All Error Markers	Removes all error markers in all files.	<code>clear_all_error_markers</code>
Configure Error Parsing	Configures regular expressions used to search for compiler messages. See Parsing Errors with Regular Expressions .	<code>configure_error_regex</code>

Build Menu Item	Description	Command
Stop Build	Sends break signal to the Build tool window.	stop_process
Show Build	Starts or activates the Build tool window.	start_process
Set Active Configuration	Submenu used to select which project build configuration is currently active.	project_config_set_active
Project Properties for Config	Displays the Project Properties dialog, which is used to edit settings for the current project with the current active configuration selected. See Project Properties Dialog	project_edit_config
Build Automatically on Save	When enabled, the project is built each time the workspace is saved.	project_toggle_auto_build

Debug (Pro only)

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 (Pro only)

The **Debug** menu contains debugging-related operations and options. The table below summarizes these items.

Debug Menu Item	Description	Command
Windows	Displays debug window tool windows. See Debug Windows Menu .	N/A
Start	Starts debugger.	<code>project_debug</code>
Suspend	Suspends execution.	<code>debug_suspend</code>
Stop Debugging	Stops debugging the program.	<code>debug_stop</code>
Restart	Restarts the program.	<code>debug_restart</code>
Start with arguments	Starts debugger current project with user-specified command line arguments and working directory.	<code>debug_run_with_arguments</code>
Attach Debugger . See Attach Debugger Menu .	Attach debugger to a process or remote server.	N/A
Detach	Detach from target process and allow application to continue running.	<code>debug_detach</code>
Debugger Information	Displays the Debugger Information dialog.	<code>debug_props</code>
Step Into	Steps into the next statement.	<code>debug_step_into</code>
Step Over	Steps over the next statement.	<code>debug_step_over</code>
Step Out	Steps out of the current function.	<code>debug_step_out</code>
Step Instruction	Steps one instruction at a time.	<code>debug_step_instr</code>

Debug Menu Item	Description	Command
Run to Cursor	Runs the program to the line containing the cursor.	<code>debug_run_to_cursor</code>
Show Next Statement	Displays the source line for the instruction pointer.	<code>debug_show_next_statement</code>
Set Instruction Pointer	Set the instruction pointer to the current line.	<code>debug_set_instruction_pointer</code>
Show Disassembly	Toggle display of disassembly.	<code>debug_toggle_disassembly</code>
Toggle Breakpoint	Toggles a breakpoint at the current line.	<code>debug_toggle_breakpoint</code>
Delete All Breakpoints	Deletes all debugger breakpoints.	<code>debug_clear_all_breakpoints</code>
Disable All Breakpoints	Disables all debugger breakpoints.	<code>debug_disable_all_breakpoints</code>
Add Watch	Add a watch on the variable under the cursor.	<code>debug_add_watch</code>
Set Watchpoint	Set a watchpoint on the variable under the cursor.	<code>debug_add_watchpoint</code>
Debugger Options	Displays the Debugger Options dialog. See Viewing Debugger Info and Setting Options for detailed information.	<code>debugger_options</code>

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.	<code>activate_call_stack</code>
Locals	Activates the Locals window.	<code>activate_locals</code>
Members	Activates the window which	<code>activate_members</code>

Debug Windows Menu Item	Description	Command
	displays member variables.	
Autos	Activates the Autos window.	activate_autos
Watch	Activates the Watch window.	activate_watch
Threads	Activates the Threads window.	activate_threads
Breakpoints	Activates the Breakpoints window.	activate_breakpoints
Registers	Activates the Registers window.	activate_registers
Memory	Activates the Memory window.	activate_memory
Loaded Classes	Activates the Loaded Classes window.	activate_classes

Attach Debugger Menu

The **Debug → Attach Debugger** menu items are summarized in the table below. See [Multiple Session Debugging](#) for more information. At this time, the LLDB related options are only available on macOS and 64-bit Linux. The WinDBG related options are only available on Microsoft Windows.

Attach Debugger Menu Item	Description	Command
Attach to Running Process (LLDB)	Attach debugger to a running process using LLDB.	debug_attach_lldb
Analyze Core File (LLDB)	Load crash dump information from a Unix core file and analyze it using the integrated LLDB debugger.	debug_corefile_lldb
Attach to Remote Process (LLDB)	Attach debugger to a remote LLDB server or executable with GDB stub.	debug_remote_lldb
Debug Executable (LLDB)	Step into a program using LLDB	debug_executable_lldb
Attach to Running Process (GDB)	Attach debugger to a running process using GDB.	debug_attach_gdb
Analyze Core File (GDB)	Load crash dump information	debug_corefile_gdb

Attach Debugger Menu Item	Description	Command
	from a Unix core file and analyze it using the integrated GDB debugger.	
Attach to Remote Process (GDB)	Attach debugger to a remote GDB server or executable with GDB stub.	<code>debug_remote_gdb</code>
Attach to Android Application Process (GDB)	Attach debugger to an Android application running on hardware device or emulator.	<code>debug_remote android</code>
Debug Executable (GDB)	Step into a program using GDB	<code>debug_executable_gdb</code>
Attach to Process (WinDBG)	Attach debugger to a running process using WinDBG	<code>debug_attach_windbg</code>
Debug Executable (WinDBG)	Step into a program using WinDBG	<code>debug_executable_windbg</code>
Open Dump File (WinDBG)	Attach debugger to a dump file	<code>debug_corefile_windbg</code>
Write Dump File (WinDBG)	Write a dump file using WinDBG	<code>windbg_write_dumpfile</code>
Attach to Java Virtual Machine	Attach to a Java virtual machine executing remotely.	<code>debug_attach_jdwp</code>
Debug Executable (Java)	Step into a program using the Java debugger	<code>debug_executable_java</code>
Attach to Mono Virtual Machine	Attach to a Mono virtual machine executing remotely.	<code>debug_attach_mono</code>
Debug Executable (Mono)	Step into a program using the Mono debugger	<code>debug_executable_mono</code>
Attach to Xdebug	Attach to a PHP session using Xdebug.	<code>debug_remote_xdebug</code>
Attach to Python (PTVSD)	Attach to a Python debugger session.	<code>debug_remote_dap</code>
Attach to perl5db	Attach to a Perl 5 debugger session.	<code>debug_remote_perl5db</code>

Attach Debugger Menu Item	Description	Command
Attach to rdbgp	Attach to a Ruby debugger session.	debug_remote rdbgp

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 Files, Buffers, and Editor Windows .	<code>next_buffer</code>
Previous Buffer	Switches to the previous buffer. See Files, Buffers, and Editor Windows .	<code>prev_buffer</code>
Close Buffer	Closes the current buffer. See Files, Buffers, and Editor Windows .	<code>close_buffer</code>
List Open Files	Displays the Files tool window, which lists all buffers and allows you to activate one. See Document Dialogs and Tool Windows .	<code>list_buffers</code>
Edit Associated File	Switch to header or source file associated with the current file.	<code>edit_associated_file</code>
Select Mode	List all modes and lets you select one. See Language Editing Mode .	<code>select_mode</code>
Language Options	Displays the Options dialog open to the language-specific General options screen for the language in the current buffer. See Language Options .	<code>setupext</code>
Tabs	Sets tab stops.	<code>gui_tabs</code>
Margins	Sets word wrap margins.	<code>gui_margins</code>

Document Menu

Document Menu Item	Description	Command
Reflow Paragraph	Reflows the text in the current paragraph according to the margins.	<code>reflow_paragraph</code>
Reflow Selection	Reflows the selected text according to the margins.	<code>reflow_selection</code>
Format Columns	Format columns according to words.	<code>format_columns</code>
Edit Doc Comment	Edits document comments for the current source file.	<code>edit_doc_comment</code>
Comment Block	Converts selected text into block comment using box comment setup characters. See Commenting .	<code>box</code>
Comment Lines	Converts selected lines into line comments using the line comment setup. See Commenting .	<code>comment</code>
Uncomment Lines	Uncomments any commented lines and ignores any that isn't commented. See Commenting .	<code>comment_erase</code>
Reflow Comment	Reflows and reformats the current block comment. See Reflow Comment Dialog .	<code>gui_reflow_comment</code>
Comment Setup	Displays the language-specific Comment options screen, which contains settings for box and line comments. See Language-Specific Comment Options .	<code>comment_setup</code>
Comment Wrap	Toggles comment wrap on/off.	<code>comment_wrap_toggle</code>
Beautify while typing	(Pro only) Toggles Beautify while typing on/off. See Beautify while typing .	<code>indent_with_tabs_toggle</code>
Indent with Tabs	Toggles indenting with tabs on/off. See Syntax Indent .	<code>indent_with_tabs_toggle</code>

Document Menu Item	Description	Command
Spell Check While Typing	Toggles spell check while typing on/off.	<code>spell_check_while_typing_toggle</code>
Word Wrap While Typing	Toggles word wrap while typing on/off.	<code>word_wrap_toggle</code>
Justify	Sets/displays word wrap justification style.	<code>gui_justify</code>
Read Only Mode	Toggles read-only mode on/off.	<code>read_only_mode_toggle</code> .
Adaptive Formatting	Toggles Adaptive Formatting on/off. See Adaptive Formatting .	<code>adaptive_format_toggle</code>

Document 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 contains three tabs that allow you to view open files, project files, and workspace files. The files can be sorted 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.).

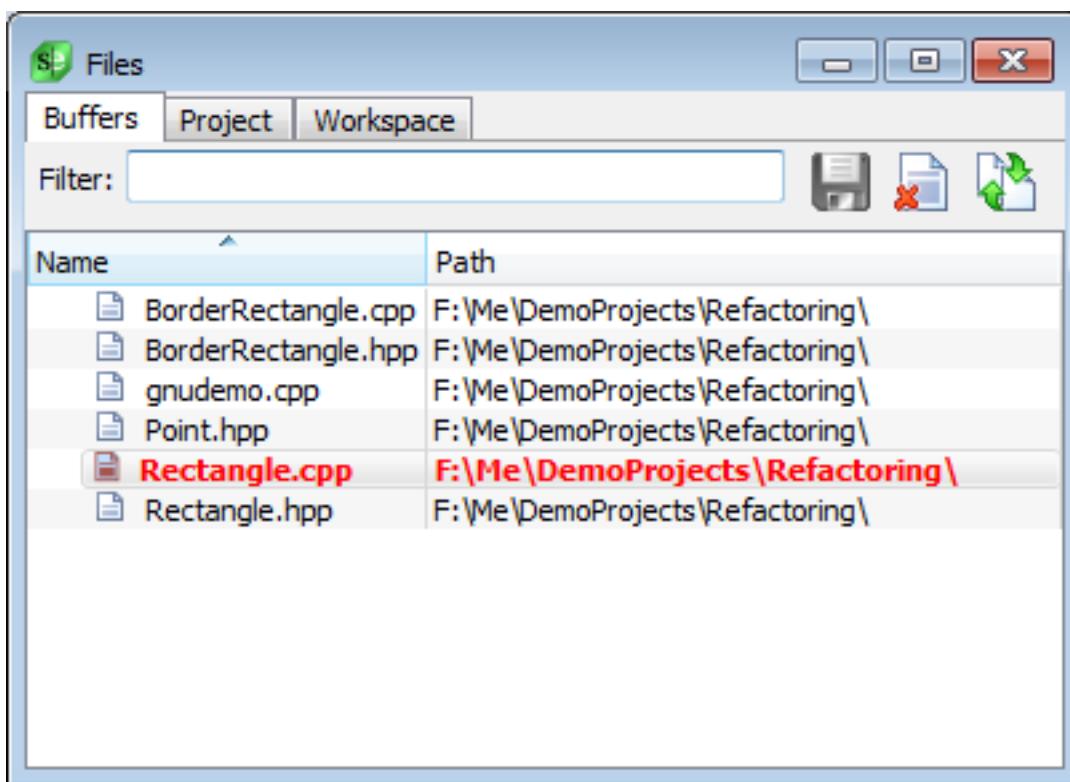
Note

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:

- Click **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 clicking **View** → **Tool Windows** → **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 with a tab to select each:

- **Buffers** - Shows the list of files that are being edited. Useful for selecting the file to edit when working on multiple files.
- **Project** - (Not in Community edition) Shows the files in the active project.
- **Workspace** - (Not in Community edition) Shows the files in this 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.

Note

The **Filter** text box supports ant-like wildcards (wildcards in path parts and use of **). Depending on what you've typed and the options you've specified (see context menu options), an ant-like wildcard specification is generated and used to match files. For example, if you specify **path-part/name-part**, the generated wildcard is ***/path-part**/*name-part***. The context menu has some options for how to handle certain **name-part** specifications. For example, if **name-part** is ***.ext**, no leading * is prepended to **name-part**, and no trailing * is appended to **name-part..**. The context menu has an option for this.

To allow 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.

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.
- 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** bitmap 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 right of the **Filter** field.
- Right-click and select **Save**.

Tip

- When you invoke a Save operation, if a file is selected, it is simply saved. If an untitled buffer of the form **Untitled<nn>** (not Untitled yyyy-mm-dd...) 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 File** screen element (**Tools** → **Options** → **Appearance** → **Colors**). 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, all windows displaying the buffer are closed as well (if the **Files per window** option **One file per window** is enabled at **Tools** → **Options** → **Editing** → **Editor Windows**). 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.

Diffing Files

In the Buffers view, you can select a modified file and compare it against the version on disk by doing one of the following:

- Right-click and select **Diff**.
- Click the **Diff 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:

- **Buffers tab** - displays all files and buffers that are currently open in the editor.
- **Project tab** - displays all files in the current project, regardless of whether they are open or not. This view does not show an indicator for modified files. Further, it does not provide icons to Save, Close, or Diff a file. See [Workspaces and Projects](#) for more information about working with projects.
- **Workspace tab** - displays the set of all files in the current workspace. This view does not show an indicator for modified files. Further, it does not provide icons to Save, Close, or Diff a file. See [Workspaces and Projects](#) for information about workspaces.
- **Save Selected File(s) icon** - if a file is selected, it is simply saved. If an untitled buffer of the form **Untitled<nn>** (not Untitled yyyy-mm-dd...) 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 context menu inside the Files list. This icon is only displayed when the Buffers tab is selected.
- **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 context menu inside the Files list. This icon is only displayed when the Buffers tab is selected.
- **Diff Selected File(s) icon** - compares the selected file(s) against the version on disk. This icon is only displayed when the Buffers tab is selected.
- **Filter** - used to display matching file names. Right-click inside the Files list area to allow **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.
- **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** bitmap 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.
- Context menu - right click in the files list to see the available operations:
 - Open - select this option to open the selected file.

- Open in Current Window - select this option to open the selected file in the current window. This is useful if you do not have the **One file per window** option on (see [Files, Buffers, and Editor Windows](#)).
- Save - saves the selected file. This is only available when the Buffers tab is selected.
- Close - closes the selected file. This is only available when the Buffers tab is selected.
- Diff - compares the selected file against the version on disk. This does nothing if the selected file is not modified.
- Dismiss on select - this option dismisses the Files tool window after you select a file. This is useful when you use the Files tool window undocked.
- Prefix match - select this option to perform a prefix match instead of matching anywhere in the filename.
- Refresh - refreshes the file list.

Tabs Dialog

The Tabs dialog (**Document** → **Tabs**), is used to specify tab stops. Note that configuring the tabs does not necessarily effect where the **Tab** and **Shift+Tab** keys move the cursor. You may need to configure your syntax indent. See [Language-Specific Formatting Options](#). Setting the tab stops always effects how text is displayed. Tab characters are expanded to spaces when displayed on the screen.

- **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.

Margins Dialog

The Margins dialog (**Document** → **Margins**), is used to configure the word wrap margin options. This word wrap feature is intended only for plain text only.

- **Automatic Left Margin** - If selected, the left margin is determined by the first non blank in the line. The right margin may be specified as follows:
 - **Fixed right column** - If selected, lines will break before the specified column.
 - **Fixed width** - If selected, specifies the maximum amount of non blank text allowed on each line.
- **Fixed left column** - If selected, allows you to specify the left margin, right margin, and new paragraph columns.
- **Partial word wrap** - When on and word wrap while typing is on, a more conservative word wrap approach is taken. This option provides word wrap similar to previous versions of SlickEdit. You may prefer this style of word wrapping if you leave word wrap while typing on for source files. This option only effects word wrap while typing characters, pressing **Backspace**, or pressing **Del**.

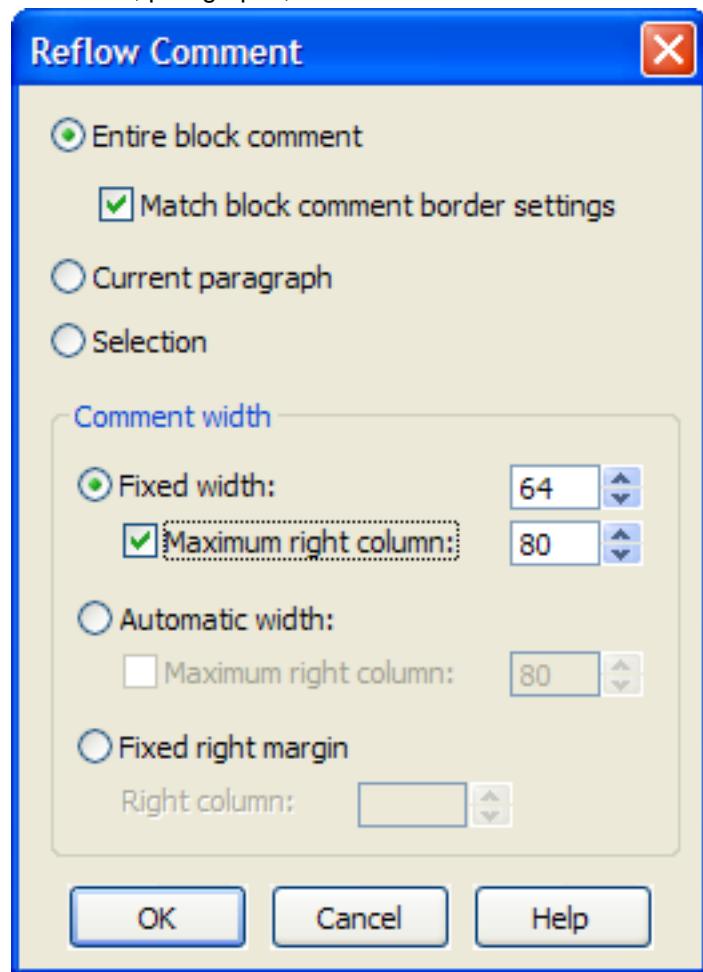
Justification Dialog

The Justification dialog (**Document → Justify**), is used to configure the justify style used when you word wrap paragraphs. The word wrap as you type features of Word Wrap do not support full justification.

- **Left and respace** - Left justification with space character reformatting. One space is placed between words except after the punctuation characters period, ?, and !, which get two spaces. To have only one space after the period, question mark, and exclamation point punctuation characters, 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 File Options](#) for more information.
- **Justified** - Full justification. Left and right edges of text will align exactly at margins.

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 border settings** - If selected, forces the borders to conform to the comment settings (**Document → Comment Setup**- see [Language-Specific Comment Options](#)).
- **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 [Language-Specific Comment Wrap Options](#) for information.

For more information about comments, see [Commenting](#).

Macro

This section describes items related to macros.

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.	<code>gui_load</code>
Unload Module	Unloads a Slick-C macro file from the state file.	<code>gui_unload</code>
List User-Loaded Macros	(Pro only) Lists user-loaded Slick-C modules. See Macro Dialogs and Tool Windows .	<code>gui_list_macfiles</code>
Record Macro	Starts recording a Slick-C language macro.	<code>record_macro_toggle</code>
Stop Recording Macro	Stops recording a Slick-C language macro.	<code>record_macro_toggle</code>
Execute last-macro	Runs last recorded macro.	<code>record_macro_end_execute</code>
Save last-macro	Saves the last recorded macro under a name you specify. See Save Macro Dialog .	<code>gui_save_macro</code>
List Macros	Lists saved, recorded macros. See List Macros Dialog .	<code>list_macros</code>
Set Macro Variable	Allows you to set global macro variables. See Set Variable Dialog and Variable Editor Dialog .	<code>gui_set_var</code>
Start Slick-C Debugger	(Pro only) Activates the Slick-C debugger window. See "Slick-C Debugger" in the Help → Index .	<code>slickc_debug_start</code>
Go to Slick-C Definition	Opens a macro source file and	<code>gui_find_proc</code>

**Macro Dialogs and Tool
Windows**

Macro Menu Item	Description	Command
	places your cursor on the definition of a macro symbol.	
Find Slick-C Error	Places your cursor on the macro source line which caused the last interpreter run-time error.	find_error
New Form	(Pro only) Opens a new form for editing with the Dialog editor.	new_form
Open Form	(Pro only) Opens an existing or new form for editing with the Dialog editor.	open_form
Selected Form	(Pro only) Displays edited form window currently selected.	show_selected
Load and Run Form	(Pro only) Loads form, loads Slick-C code, and runs the currently selected/edited form.	run_selected
Grid	(Pro only) Sets form grid settings. This affects the distance displayed between the dots on a form that is being edited. See Grid Settings Dialog .	gui_grid
Menus	Lists all menus and allows you to edit, create, delete, or show menus. Provides access to the Menu Editor dialog box. See Menu Editor Dialog .	open_menu
Insert Form or Menu Source	Inserts source code into current file for a form or menu you specify.	insert_object

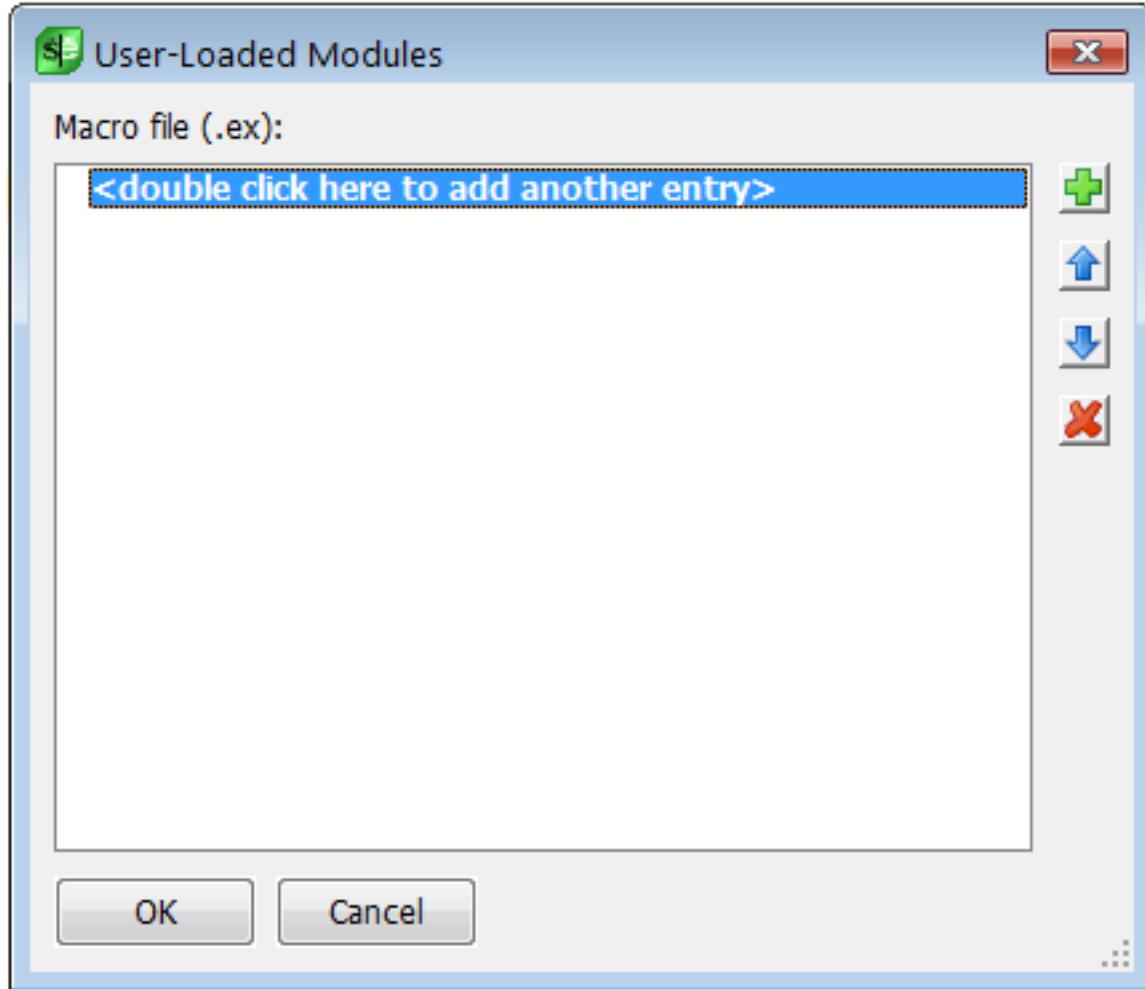
Macro Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Macro** menu items.

User-Loaded Modules Dialog (Pro only)

The User-Loaded Modules dialog is used to view a list of Slick-C® modules that you have loaded. It also lets you add, delete, and re-order modules. See [Slick-C® Modules](#) for more information.

To display the dialog, from the main menu, click **Macro** → **List User-Loaded Modules**, or, use the **gui_list_macfiles** command on the SlickEdit® command line.

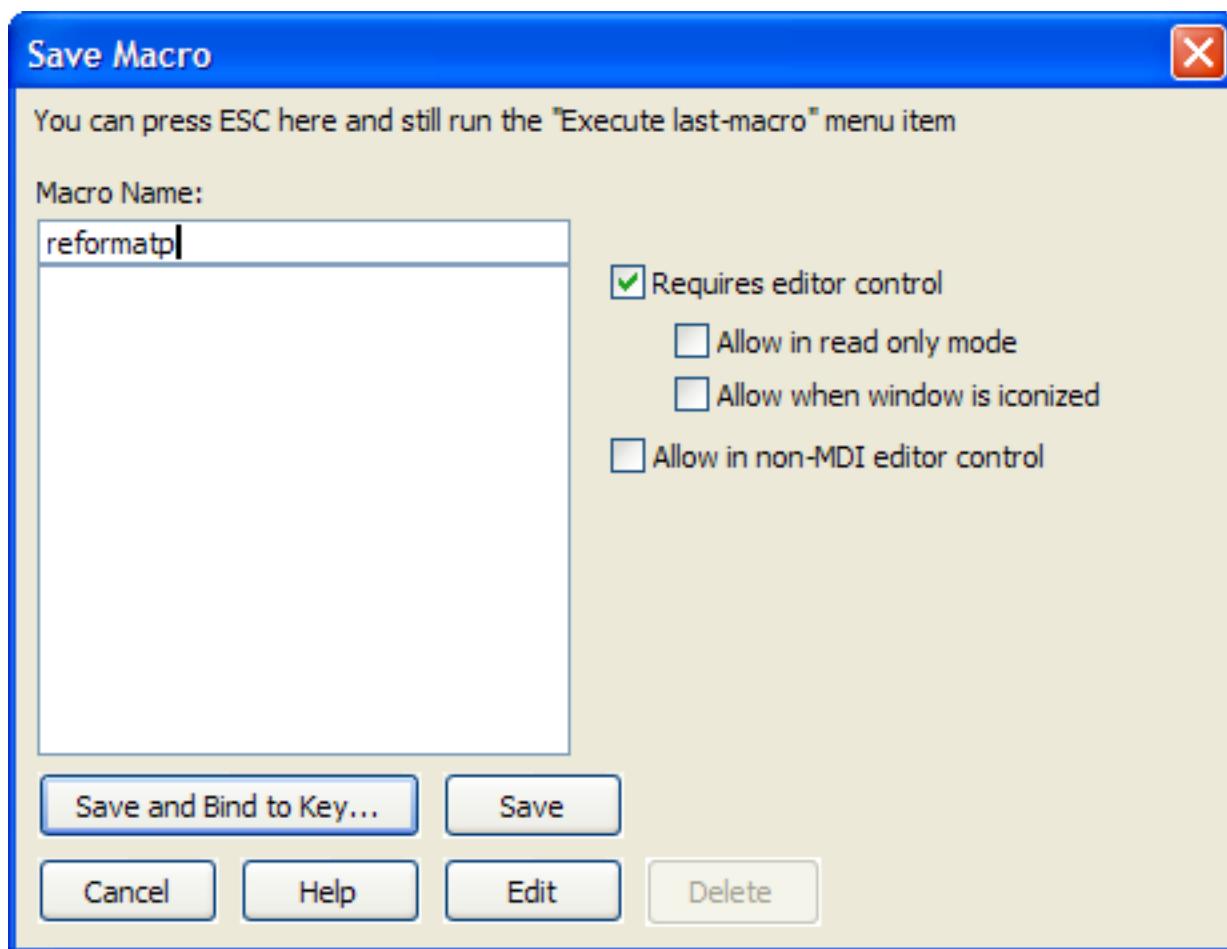


To add a module to the list, double-click where indicated or click the **Add** button. To remove a module from the list, select it, then click the **Delete** button. Use the arrow buttons to rearrange the order, moving the selected module up and down in the list. This is the order in which modules are re-loaded when you upgrade to a newer version of SlickEdit.

When you add a module to the list, SlickEdit prompts to load the module in the editor. When you delete a module from the list, SlickEdit prompts to also unload the module. See [Loading and Unloading Slick-C Modules](#) for more information.

Save Macro Dialog

The Save Macro dialog appears automatically when you end macro recording, or when you click **Macro** → **Save last-macro**. You can also display the dialog by using the **gui_save_macro** command.



The dialog contains the following elements:

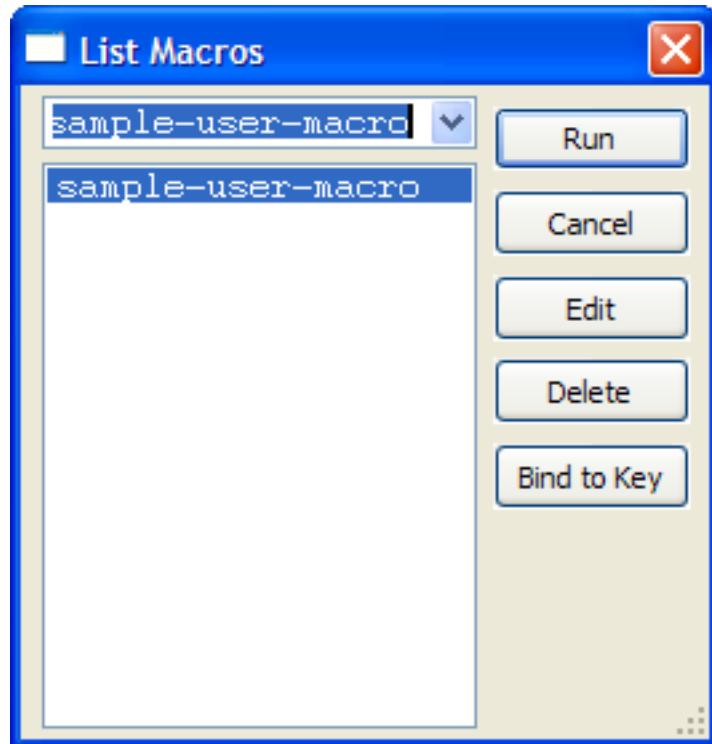
- **Macro Name** - Specifies the name for the recorded macro. If you attempt to give the macro a name that is already taken, a message is displayed asking if you want to overwrite the existing macro.
- **Macro list** - The box under the **Macro Name** field shows a list of all macros you have recorded (if any exist).
- **Requires editor control** - When selected, the macro only operates if the target is an editor control.
- **Allow in read only mode** - When selected, the macro is permitted to operate even in read-only mode. Select this option if your macro does not modify the current buffer.
- **Allow when window is iconized** - When selected, the macro is permitted to operate even when the edit window is iconized. If the macro modifies the current buffer, you may prefer to leave this option off.
- **Allow in non-MDI editor control** - When selected, the macro is permitted to operate even 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 and Bind to Key** - Saves the recorded macro by appending the source code of the macro to the `vusrmacs.e` user macros file located in your configuration directory, then displays the Key Bindings option screen so you can create a keyboard shortcut for the macro. See [Binding Recorded Macros to](#)

[Keys](#) for more information.

- **Save** - Saves the recorded macro by appending the source code of the macro to the `vusrmacs.e` user macros file located in your configuration directory.
- **Edit** - (**Alt+E**) Displays the macro source code in a new editor window (to save it, click **Macros** → **Save last-macro** or to bind the macro to a key, use the menu item **Macro** → **List Macros**). Note that this button is disabled for existing macros because with the Save Macro dialog, you can only edit the macro you have just recorded prior to saving it. To edit a macro that has been previously recorded and saved, use the List Macros dialog. See [Saving and Editing Recorded Macros](#) for more information.
- **Delete** - Deletes the selected 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.



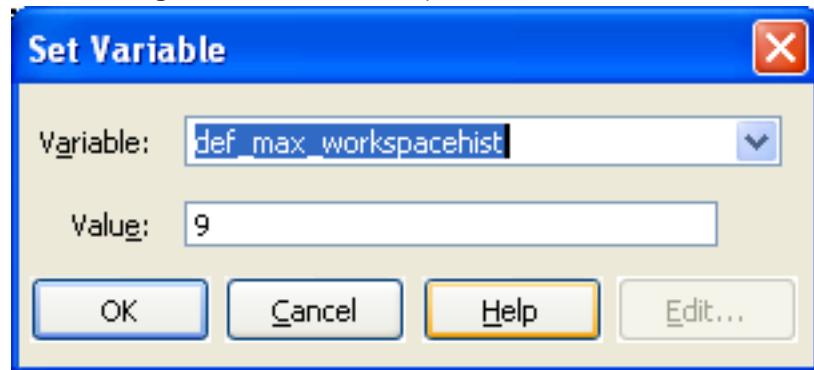
The dialog shows a list of all macros you have recorded. Use the buttons to perform the following operations:

- **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 option screen so you can assign a key or mouse shortcut to the macro. See [Binding Recorded Macros to Keys](#) for more information.

Set Variable Dialog

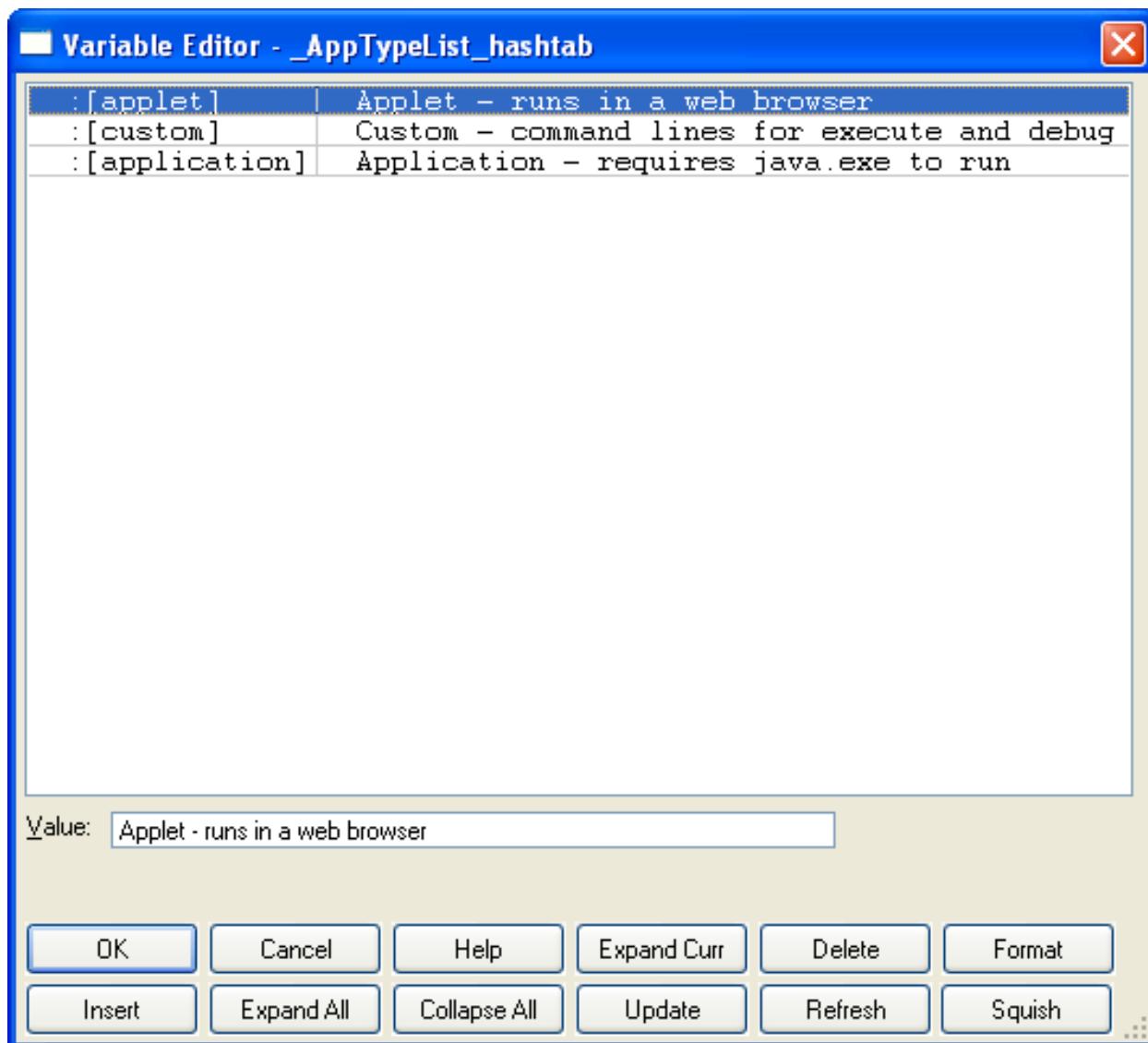
You can set Slick-C® variables to specific values using the Set Variable dialog box (**Macro → Set Macro Variable** or **gui_set_var** command).



Enter the name of Slick-C 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 and click **OK**, or click **Edit** to display the [Variable Editor Dialog](#), used for editing complex variables such as arrays, hash tables, structures, and unions.

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, click **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.

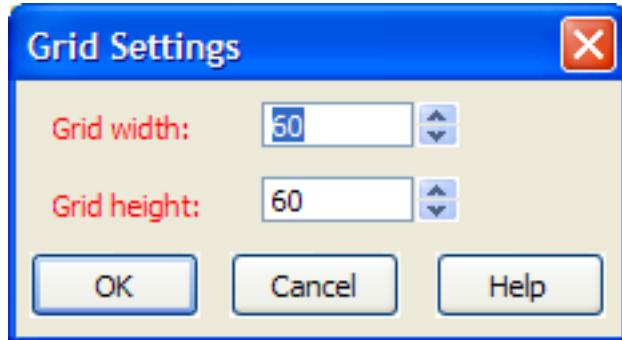
The following buttons are available:

- **Expand Curr** - Expands current item which has a **Plus (+)** 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** - Displays 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 (Pro only)

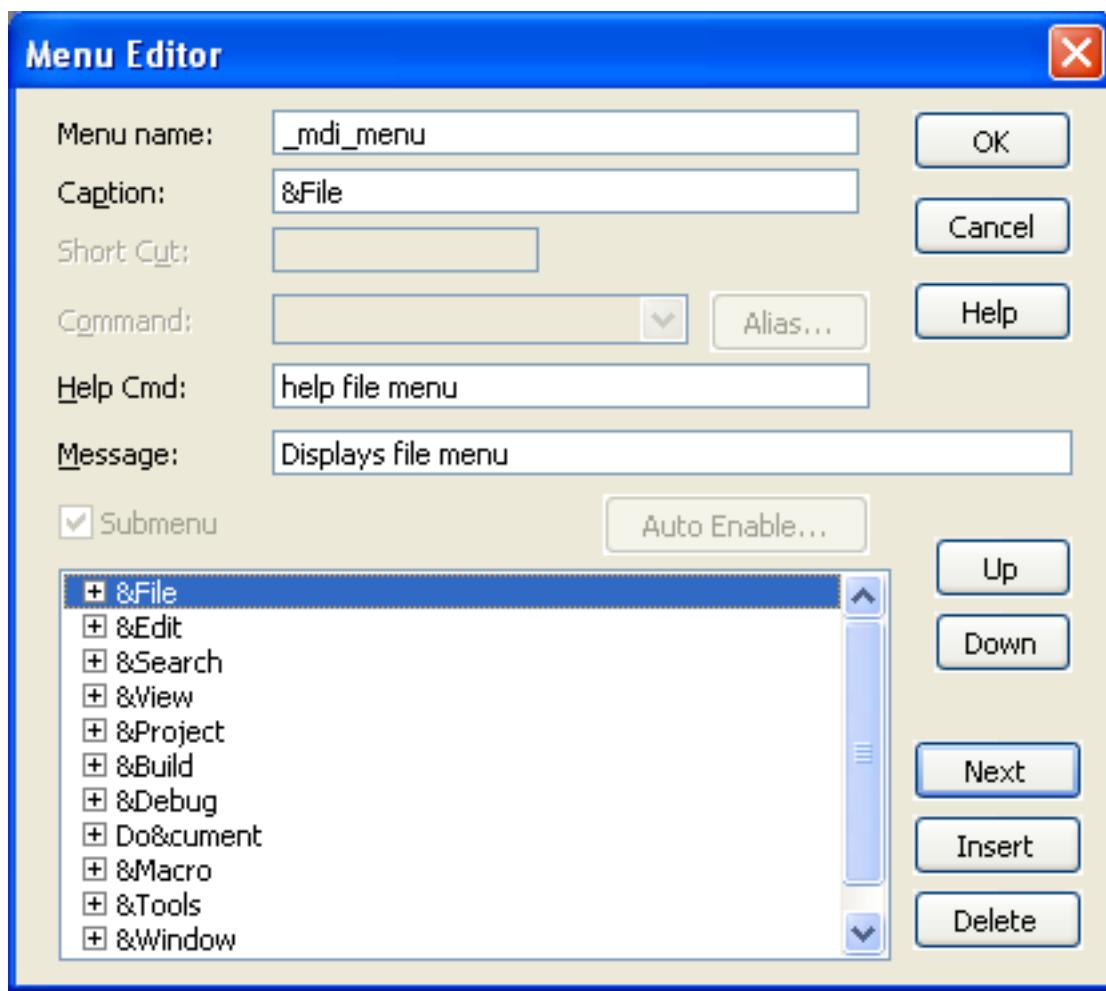
The Grid Settings dialog (**Macro → Grid** or **gui_grid** command) is used to set the width and height of grid dots displayed on forms when you use the Dialog Editor. These settings affect the distance between the dots on a form that is being edited.



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, click **Macro → Menus**, select the menu to edit from the list, then click **Open**.



The following fields and settings are available:

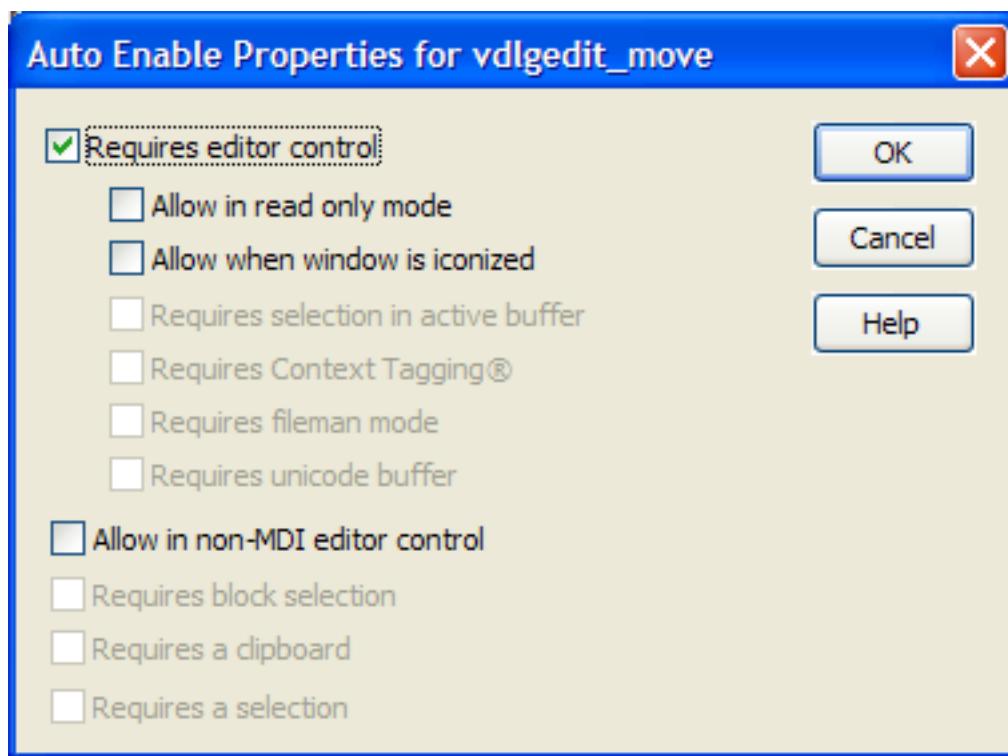
- **Menu name** - Name of the current menu resource. You can define your own menu resource which is used instead of our menu bar WITHOUT changing the name of our default menu bar **_mdi_menu**. Use the **-m** invocation option (for example, **-m mymenu**) or set the **def_mdi_menu** macro variable to your menu name (see [Setting/Changing Configuration 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 the Options dialog. See Options .	config
Quick Start Configuration	Displays the Quick Start Configuration Wizard. See Quick Start Configuration Wizard .	quick_start
Regex Evaluator	Shows the Regex Evaluator tool window. See The Regex Evaluator .	activate_regex_evaluator
OS Shell	Runs operating system command shell (DOS).	dos
OS File Browser	Runs operating system file system browser. See OS File Browser .	explore or finder
Calculator	Displays the Calculator, which allows you to evaluate mathematical expressions. See Using the Calculator and Math Commands .	calculator
Add Selected Expr	Adds the result of evaluating each line in a selected area of text.	add
ASCII Table	Opens ASCII table file.	ascii_table
Generate GUID	Generates a Globally Unique Identifier. See GUID Generator .	gui_insert_guid
Version Control	(Pro only) Displays Version Control menu. See Version Control Menu .	N/A

Tools Menu

Tools Menu Item	Description	Command
Quick Refactoring	(Pro only) Displays Quick Refactoring menu. See Quick Refactoring Menu .	N/A
Imports	(Pro only) Displays Imports refactoring menu. See Imports Menu .	N/A
Generate Debug Statement for	(Pro only) Generates debug code for symbol under the cursor.	<code>generate_debug</code>
Sort	Sorts current buffer or selected text. See Sorting Text .	<code>gui_sort</code>
Beautify	(Pro only) Displays Beautify menu. See Beautify Menu .	N/A
File Merge	(Pro only) Displays the 3-Way Merge Setup dialog, which provides settings for merging two sets of changes made to a file. See 3-Way Merge and Tools Dialogs and Tool Windows .	<code>merge</code>
File Difference	Displays the DIFFzilla® dialog, which allows you to view and edit differences between files. See DIFFzilla® and DIFFzilla® Dialog .	<code>diff</code>
Spell Check	Displays menu of spell checking commands. See Spell Check Menu .	N/A
Tag Files	(Pro only) Displays a dialog which allows you to build tag files for use by the Symbols tool window and other Context Tagging® features. See Creating Language-Specific Tag Files and Context Tagging - Tag Files Dialog .	<code>gui_make_tags</code>

Version Control Menu (Pro only)

The table below describes each item on the **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.	vccheckin
Get	Checks out current file read only.	vcget
Check Out	Checks out current file.	vccheckout
Lock	Locks the current file without checking out the file.	vclock
Unlock	Unlocks the current file without checking in the file.	vcunlock
Add	Adds current file to version control.	vcadd
Remove	Removes current file from version control.	vcremove
History	Views history for current file.	vchistory
Difference	Views differences of current file.	vcdiff
Properties	Views properties of current file.	vcproperties
Manager	Executes Version Control Manager.	vcmanager
Setup	Displays the Version Control Setup options screen, which allows you to choose and configure a Version Control System interface. See Version Control Setup Options .	vcsetup

Quick Refactoring Menu (Pro only)

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.	<code>refactor_quick_rename</code>
Extract Method	Extract the selected code block into a new function.	<code>refactor_quick_extract_method</code>
Modify Parameter List	Modify the parameter list of a method.	<code>refactor_quick_modify_params</code>
Encapsulate Field	Encapsulate field using Context Tagging®.	<code>refactor_quick_encapsulate_field</code>
Replace Literal with Constant	Replace literal value with a declared constant.	<code>refactor_quick_replace_literal</code>

Imports Menu (Pro only)

The **Tools** → **Imports** menu contains options for organizing Java imports. For more information, see [Organize Java Imports](#).

Imports Menu Item	Description	Command
Organize Imports	Organize import statements in a Java or C# file.	<code>refactor_organize_imports</code>
Add Import	Add import statement for symbol under cursor.	<code>refactor_add_import</code>
Go to Import	Jump to the import statement for symbol under cursor.	<code>refactor_goto_import</code>
Options	Displays the Options dialog open to Organize Imports node. See Organize Java Imports and Organize C# Imports .	<code>refactor_organize_imports_options</code>

Beautify Menu (Pro only)

The **Tools** → **Beautify** menu items are summarized in the table below. For more information about working with beautifiers, see [Beautifying Code](#).

Tools Menu

Beautify Menu Item	Description	Command
Beautify	Beautifies the current buffer with the current beautifier settings.	<code>beautify</code>
Beautify With	(Certain languages only) Submenu that allows you to beautify the current buffer with any of the known beautifier profiles. One caveat: your buffer always keeps the tab settings from your default profile, so if you use this menu item with a profile that has different tab settings, the indents may look off.	N/A
Edit Current Profile	(Certain languages only) Brings up the profile editor for the default beautifier profile for the buffer's language. When started from this menu, the profile editor also allows you to beautify the buffer as you're making changes to the profile.	<code>beautifier_edit_current_profile</code>
Options	Brings up the Beautifier options for the language in the current buffer.	<code>beautifier_options</code>
Beautifier Profile Overrides	Displays the beautifier profile overrides for the current buffer. Also, allows add/remove beautifier profiles to be used for files beneath a specific directory which will override the default beautifier profile.	<code>beautifier_edit_seeditorconfig</code>

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

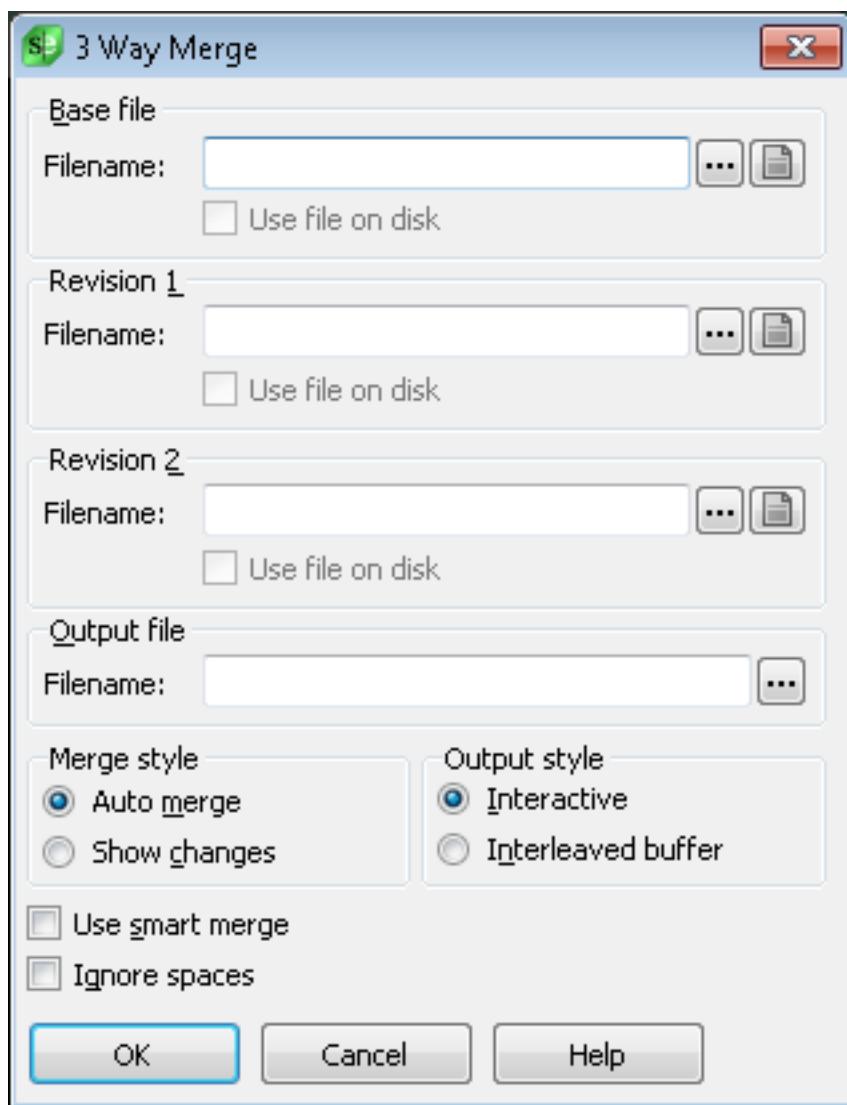
Spell Check Menu Item	Description	Command
Check from Cursor	Spell check starting from cursor.	spell_check
Check Comments and Strings	Spell check comments and strings starting from cursor.	spell_check_source
Check Selection	Spell check words in selection.	spell_check_selection
Check Word at Cursor	Spell check word at cursor.	spell_check_word
Check Files	Spell check multiple source files.	spell_check_files
Spell Options	Display/modify spell checker options.	spell_options

Tools Dialogs and Tool Windows

This section describes the dialogs and tool windows that are associated with the **Tools** menu items.

3-Way Merge Dialog (Pro only)

The 3-Way Merge dialog (**Tools > File Merge**), shown below, is used for merging file differences.



The **Ellipses** buttons to the right of the text boxes are used to select files. The **B**uttons 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 and2** - 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.

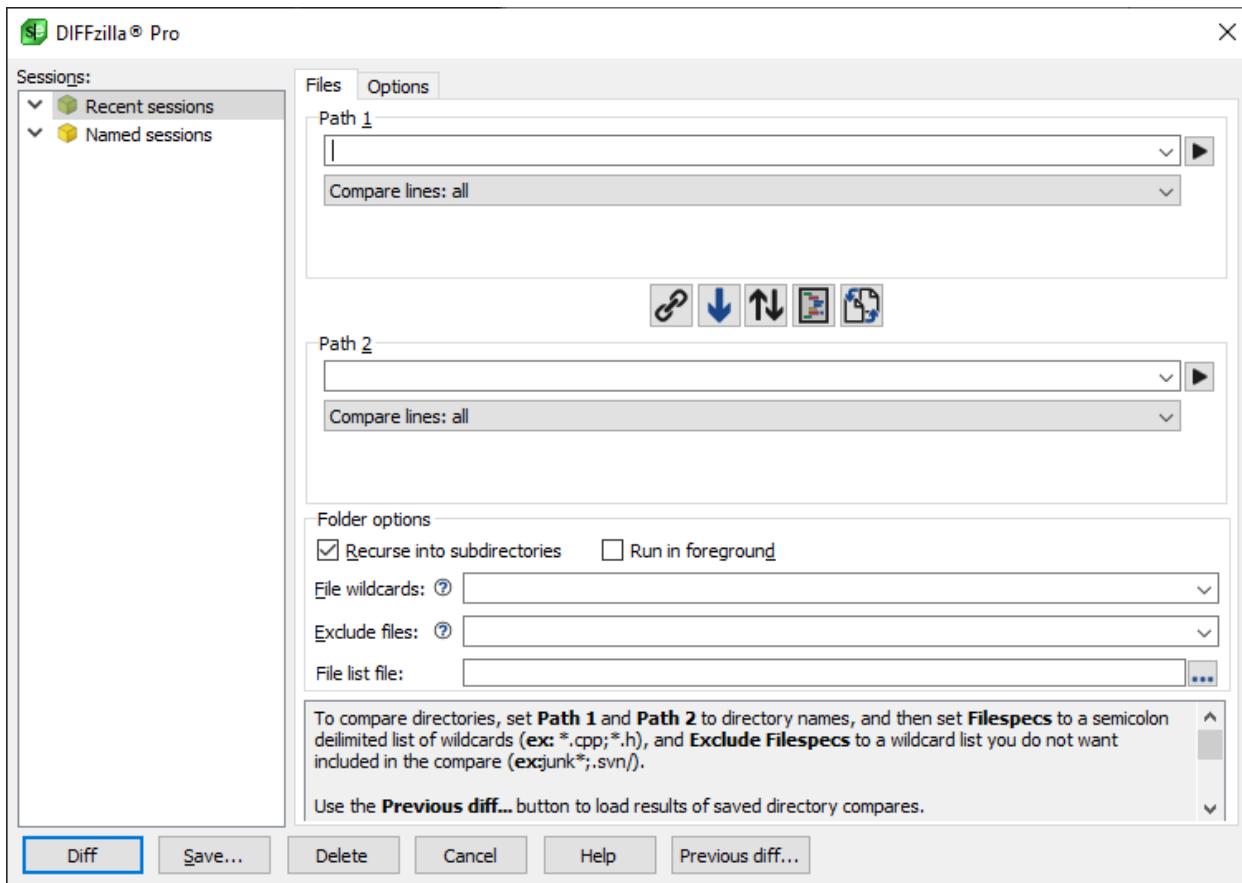
DIFFzilla

The DIFFzilla® dialog (**Tools** → **File Difference**) is used to configure a file differencing operation and begin the diff. The left side of the dialog contains a tree that shows recent diff sessions and sessions you have saved under an assigned name.

The **Sessions** tree records your last several diff sessions, and they are at the top of the tree. You can refill the dialog with the information from a previous session by clicking on it. You can save a session with a name by clicking the **Save As** button at the bottom of the dialog. If you have selected a named session from the bottom of the tree, you can save changes to it by clicking the **Save** button at the bottom of the dialog.

The dialog contains two tabs:

- [DIFFzilla® Files Tab](#) - used to select the items to compare.
- [DIFFzilla Options Tab](#) - used to specify options to control how the diff is performed and control the setup of the diff dialog.



DIFFzilla® Files Tab

Use this tab to specify the items to compare and the manner in which the comparison is performed. After filling in the needed information, click **OK** to start the diff.

Items to Compare

The dialog contains two areas used to specify the items to compare: Path 1 and Path 2. Items specified in the Path 1 section will appear on the left side of the diff output window. Items specified in the path 2 section will appear on the right side. For each you can specify the following items:

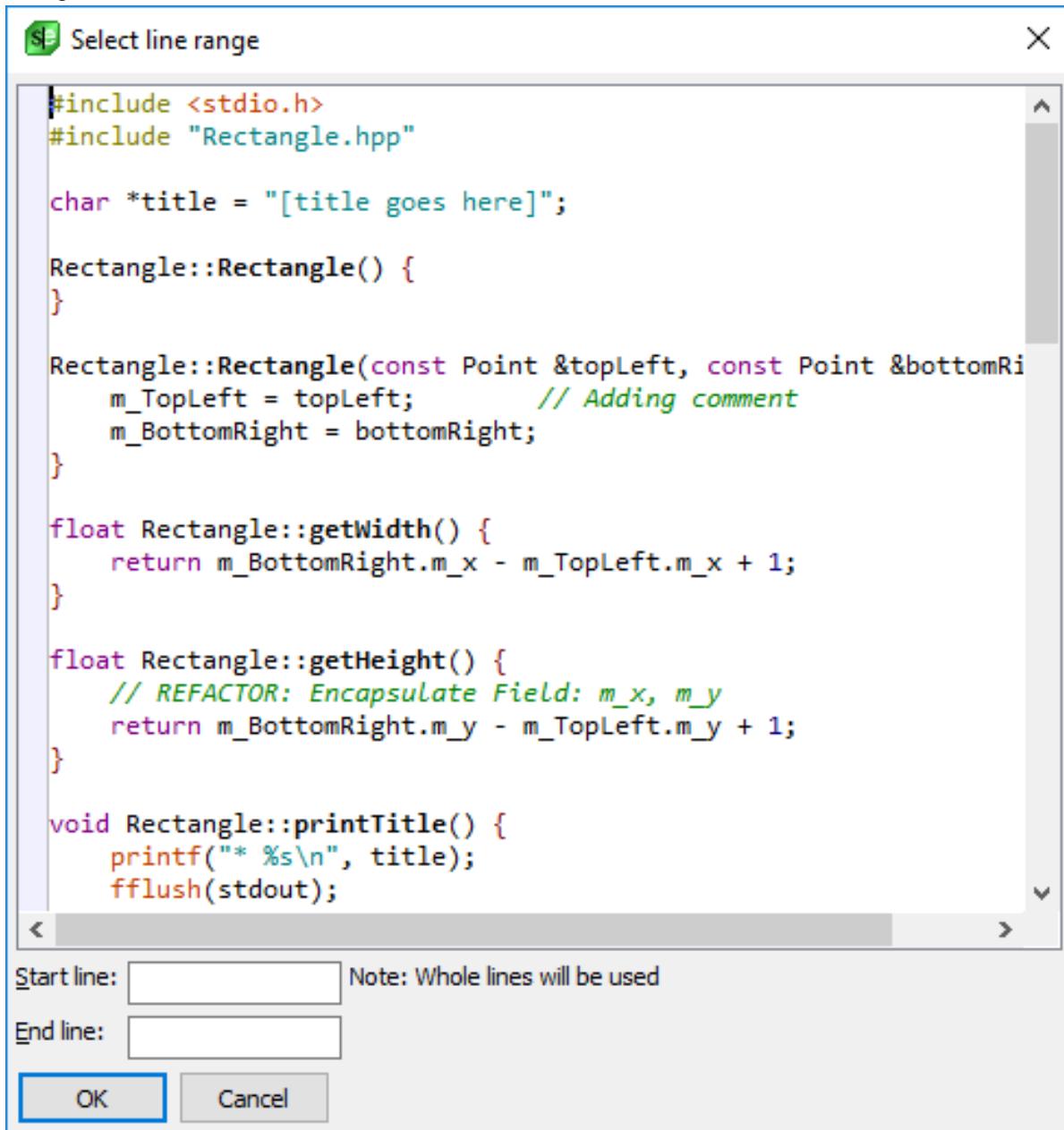
- **Path 1, Path 2** - When comparing files, set **Path 1** and **Path 2** to file names. When comparing folders, set **Path 1** and **Path 2** to directory names. If the file names only differ by path, you only need to specify a directory for **Path 2**.

Tip

By default, SlickEdit will automatically set the **Diff Type** based on whether the values for Path 1 and Path 2 contain directory names or file names.

You can use the drop-down list to select a previously used item to compare. To browse for a file or directory, click the **Ellipses** button. Click the **B** button to select an open buffer.

- **Compare type** - The second drop-down list lets you select the type of comparison to run. Select one of the following:
 - **Compare lines: all** - This is the default comparison type, comparing all of the lines in the specified files.
 - **Compare lines: range** - This option allows you to select a subset of the lines to compare, using the dialog, below.



- **Compare symbols: all** - (Pro only) This compares the symbols from the two files, ignoring differences in order. For example, if a function was declared higher up in the file in one version than the other, selecting this option would ignore that difference.

DIFFzilla Icons

In the center of the DIFFzilla dialog are icons that help to configure a comparison:

-  **Toggle automatic directory mapping** - When on, Path 2 is calculated based on other Path 2 directories used with the current Path 1 directory. The icon contains a little, red 'x' when off.
-  **Copy path** - Copies the path from Path 1 to Path 2 or vice versa. The direction of the copy is indicated by the arrow in the icon. When the arrow is pointing down, it will copy from Path 1 to Path 2. The direction is controlled by the location of the cursor, which designates the origin of the copy.
-  **Swap paths** - Click on this button to swap the paths from Path 1 to Path 2.
-  **Toggle Source Diff** - (Pro only) Turns Source Diff on or off. When Source Diff is off, the icon contains a small, red 'x'. Source Diff is on by default.
-  **Toggle compare contents** - Turns compare contents on or off. When compare contents is off, the icon contains a small, red 'x'. Compare contents is on by default. Only the filenames are compared when compare contents is off. This is useful for checking whether files exist or don't exist.

Folder options

For multi-file diff (differing two directories), you can set specific file types to compare or to exclude:

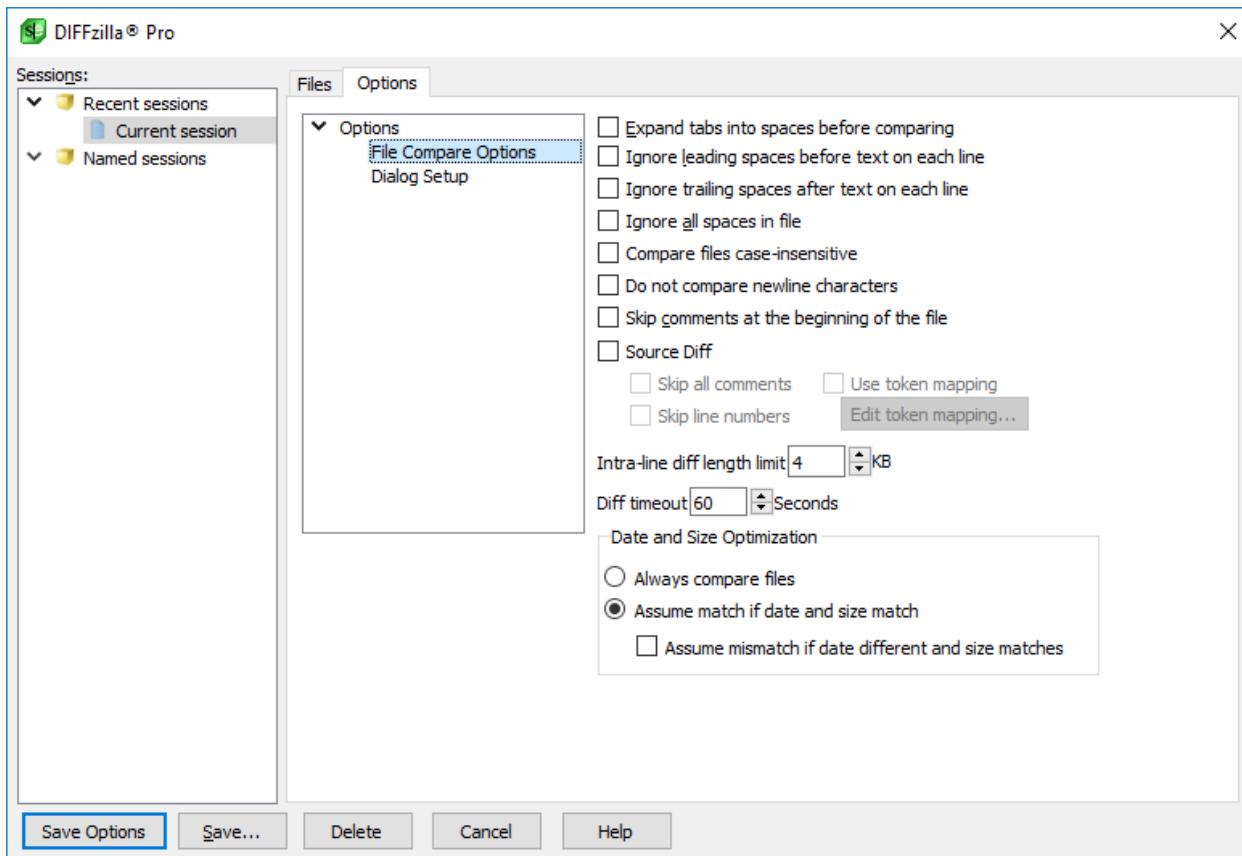
- **File wildcards** - Enter a semicolon-delimited list of wildcard file specifications to difference. For example, enter "* .c ; * .cpp ; * .h" to difference all files with .c, .cpp, and .h extensions.
- **Exclude files** - Enter a semicolon-delimited list of ant-like wildcard file specifications to be excluded from the differencing. For example, enter junk* ; test* to exclude all files with names beginning with the words "junk" or "test". To exclude a specific directory, provide a relative path to path1 and path2. To exclude any subdirectory with a particular name, put a slash at the end of the name. For example, enter ".svn/" to exclude all subdirectories named ".svn" wherever they occur. A more advanced ant-like wildcard can be used like "c* /" to exclude any directory that starts with "c". For more examples, see [Exclusion Examples](#)
- **File list file** - (Pro only) File containing relative filenames on separate lines to difference.

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 Compare Options](#) and [Dialog Setup Options](#).

File Compare Options

Tools Dialogs and Tool Windows



The file compare options, shown above, are described as follows:

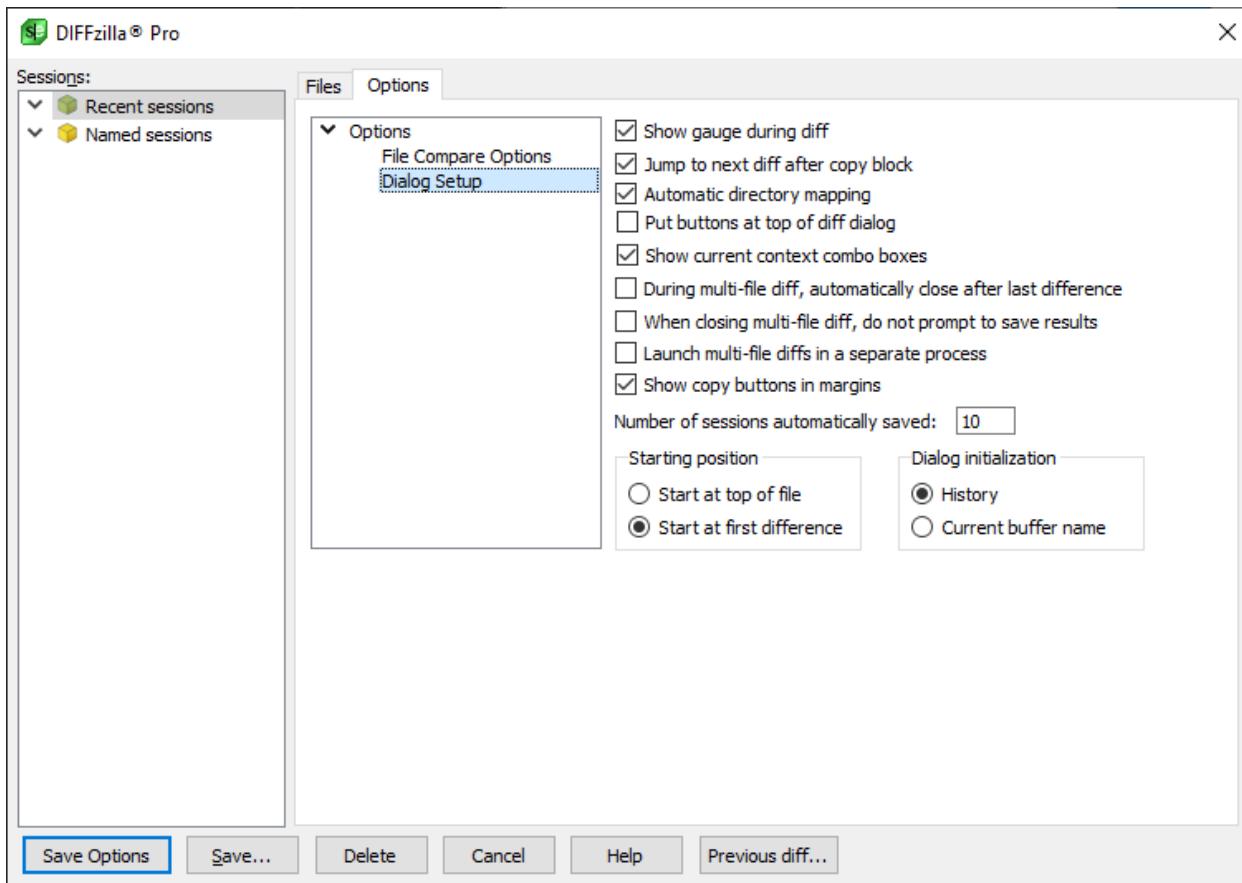
- **Expand tabs into spaces before comparing** - When selected, tabs are expanded to the appropriate number of spaces before lines from each file that is compared.
- **Ignore leading spaces before text on each line** - When selected, differences in leading spaces of lines are ignored.
- **Ignore trailing 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.
- **Source Diff** - (Pro only) Determines whether Source Diff is used. Source Diff compares the code temporarily reformatting the code from Path 2 to match the format in Path 1. This is very useful when one version has been both beautified and contains other meaningful changes. Source Diff can also be

toggled by an icon on the diff dialog.

- **Skip all comments** - (Pro only) When selected, all comments are ignored.
- **Skip line numbers** - (Pro only) When selected, line numbers are ignored (as in Cobol or Fortran).
- **Use token mapping** - (Pro only) When selected, use the current token exclusion mapping rules with **Source Diff** (for example, to ignore changes from renaming a symbol). See example below.
- **Edit token mapping** - (Pro only) Edit the set of token mapping rules. Each rule is a pair consisting of the new token text (how it appears in the source file corresponding to **Path 1**, and the original token text on the right-hand-side of the Diff dialog (how it appears in the source file corresponding to **Path 2**). See [Source Diff](#) for more information and an example.
- **Intra-line diff length limit** - If lines are shorter than this limit, perform intra-line diff.
- **Diff timeout** - Stop diff if it takes longer than this amount of seconds.
- **Date and Size Optimization** - (Pro only) These options control how DIFFzilla analyzes multi-file diffs.
 - **Always compare files** - When selected, the two files are always compared. This was how SlickEdit performed multi-file diffs in SlickEdit 2008 and earlier.
 - **Assume match if date and size match** - When selected, the two files are assumed to be the same if the date and size are the same. This will significantly speed multi-file diffs.
 - **Assume mismatch if date different and size matches** - Defines how to handle the case when the size is the same but the date differs.

Dialog Setup Options

Tools Dialogs and Tool Windows



Setup options for the DIFFzilla® dialog are described as follows:

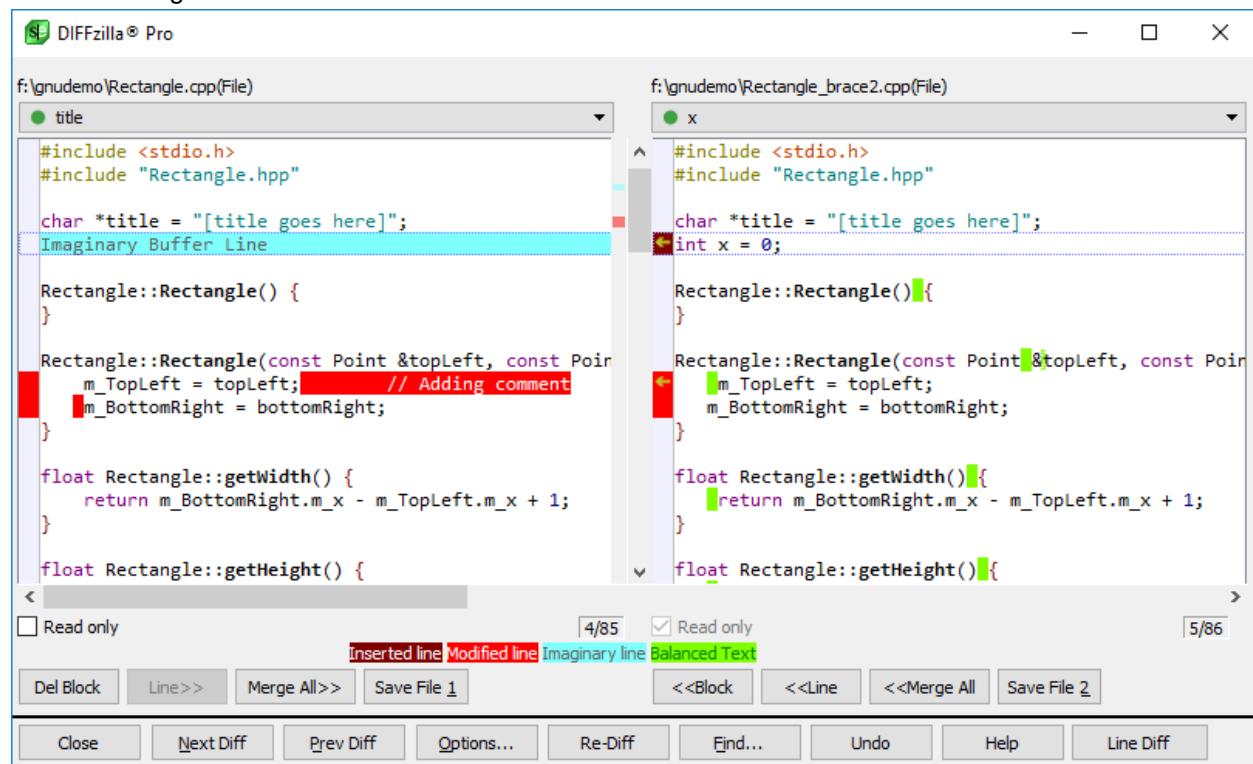
- **Show gauge during diff** - When selected, a gauge control will show various processing statistics while you wait for the differences output to complete.
- **Jump to next diff after copy block** - When selected, the cursor is moved to the next difference when you apply changes from one file to the 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.
- **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.
- **Show current context combo boxes** - When selected, the show combo boxes at the top of the dialog which show the current symbol under the cursor and allow to jump to another symbol quickly. When unselected, these combo boxes are hidden to save space.
- **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.
- **When closing multi-file diff, do not prompt to save results** - suppresses the prompt to save results after a multi-file diff.

- **Launch multi-file diffs in a separate process** - When selected, source trees are diffed in a separate process so you can continue working.
- **Show copy buttons in margins** - When this is on, arrows are displayed in the diff editor margins to allow you to transfer changes from right to left or left to right. If this is distracting, you can turn this off. Note that source diff only displays arrows on the right because the right side is read-only.
- **Number of sessions automatically saved** - Determines the number of sessions automatically saved under **Recent Sessions** in the **Sessions** tree. The default value is 10.
- **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.

DIFFzilla Diff Dialog

The **Diff** dialog is used to display the results of a file comparison. It displays the file from Path 1 on the left and the file from Path 2 on the right. Colored markers are used to indicate differences between the two files. Unlike most diff tools, the two panes are editable and support many of the same operations as the main editor windows, including syntax expansion and completions.

Imaginary Buffer Lines are inserted any time a line exists in only one file. This ensures that lines that are considered to be the same line (even if they contain changes) are displayed next to each other. This aids in viewing the differences.



The screenshot shows the DIFFzilla Pro application window titled "DIFFzilla® Pro". It displays a file comparison between two files: "f:\gnudemo\Rectangle.cpp" (File) and "f:\gnudemo\Rectangle_brace2.cpp" (File). The left pane shows the original code with an "Imaginary Buffer Line" highlighted in blue. The right pane shows the modified code with red and green markers indicating differences. The bottom bar includes buttons for "Read only", "Inserted line", "Modified line", "Balanced Text", and various save and diff options.

A contiguous set of the same type of differences (inserted lines or modified lines) is called a **block**. A block can be a whole line or several lines. If a particular code change to a file consists of three modified lines followed by three inserted lines, that comprises two blocks of three lines each. Operations you perform will act upon a block, a line, or the whole file.

The following buttons are displayed below each code pane in the **Diff** dialog. These are used to move changes from one side to the other or save the modified contents of the pane.

- **Del Block** - (Pro only) This button only appears when one side contains code that is not present on the other side. Click this button to delete the block.
- **Block** - (Pro only) Click this button to move the current block of differences from one side to the other.
- **Line** - (Pro only) Use this button to move the contents of the current line from one side to the other.
- **Merge All** - (Pro only) Merges all changes from the selected side to the other.
- **Save File** - (Pro only) Saves the file, including all changes made during the diff session.
- **Read only** - (Pro only) Lets you change the read-only status of the file.

The bottom of the **Diff** dialog contains the following buttons:

- **Close** - Close the **Diff** dialog. You will be prompted to save any unsaved changes.
- **Next Diff** - Moves to the next block of differences in the file.
- **Prev Diff** - Moves to the previous block of differences in the file.
- **Re-Diff** - Diffs the contents of the files again. This is typically used after some edits have been made where a re-diff will better align the source files.
- **Find** - Performs a search in the two files.
- **Undo** - (Pro only) Undo the last change. This includes changes made by editing and those made using the Block, Line, and Merge buttons.
- **Help** - Brings up Help on the **Diff** dialog.
- **Source Diff/Line Diff** - (Pro only) Switches from **Source Diff** to **Line Diff**. The button changes based on the kind of diff that was performed. If you did a **Code Diff**, the button will say **Line Diff** and vice versa.

To change the colors used, select **Tools** → **Options** → **Appearance** → **Colors**. The items to change are listed under the **Modifications** node in the list of elements. Set the following colors used by the **Diff** dialog:

- **Inserted Line** - Sets the color for lines that exist in one file but not the other. This color is displayed in the margin of the file that contains the inserted line. The other file will contain an **Imaginary Buffer Line**, to make sure that identical lines are always present next to each other.
- **Modified Line** - Sets the color lines that exist in both files but are different. Again, the color is rendered

in the margin of the file.

- **Modified Whitespace** - Sets the color used for whitespace adjustments created by **Source Diff**. The **Code Diff** capability adjusts the formatting of the Path 2 file to match that of the Path 1 file. Inserted or removed whitespace is shown with this color. For more information, see [Source Diff](#).
- **No Save Line** - Sets the color used for **Imaginary Buffer Lines**. These are inserted into the buffer to make sure that lines that are believed to be the same line (even if one has changed) are drawn next to each other. This helps when viewing and understanding differences.

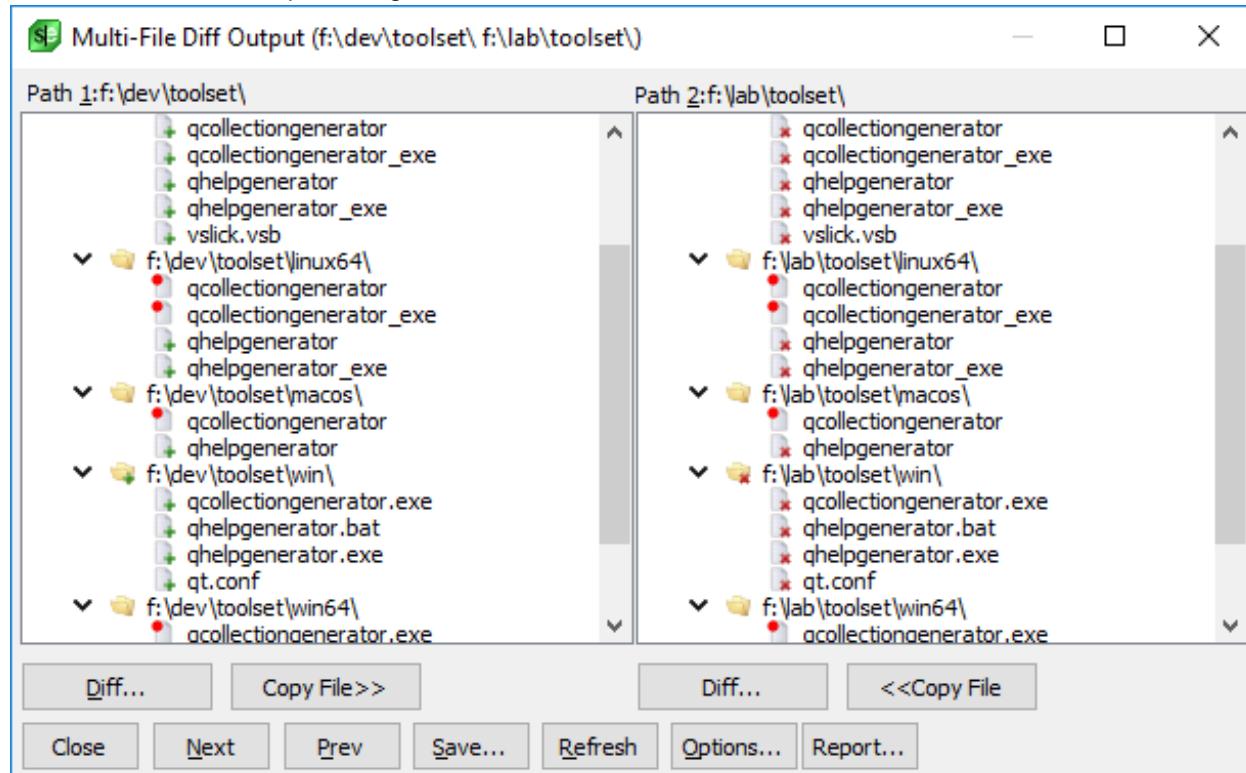
Caution

These colors are not used exclusively by the **Diff** dialog. They are used wherever that same information needs to be conveyed. For example, the **Modified Line** color is also used for margin markers when you enable the viewing of modified lines in the editor window.

For more information on setting colors see [Colors](#).

Multi-File Diff Output Dialog (Pro only)

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:

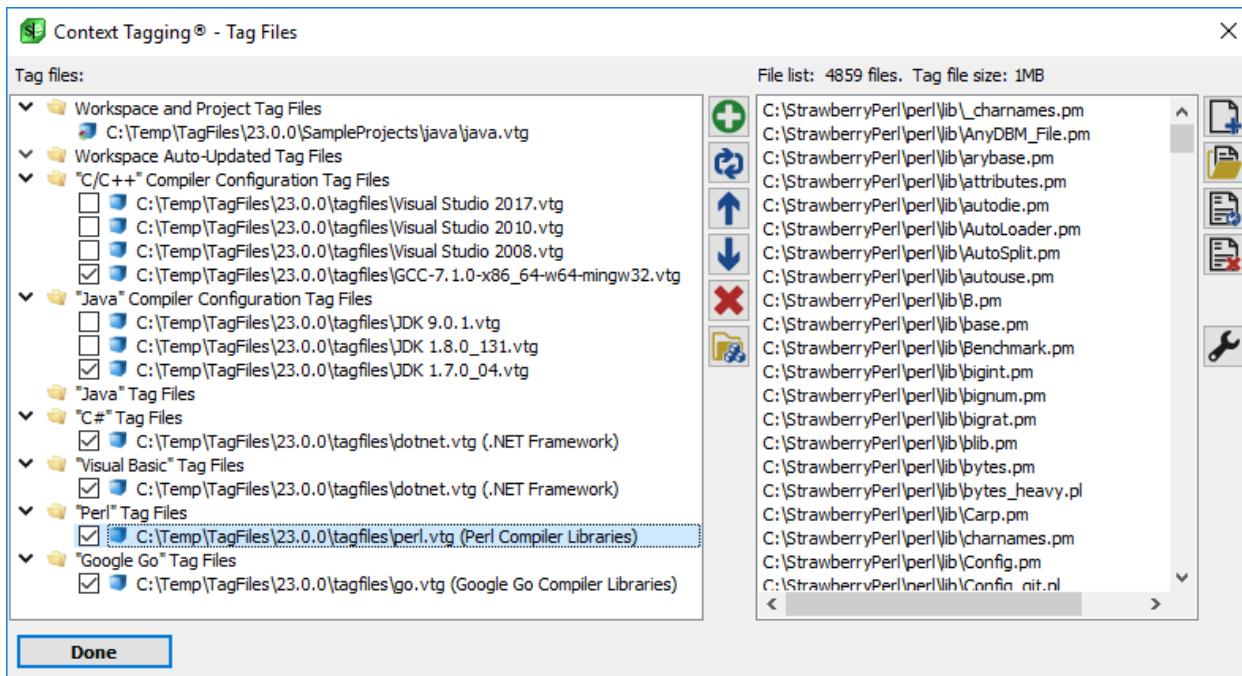
- **Diff** - Shows current files in the difference editor when the selected files differ.

- **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 **Plus** 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 (Pro only)

The Context Tagging® - Tag Files dialog, shown below, is used to manage all your tag files. For more information on tagging in general, see [Context Tagging Features](#). For more information about tag files, see [Building and Managing Tag Files](#). To access the Context Tagging® - Tag Files dialog, click **Tools** → **Tag Files**.

Tools Dialogs and Tool Windows



The left section of the dialog lists all of your tag files, separated into categories. A tag file having a **File** bitmap 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. They are divided into two columns, one for managing tag files, and one for managing the set of files in the currently selected tag file.

The following buttons are used for managing a list of tag files:

- **Plus - Add new or existing ag file** - Displays the Add Tag File dialog box, which allows you to choose from a list of languages for which to insert the tag file. For descriptions of the Add Tag File dialog box, see [Add Tag File dialog](#).

To automatically create tag files for C++, Java, and .NET, you can instead use the Create Tag Files for Compiler Libraries dialog (see [Creating Tag Files for Compiler-Specific Libraries](#)) or click on the auto-tag button to build compiler-specific or automatically generated language-specific tag files.

- **Refresh - Rebuild selected tag file** - Displays the Rebuild Tag File dialog box containing options for rebuilding the selected tag file. See [Rebuilding Tag Files](#).
- **Up** - Moves the selected tag file higher in the search order. This only applies to language-specific tag files (see [Creating Language-Specific Tag Files](#)) and workspace auto-updated tag files (see [Workspace Auto-Updated Tag Files](#)).
- **Down** - Moves the selected tag file lower in the search order. This only applies to language-specific tag files (see [Creating Language-Specific Tag Files](#)) and workspace auto-updated tag files (see [Workspace Auto-Updated Tag Files](#)).

- **Delete - Remove selected 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 language-specific tag files are automatically generated, and thus will be automatically regenerated if you delete them. This button will be unavailable for workspace and project tag files, as well as auto-updated tag files.
- **Auto Tag** - Displays the **Create Tag Files for Compiler Libraries** dialog box used to automatically create run-time library tag files for C++, Java, and .NET (see [Creating Tag Files for Compiler-Specific Libraries](#)).

The following buttons are used for managing the list of files in the currently selected tag file:

- **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 unavailable for read-only tag files, workspace and project tag files, and auto-updated tag files.

Note

By default, the **Add Source Files** dialog will not list files in the directory being viewed that are already included in the project. This makes it easier to locate files to add that are not yet part of the project. This feature can be turned off by changing [Open File Options](#).

- **Add Folder** - 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 unavailable for read-only tag files, workspace and project tag files, and auto-updated tag files.
- **Retag File(s)** - 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. This button will be unavailable for read-only tag files.
- **Remove File(s)** - 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. This button will be unavailable for read-only tag files, workspace and project tag files, and auto-updated tag files.
- **Options** - Displays the Context Tagging® Options screen for you to configure Context Tagging® options. See [Context Tagging® Options](#) for more information.

When invoked outside of the language-specific Tag Files options page, the **Tag Files** dialog will also have a **Done** button which saves tag file settings and closes the dialog box.

Note

Note that managing tag files is generally an active process, where you are creating and deleting tag files, adding and removing files, or rebuilding tag files concurrently as the dialog is active, so most modifications made in the **Tag Files** dialog can not be cancelled or undone.

In addition to the operations described above, the following menu options are available on the right-click context menu on the Context Tagging® - Tag Files dialog:

- **Edit Description** - Displays a dialog box where you can enter a text description for the currently selected tag file. This description will be shown in parenthesis in the list of tag files in the Tag Files dialog and the Symbols tool window.
- **Generate References** - This option will be checked if the currently selected tag file is built with symbol cross-referencing enabled. Unchecking the option will remove the symbol cross-referencing data. Likewise, checking the option will cause the tag file to be rebuilt with symbol cross-referencing enabled.
- **Set Workspace Tag Files Dir.** - Displays a directory chooser dialog where you can select a directory for workspace tag files (and project tag files) to be placed. This option is useful when you have workspaces that are on network drives or if you have a high-speed drive that you prefer to store your workspace and project tag files on for performance.

This option is also useful to avoid cluttering your workspace directory with tag files (in the case where you have several project-specific tag files or auto-updated tag files in your workspace).

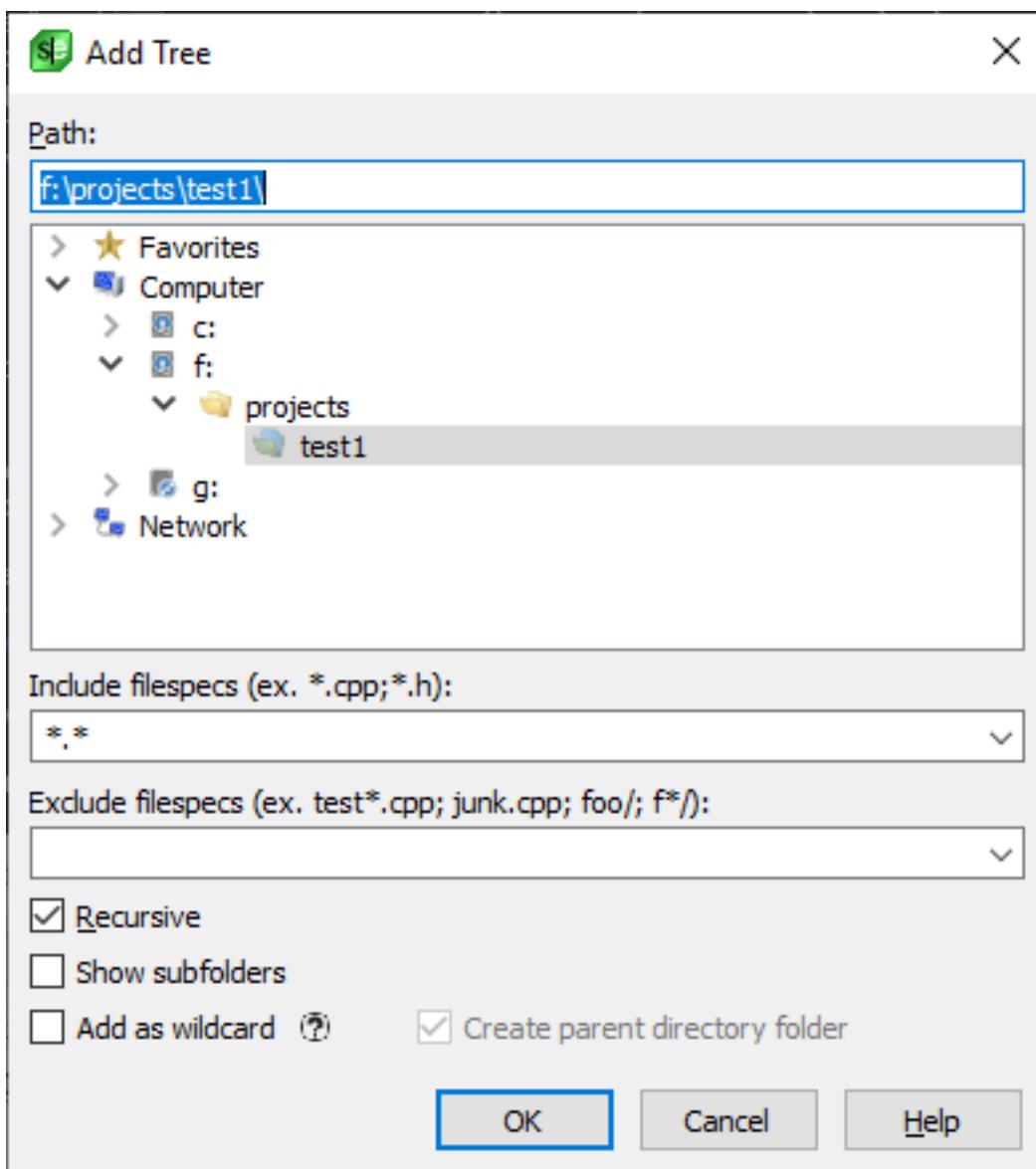
In addition, this option can be useful in order to avoid conflicts with other users when working with a workspace that is in a shared directory. By setting the workspace tagging directory to a location under your home directory or your SlickEdit configuration directory using an environment variable such as **%HOME** or **%(SLICKEDITCONFIG)**, you can insure that each user has a private copy of all the workspace tag files and the workspace history file.

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.

Tools Dialogs and Tool Windows



The dialog contains the following elements:

- **Path** - The **Path** text box lets you type out the path to the directory from which to include files. As you type, the first matching item is selected and expanded in the directory list.
- **Directory list** - The directory list box lets you pick the directory from which to add files.
- **Include filespecs** - The **Include Filespecs** combo box lets you select from predefined wildcard specifications or you can type your own. Each file spec should be separated with semicolons. For example, to include only Java files, select ***.java** from the predefined list. To include all files in a directory, type the wildcard *****. To customize the items in this list, see the [Files of Type Filter Options](#).
- **Exclude** - Use this combo box to exclude paths, files, or file types from the specified directory using ant-like wildcards. To specify multiple patterns, separate them with semicolons. No files are searched in a path that is excluded, including any files in sub-directories beneath. For examples, see [Exclusion](#)

[Examples](#) below.

- **Recursive** - If checked, the selected directory will be searched recursively.
- **Show subfolders** - If checked, project folders are created for each child directory when recursing files. Project folders are seen in the Project tool window when the project is in Custom View.
- **Add as wildcard** - When this box is unchecked, then the file tree will be traversed once and all files found at that time will be added as individual files. When this box is checked, then a wildcard specification is added instead, and any time a new file is added to the tree that matches the specifications will be included. While this is a good way to automatically keep your project updated, it can cause performance degradation.
- **Create parent directory folder** - Turn on the **Create parent directory folder** checkbox if you want a project folder created for the parent directory of a wildcard. For example, if the **Path** for this wildcard is **f:\project\source**, a **source** project folder will be created in the Projects tool window when in Custom View.

Exclusion Examples

SlickEdit supports Ant-like wildcards. It's essentially Ant syntax with some additional short hands. For example, when doing recursive file listing **path/** is treated as ****/path/**** and ***.cpp** is treated as ****/* .cpp**.

When specifying a list of file or directory exclusions, they need to be separated by semicolons. Wildcard expressions containing spaces can be placed in quotes.

Note

Absolute path file exclusions don't work. The exclusions need to be relative to the start path.

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 .
*.cpp	Exclude any file with extension .cpp .
.png;.ico;*.jpg	Exclude any file with extension .png , .ico , or .jpg .
.svn\	Exclude any files in paths named ".svn".
C*\	Exclude any files in paths that start with "C".

Example	Description
<code>**/b*/debug/**/backup/</code>	Exclude all files in this path name.
<code>*demo*</code>	Exclude any file (not directory) with "demo" in the name.
<code><Binary Files></code>	Exclude Binary files. All extensions defined by the Binary language are excluded. Also, unknown files which appear to contain binary data use the Binary language.

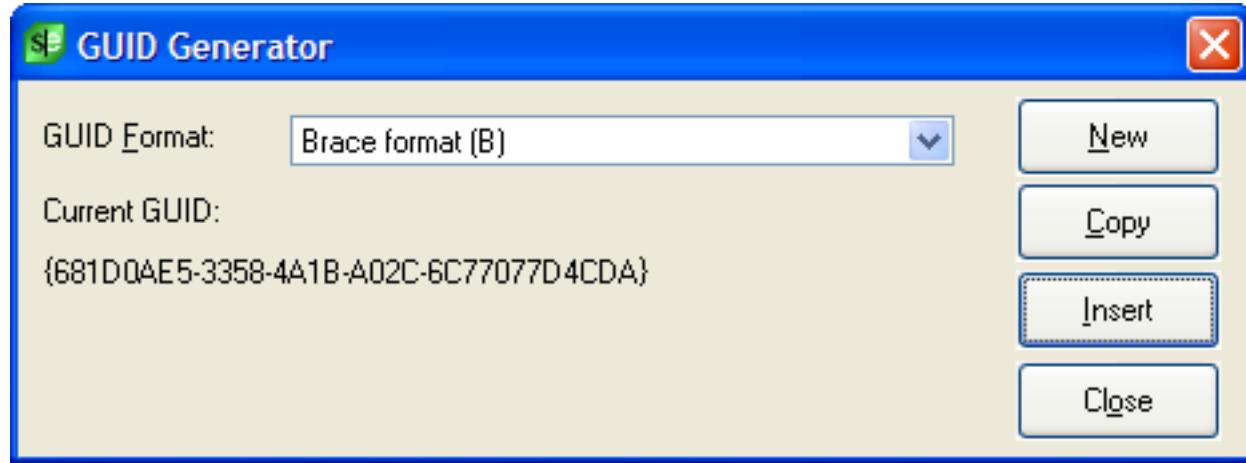
Workspace Tagging Excludes

The Workspace Tagging Excludes options allow you to specify directories of files to be excluded from tagging. by the currently selected tag file.

- **Add Full Path...** - Allows you to add an absolute path to be excluded from tagging.
- **Add Path Component...** - Allows you to add a partial path to be excluded from tagging. For example, "backup" would exclude all files beneath a backup directory.

GUID Generator

The GUID Generator (**Tools → Generate GUID**), shown below, is used to create Globally Unique Identifiers for use in your code.



The **GUID Generator** dialog contains the following fields and controls:

- **GUID Format** - Lets you select the format for the GUID.
- **Current GUID** - Displays the last created GUID. A new GUID is generated each time you invoke this dialog.

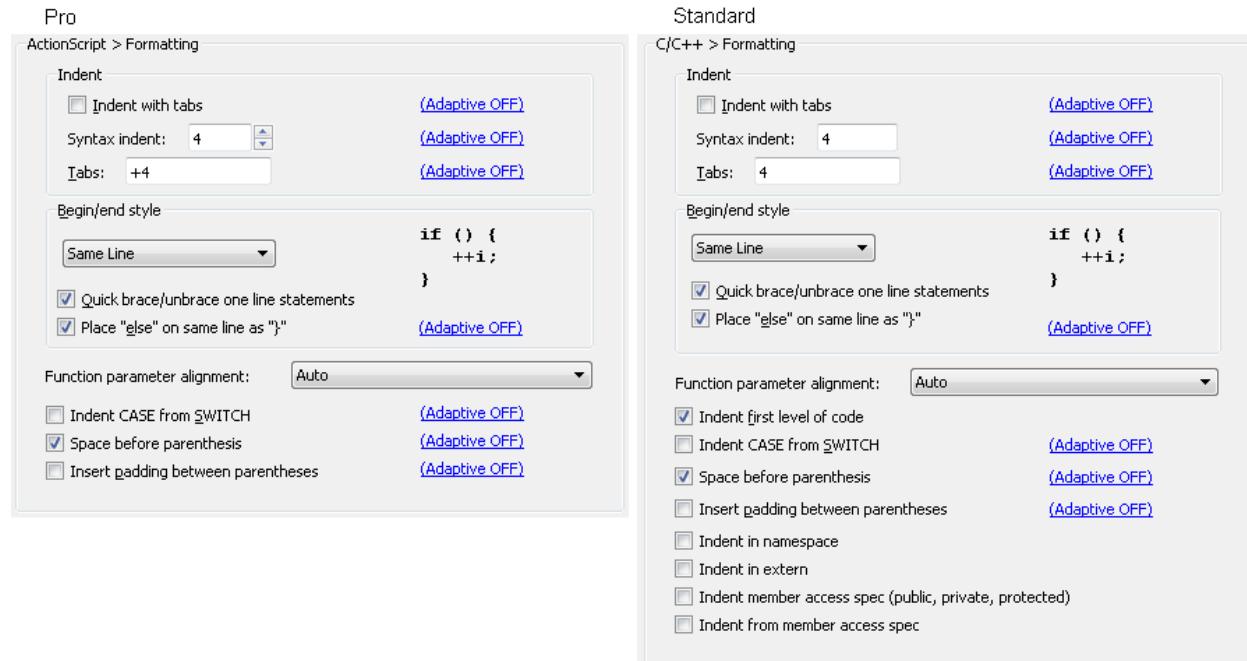
- **New** - Click this button to generate a new GUID.
- **Copy** - Click this button to copy the current GUID to the clipboard.
- **Insert** - Click this button to insert the current GUID into the active buffer at the cursor location.

Common Formatting Options for Brace-style Languages

Some common formatting options are available for brace-style languages which do not have beautifiers (ActionScript, Ansi-C, AWK, Batch, CFScript, CH, D, Google Go, IDL, J#, Perl, Ruby, Slick-C, Tcl, Vera, and Windows PowerShell). These option include the [Syntax Indent](#) and [Syntax Expansion](#) style settings. To access these options, from the main menu, click **Tools** → **Options** → **Languages**, choose a language, and click **Formatting**.

Note

Languages similar to ActionScript have similar Formatting Options screens that are not specifically documented.



The following settings are available:

- **Indent with tabs** - Determines whether **Tab** key, **Enter** key, and paragraph reformat commands indent with spaces or tabs. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Indenting with Tabs](#) for more information.
- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Syntax Indent](#) for more information.

- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **Begin/end style** - Specify the brace style to be used for [Syntax Indent](#) and [Syntax Expansion](#). The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. After specifying the brace style, choose from the following options:
 - **Quick brace/unbrace one line statements** - Enables Quick Brace/Unbrace, features that allow you to convert a single line statement to a brace-enclosed block, and vice versa. See [Quick Brace/Unbrace](#) for more information.
 - **Place "else" on same line as "}"** - When this option is selected, SlickEdit® places the **else** keyword on the same line as **}**. This is common when using brace Style 1.
- **Indent first level of code** - Specifies whether [Syntax Indent](#) should indent the cursor after declarations such as functions.
- **Function parameter alignment** - Determines whether function parameters should use the continuation indent or be aligned to the parenthesis. When "Auto" is selected, continuation indent is used unless the first parameter is on the same line as the open parenthesis.
- **Indent CASE from SWITCH** - When checked, [Syntax Expansion](#) places the case statement indented from the switch statement column. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **Space before parenthesis** - 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 ()** The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **Insert padding between parentheses** - 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 ()**. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.
- **Indent namespace** - Determines whether text inside namespace is indented.
- **Indent in extern** - Determines whether text extern braces is indented.
- **Indent member access spec (public, private, protected)** - Determines whether member access specifier is indented (public: indented from class keyword).
- **Indent from member access spec** - Determines whether declarations are indented from the member access specifier (i.e. indented from public:).

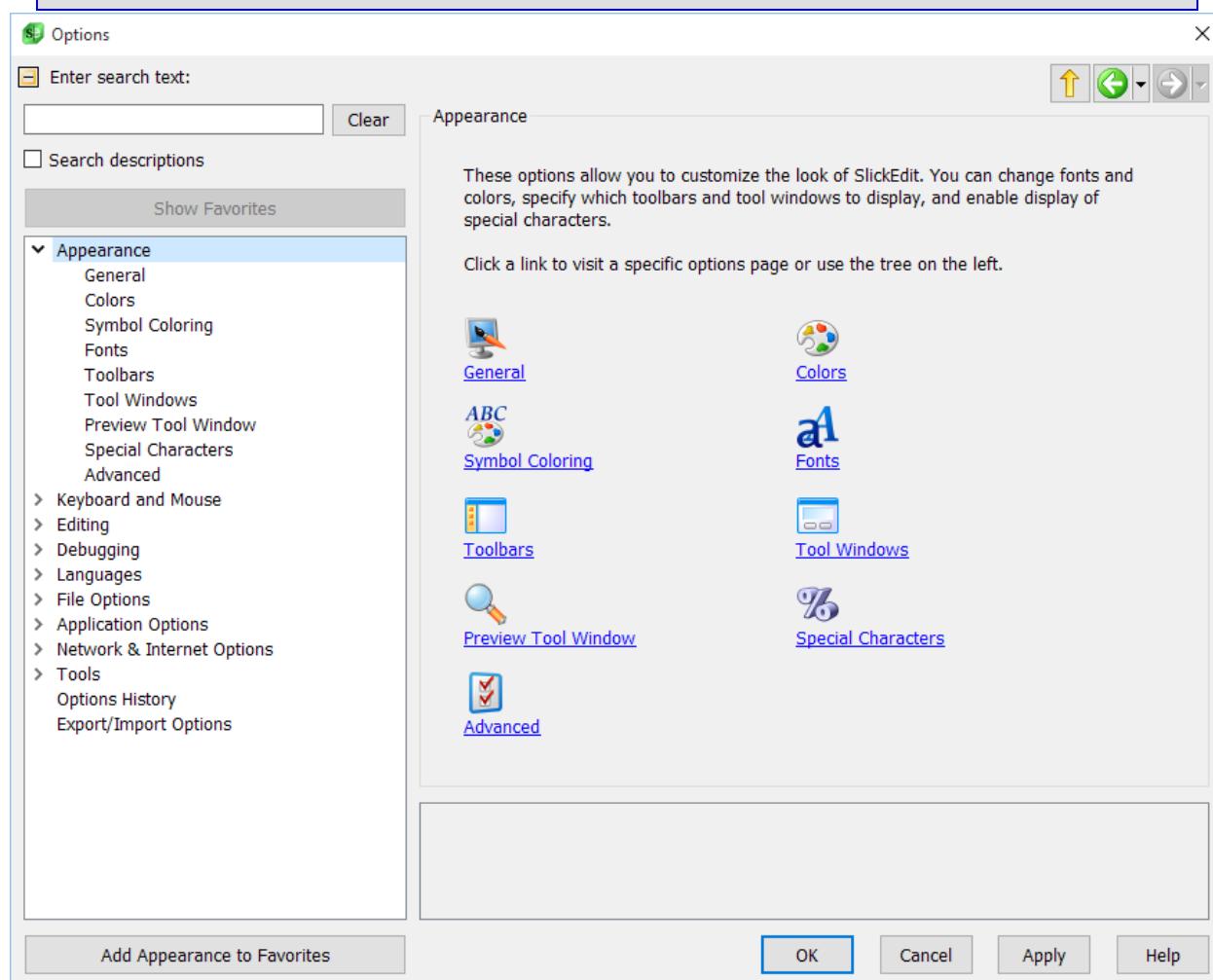
Options

Options Dialog

The Options dialog is used to configure SlickEdit®. To display it, from the main menu, click **Tools** → **Options**, or use the **config** command on the SlickEdit® command line.

Note

The Options dialog displayed below is Options dialog for the Pro edition. The Debugging and Symbol Coloring options are not available in the Standard and Community editions.



The following sections describe how to use the Options dialog. For descriptions of individual options, skip to the index located at [Option Categories](#).

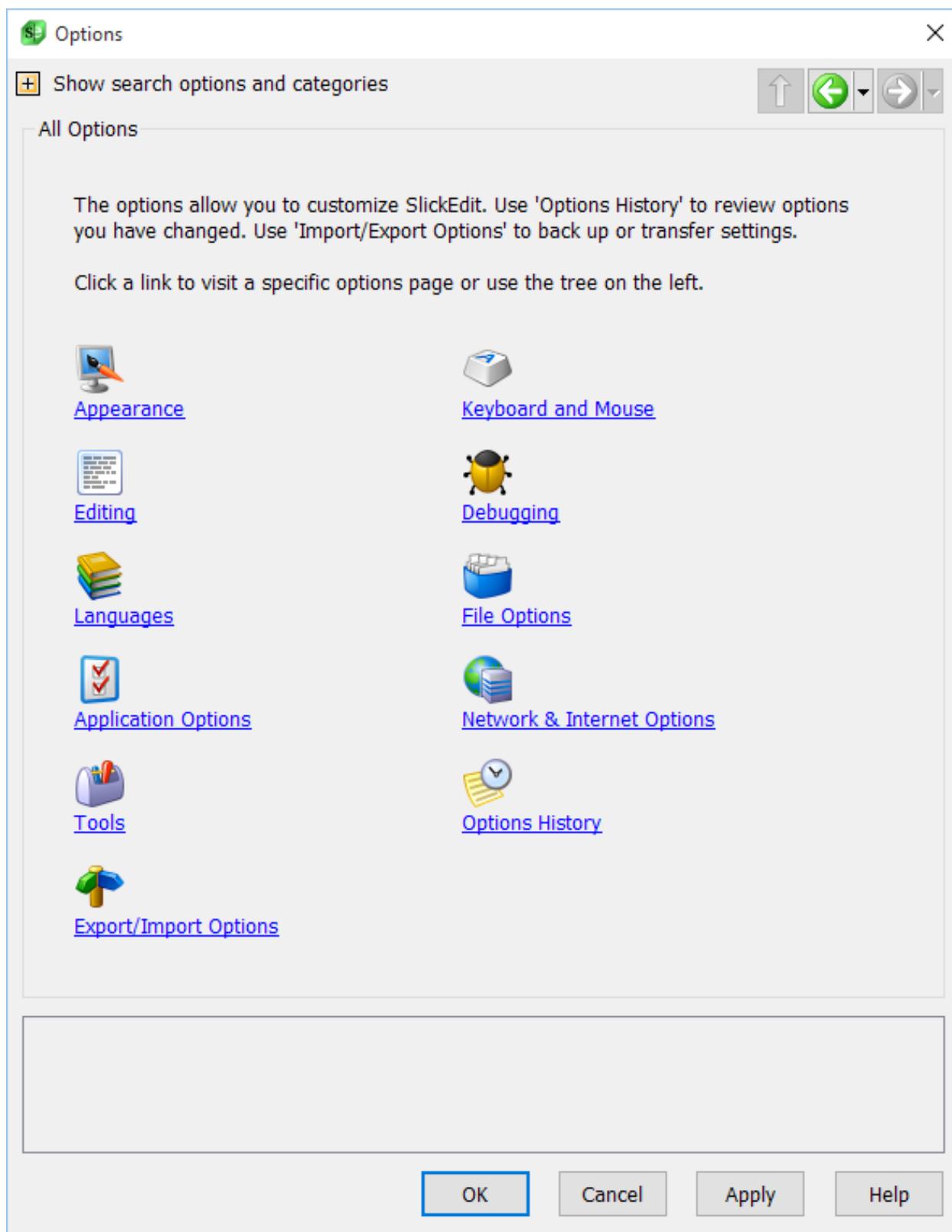
Using the Options Dialog

The Options dialog is divided into two sections: the tree on the left, which contains category nodes (see [Option Categories](#)), and the option panel on the right, which contains specific options. Right-click inside the tree area to expand or collapse all nodes in the tree.

As you click on nodes, the option panel updates to show the applicable options. The layout of the option panel can be either a property sheet in tabular format (like **Tools** → **Options** → **Appearance** → **General**), or a form with radio buttons, check boxes, etc. (like **Tools** → **Options** → **Appearance** → **Colors**).

The tree on the left, which contains category nodes, as well as the options search filters, can be collapsed to conserve screen space. When the tree is collapsed, you can still navigate between panels by clicking on their icons and by using the Up, Back, and Forward keys at the top of the Options dialog.

Options Dialog



Tip

The Options dialog supports keyboard shortcuts, so you can use the dialog and change settings without having to touch the mouse. See [Keyboard Shortcuts in the Options Dialog](#) for more information.

See the following sections for more information:

- [Section Changing and Applying Option Settings](#)
- [Navigating to Previously Viewed Panels](#)
- [Option Favorites](#)
- [Option Search](#)
- [Keyboard Shortcuts in the Options Dialog](#)

Changing and Applying Option Settings

When options are displayed as a property sheet, the name of the option is shown in the **Option** column, and the setting is shown in the **Value** column. The manner in which an option value is changed depends on the type of option:

- **For options with switches** - Click anywhere on the switch to toggle it off and on.
- **For options with combo boxes** - Use the drop-down arrow to make a selection. Alternately, to cycle through and select a setting, double-click on the option. If you're using keyboard shortcuts, you can toggle a combo box up and down using **F4** or **Alt+Up** and **Alt+Down**, respectively, and then use the **Up** and **Down** keys to make a selection.
- **For options with numeric text boxes** - Click on the option and type directly in the text box to change the value. If you're using keyboard shortcuts, use one of the keys **Right**, **Space**, or **F4** to enable the text box.
- **For color options** - Double-click on the color block or press **F4**, and the color picker is displayed.
- **For file and directory path options** - Double-click on the value or press **F4**, and a directory picker is displayed. If you have specified a file name or directory in a field and wish to change it back to the default, press the **Delete** key. This clears the value and SlickEdit® will use the default setting.

For both forms and property sheets, when you change the value of an option, an asterisk appears after the node name in the tree. If the option is in a property sheet, an asterisk also appears next to the option name. This helps you see the options that have changed when you've made a lot of settings changes at once. The asterisks remain until you click **Apply** or close the dialog. To see changes made in previous dialog sessions, click the **Options History** node in the tree. See [Options History](#) for more information.

Click **Apply** to save option changes and leave the dialog open, or click **OK** to save the changes and close the dialog. If you attempt to close the dialog by any other means, and if changes have been made but not yet applied, you are prompted to save the changes. When the Options dialog is closed, the view is saved and restored the next time the dialog is opened.

Navigating to Previously Viewed Panels

Some options link to other option panels, like options for [Adaptive Formatting](#). To make it easy to get back to an option panel after clicking a link, **Backward** and **Forward** navigation buttons are available along the top right side of the dialog. These buttons let you navigate between previously viewed panels, similar to the Windows Explorer back/forward navigation features. Click the drop-down arrows to see lists of the previously viewed option panel. Click on an item in one of the lists to navigate to the panel directly.

Tip

- If your mouse has Back and Forward navigation buttons, you can use them to navigate through previously viewed option panels.
- If you prefer to use keyboard shortcuts, use **Alt+Left** and **Alt+Right** to navigate backward and forward, respectively. See [Keyboard Shortcuts in the Options Dialog](#) for more information about keyboard navigation.

Option Favorites

You can mark a frequently used options page as a favorite for quicker access. To add the current page to your favorites list, click the **Add [OptionsPage] to Favorites** button located at the bottom of the tree. After adding favorites, click the **Show Favorites** button located above the tree, and the tree changes to only show a list of nodes which are favorites or parents of favorites. Favorite nodes are displayed in bold type. To remove the selected favorite, click **Remove from Favorites**. To reset the tree to the default view, click **Show All**. Note that initiating a search while viewing Favorites causes the tree to reset as well.

Option Search

The dialog lets you search for keywords throughout the options with an incremental search field located at the top left of the dialog under **Enter search text**. As you type each character in the search box, the tree is filtered to show only nodes that contain the search text. The node that contains a match is displayed in bold type in the tree.

To perform a more in-depth search, check the **Search descriptions** checkbox. In the case that you do not find what you are looking for using the regular search, this will enable searching of help information included in the options dialog. This search will likely yield more results. Since the additional results will be more difficult to go through, this search is off by default.

To reset the tree and clear the search box, click the **Clear** button or press **Alt+C**. Note that viewing Favorites resets the tree and clears the search box as well.

After you have initiated a search, to see a list of search results, select the **Search Results** node. This node only appears in the tree (at the bottom of the list) after you have started a search. The search results are divided into columns showing the option name and the path to the option in the dialog. Double-click on an option to navigate to the corresponding option page. For options in forms embedded in the Options dialog, the results only show the name of the form. For example, if the **Fixed Fonts Only** option on the **Fonts** form was a match, the results show "Fonts" as the option name and "Appearance" as the

path.

Keyboard Shortcuts in the Options Dialog

If you prefer to keep your hands on the keyboard, invoke the Options dialog with the **config** command and use keyboard shortcuts to navigate and change options.

Some quick tips:

- Each time the dialog is displayed, the focus is in the **Search** box at the top left of the dialog. Use **Tab** to navigate through the elements on the dialog.
- Use the **Right** and **Left** keys to expand and collapse nodes in the tree.
- Prefix matching is supported for tree nodes and in property sheets. For example, if the **Appearance** node category is expanded and focus is in the tree, you can type "F" and the first node beginning with that letter is selected (**Fonts**). Likewise, if focus is at the top of the **Tools** → **Options** → **Appearance** → **General** property sheet, you can type "C" and the first option beginning with "C" is selected (**Cursor style**). Note that only the current visible hierarchy of the tree is searched, and on property sheets, property group headings are not included in the search.
- Use **Alt+O** to jump from the option tree to the option panel.
- On property sheets, use **F4** to toggle combo boxes up and down, enable numeric text boxes, and display color and directory pickers.
- Press **Enter** to save changes and close the Options dialog, or **Esc** to prompt for changes before closing the dialog.

The table below describes all of the available shortcuts.

Summary	Keyboard Shortcut(s)	Description
Navigation	Alt+[letter] or Alt+[number]	Performs the following actions: <ul style="list-style-type: none">• If neither form nor property sheet is visible, jumps to the first Alt-prefixed shortcut on the Options dialog proper (for example, Alt+C corresponds to the Clear button next to the Search field). Note that for most operating systems, Alt-prefixed shortcuts correspond to the underlined letters in labels on forms and dialogs.• If a form is visible, jumps to the first Alt-prefixed shortcut on the form (for example, in the

Options Dialog

Summary	Keyboard Shortcut(s)	Description
		<p>Appearance Colors form, Alt+E corresponds to the Screen element list).</p> <ul style="list-style-type: none"> If a property sheet is visible, jumps to the first option starting with that letter or number (for example, on the Tools → Options → Appearance → General property sheet, Alt+T selects the first option that starts with a "T", which is Top of file line).
Navigation	Tab	Moves the focus to each area/button on the Options dialog. In a form, Tab moves between each area/button on the form until reaching the last element and placing focus back on the Options dialog proper. In a property sheet, Tab jumps back to the dialog proper.
Navigation	Up and Down	In the tree, moves up and down the visible nodes. In property sheets, moves up and down line-by-line. In combo boxes, moves up and down through the available settings.
Navigation (dialog to panel)	Shift+Tab	Jumps from the OK button on the Options dialog to the currently visible option panel.
Navigation (tree to panel)	Alt+O	Shifts focus from the tree to the currently visible option panel. Subsequently, if the panel is a property sheet, use the Up and Down keys to navigate through the options. If the panel is a form, use Tab to navigate through the elements on the form.
Navigation (between panels)	Alt+Left and Alt+Right	Navigates backward and forward,

Options Dialog

Summary	Keyboard Shortcut(s)	Description
		respectively, between previously viewed option panels. Same as using the Back and Forward buttons on the Options dialog.
Navigation or exit	Enter	Saves changes and closes the Options dialog, or, if inside a text box or combo box, shifts focus back to the option panel. Same as pressing OK on the Options dialog.
Navigation or exit	Esc	Prompts to save any changes and closes the Options dialog, or, inside a text box or combo box, shifts focus back to the option panel. Same as pressing Cancel on the Options dialog.
Expands/collapses tree nodes and enables property sheet controls	Right and Left	Expands/collapses nodes in the tree. In property sheets, enables numeric text boxes and drops down combo boxes (but not up).
Enables property sheet controls	F4	Toggles combo boxes up and down, enables numeric text boxes, and displays color and directory pickers.
Enables property sheet controls	Alt+Up and Alt+Down	Toggles combo boxes on property sheets up and down, respectively.
Enables property sheet controls	Space	In property sheets, enables a numeric text box and drops down combo boxes (but not up). In the Option Search results or Options History nodes, displays the selected option in the option panel (same as double-clicking).
Scrolling	Ctrl+Down and Ctrl+Up	Scrolls the option tree and long property sheets one line at a time.
Scrolling	PageUp and PageDown	Scrolls the option tree and long property sheets one page at a

Summary	Keyboard Shortcut(s)	Description
		time.
Clears a directory/file name property	Delete	Clears a directory or file name field, resetting the default value.
Option search	Alt+C	Clears the Search box on the Options dialog. Same as using the Clear button. See Option Search for more information.
Option favorites	Alt+F	Adds or removes the selected node to or from your favorites list. Same as using the Add to Favorites or Remove from Favorites button on the Options dialog. See Option Favorites for more information.
Option favorites	Alt+S	Trims the tree to show only your favorite option nodes (same as the Show Favorites button), or displays all nodes again when viewing favorites (same as the Show All button). See Option Favorites for more information.
Save changes	Alt+A	Saves all option changes yet leaves the Options dialog displayed. Same as using the Apply button on the Options dialog.
Help	Alt+H	Displays the Help topic for the Options dialog.

Option Categories

The tree in the Options dialog (**Tools** → **Options**) contains the following category nodes, which are described in subsequent sections:

- [Appearance Options](#)
- [Keyboard and Mouse Options](#)

- [Editing Options](#)
- [Debugging](#)
- [Language Options](#)
- [File Options](#)
- [Application Options](#)
- [Network and Internet Options](#)
- [Tool Options](#)
- [Options History](#)

Apearance Options

Apearance options (**Tools → Options → Apearance**) allow you to customize the look of SlickEdit®. You can change fonts and colors, specify which toolbars and tool windows to display, enable display of special characters, and more. Apearance option categories are:

- [General Apearance Options](#)
- [Color Options](#)
- [Symbol Coloring Options](#)
- [Font Options](#)
- [Toolbar Options](#)
- [Tool Windows Options](#)
- [Preview Tool Window](#)
- [Special Character Options](#)
- [Advanced Apearance Options](#)

General Apearance Options

General apearance options are shown below (**Tools → Options → Apearance → General**).

Appearance Options

General	
Option	Value
Application theme	Automatic
Cursor and Mouse	
Cursor style	Use vertical cursor
Cursor blink period (ms)	0
Hide mouse pointer	<input type="radio"/> OFF <input checked="" type="radio"/>
Show number base popups	<input checked="" type="radio"/> ON <input type="radio"/>
Margins	
Window left margin	0.15
* Vertical line columns	80 132
Vertical line color	<input type="color" value="#ff0000"/>
Top of file line	<input type="radio"/> OFF <input checked="" type="radio"/>
File Navigation	
List command line completions	<input checked="" type="radio"/> ON <input type="radio"/>
Change directory	<input checked="" type="radio"/> ON <input type="radio"/>
Show dot files	<input checked="" type="radio"/> ON <input type="radio"/>
Underline URLs	<input checked="" type="radio"/> ON <input type="radio"/>
Build Window	
Process recognized xterm color output in build window	<input checked="" type="radio"/> ON <input type="radio"/>
Scroll bars	
Horizontal scroll bar	<input checked="" type="radio"/> ON <input type="radio"/>
Vertical scroll bar	<input checked="" type="radio"/> ON <input type="radio"/>
Limit horizontal scroll	<input checked="" type="radio"/> ON <input type="radio"/>
Scroll style	
Smooth horizontal scroll	<input type="radio"/> OFF <input checked="" type="radio"/>
Smooth vertical scroll	<input checked="" type="radio"/> ON <input type="radio"/>
Scroll when	2
Current line highlight	
Current line highlight	None
Current line box color	<input type="color" value="#0000ff"/>
Current line column color	<input type="color" value="#333333"/>
Selective Display	
Selective Display bracketing	<input type="radio"/> OFF <input checked="" type="radio"/>
Selective Display line color	<input type="color" value="#668d4c"/>
Maximum nesting level	25
Minimum level to collapse	20
Message list colors	
Message visited color	<input type="color" value="#00ffff"/>
Message modified color	<input type="color" value="#ff0000"/>

The options are described as follows:

- **Application theme** - Effects the color scheme used by all controls except the edit window. Set to **Dark** to have tool windows drawn with a dark background color. Specifies One of the following settings:
 - **Automatic** - On Mac and Windows, this specifies that the **Dark** application theme should be selected at startup if the OS application theme is configured for Dark mode. This option is not available on Unix.
 - **System** - On Windows, this specifies the OS theme. On Mac, a default theme intended to match the OS theme is used. On Unix, a theme is chosen similar to GTK but this can be configured with a command line switch.
 - **Dark** - Specifies a dark theme designed for SlickEdit.
- **Cursor and Mouse**
 - **Cursor style** - Specifies the style of the cursor (block/text mode style, or vertical).
 - **Cursor blink period** - Specifies the period of cursor blinks in milliseconds. Set this value to **0** to use the default value for the OS.
 - **Hide mouse pointer** - When set to **On**, the mouse pointer is hidden when typing, but visible when moving the mouse or when a dialog box is displayed.
 - **Show number base popups** - When set to **On**, hovering over a number in source code will show the number in the common number bases (hex, octal, binary, decimal).
- **Margins**
 - **Window left margin** - Specifies the amount of space, in inches, between the left edge of the window and the editor text. This option has no effect when there are bitmaps displayed in the left margin, since more space may be necessary to accommodate the size of the bitmap.
 - **Vertical line columns** - Specifies the columns in which the editor is to display a vertical line. You can show multiple lines by specifying each column, separated by a space. Set the value to **0** to have no vertical lines. The vertical line is not displayed in Unicode files or when using proportional fonts.
 - **Vertical line color** - Specifies the color of the vertical line when it is displayed.
 - **Top of file line** - When set to **On**, a line 0 is inserted at the top of each buffer with the text "Top of File". Does not affect lines of code. Note that rather than setting this option, you can press **Ctrl+Shift+Enter** (**Ctrl+Enter** in the Visual C++ and Visual Studio default emulations) to insert a new line above the line where the cursor is located.
- **File Navigation**
 - **List command line completions** - When set to **On**, a pop-up list of possible commands and argument completions is displayed for partially typed commands and arguments on the SlickEdit® command line. See [Command Line Completion](#) for more information.

- **Change directory** - When set to **On**, the current directory is changed in the editor when the directory is changed in the Change Directory dialog (**File** → **Change Directory**) and the Open, Save Copy As, and Save As dialogs (**File** → **Open**, **File** → **Save Copy As** and **File** → **Save As**).
- **Show dot files** - When set to **On**, displays dot files in the Open tool window and Open, Save Copy As, and Save As dialogs. On Windows, the Open and Save As dialogs ignore this option and always display dot files. This value is also controlled by the configuration variable `def_filelist_show_dotfiles`.
- **Underline URLs** - When set to **On**, URLs in and editor window are underlined.
- **Build Window**
 - **Process recognized xterm color output in Build Window** - When set to **On**, recognized color escape sequences in the Build Window are processed. For example, "`g++ -fdiagnostics-color=always myfile.cpp`" or "`clang++ -fdiagnostics-color=always myfile.cpp`" will display colored error messages in the Build window. Cursor movement and non-color escape sequences are not supported. For Unix and macOS, using SoftWrap in the Build Window disables support for bold, italics, and underline.
- **Localization**
 - **Date display style** - (Unix and Mac only) Determines how the date is displayed. This option effects printing, alias dates, and template dates. Unfortunately, this option does not effect dates displayed in a tree control which several dialogs use. It also does not affect how the date is displayed in the file manager.
 - **Time display style** - (Unix and Mac only) Determines how the time is displayed. This option effects printing, alias times, and template times. Unfortunately, this option does not effect times displayed in a tree control which several dialogs use. It also does not affect how the time is displayed in the file manager.
- **Scroll bars**
 - **Horizontal scroll bar** - When set to **On**, each editor window displays a horizontal scroll bar. This does not affect edit window controls on dialog boxes.
 - **Vertical scroll bar** - When set to **On**, each editor window displays a vertical scroll bar. This does not affect edit window controls on dialog boxes.
- **Scroll style** options include:
 - **Smooth horizontal scroll** - When set to **On**, editor windows scroll column-by-column when the cursor moves out of view. When set to **Off**, the cursor is centered and the text is scrolled one-fourth the width of the window when the cursor moves out of view.
 - **Smooth vertical scroll** - When set to **On**, editor windows scroll line-by-line when the cursor moves out of view. When set to **Off**, the cursor is centered and the text is 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.

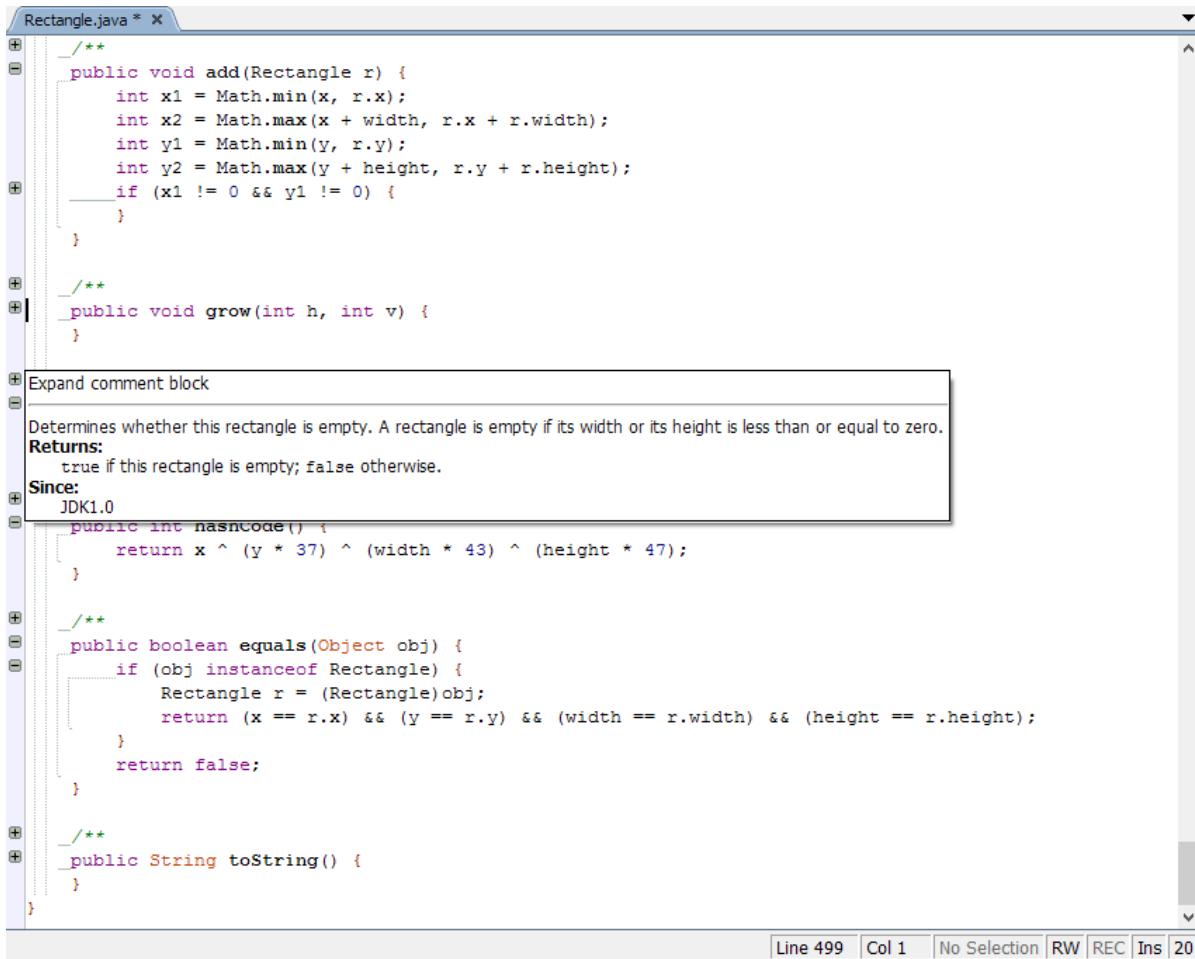
- **Current line highlight** options include:

- **Current line highlight** - Specifies the type of highlight to be drawn around the current line. When **None** is selected, the current line is not highlighted. When **Draw box only** is selected, a dotted box is drawn around the current line. When **Tabs ruler** is selected, a box is drawn around the current line with tab stops marked. When **Syntax indent ruler** is selected, a box is drawn around the current line with Syntax Indent levels marked. When **Decimal ruler** is selected, a box is drawn around the current line with marks at multiples of five and 10. For Unicode files or when using proportional fonts, only a box will be drawn.
- **Current line box color** - Specifies the color of the box outline when **Current line highlight** is enabled.
- **Current line column color** - Specifies the color for column markers when using a current line highlight with column indicators (Tabs, Syntax Indent, or Decimal rulers). Note that this is the same as the margin color.

- **Selective Display** options include:

- **Selective Display bracketing** - Specifies whether or not to draw lines to illustrate Selective Display regions. For more information, see [Selective Display](#) and [Selective Display on file open](#).

The following example shows what the selective display bracketing looks like. Note that solid horizontal connector lines indicate a collapsed selective display region.

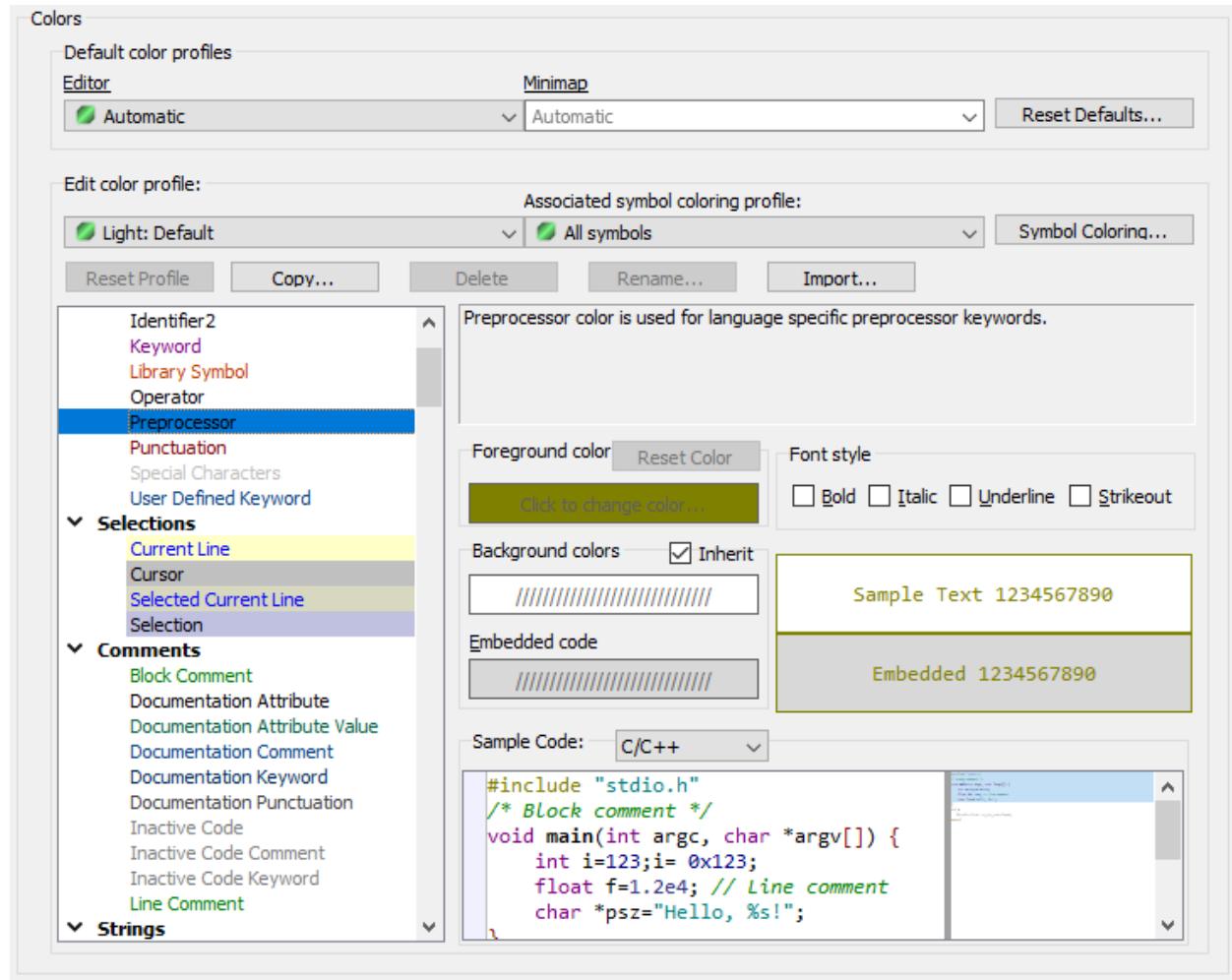


- **Selective Display line color** - Specifies the color of the lines drawn to illustrate Selective Display regions when **Selective Display bracketing** is enabled.
- **Maximum nesting level** - Specifies the maximum level of nesting for Selective Display outlining to create. For more information, see [Multi-Level outline](#) and [Selective Display on file open](#).
- **Minimum level to collapse** - Specifies the minimum level of nesting for Selective Display outlining to leave expanded. Any region above this level will be collapsed. For more information, see [Multi-Level outline](#) and [Selective Display on file open](#).
- **Message list colors** - Select from the following options that pertain to the [Message List](#) feature:
 - **Message visited color** - Specifies the color of the message in the Message List tool window after you have navigated to the associated code location.
 - **Message modified color** - Specifies the color of the message in the Message List tool window after the line at the associated code location has been modified.

Color Options

Appearance Options

Color options are shown below (**Tools** → **Options** → **Appearance** → **Colors**). These options let you specify colors for screen elements in SlickEdit® and create and manage color profiles. See [Colors](#) for more information.



The options are described as follows:

- **Default color profiles** - Specify the color profiles to use as the global default for editor windows and minimap windows. Default color profiles can be overridden on a per-language basis.

Several predefined profiles are available or you can define or import your own. See [Using Color Profiles](#) for more information. Color profiles are categorized as **Dark** or **Light** depending on the darkness of the window background color used.

- **Editor** - Specify the color profile to use from the drop-down list. This will be the global default for all editor windows. The default for this setting is **Automatic**, which indicates that the editor color profile should be chosen automatically based on the application theme. See [Application theme](#) for more information.
- **Minimap** - Specify the color profile to use from the drop-down list. This will be the global default for all minimap windows. The default for this setting is blank, which means to use the same color profile as used for editor windows.

Note

Because minimap text is generally too small to read, using a softer or more subdued color profile, like **Grayscale**, for the minimap can be less distracting. It can also be helpful to select a color profile with distinct background colors for documentation comments to provide a visual clue where documented code blocks start.

- **Reset Defaults...** - Allows you to quickly reset the selected Editor and Minimap default color profiles back to either the original settings when the Color Options was invoked, or to the installation defaults.
- **Edit color profile** - This section of the dialog is used to edit color elements in the selected profile.
 - **Profile** - Specify the color profile to edit from the drop-down list. By default, this field will be initialized to the color profile used by the current editor control.
 - **Associated symbol coloring profile** - (Pro only) Each color profile is associated with a corresponding symbol coloring profile. Because symbol coloring can define custom colors for symbol coloring rules, this is necessary in order to ensure that the symbol colors are compatible with the base colors in the color profile. When you switch color profiles, you also switch symbol coloring profiles.

(Pro only) Press the **Symbol Coloring...** button to jump to the Symbol Coloring Options dialog to take a closer look at the available profiles. For more information about symbol coloring, see [Symbol Coloring](#).

Note

The predefined symbol coloring rule bases shipped with SlickEdit do not define any custom colors. All the colors are inherited from symbol colors defined in the color profile. This allows you to use the predefined symbol coloring profiles without concern for their compatibility with specific color profiles.

This somewhat makes the **Associated symbol coloring profile** obsolete, but it is still necessary when using user-defined symbol coloring profiles that define custom colors which may not be compatible with all color profiles.

- **Managing profiles**
 - **Reset Profile...** - Allows you to reset all the changes made to a color profile since the Color Options was invoked, or to reset a predefined SlickEdit color profile back to its default configuration. All color changes to the selected profile are saved when you exit the dialog or switch profiles.
 - **Copy...** - Allows you to copy the current color settings as a new color profile with a name you specify.
 - **Delete** - Deletes the selected color profile. Only available for user defined color profiles.

- **Rename...** - Allows you to rename the selected color profile. Only available for user defined color profiles.
- **Import...** - Import a color theme from another product such as Visual Studio, Eclipse, Xcode, Sublime Text, VSCode, or TextMate. See [Importing Color Profiles](#) for more information.
- **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. For a complete list of color elements, see [Color Elements](#).
- **Canvas/Window Text** - This is the color of other text which is not a specific syntax element. This background color and embedded background color defined for **WindowText** is the color that is inherited by any other color that inherits its background color.
- **Attribute (XML and HTML only)** - This is the color used for a recognized attribute of an XML or HTML tag. For example, in HTML, the **src** attribute of the **img** tag gets this color.
- **Cursor** - This 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 only a slight tint from your normal background color, ensuring that the color-coded text is still easy to read.
- **Foreground color**
 - **Foreground color** - Click the color square to change the text color 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.
 - **Reset Color** - Resets the currently selected color back to its original value when the Color Options dialog was invoked. For predefined SlickEdit color profiles, reset the currently selected color to its original installation defaults.
- **Background color**
 - **Background color** - Click the color square to change the background color 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.
 - **Inherit** - Select the **Inherit** option to specify that the background color and embedded code color should be inherited from the basic **Window Text** canvas color. This feature can be used to keep the background colors synchronized among editor color elements.
 - **Embedded code** - Click the color square to change the background color for use when displaying the selected element in embedded code. The Color Picker dialog is displayed, allowing you to pick a color from the palette or set your own custom color using RGB values.

The **Embedded code** option is used to define the background color to be used for source code which is embedded in another language. (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 profile names to get embedded language color-coding. For an example, see [Setting Colors for Screen Elements](#)

Only color elements recognized by Color Coding, current line, and selection colors have an embedded color option.

- **Use system default** - When this option is selected, the operating system's default colors are used. Currently, this check box is only available for the **Status**, **Message**, and **Document Tab** fields. For UNIX, the system default colors are selected by the editor and not the operating system.
- **Font style** - For color-coded elements, you may choose whether the element is normal or a combination of bold, italic, and underline. For example, keywords are bold by default.
- **Sample Text** - Use the sample text and the embedded sample text to preview the foreground, background, and font attribute choices, both in normal code and embedded code.
- **Sample Code** - Use the sample code text box to view your selected color profile in a language of your choosing. You can also cut and paste small samples of text into this box in order to view specific items. The **Sample Code** will also display a preview of the minimap using the color profile selected for it.

Color Elements

Each of the following color elements which can be configured from within SlickEdit®.

- **Canvas** - This color element defines the most basic color element from which most other color elements inherit their background colors from.
- **Window Text** - This color is used for all the text in an editor control which does not match one of the other color coding elements defined for the current language.
- **General** - These color options are for syntactic elements displayed in editor windows. They correspond to items defined in the language's Color Coding specification. For more information, see [Color Coding](#).
- **Function** - This color is used to highlight identifiers which are followed by an open parenthesis, provided the language's Color Coding options specify to color them as such. Note that this color can be easily overridden by Symbol Coloring. For more information, see [Symbol Coloring](#)
- **Identifier** - This color is used to highlight symbols matching the identifier characters defined for the current language.
- **Identifier2** - This color is intended to be used for another identifier color like data members.
- **Keyword** - This color is used to highlight identifiers which match one of the keywords defined in the Color Coding for the current language.
- **Library Symbol** - This color is used to highlight identifiers which match one of the library symbols

defined in the Color Coding for the current language.

Note

If Symbol Coloring is enabled, and **View → Symbol Coloring → Override Library Symbols** is also enabled, identifiers with this color can be overridden by symbol coloring if they are recognized as a valid symbol by Context Tagging®.

- **Operator** This color is used to highlight symbols and punctuation which match one of the operators defined in the Color Coding for the current language.
- **Preprocessor** - This color is used to highlight preprocessor keywords defined in the Color Coding for the current language.
- **Punctuation** - This color is used to highlight symbols and punctuation which match one of the punctuation symbols defined in the Color Coding for the current language.
- **Special Characters** - This color is used for special characters such as tabs, newlines, and spaces. The color is only used when the option to view special characters is turned on either at the language level or for the current file.
- **User Defined Keyword** - This color is used to highlight identifiers which match one of the user-defined symbols defined in the Color Coding for the current language.

Note

If Symbol Coloring is enabled, and **View → Symbol Coloring → Override Library Symbols** is also enabled, identifiers with this color can be overridden by symbol coloring if they are recognized as a valid symbol by Context Tagging®.

- **Selections** - These color options are for text selections and the current line in the editor control.
- **Current Line** - This color is used to highlight the current line under the cursor. Underlying items will be colored using their configured foreground color and the background color specified for the current line, unless there isn't sufficient contrast, in which the foreground color for the current line will be used.
- **Cursor** - This 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.
- **Selected Current Line** - This color is used to highlight the current line under the cursor when in a selection. Underlying items will be colored using their configured foreground color and the background color specified for the current line, unless there isn't sufficient contrast, in which the foreground color for the current line will be used. Ideally, you should select a background for this color that is a combination of the **Current Line** and **Selection** colors.
- **Selection** - This color is used to highlight selections within the editor.

- **Comments** - These colors are used to highlight different comment types defined in the Color Coding specification for the current language. For more information, see [Color Coding](#).
- **Block Comment** - This color is used to highlight block comments.
- **Documentation Attribute** - This color is used for attribute names within HTML and XML tags recognized within documentation comments.
- **Documentation Attribute Value** - This color is used for attribute values within HTML and XML tags recognized within documentation comments.
- **Documentation Comment** - This color is used for documentation comments. Three types of documentation comments are supported: JavaDoc, XMLDoc, and Doxygen.
- **Documentation Keyword** - This color is used for documentation comment keywords and HTML and XML tag names recognized within documentation comments.
- **Documentation Punctuation** - This color is used for punctuation used for HTML and XML tags recognized within documentation comments.
- **Inactive Code** - This color is used for inactive code regions recognized by color coding. Inactive code regions are found in languages that support C/C++ style preprocessing provided the option to color inactive code regions is enabled for the current language.
- **Inactive Code Keyword** - This color is used for keywords within inactive code regions. Generally, the colors chosen for inactive code are such that the code appears to be grayed out. Highlighting keywords helps make inactive code still look somewhat like code, as apposed to just looking like block comments.
- **Inactive Code Comment** - This color is used for comments within inactive code regions. Generally, the colors chosen for inactive code are such that the code appears to be grayed out. Highlighting comments helps make inactive code still look slightly more like normal code.
- **Line Comment** - This color is used for line comments.
- **Strings** - These colors are used to highlight different string types defined in the Color Coding specification for the current language. For more information, see [Color Coding](#).
- **Backquoted String** - This color is used for strings which use the backwards single quote character as a delimiter, such as found in shell scripts and Perl. Because the contents of these strings are executed and evaluated, it is important to be able to visually distinguish this kind of string from a common literal string.
- **Single Quoted String** - This color is used for strings and character literals which use the single quote character as a delimiter. Because these are usually character literals, it is useful to be able to distinguish between them and double quoted strings.
- **String** - This is the general string color, used for double quoted strings, regular expressions, and anything else that is regarded as a string literal by the color coding engine. For example, this color is also used for here-documents when they have no embedded language.
- **Unterminated String** - This color is used for the background of the right hand side of the line when

the string does not yet have a closing quote. You can disable this coloring simply by allowing it to inherit its background color from **Window Text**.

- **Numbers** - These colors are used to highlight different numeric types defined in the Color Coding specification for the current language. For more information, see [Color Coding](#).
 - **Floating Point Number** - This color is used to highlight floating point numbers.
 - **Hexadecimal Number** - This color is used to highlight numbers specified in hexadecimal format.
 - **Line Number** - This color is used to highlight line numbers. This is different from the **View → Line Numbers** display option. This option is for coded line numbers, such as that found in dialects of Basic and COBOL.
 - **Number** - This color is used for all other numeric constants recognized by the Color Coding engine.
- **HTML and XML** -
 - **Attribute** - This is the color used for a recognized attribute of an XML or HTML tag. For example, in HTML, the **src** attribute of the **img** tag gets this color. Attribute values are colored using the **String** color.
 - **Tag/Element** - This is the color used for a recognized XML or HTML tag. For example, in HTML, the **img** tag gets this color.
 - **Unknown Attribute** - This is the color used for an unrecognized attribute of an XML or HTML tag. For example, in HTML, attempting to use the incorrectly spelled **sorce** attribute of an **img** tag gets this color.
 - **Unknown Tag Name** - This is the color used for unrecognized XML or HTML tags. If an XML document has no DTD or Schema, all the tags in the document will be colored using this color. Recognized XML and HTML tags are colored using the **Tag/Element** color.
 - **XHTML Element in XSL** - This color is used for an XHTML element in an XSL style sheet.
 - **XML/HTML Numeric Character References** - This color is used for XML/HTML numeric character references `&#nnnn;` (decimal) or `&#xhhh;` (hexadecimal).
- **Markdown** - Colors used when editing CSS or Less style sheets.
 - **CSS Class**
 - **CSS Element**
 - **CSS Property**
 - **CSS Selector**
- **Markdown** - Colors used when editing Markdown.
 - **Markdown Blockquote**

- **Markdown Bullet**
- **Markdown Code**
- **Markdown Emphasis** - Not specific, but generally mapped to italics.
- **Markdown Emphasis2** - Not specific, but generally mapped to bold.
- **Markdown Emphasis3** - Not specific, but generally mapped to bold/italic text.
- **Markdown Emphasis4** - Not specific, but generally mapped to overstrike text font.
- **Markdown Header**
- **Markdown Link**
- **Markdown Link2**
- **Yaml** - Colors used when editing YAML.
 - **Yaml Anchor Definition**
 - **Yaml Anchor Reference**
 - **Yaml Directive**
 - **Yaml Operator**
 - **Yaml Punctuation**
 - **Yaml Tag**
 - **Yaml Text**
 - **Yaml Text Colon**
- **Modifications** - These colors are used to show change bars in the left margin, as well as in DIFFzilla® to highlight changes.
 - **Inserted Line** - This color is used in the left-hand margin for lines that have been inserted into the current file since you started editing it. It is only displayed if you have **Modified Lines** coloring enabled for the current language. See [Language-Specific General Options](#) for more information.
 - **Modified Line** - This color is used in the left-hand margin for lines that have been changed in the current file since you started editing it. It is only displayed if you have **Modified Lines** coloring enabled for the current language. See [Language-Specific General Options](#) for more information.
 - **Modified Whitespace** - This color is used in DIFFzilla® when doing a source diff (ignoring whitespace). It is used to highlight locations where whitespace has been inserted, removed, or changed. To disable display of modified whitespace in DIFFzilla®, set the background for this color to inherit from **Window Text**.
 - **No Save Line** - This color is used to display lines which are shown in the editor or DIFFzilla® which

will not be saved when the file is saved. These lines are also known as imaginary lines. This color is also used for the top-of-file line.

- **Highlighting** - These colors are used for various types of text highlighting used to display search results.
 - **Block Matching** - This color is used to highlight matching parentheses, braces, brackets, and keyword begin-end pairs. The behavior of this option is language-specific.
 - **Compiler Errors** - This color is used to mark the position of compiler errors on the vertical scrollbar.
 - **Filename** - This color is used in the **Search Results** tool window to highlight file names. It is not used for file names in source code or for names displayed in the **File Manager**.
 - **Hex Mode** - This color is used to display hexadecimal characters when displaying text in hex mode or line hex mode.
 - **Highlight** - This color is used to highlight word matches found by word completion.
 - **Incremental Search Current Match** - This color is used to highlight the current matching word when doing an incremental search.
 - **Incremental Search Highlight** - This color is used to highlight matches to the current incremental search expression.
 - **Search Result Truncated** - This color is used in the Search Results tool window to highlight the leading and/or trailing part of search result line that is truncated.
 - **Symbol Highlight** - This color is used to highlight other references to the current symbol under the cursor within the current file.
- **Margins** - These colors are used for various marker lines and the left-hand indicator margin.
 - **Current Line Box** - This color is used for the box drawn around the current line or the ruler line drawn around the current line if a **Current line highlight** option is enabled. See [Current line highlight](#) for more information.
 - **Line Prefix Area** - This color is used for the background of the left-hand indicator margin and for line numbers when the **View → Line Numbers** display option is on.
 - **Line Prefix Divider Line** - This color is used to draw the single thin line between the left-hand indicator margin and the editor control text area.
 - **Margin Column Line(s)** - This color is used to draw a thin vertical line where the word-wrap margins are set. This line is only displayed if word-wrap is enabled for the current file. This option does not apply to comment formatting or HTML and XML text wrap options. See [Language-Specific Word Wrap Options](#) for more information.
 - **Truncation Column Line** - This color is used to draw a hard vertical line at the column where this file is to be truncated. This option is only used, generally, for languages with fixed line length restrictions, such as COBOL or certain dialects of assembly. See [Truncation Column](#) for more information.

- **Vertical Column Line** - This color is used to draw a thin vertical line at the column designated as the vertical column line. This line simply gives you a visual indicator when the current line may be getting longer than allowed by your coding conventions. See [Vertical line columns](#) for more information.
- **Document Tabs** - The following colors are used to configure colors used by the tabbed document interface.
 - **Document Tab - Active** - Active document tab displays the tab for the window with focus.
 - **Document Tab - Modified** - The color for the tab caption of a window that is modified. Only the foreground color is configurable since it overrides the caption color used on active, selected, and unselected tabs. To use this color you must turn on "Color modified document tabs" on **Tools** → **Options** → **Editing** → **Editor Windows**
 - **Document Tab - Selected** - This color is used to designate a current tab. This color is overridden if the tab is active or modified.
 - **Document Tab - Unselected** - This color is used to display a non-current tab. The color is overridden if tab is modified.

Note

By customizing the document tabs colors and using specific color profiles for different languages or individual files, you can color code the files in the tabbed document interface. For example, you may want to use a white background for document tabs containing Java source files, a yellow background for document tabs containing JavaScript, and a blue background for HTML and XML.

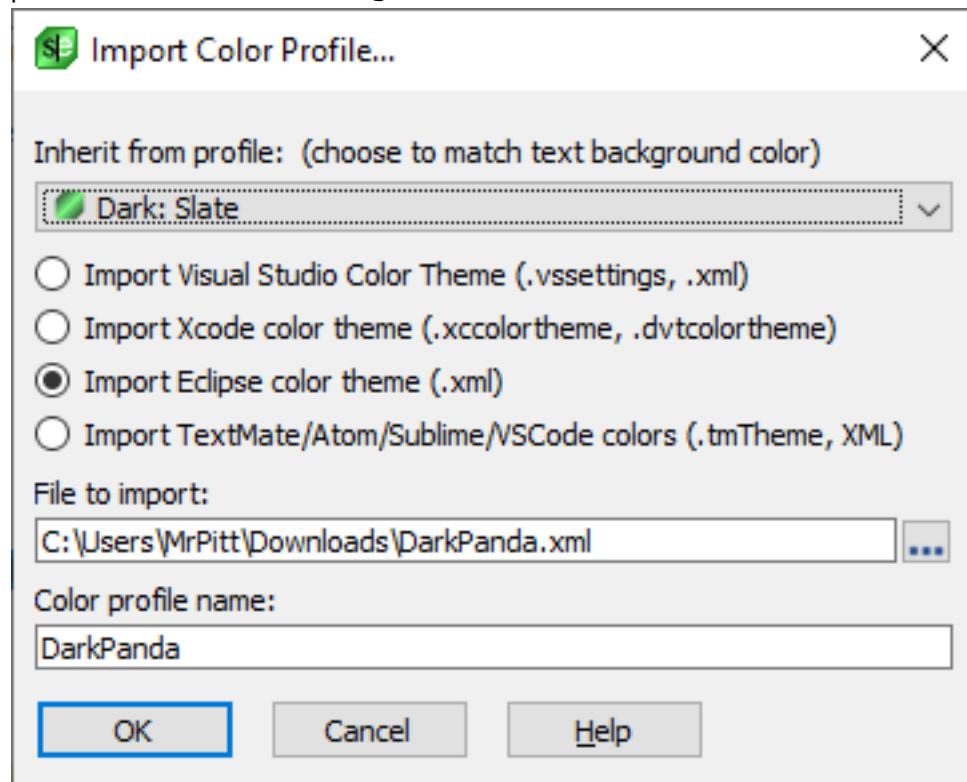
- **Application Colors** - The following colors are used to configure specific elements of the GUI displayed outside of the editor windows themselves.
 - **Message** - This color is used for the text displayed in the message bar found on the lower, left-hand side of the main SlickEdit® window.
 - **Modified variable** - This color is used in the debugger tool window and the to indicate that the value of a variable or watch has changed in the last debug stepping operation.
 - **Status** - This color is used for the text displayed in the status bar found on the lower, right-hand side of the main SlickEdit® window.
- **References** - This array of colors is used by the References tool window to highlight references to Symbols. Each one has a slightly different background shade.
- **Symbol Coloring** - This group of colors directly map to the rules defined in the default Symbol Coloring profile for **All symbols**.
- **Extended Coloring Palette** - This array of predefined colors are specifically for users to utilize when adding Symbol Coloring rules. Each color is specifically chosen because they are color safe colors for display against the window text background for the color profile.

Importing Color Profiles

Color profiles can be imported from editing color schemes assembled for several other products. This allows you to select a wide range of editor color profiles from profiles shipped with those products that you like and thousands of color themes found online.

While the import process is not perfect and can not map every color from another product's color theme to every color in a SlickEdit® color profile, it provides a good starting point. Colors not defined in the imported color theme can be inherited from a built-in SlickEdit® color profile.

You will find that after importing a color theme, you may want to fine tune certain colors. Imported color profiles are stored in `user.cfg.xml`.



The options are described as follows:

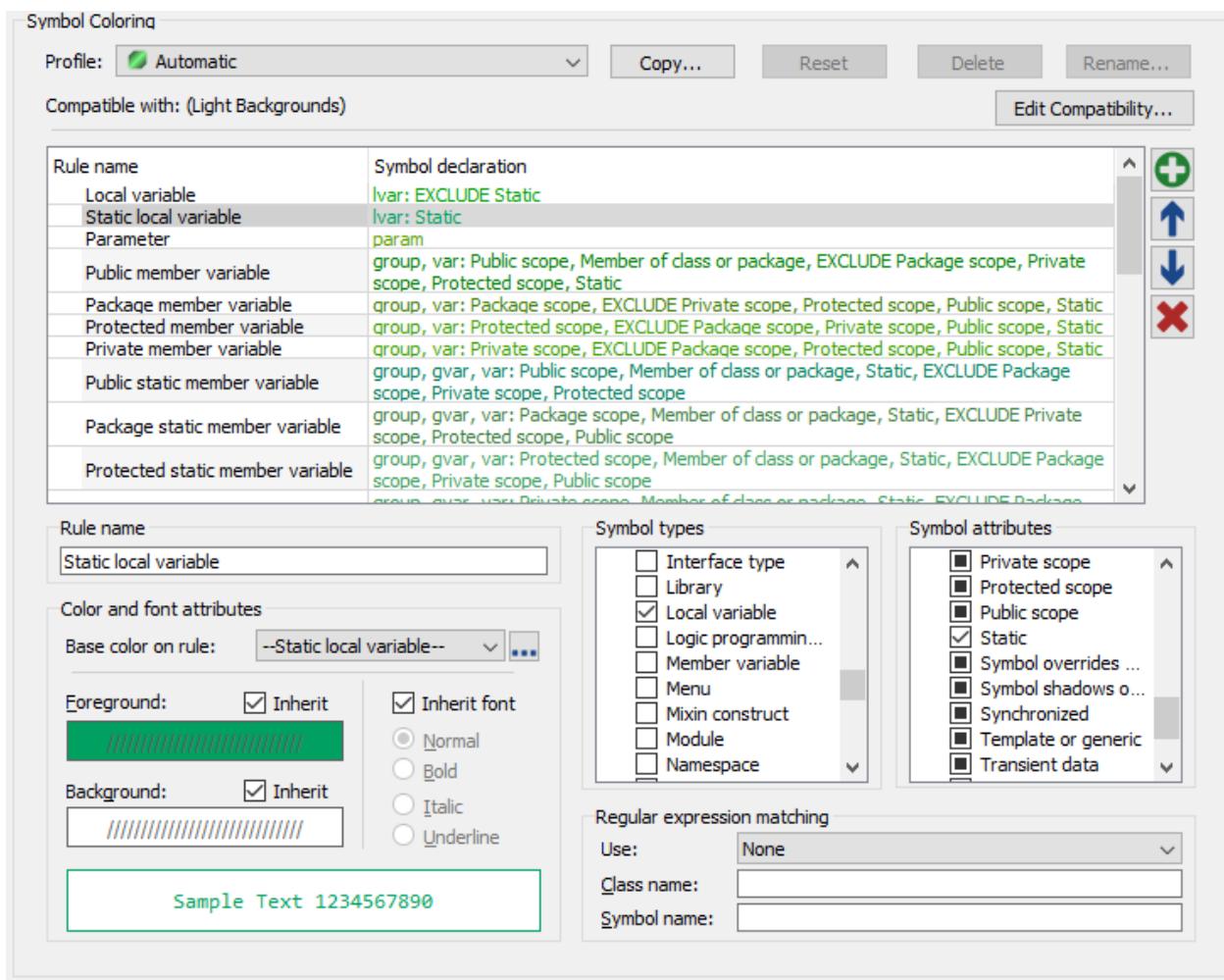
- **Inherit from profile:** - Specify a built-in SlickEdit® color profile to use as a baseline for the imported color theme. This way colors that are not mapped from the imported color theme will be defined. You must use a built-in color scheme here because it could cause problems to inherit from a user-defined scheme which may be deleted at some point in the future. When selecting a scheme to inherit from, try to match up the background colors. A dark color theme should inherit from a dark color profile and a light color theme should inherit form a light color profile.
- **Select the kind of file to import from -**
 - **Import Visual Studio Color Theme** - Visual Studio color themes are stored in `.vssettings` files. There are literally thousands available online. You can also import XML color themes for Visual Studio designed to work with the VSIX Color Editor and Compiler plugin for Visual Studio.

- **Import Xcode color theme** - Xcode color themes are either `.dvtcolortheme` or `.xccolortheme` files. There are numerous shipped with Xcode under the `FontAndColorThemes` directory, as well as hundreds of quality editor color themes for Xcode available online.
- **Import Eclipse color theme** - Eclipse color themes are XML plugin files. There are also hundreds available online. The Eclipse color themes tend to map very well to SlickEdit color themes.
- **Import TextMate/Atom/Sublime/VSCode colors** - VSCode, Atom, Sublime Text, and TextMate all share the same XML format for color themes. Several other products have also adopted this format and there are many very high quality editor color themes for these editors.
- **File to import:** - Specify the file to import here.
- **Color profile name** - Specify a name for the color profile imported. By default, this will be initialized to the name of the file imported.

Symbol Coloring Options (Pro only)

Symbol Coloring options are shown below (**Tools** → **Options** → **Appearance** → **Symbol Coloring**). These options let you specify colors for symbols identified using Context Tagging® and create and manage symbol coloring profiles. See [Symbol Coloring Profiles](#) for more information.

Appearance Options



Symbol Coloring settings

The options are described as follows:

- **Profile** - Specify the symbol coloring profile to use from the drop-down list. Several predefined profiles are available or you can define your own. See [Selecting a Symbol Coloring Profile](#) for more information.
- **Copy...** - Allows you to copy the current symbol coloring profile as a new profile with a name you specify.
- **Reset** - Allows you to reset a modified system profile back to its original configuration.
- **Delete** - Deletes the selected symbol coloring profile. Only available for user-created profiles.
- **Rename...** - Allows you to rename the selected symbol coloring profile. Only available for user-created profiles.
- **Compatible with and Edit Compatibility...** - Each symbol coloring profile can, optionally, be associated with a set of color profiles that it is compatible with. This is necessary in order to ensure that custom symbol colors are compatible with the base colors in the color profile.

If a profile does not list any specific compatible color profiles, it will be regarded as compatible with all color profiles. If a symbol coloring profile does not define any custom colors, that is, all the colors used are from the pre-defined symbol colors or the extended coloring palette, then it will be compatible with all color profiles. All of the predefine symbol coloring profiles are designed this way.

Press the **Edit Compatibility...** button to select which color profiles which are compatible with the currently selected symbol coloring profile.

Generally, compatibility hinges on the amount of contrast between the selection of foreground colors in the symbol coloring profile and the selection of background color in the base color canvas.

- **Rule list** - The symbol coloring rules are presented in order of precedence. Symbol coloring rules are matched in order from the top to bottom. For a symbol to match a rule, it must be the first rule in the symbol coloring profile that matches all of the requirements. See [Color Rules](#) for detailed descriptions of each of the standard symbol coloring rules shipped with SlickEdit®.
 - **Rule name** - This is the name of the symbol coloring rule.
 - **Symbol declaration** - This column contains a synopsis of the symbol coloring rule, including the types it matches, its attribute requirements, and regular expressions, if applicable. It is rendered using color and font attributes specified for the rule. This is also why this column appears to have a different background in color profiles which have a non-white background.
 - **Add rule** - Click the plus bitmap to add a new symbol coloring rule. Focus will be placed on the rule name field.
 - **Move Up** - Use the up arrow button to move the currently selected rule up one position in the list precedence. Remember that rules are applied in order of precedence.
 - **Move Down** - Use the down arrow button to move the currently selected rule down one position in the list precedence. Remember that rules are applied in order of precedence.
 - **Delete rule** - Use the delete button to delete the currently selected rule.
- **Rule name** - This is the name of the rule. Press Tab or Enter to apply your changes after editing the name of the currently selected symbol coloring rule.
- **Color and font attributes** - Use this panel to select how symbols which match the current symbol coloring rule will be rendered in the editor.
- **Base color on rule** - A symbol coloring rule can selectively inherit parts of its color information from another symbol coloring rule within the currently selected symbol coloring profile. It can also inherit color information from certain colors from the basic color profile, listed below.

The chooser to the right of the rule name can be used to select a predefined symbol color quickly and preview the color that will be used in association with the current color profile.

- **--Window Text--** - This color is used for all the text in an editor control which does not match one of the other color coding elements defined for the current language.

- **--Function--** - This color is used to highlight identifiers which are followed by an open parenthesis, provided the language's Color Coding options specify to color them as such.
- **--Preprocessor--** - This color is used to highlight preprocessor keywords defined in the Color Coding for the current language.
- **--Library Symbol--** This color is used to highlight identifiers which match one of the library symbols defined in the Color Coding for the current language.
- **--User Defined Keyword--** This color is used to highlight identifiers which match one of the user-defined keywords defined in the Color Coding for the current language.
- **--Highlight--** This color is used to highlight word matches found by word completion.
- **--Symbol Highlight--** This color is used to highlight other references to the current symbol under the cursor within the current file.
- **--Symbol Coloring ... --** - This group of colors directly map to the rules defined in the default Symbol Coloring profile for **All symbols**.
- **--Extended Coloring Palette NN--** - This array of predefined colors are specifically for users to utilize when adding Symbol Coloring rules. Each color is specifically chosen because they are color safe colors for display against the window text background for the color profile.

Note

For best results, always use the predefined symbol colors or colors from the extended coloring palette and inherit both the foreground and background colors. These colors are defined in each color profile. Doing this will keep your symbol coloring scheme compatible with all color profiles, whether it has a dark or light background color.

- **Foreground** - Select **Inherit** to inherit the foreground color from the base color. If **Inherit** is unchecked, you can click on the color sample to change the foreground color for this symbol coloring rule. If **Inherit** is checked, the inherited foreground color will be displayed, but clicking on the sample to change it will be disabled.
- **Background** - Select **Inherit** to inherit the background color from the base color. If **Inherit** is unchecked, you can click on the color sample to change the background color for this symbol coloring rule. If **Inherit** is checked, the inherited background color will be displayed, but clicking on the sample to change it will be disabled.

For best results, you almost always want the symbol color to inherit its background color instead of setting its own.

- **Font attributes** - Select **Inherit font** to inherit the font attributes from the base color. If **Inherit** is unchecked, you can click on **Normal**, **Bold**, **Italic**, or **Underline** to select font attributes for this symbol coloring rule. Note that font attributes can not be combined. If **Inherit** is checked, the inherited font attribute will be checked, but all the font attribute choices will be disabled.

Note

For certain languages, identifiers which are followed by a parenthesis are colored using "Function" color, as configured in the Color Options dialog. Typically, this will make those identifiers bold. Symbol coloring will preserve this information and propagate the font attributes for "Function" color forward when highlighting an identifier which is followed by a parenthesis. This makes it possible, for example, to visually distinguish between constant-like defines and function-like defines in a language such as C++.

- **Sample Text** - The sample text shows what the symbol coloring text might look like in the editor. The text is rendered in the same font as used in the editor for **SBCS/DBCS Source Windows**.
- **Symbol types** - A matching symbol's type must be one of the specified types. The special ***SYMBOL NOT FOUND*** type is used to identify symbols which Context Tagging® can not locate. See [Symbol types](#) for detailed descriptions of each symbol type.
- **Symbol attributes** - The attributes can be either required, ignored, or disallowed. A matching symbol must have all the required attributes, and none of the disallowed attributes. See [Symbol attributes](#) for detailed descriptions of each symbol attribute.
- **Regular expression matching** - In addition to the symbol type and attribute specifications, you can further refine a symbol coloring rule by adding a **Class name** or **Symbol name** regular expression, using the regular expression syntax of your choice. The class name regular expression is matched against the name of the scope (class, package, struct) which a symbol is defined in. Do not confuse this with the name of the scope in which the symbol is used. The symbol name regular expression is matched against the name of the symbol. For example, a Wildcards expression of "vs*" would match all symbols starting with the characters "vs". Case sensitivity for the regular expression matching is regulated by the language's case-sensitivity. See [Color Coding](#) for more information.
 - **Use** - This allows you to select the regular expression syntax you prefer to use for this symbol coloring rule's class and symbol name regular expressions. See [Regular Expressions](#) for more information.
 - **Class name** - A matching symbol must belong to a class matching the regular expression.
 - **Symbol name** - A matching symbol's name must match the regular expression.

Symbol types

The following symbol types may be included in a symbol coloring rule. A rule can include as many symbol types as it requires, and sometimes that is necessary to create a rule with enough generality. For example, to make a rule that matched any kind of constant value, you would need to include: **Constant**, **Enumeration value**, and **Preprocessor macro**.

- ***SYMBOL NOT FOUND*** - This is a special symbol type used when a symbol found in the editor is not recognized by Context Tagging®. This type is useful to have in a single rule as a rudimentary form of error checking.

Warning

Context Tagging® tries very hard to correctly recognize symbols, but it is not as accurate as your language's compiler. There are situations where a symbol will be highlighted as unknown, even though it is not strictly an error. This is particularly true for dynamic languages and languages that depend heavily on implicit declarations of local variables, such as most popular scripting languages.

Note

Highlighting of unknown symbols can be turned off on a per-language basis. This is a good idea for languages that are dynamically typed or have implicit declarations. It is also a good idea for Java because highlighting unknown symbols overlaps with the functionality provided by [Java Live Errors](#).

- **Annotation or attribute instance** - This is metadata. Examples include the use of a Java annotation or C# attribute in code.
- **Annotation or attribute type** - This is the definition of a metadata type. Examples include a Java annotation type or C# attribute class.
- **Build target** - Build target from Ant, a Makefile or other project build system.
- **Class constructor** - This is a constructor for a class in an object-oriented language. Note that in some languages, constructors are treated as functions with a **Constructor** symbol attribute.
- **Class destructor** - This is the destructor for a class in an object-oriented language. Note that in some languages, destructors are treated as functions with a **Destructor** symbol attribute.
- **Class property** - This is a property variable within a class type, as found in C#, Visual Basic .NET, and Managed C++.
- **Class type** - This type is used for classes in object oriented languages.
- **Constant** - This is a named literal constant.
- **Container variable** - This is used for a container (or group) variable, as found in COBOL data sections. A container variable is like an transparent structure type.
- **Control or widget** - This is used for a control type in languages that have built-in support for user interfaces, such as Slick-C®.
- **Database** - This type is used for the name of a database. It only applies to SQL dialects.
- **Database audit policy** - This type is used for a database audit policy definition. It only applies to SQL dialects.

- **Database cluster** - This type is used for a database cluster definition. It only applies to SQL dialects.
- **Database column** - This type is used for the column name of a database table. It only applies to SQL dialects.
- **Database constraint** - This type is used for a database constraint definition. It only applies to SQL dialects.
- **Database cursor** - This type is used for a database cursor type. It only applies to SQL dialects.
- **Database dimension** - This type is used for a database dimension definition. It only applies to SQL dialects.
- **Database edition** - This type is used for a database edition definition. It only applies to SQL dialects.
- **Database index** - This type is used for a database index name. It only applies to SQL dialects.
- **Database link** - This type is used for a database link name. It only applies to SQL dialects.
- **Database partition** - This type is used for a database partition name. It only applies to SQL dialects.
- **Database role** - This type is used for a database role name. It only applies to SQL dialects.
- **Database select statement** - This type is used for a database select statement. It only applies to SQL dialects.
- **Database sequence type** - This type is used for a database sequence definition. It only applies to SQL dialects.
- **Database table** - This type is used for a database table name. It only applies to SQL dialects.
- **Database table space** - This type is used for a database tablespace name. It only applies to SQL dialects.
- **Database trigger** - This type is used for a database trigger definition. It only applies to SQL dialects.
- **Database user profile** - This type is used for a database user profile name. It only applies to SQL dialects.
- **Database user** - This type is used for a database user name. It only applies to SQL dialects.
- **Database view** - This type is used for a database view name. It only applies to SQL dialects.
- **Directory** - This type is used for a file system directory name.
- **Enumerated type** - This type is used for type names of enumerated types.
- **Enumeration value** - This type is used for the names of the constants defined in an enumerated type.
- **Event monitor** - This type is used for an event monitor name.
- **Event table** - This type is used for event tables, as found in languages that have built-in support for graphical user interfaces, such as Slick-C®.

- **File descriptor** - This is a file descriptor declaration, as found in COBOL or SQL.
- **Form** - This type is used for form or dialog names, as found in languages that have built-in support for graphical user interfaces, such as Slick-C®.
- **Friend relationship** - This type is used for friend relationships. Note that friend relationships refer to other symbols, so a symbol coloring rule that colored friend relationships would only color the actual friend relationship declaration, not uses of the symbol that depended on the friend relationship.
- **Function** - This type is used for function names, both global functions and class member functions.
- **Function prototype** - This type is used for a function declaration, both global and abstract class member functions.
- **Global variable** - This type is used for global variables and variables declared at the namespace or package level. It does not apply to local variables or member variables.
- **Interface type** - This is a class interface declaration, as found in most object-oriented languages.
- **Library** - This is a library module type, as found in Pascal.
- **Local variable** - This is a local variable, that is, a variable declared within the scope of a function.
- **Member variable** - This is a class member variable or the member of a structured or record type or variable declared in a COBOL data section.
- **Menu** - This type is used for menu names, as found in languages that have built-in support for graphical user interfaces, such as Slick-C®.
- **Mixin construct** - This type is used for class mixin statements, as found in the D Programming Language. Note that, like friends, mixin's are only detected at the point of use. A function or variable pulled into a class through a mixin will not be considered as a mixin type.
- **Nested function** - This type is used for functions which are nested inside of other functions or procedures, as found in Pascal, Ada, and other languages.
- **Nested procedure or paragraph** - This type is used for procedures which are nested inside of other functions or procedures. It is also used for COBOL paragraphs.
- **Objective-C selector** - This type is used for the name of a selector, for example as found in Objective-C code.
- **Package import or using statement** - This type is used for an import or using statement. Note that imports are only detected at point of use. A class or function pulled into a module through an import will not be considered as an import type.
- **Package, module, or namespace** - This type is used for package, module, or namespace names used to divide code into logical boundaries.
- **Parameter** - This type is used for the names of formal parameters to a function or procedure. It can also be used for template parameter names in class templates.

- **Preprocessor include** - This type is used for a preprocessor include statement or COBOL copy book.
- **Preprocessor macro** - This type is used for a preprocessor macro (for example, a #define in C and C++).
- **Procedure or command** - This type is used for procedure names, both global and class members. It is also used for command names in languages that support command types.
- **Procedure prototype** - Type is used for procedure declarations, that is, forward declarations of procedures.
- **Program** - This type is used for program names, as found in Pascal, Cobol, and other languages.
- **Statement label** - This type is used for statement labels within functions or in assembly language code.
- **Structure type** - This type is used for struct types or record types.
- **Task** - This type is used for tasks, as found in Ada and Verilog dialects.
- **Type alias** - This type is used for a type definition name, or type alias.
- **Union type** - This type is used for union types, also known as variant types.
- **XML or HTML attribute** - This type is used for an attribute name in XML or HTML.

Symbol attributes

The following symbol attributes may be required or excluded in a symbol coloring rule. Each attribute can be in one of three states:

1. If the attribute is checked, then it is required to be set for a symbol to match the rule. For example, if you check the **Inline function** attribute then the rule will only match functions that are recognized as **inline**, as in C++.
2. If the attribute is unchecked, then it is required to be unset for a symbol to match the rule. For example, if you uncheck the **Abstract** attribute then the rule will only match functions which are not recognized as **abstract** (also known as pure virtual in C++).
3. If the attribute is in the grayed state, then it is ignored with respect to rule matching.

Some attributes are mutually exclusive, such as **Public**, **Protected**, **Package**, and **Private**. In these cases, checking two mutually exclusive attributes will produce a rule which will not match anything. Instead, you need to use boolean logic and create a rule which unchecks the mutually exclusive attributes that you do not want.

- **01 level in Cobol linkage section** - Variables at the 01 level in COBOL data sections are given this flag to indicate that they are at the top-most level of the data section. This may be a very useful attribute to configure specialized coloring for if you are a Cobol programmer.
- **Abstract** - This attribute is set for abstract classes as well as pure-virtual methods or abstract methods within a class definition.

- **Ambiguous prototype/var declaration** - In C and C++ and some other languages, there is an ambiguity in syntax between function prototype declarations and variable declarations with initializers. See the example below, not knowing type information, it could either be the prototype for a function named **ambiguousDeclaration** which returns a **ClassName** and takes one parameter of type **Argument**, or a variable named the same thing which is of type **ClassName** and is initialized with **Argument**.

```
ClassName ambiguousDeclaration(Argument);
```

- **Class constructor** - This attribute is set for functions or procedures which act as a class constructor or static initializer for a class type.
- **Class destructor** - This attribute is set for class destructors.
- **Const** - This attribute is set for variables which are declared with a **const** return type, meaning that they can not be modified, or that they point to data which can not be modified. It is also set for class member functions that are declared as **const**, meaning that the function is not allowed to modify members of the class or call other non-const functions.
- **Created by preprocessor macro** - This attribute is set for symbols which are declared in a section of code that was expanded from a preprocessor macro, for example, in C++ where it is common to use preprocessor macros to generate code.
- **External function or data** - This attribute is set for functions or global variables which are declared as **extern**, meaning that they are defined in another module.
- **Final** - This attribute is set for functions which can not be overridden in derived classes. It is also set for variables, for example, in Java, which are initialized only once and do not change.
- **Forward declaration** - This attribute is set for a forward declaration of a symbol. Since there is a separate symbol type for function prototypes, this attribute is primarily used for forward declarations of classes and structured types.
- **Ignore/placeholder** - This attribute is set for symbols which are merely placeholders, and should be ignored by symbol searches.
- **Implicitly defined local variable** - This attribute is set for local variable which are implicitly defined. This is used in some scripting languages where local variables do not have to be explicitly declared.
- **Inline function** - This attribute is set for functions which are marked as **inline**, meaning that instead of being compiled into separate functions, their function bodies may be pulled inline at the point where the function is called.
- **Member of class or package** - This attribute is set for symbols which are declared inside a class, package, namespace, or other structured type.
- **Mutable** - This attribute is set for variables which are declared as **mutable** in, for example, C++.
- **Native code function** - This attribute is set for functions which are implemented in native code, for example, in Java, where certain functions are implemented in DLLs for better performance.

- **Opaque enumerated type** - In C++, the constants declared in an enumerated type do not need to be qualified with the name of the enumerated type. In this sense, they are transparent. In certain other languages, enumeration constants need to be qualified with the name of the enumerated type. These are considered as *opaque*. This attribute is set for enumerated types whose constants are opaque.
- **Overloaded operator** - This attribute is set for functions whose purpose is to overload standard language operators, such as multiplication, division, assignment.
- **Package scope** - This attribute is set for symbols which are scoped at the package or namespace level. They are visible within the package they are declared in, but considered as private outside of that package. Package scope is the default scope in Java if no other scope (public, protected, or private) is specified.
- **Part of an external file** - This attribute is set for symbols which are declared in an external file which was parsed as part of the process of parsing the current file. For example, this attribute is set for symbols in COBOL copy books.
- **Partial class** - This attribute is set for classes which are marked as **partial**, meaning that the complete class definition may be spread across several modules.
- **Private scope** - This attribute is set for symbols declared in classes which have **private** scope. These symbols are visible within the class they are declared in, but not in derived classes, and not outside of the declaring class.
- **Protected scope** - This attribute is set for symbols declared in classes which have **protected** scope. These symbols are visible within the class they are declared in, derived classes, but not outside of the declaring class.
- **Public scope** - This attribute is set for symbols declared in classes which have **public** scope. These symbols are visible everywhere. Note that in many languages, public scope is the default scope if none is otherwise specified (private, protected, or package).
- **Static** - This attribute is set for symbols which are marked as **static**. The **static** attribute can be applied in several contexts:
 - Local variables which are marked as **static** will retain their last value between function calls.
 - Global variables and functions which are marked as **static** are visible only with the current module or compilation unit.
 - Class members which are marked as **static** do not require an instance of the class in order to be accessed. In many respects, they are like globals, but scoped within the class declaration.
- **Synchronized** - The synchronized attribute is set for functions which are marked as **synchronized**, meaning that the function is not re-entrant and can not allow two threads to enter it at the same time.
- **Template or generic** - The template attribute is set for class and function templates, also known as generic functions or generic classes.
- **Transient data** - The transient attribute is set for variables which contain non-persistent data. This is a Java-specific attribute.

- **Unnamed structure** - This attribute is used for anonymous structure types and other anonymous (unnamed) types.
- **Virtual function** - This attribute is used for virtual functions in class definitions.
- **Volatile** - This attribute is used for variables and functions which are marked as **volatile**. A volatile variable is one whose value can change unpredictably, such as a memory address that echoes the value of the system clock. A volatile function is one that accesses a volatile variable.

Color Rules

The standard color rules in the **All symbols** symbol coloring profile are described below.

The list below not only explains the rule types, but also tries to explain the reasoning behind the coloring choices that were made for various rules. Not all color profiles follow these guidelines.

- **Local variable** - This rule matches local variables within a function, excluding static local variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window.
- **Static local variable** - This rule matches static local variables. It is colored slightly more blue than the variable rule, in order to indicate the static (or frozen) nature of the local variable.
- **Parameter** - This rule matches function parameter names. Like local variables, they are colored green, matching the color used for variables in the Defs tool window.
- **Template Parameter** - This rule matches class or function template parameter names. Like local variables, they are colored green, matching the color used for variables in the Defs tool window.
- **Public member variable** - This rule matches public member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window.
- **Package member variable** - This rule matches package scope member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window.
- **Protected member variable** - This rule matches protected member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.
- **Private member variable** - This rule matches private member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.
- **Public static member variable** - This rule matches public member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the variable.
- **Package static member variable** - This rule matches package scope member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the variable.

- **Protected static member variable** - This rule matches protected static member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the variable. Protected and private members are shown in italic in order to indicate their limited scope.
- **Private static member variable** - This rule matches private static member variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the variable. Protected and private members are shown in italic in order to indicate their limited scope.
- **Global variable** - This rule matches global variables. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window.
- **Static global variable** - This rule matches static global variables, which are visible only within the current module. In most profiles, variable types are colored green, mirroring the color used for variables in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the variable.
- **Global function** - This rule matches global functions and functions declared within namespaces and packages. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window.
- **Static global function** - This rule matches static global functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the function.
- **Class constructor** - This rule matches class constructors. Class constructors, destructors, and class names are colored blue, mirroring the color used for Class constructors and destructors in the Defs tool window.
- **Class destructor** - This rule matches class destructors. Class constructors, destructors, and class names are colored blue, mirroring the color used for Class constructors and destructors in the Defs tool window.
- **Public member function** - This rule matches public class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window.
- **Package member function** - This rule matches package scope class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window.
- **Protected member function** - This rule matches protected scope class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.
- **Private member function** - This rule matches private scope class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.

- **Public static member function** - This rule matches public static class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the function.
- **Package static member function** - This rule matches package scope static class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the function.
- **Protected static member function** - This rule matches protected static class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the function. Protected and private members are shown in italic in order to indicate their limited scope.
- **Private static member function** - This rule matches private static class member functions. In most profiles, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs tool window. It is colored slightly more blue, in order to indicate the static nature of the function. Protected and private members are shown in italic in order to indicate their limited scope.
- **Public class property** - This rule matches public class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window.
- **Package class property** - This rule matches package scope class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window.
- **Protected class property** - This rule matches protected class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.
- **Private class property** - This rule matches private class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.
- **Public static class property** - This rule matches public static class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window.
- **Package static class property** - This rule matches package scope static class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window.
- **Protected static class property** - This rule matches protected static class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.
- **Private static class property** - This rule matches private static class property names. In most profiles, properties are colored cyan, mirroring the color used for properties in the Defs tool window. Protected and private members are shown in italic in order to indicate their limited scope.

- **Class** - This rule matches Class names. Class names are colored blue, mirroring the color used for class constructors in the Defs tool window.
- **Template class** - This rule matches template or generic class names. Class names are colored blue, mirroring the color used for class constructors in the Defs tool window.
- **Abstract class** - This rule matches an abstract class name. Class names are colored blue, mirroring the color used for class constructors in the Defs tool window.
- **Interface class** - This rule matches class interface names. Class and interface names are colored blue, mirroring the color used for class constructors in the Defs tool window.
- **Struct** - This rule matches a structured type, such as a **struct** in C/C++ or a **record** type in Pascal or Modula. Structure types are colored blue-green.
- **Union or variant type** - This rule matches union types or variant record types in Pascal. They are colored yellow, mirroring the color used for union types in the Defs tool window.
- **Type definition or alias** - This rule matches type definitions or type aliases.
- **Preprocessor macro** - This rule matches preprocessor macro names. The color inherits it's color from the preprocessor keyword color of the base color profile.
- **Package or namespace** - This rule matches a package or namespace name. In most profiles, they are colored red, mirroring the color used for package, namespace, and programs in the Defs tool window.
- **Symbolic constant** - This rule matches symbol constants used to give names to constant literal values. In most profiles, they are colored gray, mirroring the color used for constants in the Defs tool window.
- **Statement label** - This rule matches statement labels. They are colored blue, matching the color used for labels in the Defs tool window.
- **Enumerated type** - This rule matches enumerated types. In most profiles, they are colored cyan, mirroring the color used for enums in the Defs tool window.
- **Enumerated type constant** - This rule matches constants defined in enumerated types. In most profiles, they are colored cyan, mirroring the color used for enums in the Defs tool window.
- **Database table** - This rule matches tables defined in the schema for a relational database.
- **Database table column** - This rule matches a column name of a table defined in the schema for a relational database.
- **Database table index** - This rule matches an index defined for a table in the schema for a relational database.
- **Database table view** - This rule matches an view of a table in the schema for a relational database.
- **Module** - This rule matches the name of a logical module.
- **Concept** - This rule matches the name of a concept template class.

- **Rule** - This rule matches the name of a rule in a logic-oriented programming language.
- **Annotation type** - This rule matches the name of an annotation type.
- **Annotation** - This rule matches the name of an annotation instance.
- **Control** - This rule matches the name of a GUI control.
- **Menu** - This rule matches the name of a GUI menu.
- **Form** - This rule matches the name of a GUI form or dialog type.
- **Event table** - This rule matches the name of a GUI event table.
- **Nested function** - This rule matches the name of a nested function or procedure.
- **Protected symbol** - This rule is a catch-all for any other symbol declared in a protected scope.
- **Private symbol** - This rule is a catch-all for any other symbol declared in a private scope.
- **Symbol not found** - This rule is present as a catch-all for the case where a symbol is not found by Context Tagging®. It is colored bright red in order to indicate that there is a fair likelihood that the source file contains an error.

Font Options

You can specify which fonts are used by screen elements using the options shown below (**Tools** → **Options** → **Appearance** → **F**onts). If you want to change the font used in editor windows, use **Window** → **Font** instead. See [Fonts](#) for more information about changing fonts and a list of recommendations.

Appearance Options

Fonts

Element	Font Name	Size	Style
▼ Editor Windows			
SBCS/DBCS Source Windows	Consolas	10	
SBCS/DBCS Minimap Windows	Consolas	1	
Hex Source Windows	Consolas	10	
Unicode Source Windows	Consolas	10	
Unicode Minimap Windows	Consolas	1	
File Manager Windows	Consolas	10	
Diff Editor SBCS/DBCS Source Windows	Consolas	10	
Diff Editor Unicode Source Windows	Consolas	10	
▼ HTML Controls			
Parameter Information	Tahoma	9	
Parameter Information Fixed	Consolas	9	
HTML Proportional	Times New Roman	12	
HTML Fixed	Courier New	12	
▼ Application			
Command Line	Consolas	10	
Status Line	Tahoma	10	
Selection List	Courier	10	
Dialog	Tahoma	8	
Document Tabs	Tahoma	8	

Description

Editor windows that are displaying non-Unicode content (Windows: plain text).

Sample

```
== Line before ==
Aa_Bb_Cc = (11 + 00);
== Line after ==
```

Use fixed spacing for bold and italic fixed Unicode fonts
 Use anti-aliasing

The options are described as follows:

- **Element** - This table contains the screen elements for which fonts can be changed. When an element is selected, a description of the element will be displayed in the **Description** area and a preview of the font will be displayed in the **Sample** area. The table supports selecting multiple fonts so you can modify them using the zoom in/out and font chooser buttons to the right of the table. You can also modify an

entire category of fonts by selecting a category (Editor Windows, HTML Controls, Application).

Select from the following elements:

- **Editor Windows** - This category of fonts determine what fonts are used in various types of editor windows.
 - **SBCS/DBCS Source Windows** - Editor windows that are displaying non-Unicode content (for example, plain text).
 - **SBCS/DBCS Minimap Windows** - Editor minimap 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 or Java).
 - **Unicode Minimap Windows** - Editor minimap windows that are displaying Unicode content (for example, XML or Java).
 - **File Manager Windows** - Controls the display of the SlickEdit® File Manager (**File → File Manager**).
 - **Diff Editor SBCS/DBCS Source Windows** - The editor windows used by DIFFzilla® that are displaying non-Unicode content.
 - **Diff Editor Unicode Source Windows** - The editor windows used by DIFFzilla® that are displaying Unicode content.
- **HTML Controls** - This category of fonts determine what fonts are used by default in HTML controls displaying documentation and code help comments.
 - **Parameter Info** - Controls the fonts used to display pop-ups with information about symbols and parameters.
 - **Parameter Info Fixed** - Used when SlickEdit® needs to display a fixed-width font for parameter info, such as when displaying example code.
 - **HTML Proportional** - The default font used by HTML controls for proportional fonts. In particular, this affects the Version Control History dialog, the Preview tool window, the About SlickEdit dialog, and the Cool Features dialog.
 - **HTML Fixed** - The default font used by HTML controls for fixed-space fonts.
- **Application** - This category of fonts determine what fonts are used in by various other parts of the SlickEdit® user interface.
 - **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.
 - **Selection List** - The font used for selection lists, like the document language list (**Document →**

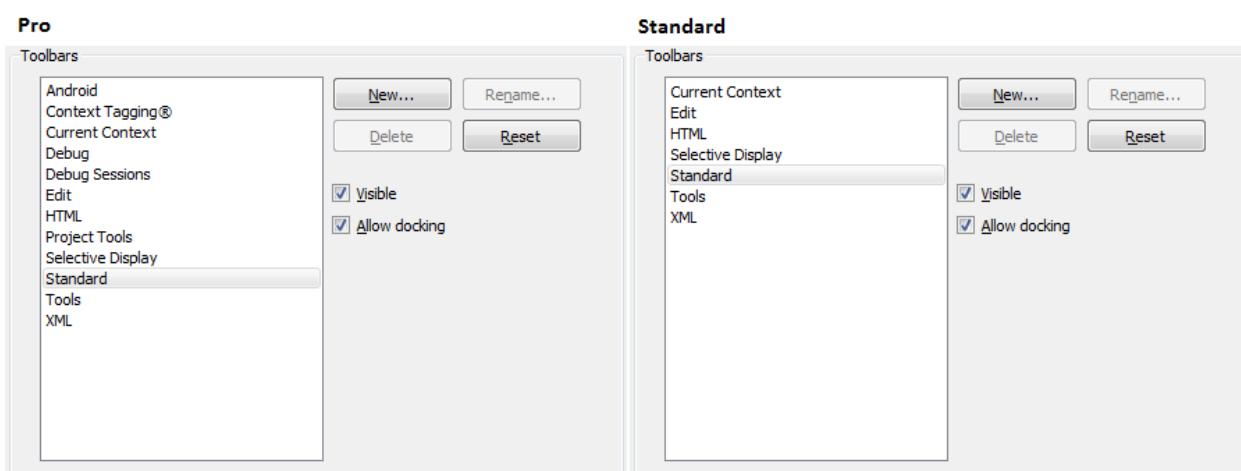
Select Mode).

- **Dialog** - Controls the font used in SlickEdit® dialogs and tool windows.
- **Document Tabs** - The default font used by tabs used to switch between documents.
- **Font Name** - The **Font Name** selected for each element can be changed directly within the table by clicking on it and selecting a font typeface from the pop-up list. The fonts that are listed are the fonts that are installed on your computer.
- **Size** - The **Size** select for each element can be changed directly within the table by clicking on it and selecting a size from the pop-up list, or typing a custom size between 1 and 99. Note that some fonts only support specific sizes.
- **Style** - This column displays the font style options (bold, italic, underline, strikethrough) selected for each element. Use the font chooser described below to modify individual font styles.
- **Zoom In Button** - Increase the size of all selected font elements by 10%. If only one item is selected, increase the size to the next larger font size available.
- **Zoom Out Button** - Decrease the size of all selected font elements by 10%. If only one item is selected, decrease the size to the next smaller font size available.
- **Font Chooser** - This button brings up a font chooser dialog where you can select a font typeface, size, and style options which will be applied to all the elements selected.
- **Sample area** - This area provides a preview of the selected font, size, and style.
- **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).
- **Use anti-aliasing** - Select this option to use anti-aliasing when displaying fonts in the edit window. This does not effect fonts displayed in the minimap window. Use the minimap context menu to toggle anti-aliasing for a particular font size the minimap window.

Toolbar Options

Toolbar options (**Tools** → **Options** → **Appearance** → **Toolbars**) let you modify, create, and change the behavior of toolbars. It contains a list of the default toolbars within SlickEdit®, and settings made here affect each toolbar individually. See [Toolbars and Tool Windows](#) for more information.

Appearance Options



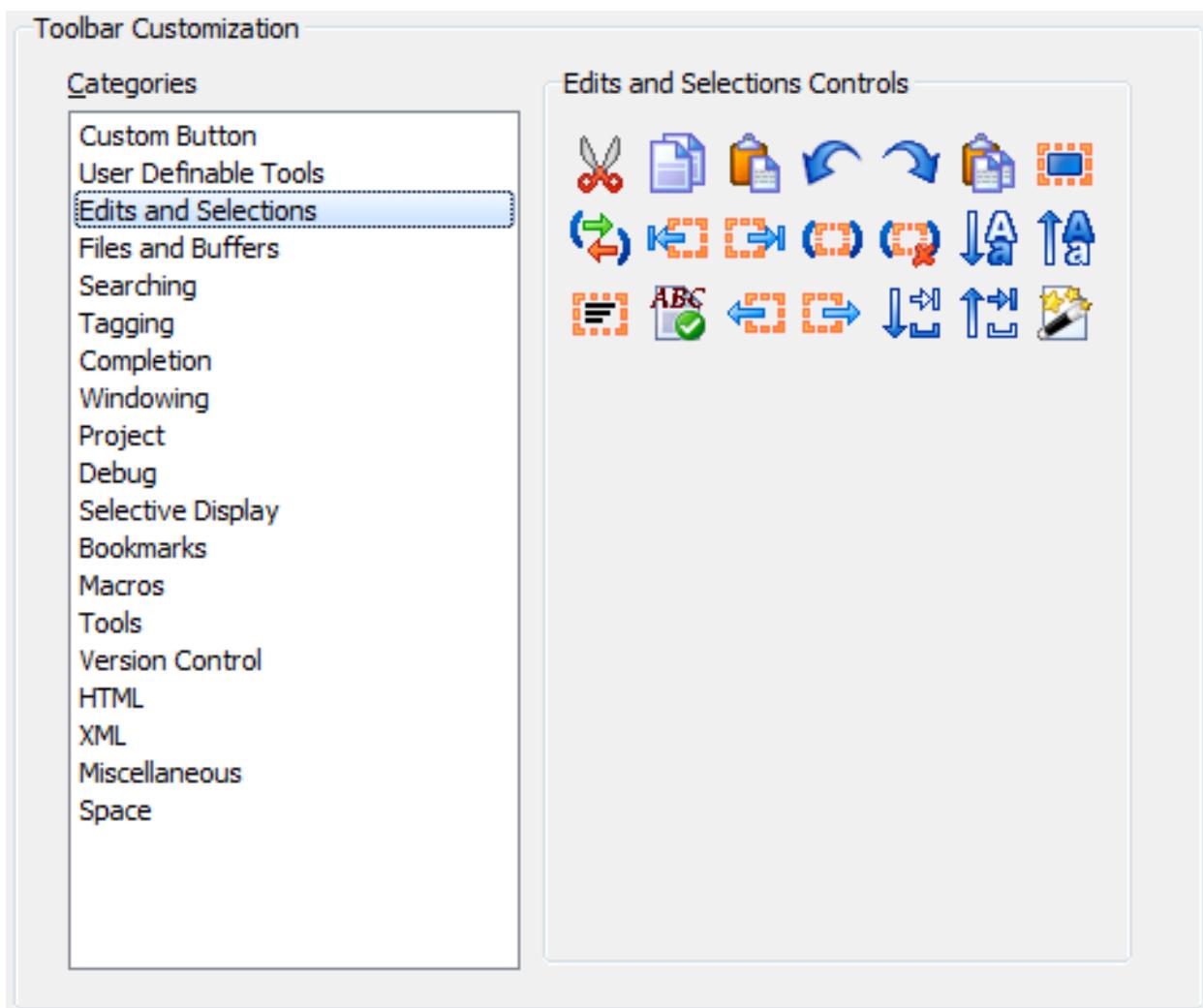
The following options are available:

- **New** - Creates a new, empty toolbar.
- **Rename** - Renames the selected toolbar. Note that you can only rename custom toolbars.
- **Delete** - Deletes the selected toolbar. Note that you can only delete custom toolbars.
- **Reset** - Restores the default buttons to the selected toolbar (not applicable for custom toolbars).
- **Visible** - When checked, the selected toolbar is displayed, if it is not already displayed. When unchecked, the selected toolbar is closed.
- **Allow docking** - When unchecked, the selected toolbar, when non-docked, cannot accidentally be docked.

Toolbar Customization

Toolbar Customization options (**Tools** → **Options** → **Appearance** → **Toolbar Customization**) let you add or remove buttons from toolbars. You can also access these options from the main menu by selecting **View** → **Toolbars** → **Customize** or by right-clicking on the toolbar and selecting **Customize**. See [Toolbars and Tool Windows](#) for more information.

This dialog categorizes the toolbar controls (buttons) and allows you to drag and drop them onto existing toolbars.

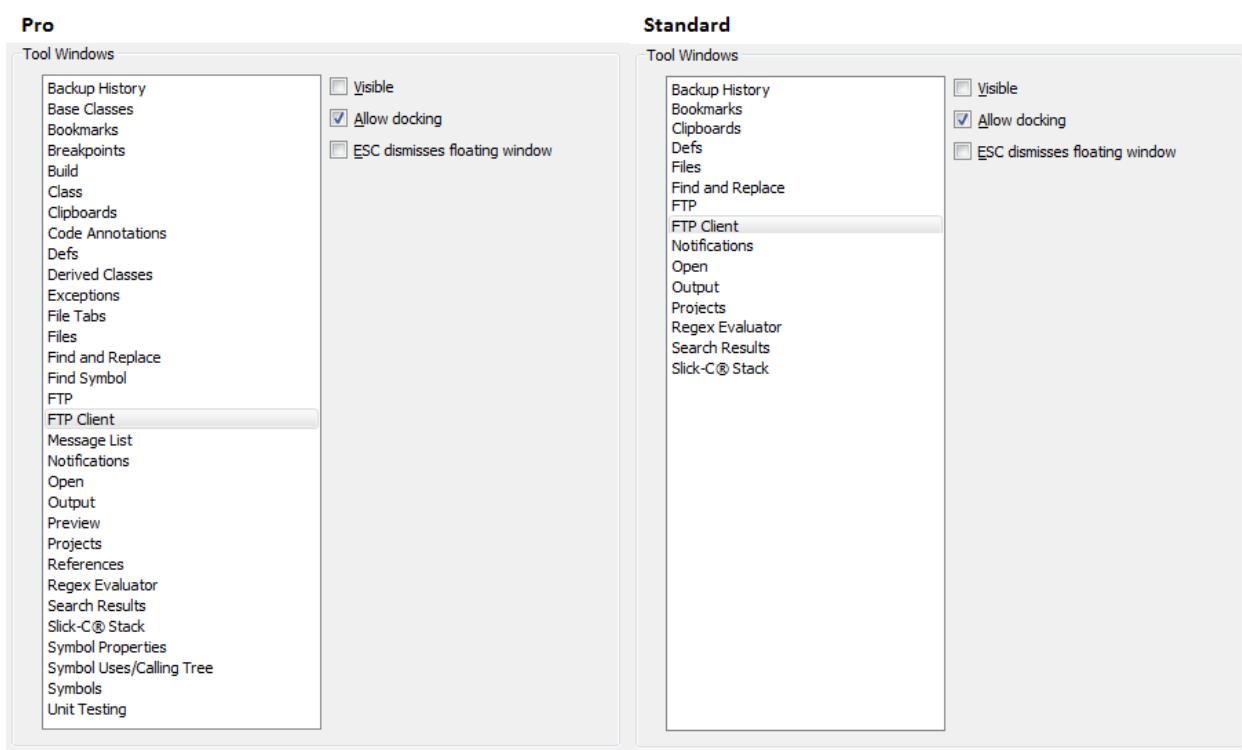


Select a category from the **Categories** list and the associated controls are displayed in the **Controls** box. To add a control, click on the control you wish to add, and drag it onto a toolbar. To remove a control, drag it from the toolbar onto the Categories screen. See also [Toolbar Control Properties Dialog](#).

Tool Windows Options

Tool Windows options (**Tools** → **Options** → **Appearance** → **Tool Windows**) let you control the behavior and visibility of tool windows. You can also access these options from the main menu by selecting **View** → **Tool Windows** → **Customize** or by right-clicking on any tool window's title bar (or on UNIX/Mac, the tool window's background) and selecting **Customize**.

Appearance Options



The following options are available:

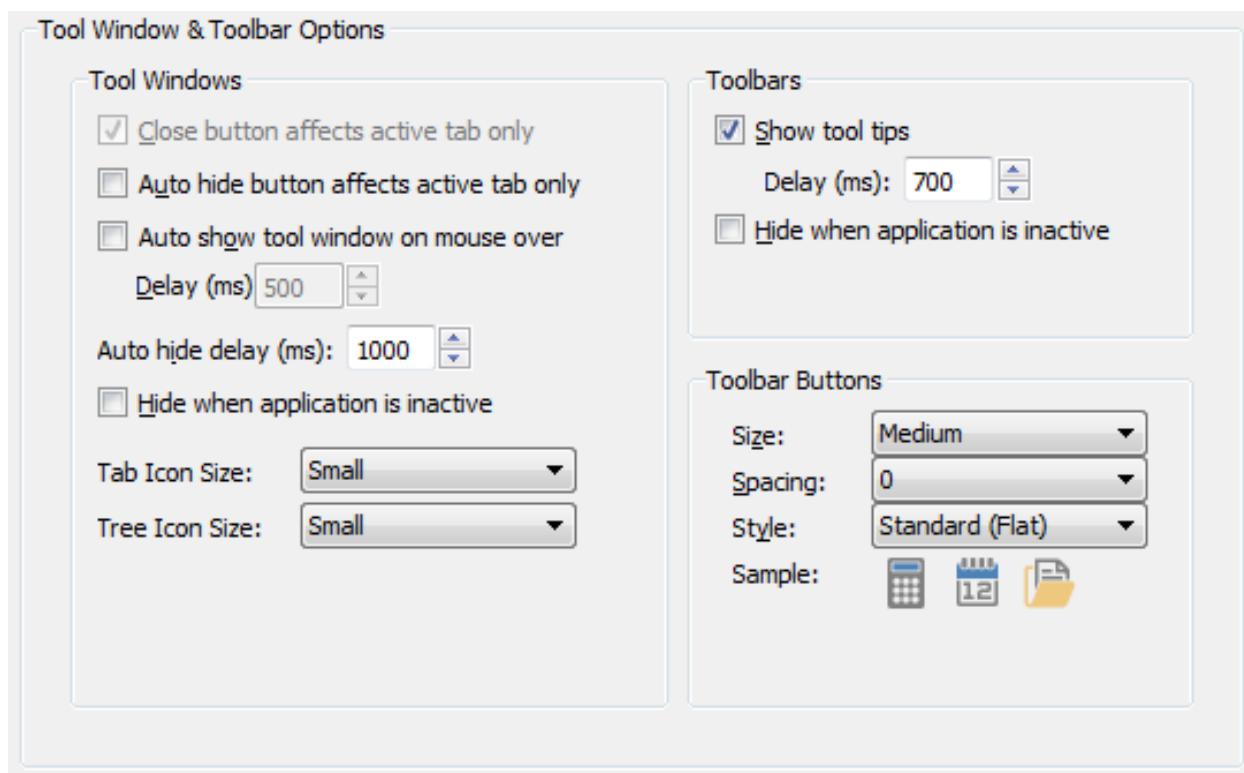
- **Visible** - When checked, the selected tool window is displayed, if it is not already displayed. When unchecked, the selected tool window is closed.
- **Always on top** - When checked, the selected tool window, when non-docked, will remain on top of the editor window.
- **Allow docking** - When unchecked, the selected tool window, when non-docked, cannot accidentally be docked.
- **ESC dismisses floating window** - When checked, pressing **Esc** on a floating tool window (not a docked or auto-hide window) will dismiss the tool window as if it were a dialog. When unchecked, pressing Esc on a floating tool window puts focus back to the active MDI child.

Tip

Pressing Esc on a docked tool window puts focus back to the active MDI child. Pressing Esc on an auto-hide tool window will hide the tool window.

Tool Window & Toolbar Options

The Tool Window & Toolbar options (**Tools** → **Options** → **Appearance** → **Tool Window & Toolbar Options**) provides general options to control the behavior of all tool windows and toolbars.



- **Tool Windows**

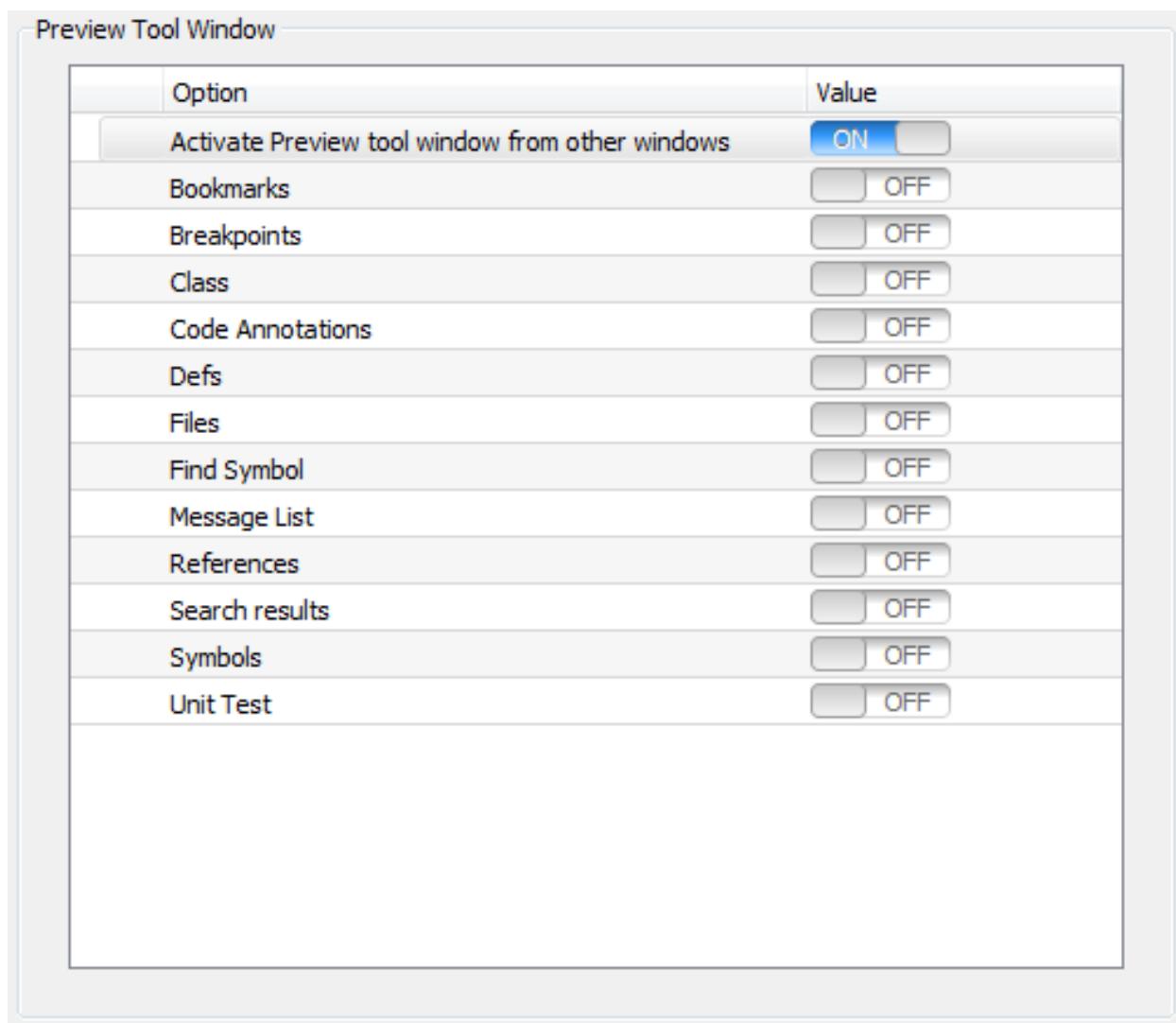
- **Close button affects active tab only** - When checked, the **Close** button on a tool window will close only that tool window. When unchecked, and the tool window is tab-linked with other tool windows, all tab-linked tool windows are closed. By default, this option is selected.
- **Auto Hide button affects active tab only** - When checked, the **Auto Hide** button (the **Pushpin** button) 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.
- **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 milliseconds) before the mouse triggers an auto-hidden tool window to auto-display.
- **Auto hide delay** - Specifies how long to wait (in milliseconds) before an auto-displayed tool window auto-hides itself.
- **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.
- **Tool window menu toggles hide/show** - When checked, tool windows that are already docked, floating, or auto-shown, if selected from the tool window menu, will be closed (effectively toggled). Otherwise, selecting an item from the tool window menu activates the tool window.
- **Tab Icon Size** - Specifies size of the icon to use for bitmaps displayed on the tabs of tool window. By

default, small icons are used, which tend to match the size of the default dialog font.

- **Tree Icon Size** - Specifies size of bitmaps to use for bitmaps displayed in tree controls on various tool windows. By default, this icon size is determined automatically based on the height of the Dialog font (see [Font Options](#) for more information). On smaller, high-resolution displays, it can be useful to use a larger size to improve visibility.
- **Toolbars**
 - **Show tool tips** - When set to **On**, pop-up tool tips are displayed when the mouse pointer rolls over a button. This option is global to all toolbars.
 - **Delay (ms)** - Specifies the delay, in milliseconds, before tool tips are displayed when **Show tool tips** is enabled.
 - **Hide when application is inactive** - When checked, non-docked toolbars are hidden when you switch to another application. When you switch back to SlickEdit®, the toolbar is made visible again. This option is global to all toolbars.
- **Toolbar Buttons** - Changes to these options can be previewed here as you make changes. Click **Apply** to save changes.
 - **Size** - Specify the size of buttons on a toolbar.
 - **Spacing** - Specify the amount of vertical and horizontal spacing between buttons on a toolbar.
 - **Style** - Specify the style of icons to use for toolbar buttons. There are several styles available to suite your preferences for look and feel including traditional color 3-Dimensional buttons, limited color flat buttons, two-color buttons, and monochrome buttons in both dark grey and white. The white monochrome buttons are best suited when using the Dark application theme. See [Application theme](#) for more information.

Preview Tool Window (Pro only)

These options are used to control when the Preview window is activated with a new symbol lookup. By selecting a symbol in another tool window, you can activate the Preview tool window and see the preview of that symbol.



The options are described as follows:

- **Activation:**
 - **Activate Preview tool window from other windows** - Setting this option to **On** allows you to activate the Preview tool window automatically from other windows. You can further specify which windows by setting the options below.
 - **Bookmarks** - When set to **On**, selecting a bookmark in the Bookmarks tool window will activate the Preview tool window with the bookmarked line shown.
 - **Breakpoints** - When set to **On**, selecting a breakpoint in the Breakpoints tool window will activate the Preview tool window and show the line with the selected breakpoint.
 - **Class** - When set to **On**, selecting an item in the Class tool window will activate the Preview tool window with the corresponding symbol shown.
 - **Code Annotations** - When set to **On**, selecting an item in the Code Annotations tool window will activate the Preview tool window with the line containing the annotation shown.

- **Defs** - When set to **On**, selecting an item in the Defs tool window will activate the Preview tool window with the corresponding symbol shown.
- **Files** - When set to **On**, selecting an item in the Files tool window will activate the Preview tool window with the selected file shown.
- **Find Symbol** - When set to **On**, selecting an item in the Find Symbol tool window will activate the Preview tool window with the corresponding symbol shown.
- **Message List** - When set to **On**, selecting an item in the Message List tool window will activate the Preview tool window with the line relevant to the selected message shown.
- **References** - When set to **On**, selecting an item in the References tool window will activate the Preview tool window with the corresponding reference shown.
- **Search results** - When set to **On**, selecting an item in the Search results tool window will activate the Preview tool window with the located search item shown.
- **Symbols** - When set to **On**, selecting an item in the Symbols tool window will activate the Preview tool window with the corresponding symbol shown.
- **Unit Test** - When set to **On**, selecting an item in the Unit Test tool window will activate the Preview tool window with the selected unit test shown.
- **Select Symbol** - When set to **On**, selecting an item in the Select Symbol dialog will activate the Preview tool window with the selected symbol shown.
- **Current line highlight:**
 - **Draw box around current line** - When set to **On**, draw the default box highlight around the current line in the Preview tool window, using the style specified in **Tools → Options → Appearance → General**.
 - **Highlight current line** - When set to **On**, highlight the current line in the Preview tool window using the current line highlight color.
 - **Highlight current line (per Language)** - When set to **On**, highlight the current line in the Preview tool window only if enabled for the current language.

Special Character Options

Special character options are shown below (**Tools → Options → Appearance → Special Characters**). These options are used to define the characters that are displayed when the view of special characters is enabled. Enabling special characters inserts characters into your file to show such items as tabs, spaces and line endings that are otherwise invisible. See [Viewing Special Characters](#) for more information about these settings.

The graphic displayed for tab and space characters is not configurable. Since some customers like to use background color to view tabs or spaces, you can turn off drawing for the tab and/or space character with the "Display tab graphic" and "Display space graphic" check boxes.

Appearance Options

Special Characters

Non-Unicode Editor Windows

	Char	Code
End-Of-File	¬	172
Formfeed	¶	164
End-Of-Line	¶	182
Carriage Return	:	58
Line Feed)	41
Other Ctrl Characters	¤	164

View numbers in Hex Dec

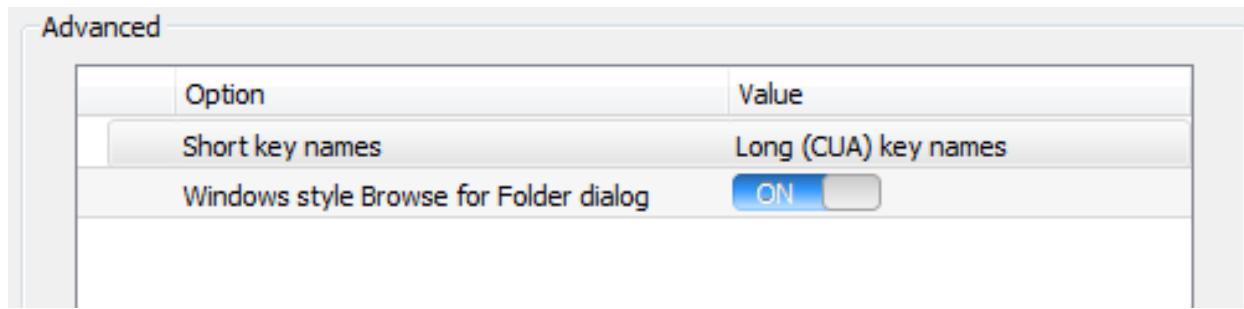
Unicode Editor Windows

ASCII	Char	Name	Value
		End-Of-Line	¶
0	^@	NUL	<NUL>
1	^A	SOH	<SOH>
2	^B	STX	<STX>
3	^C	ETX	<ETX>
4	^D	EOT	<EOT>
5	^E	ENQ	<ENQ>
6	^F	ACK	<ACK>
7	^G	BEL	<BEL>
8	^H	BS	<BS>
9	^I	HT	<HT>
10	^J	LF	<LF>
11	^K	VT	<VT>
12	^L	FF	<FF>
13	^M	CR	<CR>
14	^N	SO	<SO>
15	^O	SI	<SI>
16	^P	SLE	<DLE>

Display tab graphic Display space graphic

Advanced Appearance Options

Advanced appearance options are shown below (**Tools** → **Options** → **Appearance** → **Advanced**).



The options are described as follows:

- **Short key names** - Specifies how keyboard shortcuts are displayed in the MDI menu bar. For example, when **Long** is selected, the **Edit → Undo** menu item shows **Ctrl+Z** for the shortcut. This is traditional for the CUA emulation. When **Short** is selected, **C-Z** is displayed, traditional in other emulations.
- **macOS style Browse for Folder dialog** - (Mac only) When set to **On**, a Mac-style Browse for Folder dialog box is used to choose directories when possible. When set to **Off**, the standard Choose Directory dialog is used at all times. This dialog displays the navigation tree on a disk-drive basis, starting with the current drive, and includes a text box that supports [Directory Aliases](#) that help you quickly type the directory name.
- **Windows style Browse for Folder dialog** - (Windows only) When set to **On**, a Windows-style Browse for Folder dialog box is used to choose directories when possible. This dialog displays the navigation tree in a more Windows-friendly structure, letting you navigate to items such as the Desktop, My Computer, etc. When set to **Off**, the standard Choose Directory dialog is used at all times. This dialog displays the navigation tree on a disk-drive basis, starting with the current drive, and includes a text box that supports [Directory Aliases](#) that help you quickly type the directory name.
- **Maximize editor in full screen mode** - When set to **On**, fullscreen mode maximizes the editor to use the entire screen. Otherwise, fullscreen just swaps in the fullscreen toolbars, but the editor maintains the same size and position. On Unix, maximize is not guaranteed to work since the window manager controls sizing.
- **Show MDI menu in full screen mode** - (Windows and Unix only) When set to **On**, fullscreen mode shows the MDI menu. Otherwise, the MDI menu is hidden in fullscreen mode.

Keyboard and Mouse Options

Keyboard and Mouse options (**Tools → Options → Keyboard and Mouse**) pertain to use of the keyboard and mouse, and include options for setting the emulation you want to use and creating custom key bindings.

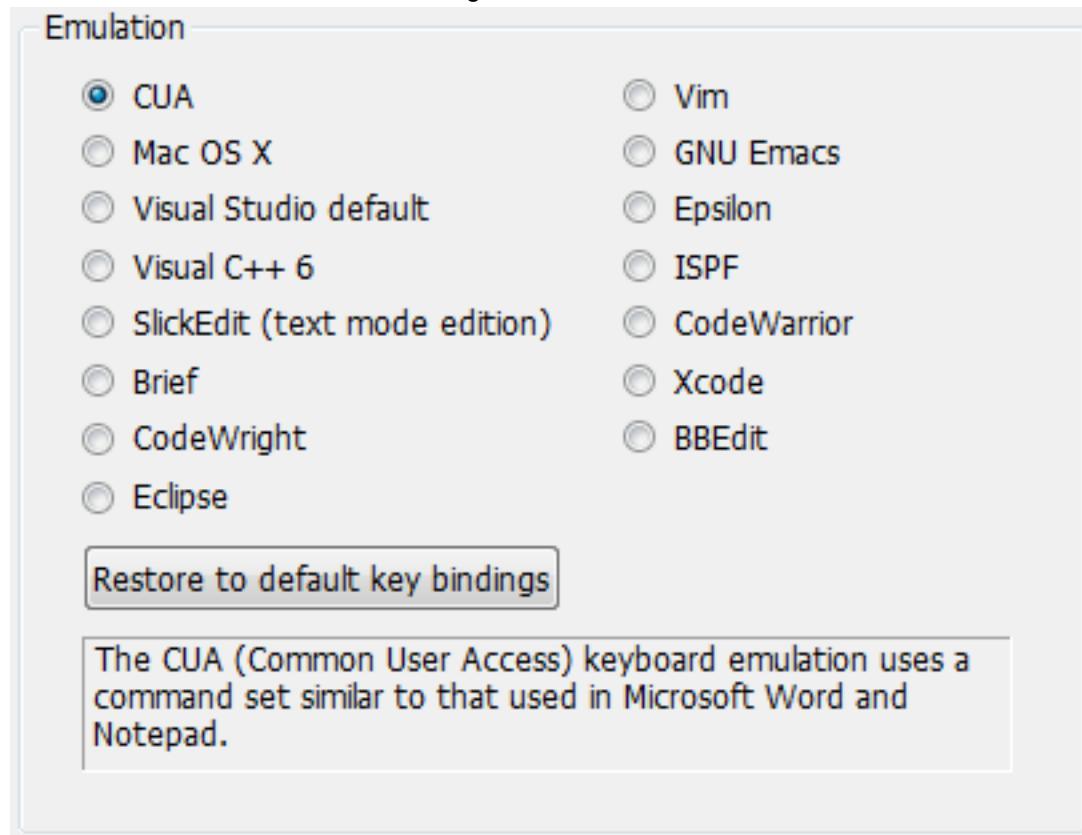
Keyboard option categories are:

- [Emulation Options](#)
- [Key Binding Options](#)
- [Redefine Common Key Options](#)

- [Advanced Keyboard and Mouse Options](#)
- [Vim Options](#)
- [ISPF Options](#)

Emulation Options

Emulation options (**Tools** → **Options** → **Keyboard and Mouse** → **Emulation**) are shown below. Use these options to specify the editor's emulation mode and to restore default key bindings. Be sure to save your custom bindings before switching emulations. This can be done by exporting (click **Key Bindings** in the Options tree) or by using the prompt that is displayed when you switch emulations. See [Emulations](#) for more information about these settings.



Key Binding Options

Key binding options are shown below (**Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**). From here, you can view, create, and manage key binding associations for SlickEdit® commands and user-recorded macros. You can also import, export, or save a chart of your key bindings. See [Managing Bindings](#) for more information.

Keyboard and Mouse Options

Key Bindings

Search by command: Search by key sequence: OR

Command	Key Sequence	Mode	Recorded
+			N
-			N
--ex-bufdo			N
-SetSymbolColoringSchemeNam			N
/			N
0			N
abort			N
actionscript-begin			N

Moves the cursor the specified number of lines down.

Parameters:
NofLines - Number of lines

AppliesTo:
Edit_Window, Editor_Control

Categories:
CursorMovement_Functions

Import... Export... Save Chart... Remove Add...

Click Add to add binding to current command. Click Remove to unbind current selected command.

Note

- The first time the Key Bindings option screen 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). To change the emulation mode, click **Tools** → **Options** → **Emulation**. For more information, see [Emulations](#).

The Key Bindings option screen is described as follows:

- 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 button 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 button 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

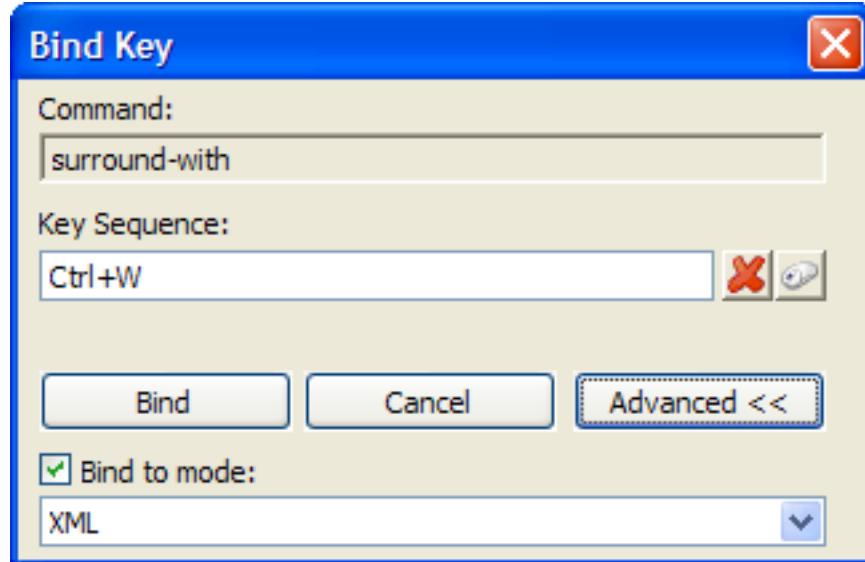
To change the mode for a command/macro that is already bound, first you should unbind the command/macro, then recreate the binding with the mode you want to use. See [Editing Bindings](#) for more information. For information about editing modes, see [Language Editing Mode](#).

- **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 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 above it. The size is remembered the next time the screen 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.
- **Remove** - This button clears the binding for the selected command/macro. You can also press the **Delete key** to clear the binding.

- **Add** - This button displays the Bind Key dialog, which is used to initiate a new binding. See [Bind Key Dialog](#) and [Creating Bindings](#) for more information.

Bind Key Dialog

This dialog is used to initiate a new key binding and is displayed when you click **Add** on the Key Bindings option page.



The dialog is described as follows:

- **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** button to clear the filter.
- **Mouse Event button** - Click this button located next to the red **X** button to pick a mouse event to use for the binding. If the event involves pressing a modifier key or keys, such as **Ctrl**, **Alt**, **Shift**, **Cmd**, **Ctrl+Alt**, etc., in conjunction with a mouse click, for example, **Ctrl+RButtonDn**, press the modifier key(s) when clicking the **Mouse Event** button. Then the Select Mouse Event dialog shows a list of modifier-prefixed mouse events. After selecting the mouse event you want to look up, click **OK**. The Key Sequence field updates to show the selected mouse event.
- **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. 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 **Bind to mode** and click

the language editing mode from the drop-down list.

Redefine Common Key Options

Redefine Common Key options (**Tools** → **Options** → **Keyboard and Mouse** → **Redefine Common Keys**) allow you to change the behavior of certain common keys. See also [Redefining Common Keys](#) for more information.

Redefine Common Keys	
Option	Value
Backspace in Replace mode	Remove previous character only
Backspace over tab	Remove tab
Redefinable keys	
Backspace key	Cursor wraps to previous line (Default)
Delete key	Next line always joins (Default)
End key	Moves cursor to end of line (Default)
Enter key	Splits current line at cursor (Default)
Home key	Toggles cursor between first non-blank character and column one (Default)

The options are described as follows:

- **Backspace in Replace mode** - Specifies the behavior of the Backspace key when the Start mode is set to Replace (**Tools** → **Options** → **Editing** → **General** → **Start mode**). When **Remove previous character only** is selected, Backspace removes the previous character and moves the cursor left. Otherwise the previous character is replaced with a space.
- **Backspace over tab** - Specifies the behavior of the Backspace key when the previous character is a tab. When **Convert tab to spaces and remove 1 space** is selected, the Backspace key deletes through tab characters one column at a time.
- **Redefinable keys** options are:
 - **Backspace key** - Specifies when the cursor is allowed to wrap to the previous line when pressing the Backspace key at the left margin.
 - **Delete key** - Specifies the behavior of the Delete key when the rest of the current line is empty. When **Next line always joins** is selected, the line below the current line is joined with the current line.
 - **End key** - Specifies where the cursor is placed when pressing the End key. The Toggle option is useful for trimming extra spaces from long lines, because it gives you a natural and quick way to get to the last non-blank column. The Ignore trailing whitespace option is similar, however it does not move the cursor to the real end of a line that has trailing whitespace.
 - **Enter key** - Specifies whether a line is split when pressing Enter and how the cursor is aligned on the new line. **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** means that if the **Start mode** is set to **Insert** (**Tools** → **Options** → **Editing** → **General** → **Start mode**), 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 the **Start mode** is set to **Replace**, the cursor is moved to column one of the next line.

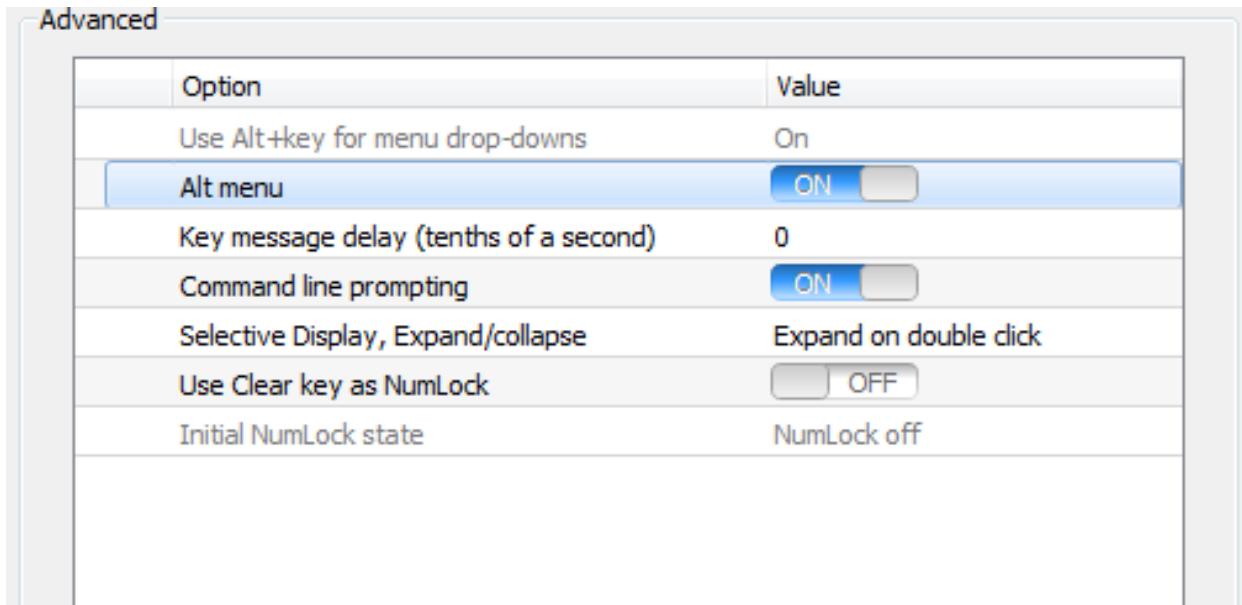
Note

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 key** - Specifies where the cursor is placed when pressing the Home key.
- **Home/End behavior** options are:
 - **Home/End within soft wrap lines** - When set to **On** and Soft Wrap is enabled, pressing the **Home** or **End** keys will move the cursor within the current partial line. For more information, see [Soft Wrap](#).
 - **End stops at vertical line column(s)** - When set to **On**, pressing the **End** key repetitively will stop at the vertical line column(s) before stopping at the end of the line. For more information, see [Vertical line columns](#).
 - **End aligns multiple cursors to longest line** - When set to **On**, pressing the **End** key with multiple cursors will toggle between the actual line end and aligning the cursors at the end of the longest line. For more information, see [Multiple Cursors and Selections](#).

Advanced Keyboard and Mouse Options

Advanced keyboard options are shown below (**Tools** → **Options** → **Keyboard and Mouse** → **Advanced**).



The options are described as follows:

- **Use Alt+key for menu drop-downs** - (Non-CUA emulation modes and non-Mac systems only) When set to **On**, "Alt"-prefixed keyboard shortcuts display the corresponding drop-down menu. When set to **Off**, you can be more selective about key bindings because you are permitted to bind Alt keys you normally could not, such as Alt+F. Set to **Off** if you bind Alt keys that are normally menu keys; otherwise, you will lose these key bindings.
- **Alt menu** - When set to **on**, pressing the Alt key by itself causes the focus to shift to the menu bar. The hotkeys in the menu names are underlined. When this value is set to **off**, pressing the Alt key has no effect.
- **Use Command+key for dialog hotkeys** - (Mac only) When this option is set to **On**, the Command key is used for dialog hotkeys.
- **Use Command+key for menu drop-downs** - (Non-macOS emulation modes and Mac systems only) When set to **On**, "Command"-prefixed keyboard shortcuts display the corresponding drop-down menu. When set to **Off**, you can be more selective about key bindings because you are permitted to bind Command keys you normally could not, such as Command+F. Set to **Off** if you bind Command keys that are normally menu keys; otherwise, you will lose these key bindings.
- **Mac Option/Alt key behavior** - (Mac only) Select **Default Mac IME (Extended ASCII entry)** to use Option+key for entering extended ASCII symbols (default macOS behavior). Set this option to **Use as Windows-style Alt key modifier** to use Option+key for user-defined key bindings.
- **Key message delay** - Specifies the maximum delay, in tenths of a second, between two key combinations when used as a single key binding (for example, Ctrl+X,Ctrl+C). If the time limit is exceeded between when the two key combinations are pressed, the key sequence is interpreted as two separate bindings.
- **Command line prompting** - When set to **On**, pressing a key binding that normally opens a dialog box causes the SlickEdit® command line to prompt for arguments instead of opening the dialog. For

example, instead of displaying the Open file dialog, Ctrl+O (bound to **gui_open**) opens the command line prompting for the file to open. See [Command Line Prompting](#) for more information.

- **Selective Display, Expand/collapse** - Specifies how Selective Display expand/collapse bitmaps ([View → Selective Display](#)) are clicked in order to expand/collapse areas. For more information, see [Selective Display](#).
- **Use Clear key as NumLock** - When set to **On**, the Clear key will behave as the NumLock key.
- **Initial NumLock state** - Sets the initial value of NumLock when the application is started. Only applies when **Use Clear key as NumLock** is set to **On**.

Vim Options

The following options are specific to the Vim emulation and are only available after you have selected it. See [Emulation Options](#).

Vim Options	
Option	Value
Enter command mode on ESC during codehelp	<input type="button" value="OFF"/>
Change cursor shape between modes	<input checked="" type="button" value="ON"/>
Verbose Ex mode prompt	<input checked="" type="button" value="ON"/>
Always highlight search results	<input checked="" type="button" value="ON"/>
Start in command mode	<input type="button" value="OFF"/>

- **Enter normal mode on ESC during codehelp** - when set to **On**, pressing the Escape key during any codehelp or auto-complete will dismiss the dialog and switch to normal mode.
- **Change cursor shape between modes** - when set to **On**, the cursor will change shape when switching between insert mode and normal mode.
- **Verbose Ex mode prompt** - when set to **On**, a warning is displayed in front of the prompt when staying in Ex mode.
- **Always highlight search results** - when set to **On**, the editor will always highlight search results.
- **Start in normal mode** - when set to **On**, the editor will switch to normal mode any time you switch to a different buffer.

ISPF Options

The following options are specific to the ISPF emulation and are only available after you have selected it. See [Emulation Options](#).

ISPF Options	
Option	Value
Prefix area width	6
Display prefix area for readonly files	<input checked="" type="checkbox"/> ON
Enter places cursor in prefix area	<input type="checkbox"/> OFF
Right CTRL = Enter/Send	<input type="checkbox"/> OFF
Cursor page up/down	False
END command saves the file	<input checked="" type="checkbox"/> ON
XEDIT line commands	<input type="checkbox"/> OFF
Home key places cursor on command line	<input type="checkbox"/> OFF

- **Prefix area width** - Sets the width of the prefix area.
- **Display prefix area for readonly files** - By default, the prefix area is not displayed for read-only files. Since the prefix area can be used to enter commands, you may wish to have the prefix area visible for these files.
- **Enter places cursor in prefix area** - When set to **On**, pressing the **Enter** key places the cursor in the prefix area on the next line.
- **Right CTRL = Enter/Send** - When set to **On**, pressing the right-hand **CTRL** key sends the command.
- **Cursor page up/down** - When set to **On**, pressing **PageUp** or **PageDown** will move the current line to the top or bottom of the screen, respectively. If the current line was already at the top/bottom of the screen, then the display is scrolled one page. When **Off**, the display is always scrolled one page.
- **END command saves the file** - When set to **On**, the END command saves the file before closing the file. When set to **Off**, you will be prompted whether to save.
- **XEDIT line commands** - When set to **On**, allows the use of XEDIT commands.
- **Home key places cursor on command line** - When set to **On**, pressing the **Home** key puts the cursor on the SlickEdit command line.

Editing Options

Editing options (**Tools** → **Options** → **Editing**) directly impact your SlickEdit® coding experience. By customizing these options so that SlickEdit® works the way you prefer and to which you are accustomed, you can greatly improve your coding speed and efficiency. Editing options include default search/replace values, selection styles, specifying the size of new editor windows, and more.

Editing option categories are:

- [General Editing Options](#)

- [Editor Window Options](#)
- [Cursor Movement](#)
- [Context Tagging Options](#)
- [Selection Options](#)
- [Search Options](#)
- [Bookmarks](#)
- [Auto-Close](#)
- [Global Alias Options](#)

General Editing Options

General editing options are shown below (**Tools** → **Options** → **Appearance** → **General**).

Editing Options

General	
Option	Value
Apply .editorconfig, .seeditorconfig.xml settings	Off <input type="button" value="..."/>
Start mode	Start in Insert mode
Line insert style	After
Next word style	End
Maximum clipboards	50
Allow drag drop of text	<input checked="" type="radio"/> ON <input type="radio"/> OFF
Throw away file lists	<input checked="" type="radio"/> ON <input type="radio"/> OFF
Auto exit build window	<input checked="" type="radio"/> ON <input type="radio"/> OFF
Reflow next	Cursor at same position
Protect read-only mode	<input checked="" type="radio"/> ON <input type="radio"/> OFF
CUA text box	<input checked="" type="radio"/> ON <input type="radio"/> OFF
Preserve column on top/bottom	<input type="radio"/> OFF <input checked="" type="radio"/> ON
CR w/o LF erases line in build window	<input type="radio"/> OFF <input checked="" type="radio"/> ON
Show VSLICKERRORPATH in build window	<input type="radio"/> OFF <input checked="" type="radio"/> ON
Show change directory commands in build window	<input type="radio"/> OFF <input checked="" type="radio"/> ON
Maximum tab expansion column	256
Show extra line after last newline	<input type="radio"/> OFF <input checked="" type="radio"/> ON
Parenthesis matching	
Parenthesis matching style	Highlight
Highlight matching blocks	<input checked="" type="radio"/> ON <input type="radio"/> OFF
Highlight matching blocks after (ms) idle	200
Highlight matching blocks timeout (ms)	1000
Maximum nesting level for Highlight matching blocks	500
Maximum distance for Highlight matching blocks (KB)	100
Turn off Highlight matching blocks when file larger than (KB)	1024
Large File Editing	
Use Plain Text mode when file larger than (KB)	50000
Turn off undo when editing file larger than (KB)	100000
Turn off soft wrap when editing file larger than (KB)	100000
Turn off view line numbers when editing file larger than (KB)	100000
Turn off minimap when editing file larger than (KB)	100000
Turn off check for inconsistent line endings when editing file larger than (KB)	8000

The options are described as follows:

- **Apply .editorconfig, .seeditorconfig.xml settings** - Enables/disables .editorconfig and .seeditorconfig.xml settings. The supported .editorconfig properties are indent_style, indent_size, tab_width, end_of_line, and trim_trailing_whitespace. For documentation on .editorconfig files, go to <http://editorconfig.org/>.

.seeditorconfig.xml support allows you to specify beautifier profile overrides for specific source trees (i.e. different project source trees). Use Tools>Beautify>Beautifier Profile Overrides to create beautifier profile overrides for a specific source tree.

- **Start mode** - Specifies the default insert/replace editing mode to use each time the editor is invoked. The editing mode is indicated in the status line of the editor (**Ins** or **Rep**). You can also click on the indicator to toggle the editing mode.
- **Line insert style** - SlickEdit® treats line selections differently than character selections. This option controls whether lines are inserted before or after the current line when you paste a line selection. This feature saves you from having to tediously position the cursor at the beginning or end of a line prior to pasting.
- **Next word style** - Specifies the cursor behavior when navigating with the **next_word** command (Ctrl+Right). When **Begin** is selected, the cursor is placed on the beginning of the next word. When **End** is selected, the cursor is placed at the end of the next word.
- **Maximum 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 **Ctrl+Shift+V**.
- **Allow drag/drop of text** - When set to **On**, selected text can be copied or moved by dragging and dropping the selection using the left mouse button.
- **Throw away file lists** - When set to **On**, File Manager file lists (**File → File Manager**) can be modified and closed without a save prompt.
- **Auto exit build window** - When set to **On**, the concurrent build window is automatically exited when the buffer is closed or when exiting the editor.
- **Reflow next** - Specifies where the cursor is placed when running the **reflow_paragraph** command. When **Cursor on next paragraph** is selected, the cursor is placed on the next paragraph after it has reformatted the current paragraph.
- **Protect read-only mode** - When set to **On**, read-only files cannot be modified. If you attempt to modify a read-only file, SlickEdit® displays a notification. If you attempt to save a read-only file, SlickEdit® prompts for a different output file name.
- **CUA text box** - When set to **On**, the keys Ctrl+X, Ctrl+C, and Ctrl+V perform cut, copy, and paste commands respectively for text boxes other than the command line, regardless of the emulation. When **Off**, these keys operate the same in a text box as they do in the command line and edit windows, which could be useful if you're using a non-CUA emulation or prefer to use your own editing key bindings.
- **Preserve column on top/bottom** - When set to **On**, 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.
- **CR w/o LF erases line in build window** - When set to **On**, lines of output sent to the Build tool window that contain carriage return (CR) characters without subsequent line feed (LF) characters are erased. When set to **Off**, these lines are not erased.

Tip

Where does this concept originate? On manual typewriters, carriage return (CR) moves the carriage back to the first column of text, and line feed (LF) moves to the next line. Older computer terminals, such as VT100 and its successors (Windows and UNIX shells), used this terminology as a metaphor to redraw lines. For example, to draw a line that showed percent complete, you would output "CR <number>%" repeatedly, then issue the line feed when it finished. In SlickEdit® therefore, due to this practice, it is possible that output sent to the Build tool window may contain carriage return characters without subsequent line feed characters. When this happens, the line is erased in the Build window.

- **Show VSICKERRORPATH in build window** - When set to **On**, VSICKERRORPATH lines sent to the Build tool window are displayed. Otherwise, they are marked as hidden lines.
- **Show change directory commands in build window** - When set to **On**, change directory commands sent to the Build tool window are displayed. Otherwise, they are marked as hidden lines.
- **Show extra line after last newline** - When set to **On** and the last line of a file has a newline, an extra line is displayed. When set to **On**, newly created files will start with a single line that is not terminated with a newline.

Tip

If you like turning this option on, you may want to rebind your select all key (Ctrl+A by default) to **select_all_char**. Otherwise, the **select_all** command will use a LINE selection which causes paste to work differently than products that typically work this way.

- **Parenthesis matching** - See [Begin/End Structure Matching](#) for more information about parenthesis matching. The following options available:
 - **Parenthesis matching style** - When **Highlight** is selected, after typing a closing parenthesis, SlickEdit® temporarily block-selects the text within the parenthesis pair. When **Cursor to Begin Parenthesis** is selected, after typing a closing parenthesis, SlickEdit® temporarily places the cursor on the matching begin parenthesis. When **None** is selected, SlickEdit® just inserts the closing parenthesis.
 - **Highlight matching blocks** - When set to **On**, the corresponding parenthesis, brace, bracket, or begin/end word pair under the cursor is automatically highlighted.
 - **Highlight matching blocks after (ms) idle** - Determines idle time delay before matching blocks are highlighted.
 - **Highlight matching blocks timeout (ms)** - Determines timeout for aborting highlighting matching blocks.
 - **Maximum nesting level for Highlight matching blocks** - Specifies the maximum level of recursive block searching to perform to highlight the corresponding parenthesis, brace, bracket, or begin/end

word pair under the cursor before returning control to the editor.

- **Maximum distance for Highlight matching blocks (KB)** - Specifies the maximum distance, in kilobytes, to search to highlight the corresponding parenthesis, brace, bracket, or begin/end word pair under the cursor before returning control to the editor.
- **Turn off Highlight matching blocks when file larger than (KB)** - When the current file is larger than this value, SlickEdit will not automatically highlight the corresponding parenthesis, brace, bracket, or begin/end word pair under the cursor.

Tip

To customize the highlight color, go to **Tools** → **Options** → **Appearance** → **Colors**, 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/Changing Configuration Variables](#) for more information.

- **Large File Editing** - These options apply when working with large files. Use these settings to improve performance when working with such files.
 - **Use Plain Text mode when file larger than (KB)** - When the current file is larger than this value, it will automatically be opened in Plain Text mode, turning off most language-related editing features and improving performance.
 - **Turn off undo when editing file larger than (KB)** - When editing a file larger than this value, undo capabilities will be turned off to improve performance.
 - **Turn off soft wrap when editing file larger than (KB)** - When editing a file larger than this value, soft wrap capabilities will be turned off to improve performance.
 - **Turn off view line numbers when editing file larger than (KB)** - When editing a file larger than this value, viewing line numbers will be turned off to improve performance.
 - **Turn off minimap when editing file larger than (KB)** - When editing a file larger than this value, the minimap window will be turned off to improve performance.
 - **Turn off check for inconsistent line endings when editing file larger than (KB)** - When editing a file larger than this value, SlickEdit will turn off checking for inconsistent line endings on open to improve performance.

Editor Window Options

Editor window options are shown below (**Tools** → **Options** → **Editing** → **Editor Windows**). See also [Files, Buffers, and Editor Windows](#) for more information.

Editor Windows	
Option	Value
Zoom (hide tabs) when one window	Auto
Files per window	One file per window
Smart next window style	Smart next window
Place cursor on focus click	<input type="button"/> OFF
File tab sort order	Alphabetical
New file tab position	New files on right
Abbreviate file tab captions	<input checked="" type="button"/> ON
Show close buttons on document tabs	<input checked="" type="button"/> ON
File tab title	Name only
Color modified document tabs	<input type="button"/> OFF
Split window style	Evenly
Tab double-click action	Do nothing
Tab middle-click action	Do nothing

The following options are available:

- **Zoom (hide tabs) when one window** - When there is only one editor window, the application can hide the document tabs. Select from the following choices:
 - **Always** - Always hide the document tabs when there is one editor window.
 - **Auto** - Figures out to hide the document tabs based on whether the last use of the `zoom_toggle` command.
 - **Never** - Never hide the document tabs when there is one editor window.
- **Files per window** - Specifies whether each buffer opened in SlickEdit® is allocated in its own editor window or in the same editor window.
- **Hide maximized child window titlebars** - (Mac only) When set to **On**, maximized editor windows will not display a titlebar.
- **Mac resize borders** - (Mac only) When set to **On**, editor windows can be resized from any edge or corner.

- **Smart next window style** - Specifies preferences for navigating between editor windows. Select from the following values:
 - **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.

Note

Under the Gnome desktop environment, **Smart next window** may not work correctly when the mouse option 'Highlight the pointer when you press Ctrl' is enabled.

- **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. 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 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.

Note

In all cases, cycling through the windows using **Ctrl+Tab** or **Ctrl+Shift+Tab** does not affect the order of the windows. Specifically activating a file by clicking on a file tab or selecting it in the Files tool window will reorder the windows unless you have set this option to "No window reordering".

- **Place cursor on focus click** - When set to **On**, clicking in an editor window that does not have focus will set focus and also place the text cursor. When set to **Off**, clicking in an editor window that does not have focus only sets the focus.
- **File tab sort order** - This setting controls the order of the file tabs and document tabs. The following choices are available:
 - **Alphabetical** - File tabs are listed alphabetically by file name.
 - **Most recently opened** - File or document tabs are listed in the order they were opened. Most recently opened files are positioned according to the **New file tab position** setting. This order is useful if you prefer a static order, since switching buffers does not change the order.
 - **Most recently viewed** - File or document tabs are listed in the order in which they were last viewed. The current (and most recently viewed) file is at the left edge. Switching buffers reorders the tabs.

- **Manual** - Enables you to drag and drop file or document tabs to the position where you want them. New files are opened according to the **New file tab position** setting.
- **New file tab position** - This option specifies whether file or document tabs for newly opened files appear on the right or the left of the file tabs, or to the right or the left of the current file tab. This option is only available when **File tab sort order** is set to **Most recently opened** or **Manual**.
- **Show close buttons on document tabs** - When set to **On**, individual file or document tabs will have a close button to allow you to quickly close files.
- **Document tab title** - Sets how the file name is displayed on the document tabs. Choose from the following options:
 - **Full path** - display the full path of the file (Example: **C:\work\foo.cpp**).
 - **Name followed by full path** - display the file name followed by the full path, including the file name (Example: **foo.cpp - C:\work\foo.cpp**).
 - **Name followed by path** - to see the file name, followed by the path, without the file name at the end (Example: **foo.cpp - C:\work**).
 - **Name only** - display the file name only (Example: **foo.cpp**).
- **Abbreviate file tab captions** - When set to **On**, files with the same name but different extension will have their captions abbreviated for all but the first file. For example, if you have files **foo.cpp** and **foo.h**, then the file tab for **foo.h** will directly follow **foo.cpp**, but the caption will only say **.h**. This feature is only available when **File tab sort order** is set to **Alphabetical**.
- **Hide known file extensions** - When set to **On**, files with specified file extensions will be displayed without their extension in order to conserve space. When set to **Off**, the full file name is shown for every file. This is useful for languages such as Java where all the source files have the same extension, so it is a bit redundant to show the file extensions in the file or document tabs. This feature is only available when **File tab title** is set to **Name only**.
- **Hidden file extensions** - Determines which file extension to hide in order to conserve space in the document tabs.
- **Color modified document tabs** - When set to **On**, document tab captions for modified files will be colored.
- **Split window style** - Determines how newly split windows are sized when using the **hsplit_window** (**Window → Split Horizontally**) or **vsplit_window** (**Window → Split Vertically**) commands. The following choices are available:
 - **Evenly** - Divides up available space evenly among all current windows.
 - **Strict halving** - Divides the window being split into two evenly-sized windows. Does not affect other currently open windows.
 - **Tab double-click action** - Specifies the action to be performed when a document tab is double-clicked.

The following choices are available:

- **Close file** - Closes the selected file.
- **Do nothing** - Performs no action.
- **One window** - Zooms the current window and deletes all other windows.
- **Split horizontal** - Splits the selected file horizontally.
- **Split vertical** - Splits the selected file vertically.
- **Zoom toggle (Document Tabs)** - Zooms the selected file, hiding the document tabs.
- **Tab middle-click option** - Specifies the preferred action when a document tab is middle-clicked. For the available choices, see [Tab double-click action](#).
- **Vertical scrolling switches tabs** - When set to **On**, allow mouse vertical scroll wheel actions to cycle through document tabs. When set to **Off**, vertical scrolling has no effect on the document tabs.
- **Horizontal scrolling switches tabs** - When set to **On**, allow mouse horizontal scroll wheel actions to cycle through document tabs. When set to **Off**, horizontal scrolling only shifts the document tabs.

Cursor Movement

Cursor Movement options are shown below (**Tools** → **Options** → **Editing** → **Cursor Movement**). These options control the movement of the cursor within editor windows.

Option	Value
Cursor right/left wraps to next/previous line	<input checked="" type="checkbox"/> ON
Cursor up/down places cursor in virtual space	<input type="checkbox"/> OFF
Click past end of line	<input type="checkbox"/> OFF
Cursor up/down within soft wrapped lines	<input checked="" type="checkbox"/> ON
Cursor left/right in leading spaces	Move cursor by one physical character
Line wrap	Wrap when column one is reached
Strip leading spaces when joining lines	<input checked="" type="checkbox"/> ON
Jump over tab characters	<input checked="" type="checkbox"/> ON
Subword Navigation	<input type="checkbox"/> OFF
Undo affects cursor movement	<input type="checkbox"/> OFF

The following options are available:

- **Cursor right/left wraps to next/previous line** - When set to **On**, the **cursor_left** and **cursor_right** commands wrap to the previous or next line respectively.

- **Cursor up/down places cursor in virtual space** - When set to **Off**, the **cursor_up** and **cursor_down** commands place the cursor up or down, respectively, at either the end of the line or at the column of the original location, whichever comes first. The cursor is never placed past the end of the line. When set to **On**, **cursor_up** and **cursor_down** go to the same column of the next or previous line, regardless of the length of the line.
- **Click past end of line** - When set to **On**, the cursor can be placed past the end of a line into virtual space.
- **Cursor up/down within soft wrapped lines** - When set to **On**, if Soft Wrap is enabled (**View → Soft Wrap**), the **cursor_up** and **cursor_down** commands move the cursor up to the next or previous visible line, including line continuations. When set to **Off**, **cursor_up** and **cursor_down** moves the cursor to the previous or next physical line (the same position to which the cursor would move if Soft Wrap was off).
- **Cursor left/right in leading spaces** - Specifies the behavior of the **cursor_left** and **cursor_right** commands when moving the cursor within leading space. The purpose of this option is to emulate the "feel" of real tab characters even if you only use spaces for indentation.
- **Line wrap** - Specifies when line wrapping occurs, either when the cursor reaches column one or when it reaches the left margin. When language-specific Word Wrap is on (**Tools → Options → Languages → [Language Category] → [Language] → Word Wrap**), wrapping occurs when the left margin is reached regardless of this setting.
- **Strip leading spaces when joining lines** - When set to **On**, hitting Delete at the end of a line to join with the following line will strip the leading spaces from the following line, effectively joining to the first non-blank character. When set to **Off**, the leading spaces from the following line will be preserved.
- **Jump over tab characters** - When set to **On**, moving the cursor over a tab character with the Left or Right arrow key causes the cursor to jump across the virtual space. When set to **Off**, the Left and Right arrow keys move the cursor into the virtual space of tab characters. Note that this setting also controls where the cursor is placed when clicking in a buffer or making a selection.
- **Subword navigation** - When set to **On**, the word navigation commands, like **next_word**, behave like their subword navigation counterparts, like **next_subword**. Word navigation jumps to the next word based on the **Word chars** value set at **Tools → Options → Languages → [Language Category] → [Language] → General**. Subword navigation stops within a word at each capital letter or after each underscore or dash, making it easier to edit the name of a symbol. For more information, see [Subword Navigation](#).
- **Undo affects cursor movement** - When set to **On** cursor movement is added to the undo stack, so undo operations will also undo cursor movement and edits.
- **Create multiple cursors with Ctrl+Click** - When set to **On**, **Ctrl+Click** can be used to create multiple cursors. Turning off this feature is useful when working on a laptop keyboard where it is easy to accidentally click by touching the touchpad. You can still create multiple cursors using **Ctrl+J**. See also [Multiple Cursors and Selections](#) for more information.

Background Tagging Options (Pro only)

Editing Options

Background Tagging options are shown below (**Tools** → **Options** → **Editing** → **Background Tagging**). These options let you set general parameters for background tagging. You can designate whether or not Background Tagging is performed and where or not to utilize threaded Background Tagging, as well as tune the application to maximize performance. See also [Building and Managing Tag Files](#) for more information.

Background Tagging	
Option	Value
General	
Use background tagging	<input checked="" type="checkbox"/> ON
Use background tagging threads	<input checked="" type="checkbox"/> ON
Tag file on save	<input checked="" type="checkbox"/> ON
Background tagging of open files	
Background tagging of open files	<input checked="" type="checkbox"/> ON
Use background thread when possible	<input checked="" type="checkbox"/> ON
Start after seconds idle	2
Background updating of tag files	
Update workspace tag file on open	<input checked="" type="checkbox"/> ON
Update workspace tag file on activate	<input checked="" type="checkbox"/> ON
Background tagging of other files	<input type="checkbox"/> OFF
Workspace tag file only	On
Start after minutes idle	1
Minutes before restarting	10
Background Tagging Threads	
Number of tagging threads to start	4
Use background thread to build workspace tag file when possible	<input checked="" type="checkbox"/> ON
Use background thread to build language support tag files when possible	<input checked="" type="checkbox"/> ON
Report background tagging progress on status bar	<input checked="" type="checkbox"/> ON
Process background tagging jobs after (ms) idle	500
Background tagging timeout (ms)	250
Maximum number of active tagging jobs	1000
Maximum amount of background tagging memory usage (MB)	31

The following options are available:

- **General:**
 - **Use background tagging** - When set to **On**, background tagging will keep your tag files up-to-date when source files change on disk or files are added and/or removed from your project.

When set to **Off**, all background tagging features except for **Tag file on save** and **Tag file on switch buffer** will be disabled. In this case, the only way to update tag files will be by using the **Tag Files** dialog (invoked from **Tools** → **Tag Files**) or by updating your workspace or project tag files manually

using **Project → Retag workspace** or **Project → Retag project**, respectively.

- **Use background tagging threads** - When set to **On**, background tagging will use background threads to parse source files and update tag files. This is designed to maximize performance by parsing multiple files in parallel, as well as minimize delays in the editor. When set to **Off**, all background tagging features will operate on a timer in the main GUI thread, which can lead to delays.
- **Tag file on save** - When set to **On**, files are retagged when you save a modified file.
- **Tag file on switch buffer** - When set to **On**, files are retagged when you switch away from a modified file.
- **Background tagging of open files:**
 - **Background tagging of open files** - When set to **On**, all open files are retagged in the background if they have been modified.
 - **Use background thread when possible** - When set to **On**, open files are tagged using a separate thread instead of using a timer on the main thread.
 - **Start after seconds idle** - Specifies the amount of time, in seconds, the editor remains idle (no keyboard or mouse movements) before retagging of buffers starts, when **Background tagging of open files** is enabled.
- **Background updating of other files:**
 - **Update workspace tag file on open** - When set to **On**, background tagging will update the workspace tag file on a thread when the workspace is opened.
If **Use background tagging threads** is not enabled, this option will also be disabled in order to prevent delays at startup or when switching workspaces.
 - **Update workspace tag file on activate** - When set to **On**, background tagging will update the workspace tag file on a thread when SlickEdit® loses focus or gains focus.
If **Use background tagging threads** is not enabled, this option will also be disabled in order to prevent delays when SlickEdit® regains focus.
 - **Background tagging of other files** - When set to **On**, tag files are updated when another application modifies a file. Note that this causes SlickEdit® to constantly perform disk I/O to check dates of files on disk.
 - **Workspace tag file only** - When set to **On**, background tagging cycles through only the workspace tag file. When set to **Off**, background tagging cycles through all of your language-specific tag files (listed under **Tools → Tag Files**) in addition to the workspace tag file.
 - **Start after minutes idle** - Specifies the amount of time, in minutes, the editor remains idle (no keyboard or mouse movements) before retagging of files on disk starts, when **Background tagging of open files** is enabled.
 - **Minutes before restarting** - Specifies the number of minutes to wait for background tagging to start

again after all files have been fully tagged.

- **Background Tagging Threads:**

- **Number of tagging threads to start** - The number of threads to be used for background tagging. This number should be based on the number of processors/cores on your computer and the other tasks you may be running while editing.
- **Use background thread to build workspace tag file when possible** - When set to **On** workspace files are initially built on a background thread instead of being built synchronously. This option applies only to languages that support threaded tagging.
- **Use background thread to build language support tag files when possible** - When set to **On** tag files for language support, like compiler libraries, are built initially using a background thread instead of being built synchronously. This option applies only to languages that support threaded tagging.
- **Report background tagging progress on status bar** - When set to **On** messages are written to the message line indicating the progress of background tagging.
- **Process background tagging jobs after (ms) idle** - Specifies the amount of idle time to wait before polling for completed background tagging jobs.
- **Background tagging timeout (ms)** - Specifies the maximum amount of time to spend gathering background tagging results before returning control to the editor.
- **Maximum number of active tagging jobs** - Specifies the maximum number of background tagging jobs to allow to be active in the background tagging processing queues at once. This setting is used to limit the amount of memory that background tagging can consume while running. You have to restart the editor for a change to this setting to take effect.
- **Maximum amount of background tagging memory usage (MB)** - Specifies the maximum amount of memory, in megabytes, that can be consumed by background tagging jobs in any stage before the tagging jobs need to be throttled back from reading any more files off of disk. This setting is used to limit the amount of memory that background tagging can consume while running. You have to restart the editor for a change to this setting to take effect.

Context Tagging® Options (Pro only)

Context Tagging options are shown below (**Tools** → **Options** → **Editing** → **Context Tagging**). These options let you set general parameters for the Context Tagging features. You can designate the way Context Tagging is performed and how the references function within the application, and you can also tune the application to maximize performance. See also [Building and Managing Tag Files](#) for more information.

Editing Options

Context Tagging®	
Option	Value
Tagging Tool Windows	
Update tool windows after (ms)	500
Additional time to wait before updating tool windows (ms)	250
Preview window symbol lookup timeout (ms)	1000
Show preview of symbols in tagging tool windows on mouse-over	<input checked="" type="checkbox"/> OFF
Show preview of symbols in tagging tool windows after (ms)	500
References	
Build workspace tag file with references	<input checked="" type="checkbox"/> ON
Find references incrementally (faster)	<input type="checkbox"/> OFF
Update references and call tree on single click	<input type="checkbox"/> OFF
Jump to first item when finding references	<input checked="" type="checkbox"/> ON
Search for word matches if symbol is not found	<input type="checkbox"/> OFF
Highlight references in editor	<input type="checkbox"/> OFF
Allow mixed language references	<input type="checkbox"/> OFF
Tag File Cache (tune for performance)	
Use memory mapped files	<input checked="" type="checkbox"/> ON
Tag file cache size (MB)	64
Tag file cache maximum (MB)	512
Use independent database file caches	On
Maximums (tune for performance)	
Maximum size of files to tag (KB)	2048
Maximum size of files for statement tagging (KB)	512
Maximum size of files for building token list (KB)	1024
Maximum number of tags per file	131072
Maximum time for parsing current file (ms)	2500
Maximum functions found by parameter help	100
Maximum class/struct members shown	1000
Maximum response time for list members (ms)	1000
Maximum candidates for list parameters	200
Maximum response time for list parameters (ms)	1000
Maximum tags found in symbol search	1000
Maximum items found in references search	1024
Update after (ms) idle (0 implies no delay)	0
Maximum response time (ms) for highlighting matching symbols	1000
Symbol coloring performance	
Update after (ms) idle	500
Timeout after (ms)	1000
Number of lines to color above and below the current page	100
Number of off-page lines to color per pass (chunk size)	20
Windows to color	Current window
Auto-Complete performance tuning	
Maximum symbols	100
Maximum function prototypes	20
Maximum word completions	100
Display after (ms) idle	250
Update after (ms) idle	50
Timeout after (ms) when automatic	500
Timeout after (ms) on demand	15000

The following options are available:

- **Tagging Tool Windows :**
 - **Update tool windows after (ms)** - Specifies the amount of idle time before the Preview window is updated to match the current location. Prevents the Preview window from showing results as you cursor through the code.
 - **Additional time to wait before updating tool windows (ms)** - Additional delay before updating other tool windows. This gives background threads more time to update the current file before other windows are updated.
 - **Preview window symbol lookup timeout (ms)** - Specifies the maximum amount of time that the preview window should spend trying to look up a symbol. If the symbol is not found in that time, it will not display the symbol preview. Smaller values will help prevent typing delays.
 - **Show preview of symbols in tool windows on mouse-over** - When set to **On**, hovering over symbols in tagging-related tool windows or in the file tabs will cause symbols to be shown in the preview window.
 - **Show preview of symbols in tool windows after (ms)** - Specifies the amount of time to delay, in milliseconds, when hovering over symbols in tool windows or in the file tabs before showing the item under the mouse in the preview window.
 - **Show info for symbol under mouse after (ms)** - Specifies the amount of time to delay, in milliseconds, when hovering over a symbol in an editor window before showing a symbol information popup for the symbol if available.
- **References** - Select from the following:
 - **Build workspace tag file with references** - When set to **On**, newly created tag files are built with support for symbol cross-references.
 - **Find references search strategy** - The References tool window supports multiple search strategies, depending on how immediately you want to see all the results of a references search.
 - **Find references in background** - By default, all files with potential references are searched and analyzed in the background. You may temporarily see files which do not have any valid references to the symbol you are looking for listed in the References tool window. Files which do not contain any references will be removed from the file list as they are processed. This option allows you to keep working while the references search completes.
 - **Find references incrementally** - This option allows reference searches to return control to the editor faster because analysis stops at the first file found containing a valid reference. However, you may see files which do not have any valid references to the symbol you are looking for listed in the References tool window. It will then resume the references search when you go to the next occurrence by pressing **Ctrl+G (Search → Next Occurrence)** or invoke the **find_next** command).
 - **Find all references immediately** - If a references search involves four or more files, a progress dialog is displayed while all the files with potential references are analyzed. Files which do not

contain any valid references are removed from the file list immediately. You can cancel the search by hitting **Cancel**.

- **Update references and call tree on single click** - When set to **On**, references in the References tool window are updated when you click on a new symbol in the Classes, Defs, or Symbols tool window.
- **Jump to first item when finding references** - When set to **Off**, Find Reference searches for references but does not jump immediately to the first reference. When set to **On**, Find Reference searches for references and automatically jumps to the first one. Note that you can find the next reference by using the **find_next** command (**Search → Find Next** or **Ctrl+G**).
- **Search for word matches if symbol is not found** - When set to **On**, Go to Reference will search for simple word matches if the symbol under the cursor is not found by Context Tagging®.
- **Highlight references in editor** - When set to **On**, each reference is highlighted within files.
- **Show bitmap in margin for each reference** - When set to **On**, each reference will have a bitmap in the left margin of the editor.
- **Only highlight current set of references** - When set to **On**, only the current set of references will be highlighted, not each one on the references stack.
- **Allow mixed language references** - When set to **On**, allow the system to also search for references in files that do not match the source language for the symbol in question.
- **Automatically build references stack** - When set to **On**, Go to Reference will always create a new item on the top of the references stack. If not enabled, you can still manually add searches to the stack using the 'Add' tool button on the References tool window.
- **Automatically pop references stack** - When set to **On**, Pop Bookmark will automatically remove the top-most item from the references stack if the originating bookmark was created using Go to Reference.
- **Automatically pop when no more references** - When set to **On**, Find Next will automatically remove the top-most item from the references stack when there are no more occurrences.
- **Tag File Cache (tune for performance)** - You can tune Context Tagging performance by dedicating more memory to caching the contents of tag files.
- **Use memory mapped files** - When set to **On**, use memory mapped files for reading and writing tag database files. This option may not be available on all platforms. If you are using very large tag files on a 32-bit system, for example, tag files in excess of 1000 megabytes, it is recommended to turn this option off.

On Windows, memory mapped tag files are automatically disabled for network file systems.

- **Tag file cache size** - Specifies the cache size, in megabytes, for tag files. Tagging performance can be improved by adjusting this setting 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 following recommendations as a guide:

- * **Minimum** - 8 MB
- * **Default** - 64 MB
- * **Ideal** - Sum of tag file sizes
- * **Maximum** - 25% of physical system memory

Note that this is the same as the **Tag file cache size** option under **Tools → Options → Application Options → Virtual Memory**.

- **Tag file cache maximum** - Specifies the maximum cache size, in megabytes, for tag files. The tag file cache size can be dynamically adjusted as high as this amount depending on the amount of available memory on your machine at the time SlickEdit is started.

Note that this is the same as the **Tag file cache maximum** option under **Tools → Options → Application Options → Virtual Memory**.

- **Use independent database file caches** - When set to **On**, use independent file caches for each tag database, rather than using a single multi-file database cache. Turning on this feature can allow memory usage to increase by a factor equal to the number of open tag files, whereas with the feature turned off, memory usage is limited to the size of the single shared database file cache. It is recommended to turn this feature off if you are running on a system with a low amount of available memory.
- **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 symbols to be omitted.
 - **Context Tagging® result cache maximum** - Specifies the maximum number of symbol sets to store for current context tagging, statement tagging, local variable tagging, and context tagging search results. The minimum allowed is 10 items, but if you have enough memory available, it is recommended to set this to at least five times the number of files you typically keep open at one time. For a very liberal estimate, on average, each item in the cache can consume a megabyte of memory.
 - **Maximum functions found by parameter help** - Specifies the maximum number of overloaded functions to display when function parameter help (Parameter Information) is invoked.
 - **Maximum class/struct shown in list members** - Specifies the maximum number of class/struct symbols to display in the Class tool window.
 - **Maximum response time for list members (ms)** - Specifies the maximum amount of time, in milliseconds, that SlickEdit spends finding symbols to display while using list members or completing symbols.
 - **Maximum candidates for list parameters** - Specifies the maximum number of local variables and class members that are evaluated to determine assignment compatibility when Auto List Compatible

Parameters is invoked.

- **Maximum response time for list parameters (ms)** - Specifies the maximum amount of time, in milliseconds, that SlickEdit® spends 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.
- **Maximum tags found in symbol search** - Specifies the number of tags found when Find Tag is invoked (right-click in the [Symbols Tool Window](#) and select **Find Tag**). This setting also controls the number of duplicate tags that are tried when SlickEdit® is attempting to evaluate the type of a symbol.
- **Maximum items found in references search** - Specifies that a symbol references search should stop after this many hits are found.
- **Display parameter information after (ms) idle (0 implies no delay)** - This option controls the idle time in milliseconds before the Auto-display parameter information feature displays function help after you type a function call operator such as '('.
- **Maximum response time (ms) for highlighting matching symbols** - Specifies the maximum amount of time to spend locating matching symbols. Only symbols found in this amount of time will be displayed.
- **Update Context performance tuning:** - Use these options to fine tune the editors performance when updating the list of symbols for the current file. This is needed by Context Tagging®, as well as many tagging tool windows and various smart editing features.
 - **Maximum size of files to tag** - Specifies the maximum size, in kilobytes, a file is allowed to have in order to be tagged.
 - **Maximum size of files to tag (slow files)** - Specifies the maximum size, in kilobytes, a file is allowed to have in order to be tagged when the file's language mode uses a slower proc-search based tagging function.
 - **Maximum size of files for statement tagging** - Specifies the maximum size, in kilobytes, a file is allowed to have in order to be tagged using statement tagging.
 - **Maximum size of files for building token list** - Specifies the maximum size, in kilobytes, a file is allowed to have in order to be allowed to construct a token list. Token lists are used to optimize gathering expression information for symbol analysis, symbol coloring, and positional keyword coloring.
 - **Maximum number of tags per file** - Specifies the maximum number of tags (including statements, if [Statement Level Tagging](#) is enabled) that a file is allowed to have in order to appear in the Defs and Current Context tool windows.
 - **Maximum time for parsing current file (ms)** - Specifies the maximum amount of time in milliseconds to spend updating the current context. This setting exists to prevent delays when updating the symbols in the current file for files that have slower proc-search functions instead of tagging callbacks written in C++.

- **Avoid updating context if average time exceeds (ms)** - If the maximum time in milliseconds to update the context for the current file exceeds this threshold, avoid non-essential operations, like auto-complete, symbol coloring and highlighting, and beautify while typing which require the current context to be updated on demand.
- **Immediately update context if maximum observed time is less than (ms)** - If the maximum time in milliseconds to update the context for the current file is less than this threshold, do not hesitate to update the current context when it is needed for updating a tool window or auto-complete or any other tagging operation.
- **Symbol coloring performance** - Use these options to fine tune symbol coloring performance.
 - **Update symbol coloring after (ms) idle** - Specifies the amount of time (in milliseconds) to wait to update the symbol coloring information for a file after the file has been modified. Based on average typing speed, we do not recommend setting this value to less than 250 ms.
 - **Timeout after (ms)** - Symbol coloring will be performed in time slices no greater than the amount specified. Setting this value very low will protect against typing delays. However, you may see the symbols coloring from top to bottom rather than seeing the whole page colored in one shot.
 - **Number of lines to color above and below the current page** - For best performance, symbol coloring only colors the current visible page of lines and a small window of surrounding lines. This setting allows you to configure how many lines before and after the current visible page of lines are also colored. Set this to 0 for optimal performance with no prefetch of symbol coloring information.
 - **Number of off-page lines to color per pass (chunk size)** - After calculating the symbol coloring for the lines on the current page, symbol coloring will start prefetching coloring for surrounding lines. This setting controls the number of lines calculated per pass.
 - **Windows to color** - This setting controls which windows to color. Select **Current window** to color the current window only (best performance). To color all currently visible windows, select **All visible windows**. To color all currently open windows, select **All windows**.
- **Highlight tool window performance** - Use these options to fine tune the highlight tool window performance.
 - **Update highlighting after (ms) idle** - Specifies the amount of time (in milliseconds) to wait to update the highlighting information for a file after the file has been modified. Based on average typing speed, we do not recommend setting this value to less than 250 ms.
 - **Timeout after (ms)** - Highlighting will be performed in time slices no greater than the amount specified. Setting this value very low will protect against typing delays. However, you may see the highlighting from top to bottom rather than seeing the whole page colored in one shot.
 - **Number of lines to color above and below the current page** - For best performance, highlighting only colors the current visible page of lines and a small window of surrounding lines. This setting allows you to configure how many lines before and after the current visible page of lines are also colored. Set this to 0 for optimal performance with no prefetch of highlighting information.
 - **Number of off-page lines to color per pass (chunk size)** - After calculating the highlighting for the

lines on the current page, highlighting will start prefetching coloring for surrounding lines. This setting controls the number of lines calculated per pass.

- **Windows to color** - This setting controls which windows to color. Select **Current window** to color the current window only (best performance). To color all currently visible windows, select **All visible windows**. To color all currently open windows, select **All windows**.
- **Auto-Complete performance tuning:**
 - **Maximum symbols** - For performance tuning, you can limit the maximum number of symbols displayed by Auto-Complete. This setting affects all file extensions.
 - **Maximum function prototypes** - Limits the maximum number of symbols displayed with their function arguments.
 - **Maximum word completion** - For performance tuning, you can limit the maximum number of word completions displayed by Word Completion. This setting affects all file extensions. This is especially useful when editing large files.
 - **Maximum completion history matches** - This option specifies the maximum number of history matches displayed by Auto-Complete. When there are many history matches, the most used and most recent matches are given priority.
 - **Maximum number of auto-complete history to store** - This option specifies the maximum number of auto-complete history entries to store per language. When there are more entries than this, a cleanup routine is used to reduce the number of items stored to half this amount by collating repeated items and removing old and infrequently used completions.
 - **Display auto-complete after (ms) idle** - The number of milliseconds the editor must be idle before auto-completion suggestions will be displayed. This setting affects all extensions.

Note

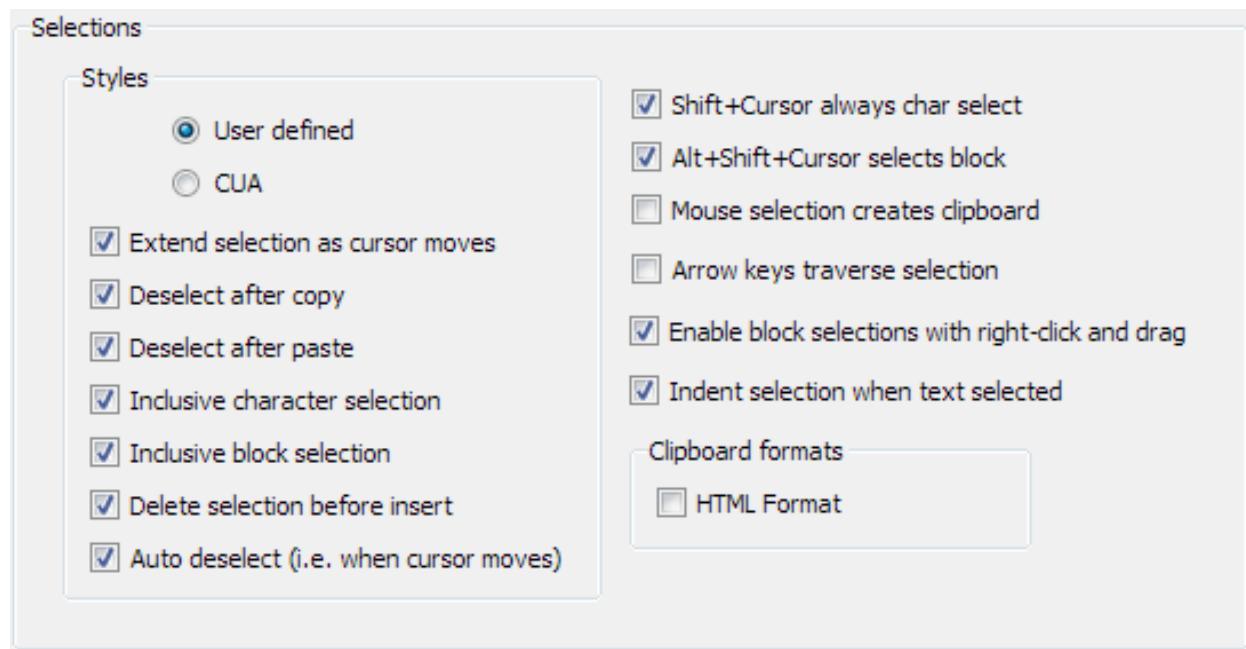
This setting does not effect Auto-list members or Auto-list compatible values. Those features are always displayed immediately, unless disabled. See [Auto-list members](#) and [Auto-list compatible values](#) for more information.

- **Update auto-complete after (ms) idle** - The number of milliseconds the editor must be idle before auto-completion suggestions will be refreshed. This setting affects all extensions.
- **Timeout after (ms) when automatic** - For performance tuning, you can limit the amount of time that Auto-Complete spends finding suggestions when it comes up automatically. Set this to less than a second to avoid typing delays.
- **Timeout after (ms) on demand** - For performance tuning, you can limit the amount of time that Auto-Complete spends finding suggestions when it is invoked manually. Set this to as much as a minute, depending on how long you might be willing to wait for results.
- **Background Wildcard caching:**

- **Update wildcard cache** - When set to On, will scan workspace directories to add or remove from projects with wildcard file specifications.
- **Maximum files for cache updating** - Background wildcard cache updating is disabled when specified maximum number of files are found. Only counts number of files found by wildcard specification, not the total number of files.
- **Minimum wildcard cache update time (ms)** - Specifies the minimum time between wildcard cache updates occur.

Selection Options

Selection options are shown below (**Tools** → **Options** → **Editing** → **Selections**). See also [Selections](#) for more information.



The following options 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 for most emulations and the default emulation) 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.

- **Extend selection as cursor moves** - When checked, 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 checked, a character selection includes the character following the cursor. This option is not available if using Brief or Emacs emulation.
- **Inclusive block selection** - When checked, a block selection includes the character following the cursor.
- **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 (i.e. when cursor moves)** - Check this box 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.
- **Alt+Shift+Cursor selects block** - When checked, pressing **Alt+Shift+<arrow keys>** will create a block selections.
- **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 checked, 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.
- **Enable block selections with right-click and drag** - If checked, then clicking the right mouse button and dragging allows you to make block selections.
- **Indent selection when text selected** - When this option is selected, pressing **Tab** or **Shift+Tab** indents or unindents the selected text.
- **HTML Clipboard formats** - (Windows only) Check this option to enable pasting of HTML-formatted and color-coded text to other applications (as well as plain text).

Search Options

Search options are shown below (**Tools** → **Options** → **Editing** → **Search**). This option screen can also be displayed from the Find and Replace tool window (**Search** → **Find** or **Ctrl+F**), or right-click in the background and select **Configure Options**.

These are the default search options that control the behavior of Find and Replace operations in the

following instances:

- The very first time the Find and Replace tool window is displayed. After that, dialog history takes over, unless:
 - You set the default search option **Initialize with default options** setting to **On**, then the options are reset to default every time the tool window is invoked, or
 - You use the right-click context menu in the Find and Replace tool window to select **Use Default Options**, which resets to the default state.
- The default options are always applied when using:
 - Quick Search and Quick Replace.
 - Incremental Search.
 - Command-line searches (**find** and **/**) if you don't specify options explicitly.
 - Selective Display commands when searching by text.

For more information, see [Find and Replace](#).

Search	
Option	Value
Default search options	
Match case	<input type="radio"/> OFF
Match whole word	<input type="radio"/> OFF
Regular expression	<input type="radio"/> OFF
Regular expression syntax	Perl
Wrap at beginning/end	Prompt every time
Search backward	<input type="radio"/> OFF
Place cursor at end	<input type="radio"/> OFF
Search hidden text	<input type="radio"/> OFF
Tool window options	
Close after find/replace	<input checked="" type="radio"/> ON
* Preview All shows modified file(s) on the left	<input checked="" type="radio"/> ON
Mini window options	
Close on Enter key	<input type="radio"/> OFF
Maximum buffer size for incremental search (KB)	50000
Maximum occurrence matches	8192
Search string initialization	
Initialize search string	History retrieval
Selected text (if exists)	<input type="radio"/> OFF
Auto escape regular expression	On
Additional options	
Default Find and Replace GUI	Mini
Initialize with default options	<input type="radio"/> OFF
Restore cursor after replace	<input checked="" type="radio"/> ON
Leave selected	<input checked="" type="radio"/> ON
Incremental search highlighting	<input checked="" type="radio"/> ON
Maximum search results output (KB)	2048
Maximum search result line length	500
Truncated search result width	0
Starting number of Search Results Buffers	2
Sort Find in Files search results by filename	<input type="radio"/> OFF
Find in Files threads	8
Default Excludes	<Binary Files>

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 enabled.
 - **Match case** - When set to **On**, search commands default to case-sensitive searches.
 - **Match whole word** - When set to **On**, search commands default to only finding matches to the word as a whole. When set to **Off**, search commands default to finding all instances of the word, ignoring characters that are to the left and right of the occurrence.
 - **Regular expression** - When set to **On**, search commands default to regular expression searching.
 - **Regular expression syntax** - Specifies which regular expression syntax to use for default regex searching, when **Regular expression** search is enabled.
 - **Wrap at beginning/end** - Specifies whether or not search commands always wrap at the beginning or end of a buffer during a search/replace operation when the range is the current buffer.
 - **Search backward** - When set to **On**, searches are always performed from the end to the beginning.
 - **Place cursor at end** - When set to **On**, the cursor is placed at the end of the found occurrence.
 - **Search hidden text** - When set to **On**, text hidden by Selective Display is allowed to be searched. To set Selective Display options, from the main menu click **View** → **Selective Display**. See [Selective Display](#) for more information.
- **Tool Window options** - Select from the following:
 - **Close after find/replace** - When set to **On**, the Find and Replace tool window is closed after finding text in the buffer.
 - **Preview All shows modified file(s) on the left** - When set to **On**, Hitting **Preview All...** after a **Replace** or **Replace in Files** will show the modified file(s) on the left side rather than the right.
- **Mini window options** - Select from the following:
 - **Close on Enter key** - When set to **On**, Mini Find is closed on **Enter** key in addition to **Esc** key.
 - **Maximum buffer size for incremental search (KB)** - If the buffer size is larger than this size, incremental searching will not be performed to improved performance.
 - **Maximum occurrence matches** - Determines that maximum number of matches for **Incremental search highlighting**.
- **Search string initialization** - The following options provide starting values for when a search and replace operation is activated:
 - **Initialize search string** - Specifies the initial value to be used in the **Search for** fields of the Find and Replace tool window when the window is activated. When **History retrieval** is selected, the Find and Replace tool window uses the last item that was searched for as the word used when performing a

search. When **Word at cursor** is selected, the Find and Replace tool window uses the word that is at the cursor when performing a search.

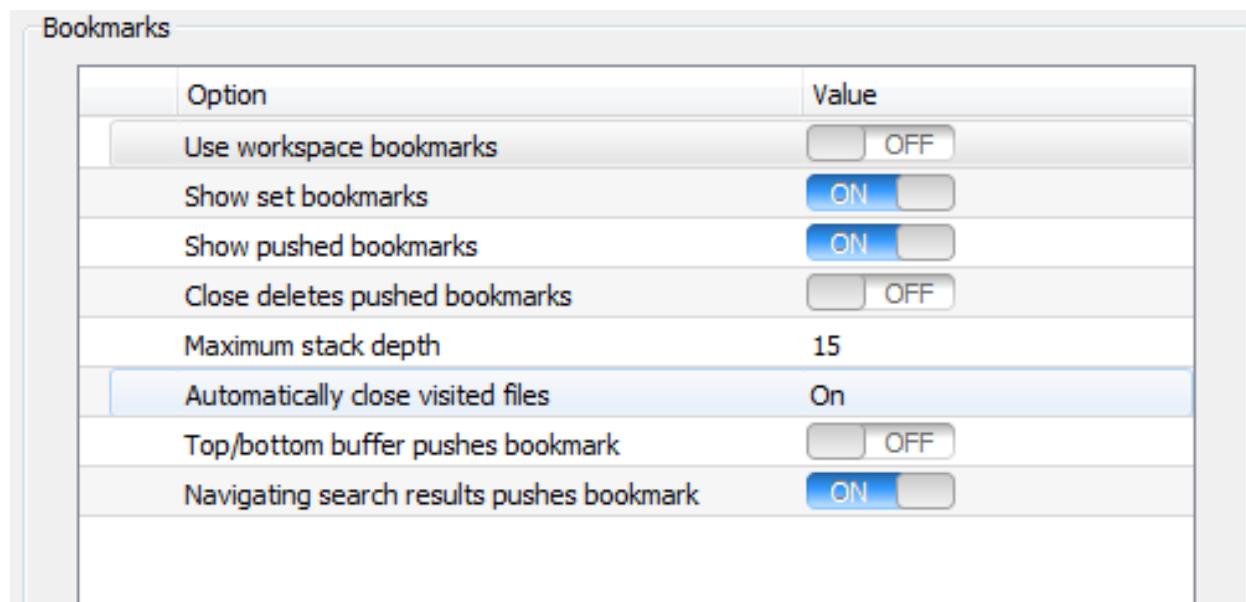
- **Selected text (if exists)** - When set to **On**, the current selection is used as the initial value in the **Search for** fields of the Find and Replace tool window when the window is activated. If a selection doesn't exist, the value specified by the **Initialize search string** option is used.
- **Auto escape regular expression** - When set to **On** and the string is initialized with selected text or current word, add escape (Backslash) to any regular expression metacharacters.
- **Additional options** - Select from the following:
 - **Default Find and Replace GUI** - Determines the dialog displayed by the gui-find (Ctrl+F) and gui-replace (Ctrl+R) commands. **Mini** displays the Mini Find and Replace dialog. **Tool Window** displays the Find and Replace tool window.
 - **Initialize with default options** - When set to **On**, options on the Find and Replace tool window and Mini Find are reset to the original default values each time the window is launched.
 - **Restore cursor after replace** - When set to **On**, the cursor is restored to its original position after a search/replace operation completes without being cancelled.
 - **Leave selected** - When set to **On**, the last occurrence of a matching search string is left selected when a search operation completes. This also affects whether pressing **Esc** during a search and replace leaves the search string selected.
 - **Incremental search highlighting** - When set to **On**, incremental searching highlights matching occurrences with two colors: one for the current match at the cursor and one for all possible matches. Highlights are removed when the search terminates. These colors are controlled by the I-Search Current Match and I-Search Highlight screen elements (**Tools** → **Options** → **Appearance** → **Colors**). See [Incremental Searching](#) for more information.
 - **Maximum search results output (KB)** - Specifies the maximum amount of search results, in kilobytes, to return after a search operation.
 - **Maximum search result line length** - Specify maximum line length for printing search result line in full. If the line length is greater than the setting, the line will be truncated around the match. Set this value to 0 to disable this check.
 - **Truncated search result width** - Specify the number of columns before and after the match for a truncated search result line.
 - **Search Results Buffers** - Specify starting number of tabs for Search Results tool windows.
 - **Sort Find in Files search results by filename** - When set to **On**, search results are sorted by filename. Turning this option on can affect performance because the entire filelist must be scanned and sorted before threads can search files.
 - **Find in Files threads** - When using Find in Files, this determines the number of threads used to search files. Using more threads can increase performance but it depends on the number of cores your CPU has and hard drive speed. Using more threads can use more memory but it's typically not

significant.

- **Default Excludes** - Configure set of semicolon delimited list of excludes that can be used in Find and Replace Excludes. Choose <Default Excludes> from the Excludes combo box to make use of the list. The Default Excludes typically excludes binary files so that a wildcard specification like '*' or '*.*' excludes binary files. Defaults to <Binary Files>, which specifies all extensions associated with Binary document mode. See [File Extension Manager](#) for more information on adding extensions and associating language modes with extensions.

Bookmarks

Bookmark options are shown below (**Tools** → **Options** → **Editing** → **Bookmarks**) See [Bookmarks](#) for more information.



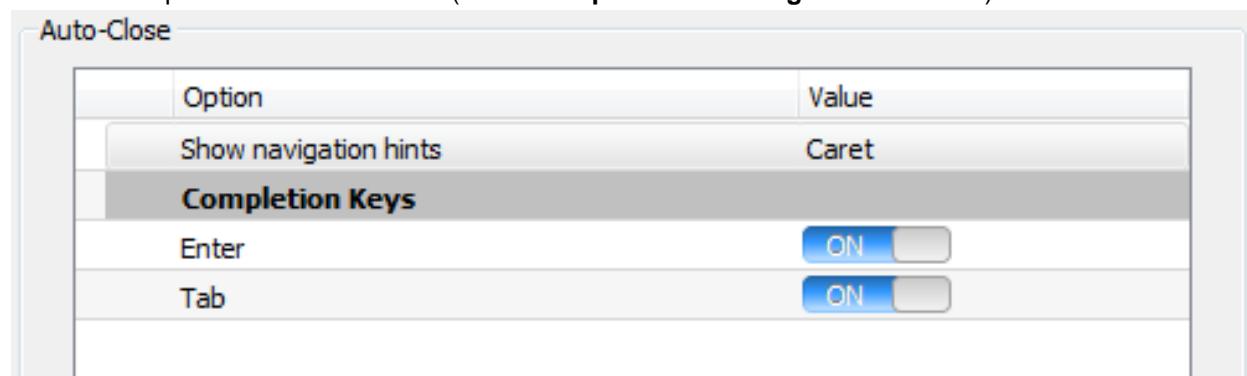
The following options are available:

- **Use workspace bookmarks** - When set to **On**, 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 updates to show only the bookmarks associated with the current workspace. See [Using Workspace Bookmarks](#) for more information.
- **Show set bookmarks** - When set to **On**, a green Bookmark bitmap is displayed in the left margin of the editor window at the location of each set bookmark.
- **Show pushed bookmarks** - When set to **On**, a blue Bookmark bitmap is displayed in the left margin of the editor window at the location of each pushed bookmark. This helps you see where "Pop Bookmark" will go.
- **Close deletes pushed bookmarks** - When set to **On**, any pushed bookmarks remaining are removed when a buffer is closed. 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.

- **Maximum stack depth** - Specifies the maximum number of bookmarks kept in the bookmark stack. When this number is exceeded, the oldest bookmark is removed from the stack.
- **Automatically close visited files** - Specifies the closing of visited files. A file is considered visited if it is opened as a result of a symbol navigation or search operation, not modified, and subsequently navigated away from. Features that open files for visiting include Go to Definition, Go to Declaration, Go to Reference, Find in Files, and Pop Bookmark (see [Symbol Navigation](#)), and some search operations. Select from the following values:
 - **On** - When this setting selected, visited files are automatically closed when you navigate away from them.
 - **Off** - When this setting is selected, the option is not enabled and visited files are not closed.
 - **Prompt me each time** - When this setting is selected, you will be prompted with a choice each time you navigate away from a visited file.
- **Top/bottom buffer pushes bookmark** - When set to **On**, a bookmark is pushed 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. This option corresponds to the configuration variable **def_top_bottom_push_bookmark**.
- **Navigating search results pushes bookmark** - When set to **On**, a bookmark is pushed when you navigate between search results. This allows you to use **pop-bookmark** to jump back to the previous location.

Auto-Close

Auto-Close automatically inserts matching closing punctuation when opening punctuation is entered. Auto-Close options are shown below (**Tools → Options → Editing → Auto-Close**).



The following Auto-Close options are available:

- **Show navigation hints** - This option controls whether a navigation hint is shown when Auto-Close is used. These hints will tell you where you can jump by using the completion keys. Select from the following choices:

- **Caret** - The navigation hint will appear as a small triangle at the bottom of the line of code.
- **Vertical pipe** - The navigation hint will appear as a vertical pipe, similar to a non-blinking cursor.
- **None** - Select this choice if you do not wish to see any navigation hints.
- **Completion Keys** - These Options determine which keys can be used to jump to the end of automatically inserted punctuation.
 - **Enter** - When set to **On**, pressing Enter will jump to the end of automatically inserted punctuation. When set to **Off**, Enter will perform its usual function when in the middle of automatically inserted punctuation.
 - **Tab** - When set to **On**, pressing Tab will jump to the end of automatically inserted punctuation. When set to **Off**, Tab will perform its usual function when in the middle of automatically inserted punctuation.

Hotspot Options

Hotspots are used with the [Syntax Expansion](#) and [Aliases](#) features. When expanding a block of text, markers are inserted where you are likely to want to jump to as you edit. You can use Tab to jump to the next hotspot, or use the `next_hotspot` and `prev_hotspot` commands.

The Hotspot options are shown below (**Tools** → **Options** → **Editing** → **Hotspots**).

Hotspots	
Option	Value
Hotspot Navigation	<input checked="" type="button"/> ON
Allow Tab key navigation	<input checked="" type="button"/> ON
Show navigation hints	Caret

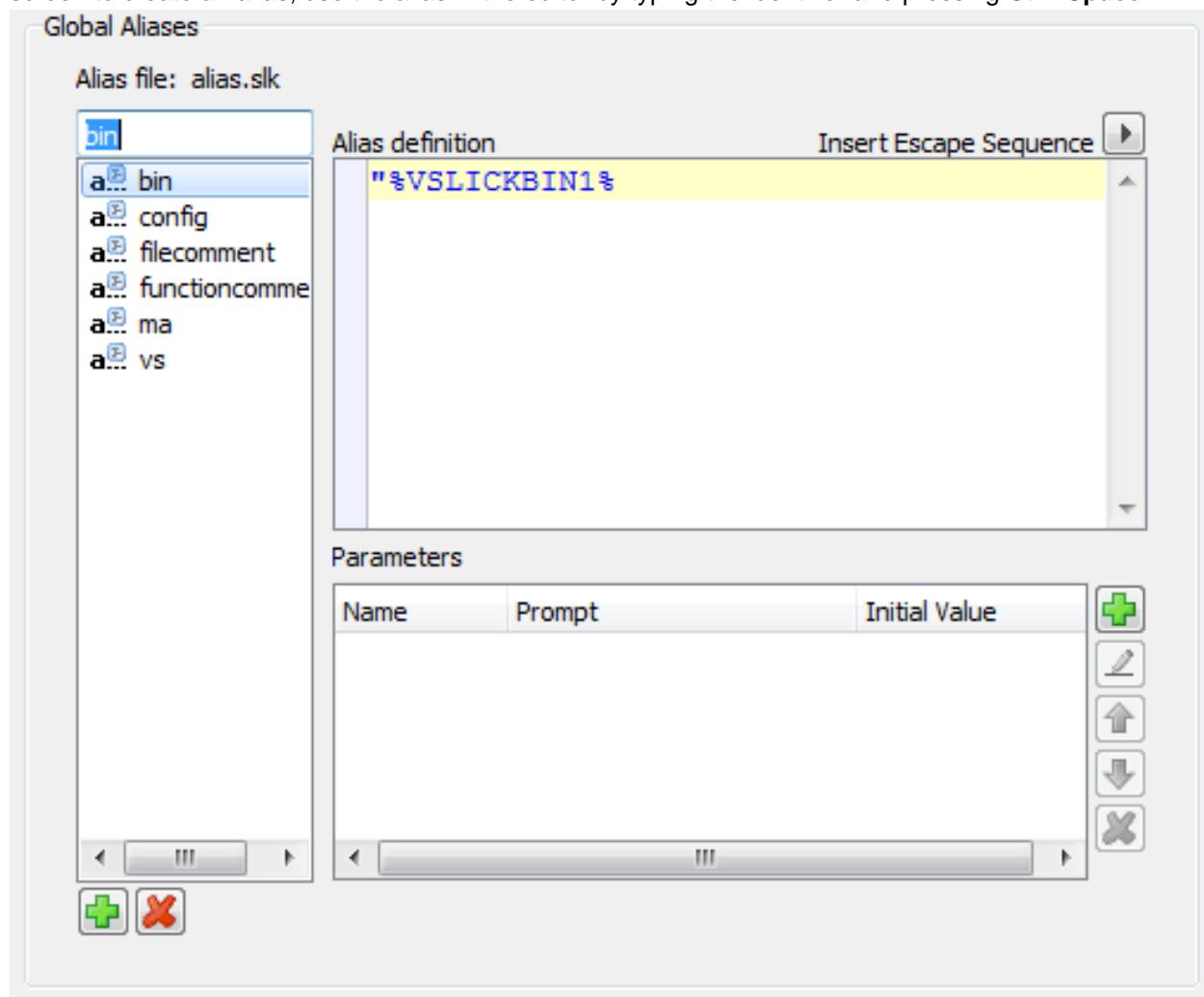
The following Hotspot options are available:

- **Hotspot navigation** - Enables hotspots for syntax expansion and alias expansion.
- **Allow Tab key navigation** - Use the Tab key to jump to the next hotspot. When turned **Off**, then the `next_hotspot` and `prev_hotspot` commands can be used (or bound to keys using the Keybindings options).
- **Show navigation hints** - Determines how to display hotspots in the editor window. The following choices are available:

- **None** - Do not display hotspots.
- **Caret** - Use the caret character (^) to represent hotspots.
- **Vertical pipe** - Use the vertical pipe character (|) to represent hotspots.

Global Alias Options

Global alias options are shown below (**Tools** → **Options** → **Editing** → **Global Aliases**). After using this screen to create an alias, use the alias in the editor by typing the identifier and pressing **Ctrl+Space**.



Note that the Global Aliases options page uses the same form that is used to create and manage Language-Specific Aliases. See [Global Aliases](#) for more information about this feature.

Debugging Options (Pro only)

Debugging options (**Tools** → **Options** → **Debugging**) are used for tuning the run-time performance of

the integrated debugger, examining the properties of the underlying debugger system, setting class filters, and controlling the directories searched for source files.

You can also access these options by clicking the **Debug** → **Debugger Options** from the main menu (or by using the **debugger_options** command).

Note

To see information about the underlying debugger system, including a general description retrieved from the debugger, version number, run-time version, and debugger name, make sure you're in debug mode, then click **Debug** → **Debugger Information** (or use the **debug_props** command). See [Viewing Debugger Info and Setting Options](#) for more information.

Debugging options categories are:

- [Debugging General Options](#)
- [Debugging Numbers Options](#)
- [Debugging Runtime Filter Options](#)
- [Debugging Directories Options](#)
- [Debugging Configurations Options](#)

Debugging General Options

The Debugging **General** Options screen is shown below (**Tools** → **Options** → **Debugging** → **General**). Use these options to increase debugger performance.

Debugging Options (Pro only)

General	
Option	Value
Performance	
Number of lines to scan for Autos	3
Number of elements to expand in arrays	1000
* Response timeout (seconds)	30
Minimum running update time (ms)	250
Maximum suspended update time (ms)	1000
Toolbar update delay (ms)	100
Asynchronous message duration (seconds)	5
Features	
Allow edit and continue (hot swap) where available	<input checked="" type="checkbox"/>
Allow editing of source files during debugging	<input checked="" type="checkbox"/>
Show value of symbol under mouse	<input checked="" type="checkbox"/>
Automatically correct breakpoint scope	<input checked="" type="checkbox"/>
Input/Output	
vsdebugio connection port	8001
GDB Options	
(Windows only) Use remote proxy	<input checked="" type="checkbox"/>
(Windows only) Remote proxy port	
Enable Python pretty printing	<input checked="" type="checkbox"/>
Disable auto-loading of scripts	<input checked="" type="checkbox"/>

The options are described as follows:

- **Performance:**

- **Number of lines to scan for Autos** - Specifies 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 tool window.
- **Number of elements to expand in arrays** - Specifies the maximum number of elements to expand when examining the contents of an array or string in the debugger. Note: This setting is not enforced by all debugger environments, nor is it an absolute maximum in all debugger environments.
- **Response timeout(s)** - Specifies the number of seconds to wait for a connection or response from the underlying debugger system before giving up.
- **Minimum running update time(ms)** - Specifies the minimum amount of time in milliseconds to spend polling for asynchronous events coming from the underlying debugger system. This value is also used to determine the frequency with which to poll for events when the application is running.

- **Maximum suspended update time(ms)** - Specifies the maximum amount of time in milliseconds to spend polling for asynchronous events coming from the underlying debugger system. This value is also used to determine the frequency with which to poll for events when the application is suspended.

Tip

Decreasing the minimum and maximum update times will make the editor more responsive in some cases during debugging, however, increasing these times can improve overall performance.

- **Toolbar update delay(ms)** - Specifies the amount of time to wait in milliseconds 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(s)** - Specifies the amount of time in seconds to display certain informative messages caused by asynchronous events (such as loading classes in Java) before they are erased.
- **Features:**
 - **Allow edit and continue (hot swap) where available** - (Edit and continue - Java only) When set to **On**, you can edit a file during a Java debugging session, compile or rebuild, and then continue to debug. Keep in mind that when using this feature, there are certain feature limitations that you might encounter that are defined by the Java Virtual Machine.
 - **Allow editing of source files during debugging** - When set to **On**, you can edit files during debugging sessions.
 - **Show value of symbol under mouse** - When set to **On**, as the mouse cursor floats over a symbol, the information about the symbol, including its value for variables, are displayed.
 - **Automatically correct breakpoint scope** - When a breakpoint is set in the debugger, the name of the function and the class it is in is recorded. If the class name or function name subsequently changes between debugging sessions, having this option enabled will allow the debugger to correct this information stored with the breakpoint.
- **Input/Output:**
 - **vsdebugio connection port** - Specifies the TCP/IP port to connect to **vsdebugio** running locally on the same machine.
- **Advanced GDB Options:**
 - **(Windows only) Use remote proxy** - (Windows only). Set to true to use gdb remote proxy application to mediate connection between gdb and remote target. This setting only applies to attaching to remote target.
 - **(Windows only) Remote proxy port** - (Windows only). Specifies the TCP/IP port to use when gdb connects to a remote target through the gdb remote proxy application. 'def_gdb_use_proxy' must be

set to true for this to have any effect. The default port number is 8002.

- **Enable Python pretty printing** - Specifies to enable pretty printing using Python scripts provided that the GDB executable has supports for python built-in.
- **Disable auto-loading of scripts** - Specifies to disable auto-loading of pretty printing modules and other scripting that a user's GDB may have pre-configured. This is generally necessary when using the integrated pretty printing because the scripts can be prone to cause long delays.

Debugging Numbers Options

The Debugging **Numbers** Options screen is shown below (**Tools** → **Options** → **Debugging** → **Numbers**). Use these options to configure settings for viewing numbers in multiple bases (hex, octal, and binary views) or for viewing floating point values using a specific style, or for viewing strings with Unicode characters or with non-ascii characters escaped.

Option	Value
Integer numbers	
Short	Decimal
Integer	Decimal
Long	Decimal
Unsigned	Decimal
Floating point numbers	
Float	Decimal
Double	Decimal
Characters	
Character	Char

These settings allow you to specify how different numeric types are formatted by the debugger by default. You can specify settings for integer numbers, floating point numbers, characters, and strings. Each option contains the same values for you to pick from. The table below shows examples of each type of configuration for numbers.

Option Value	Base	Example
Binary	2	0b01000001
Char	N/A	'A'
Decimal	10	65

Option Value	Base	Example
Hexadecimal	16	0x41
Octal	8	0101

The following options are for floating point numbers:

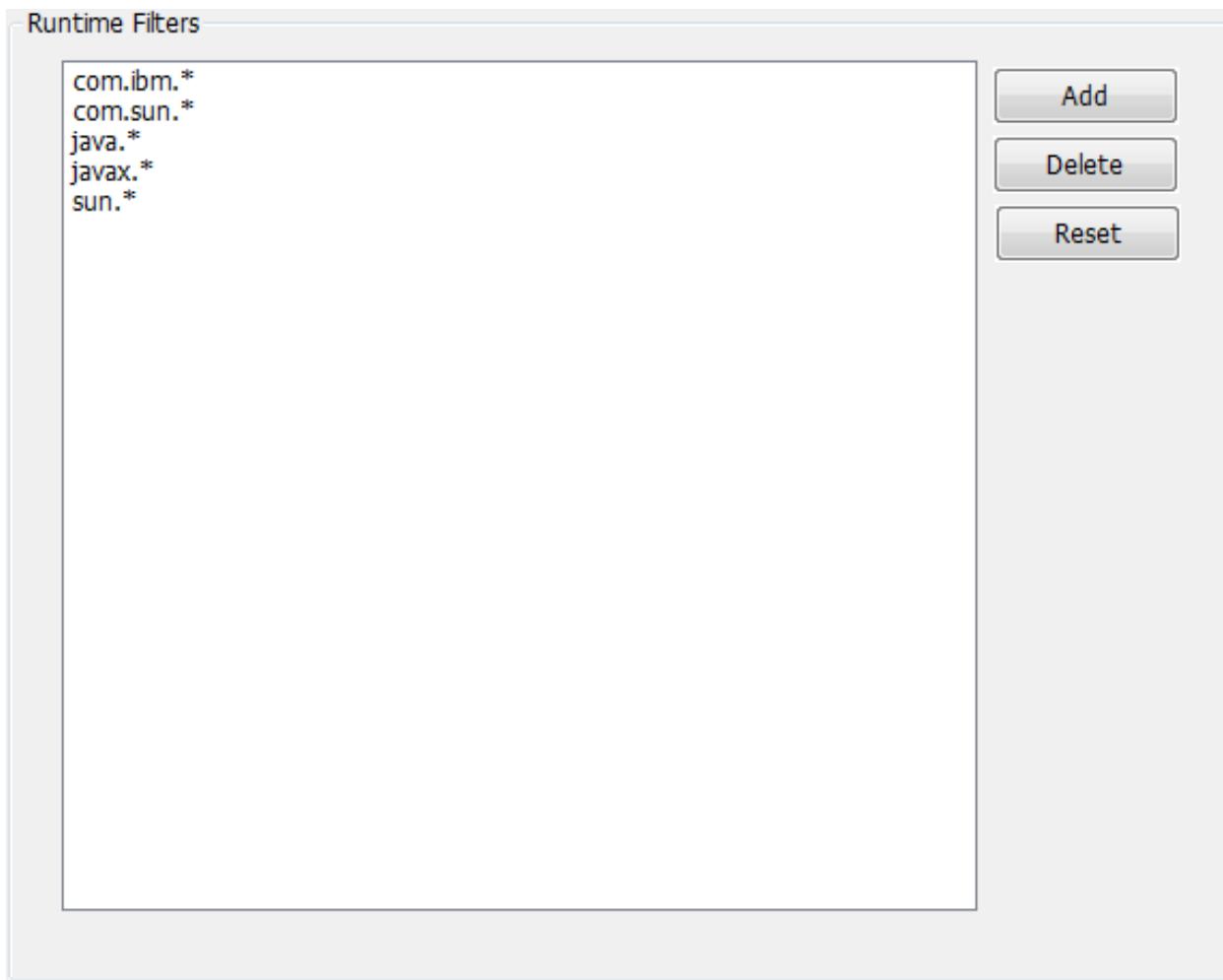
Option Value	Example	
Default	0.000123	
Scientific	1.23e-4	
Floating Point	0.0001230000	

For string values, they can be displayed with native Unicode characters, or they can be displayed with all Unicode characters escaped to make the string plain ascii text.

These settings can be overridden by using the right-click context menu in any of the debugger variables tool windows. They may also be overridden on a per-variable basis, again by using the right-click context menu in the debugger variables tool windows (Autos, Locals, Members, Watches). See [Debug Tool Windows](#) for more information.

Debugging Runtime Filters Options

The Debugging **Runtime Filters** Options screen is shown below (**Tools** → **Options** → **Debugging** → **Runtime Filters**). Use this screen to configure the Step Into command (**Debug** → **Step Into**, **debug_step_into**) to skip certain runtime functions and methods.

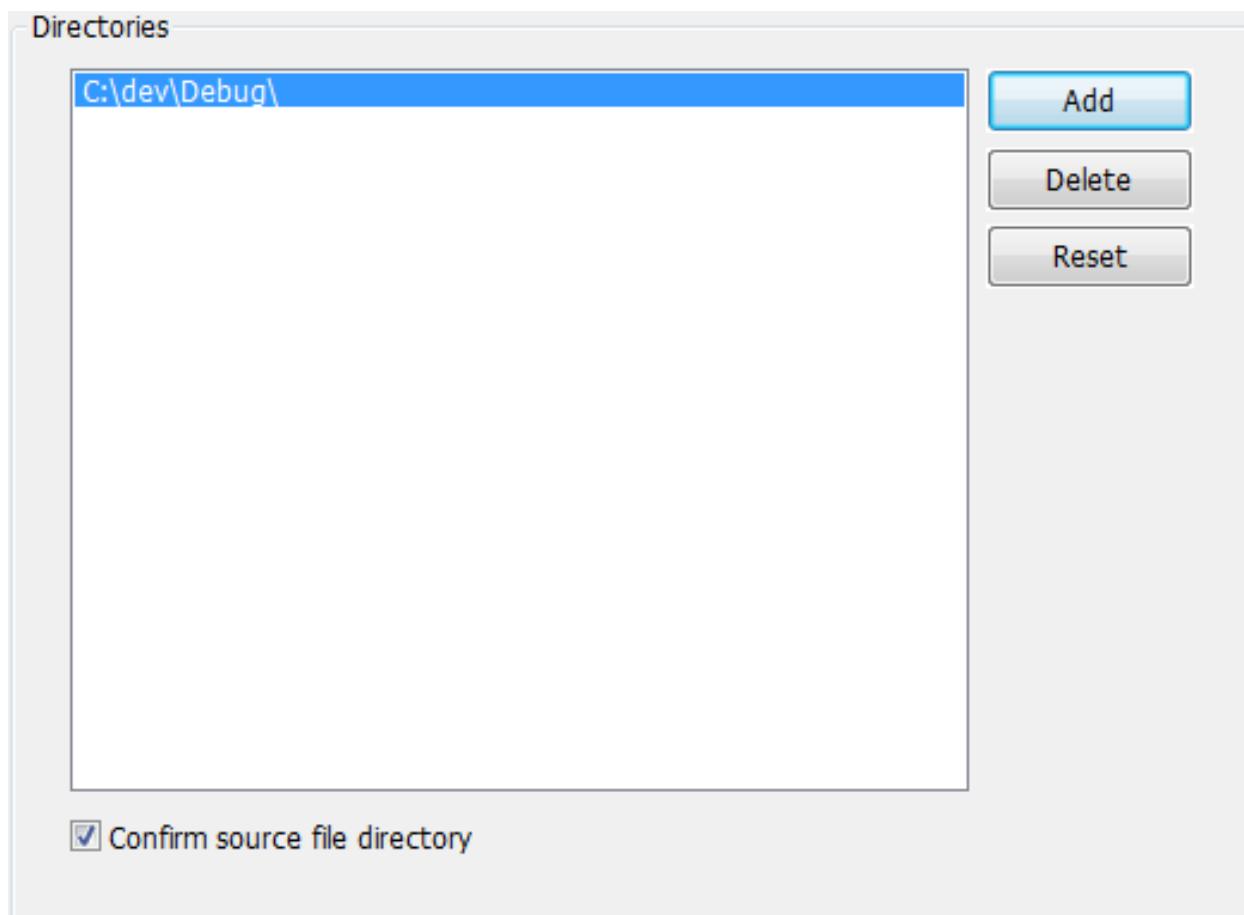


For GNU C/C++, click **Add** to specify a function, or class and method patterns (ex. **strcpy**, **str***, **MyClass::***, **MyClass::Insert***) for the debugger to consider as run-time functions. This will affect the behavior of Step Into. Step Into will step over statements that call into run-time functions. The **Reset** button will reset the filters back to the defaults.

For Java, click **Add** to specify packages or specific classes or class patterns for the debugger to consider as run-time classes. This will affect the behavior of Step Into and the Debug Loaded Classes tool window. Step Into will step over statements that call into run-time 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.

Debugging Directory Options

The Debugging **Directories** Options screen is shown below (**Tools** → **Options** → **Debugging** → **Directories**). Here, you can tune the search path used to find source files while debugging.

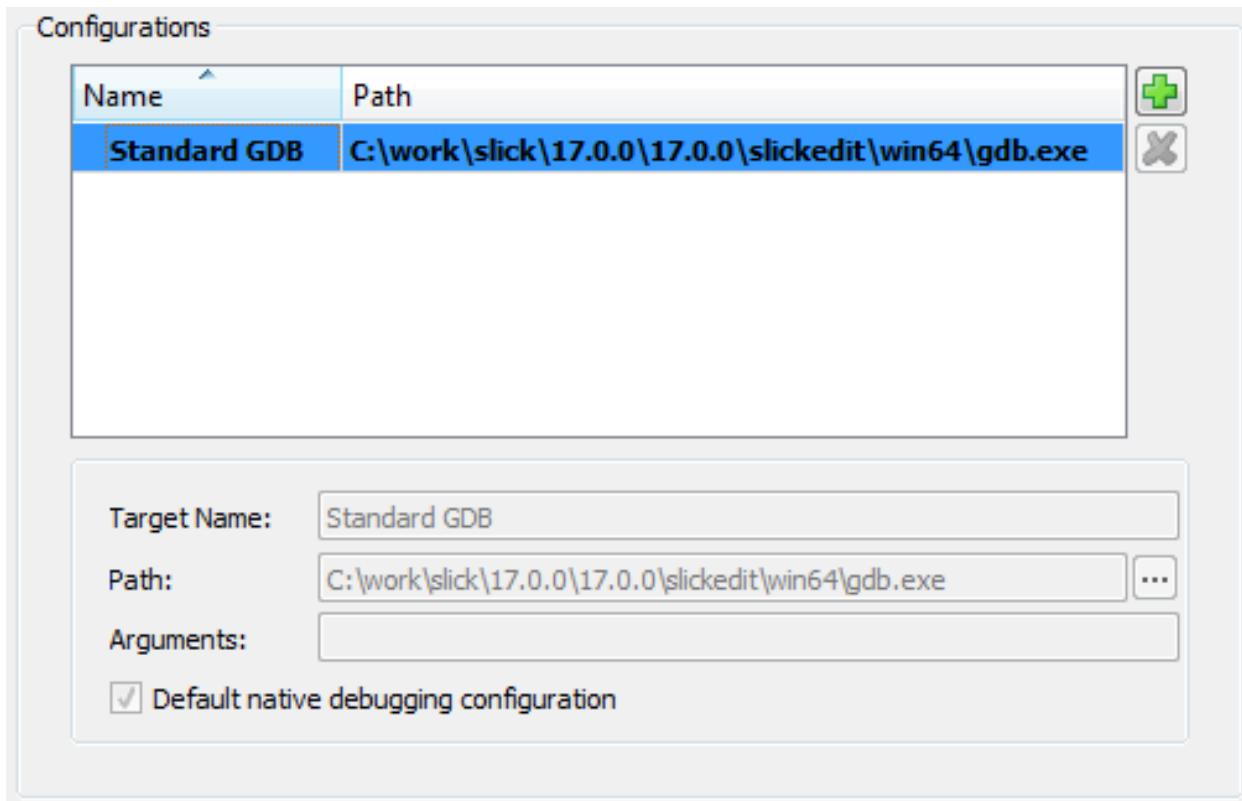


When debugging, if a source file path cannot be resolved using the current directory or the class path, you will be prompted for the source file. The debugger then saves the path in which the source file was found, so you will not be prompted again when that file or another in that directory is needed during a debugging session. The **Reset** button clears all stored source paths.

Debugging Configurations Options

The Debugging **Configurations** Options screen is shown below (**Tools** → **Options** → **Debugging** → **Configurations**). 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 to be used when debugging local, native executables.

These settings are especially useful for development teams who use cross-compiler platforms for applications such as embedded systems. They are also useful for adding newer versions of the GDB debugger, when they become available, for use in the debugging processes.



The debuggers in this list also appear on the **Remote Options** tab of the **Debug → Attach Debugger → Attach to Remote Process(GDB)** dialog box.

Language Options

SlickEdit® provides many options that can be configured on a language-specific basis, such as indenting and word wrap. For example, settings for coding in C/C++ can be set differently than those used for Java. SlickEdit® uses the extension of the current file to determine what language you are using, thereby only making available the features that are possible in that language and applying the associated settings. For convenience, we provide a means to set most of these values for all languages, too.

Use the **[Language]** section of the Options dialog (**Tools → Options → Languages → [Language Category] → [Language]**) to control the behavior of SlickEdit for specific a language. Each supported language is categorized by its language type. Expand the applicable category node to find your language. For example, C/C++ and Java are located under the **Application Languages** node.

Tip

A shortcut method to access language options for the current buffer is to use the **Document → [Language] Options** menu item. This will open the Options dialog to the **General** language-specific option screen for that language.

The options available for each language are further categorized by type. In general, the option categories

are the same for each language (General, Editing, View, Formatting, etc.), although if a particular language does not support a particular set of options, that options category is not included. Conversely, some languages include additional options that are not available in other languages. While most of the options are common among all languages, the default settings for these options vary.

Many of the language options can be set for all languages as well, using **Tools** → **Options** → **Languages** → **All Languages**. This avoids having to repetitively set options for things like viewing line numbers. The **All Languages** options are arranged in the same hierarchy and screens used for the individual language options. For more information, about this section, see [All Languages](#)

The **Language Manager** and **Extension Manager** nodes are used to add and remove languages and manage language extension associations. See [Managing Languages](#) and [Managing File Extensions](#) for more information on these screens. You can also use **Advanced File Mappings** to associate files without extensions to languages. See [Managing Extensionless Files](#) for more information about this screen.

The common categories for each language (when supported) are:

- [Language-Specific General Options](#)
- [Language-Specific Editing Options](#)
- [Language-Specific View Options](#)
- [Language-Specific Formatting Options](#)
- [Language-Specific Adaptive Formatting Options](#)
- [Language-Specific Comment Options](#)
- [Language-Specific Comment Wrap Options](#)
- [Language-Specific Word Wrap Options](#)
- [Language-Specific Alias Options](#)
- [Language-Specific Auto-Complete Options](#)
- [Language-Specific Context Tagging Options](#)
- [Language-Specific Color Coding Options](#)
- [Language-Specific File Options](#)
- [Language-Specific Live Errors Profiles](#)
- [Language-Specific Compiler Properties](#)

Language Manager

The Language Manager provides and alphabetical listing of all languages known to SlickEdit. By selecting a language and clicking the **Settings** button, you can navigate to the general options page for that language. See [Language-Specific General Options](#) for more information.

Language Manager

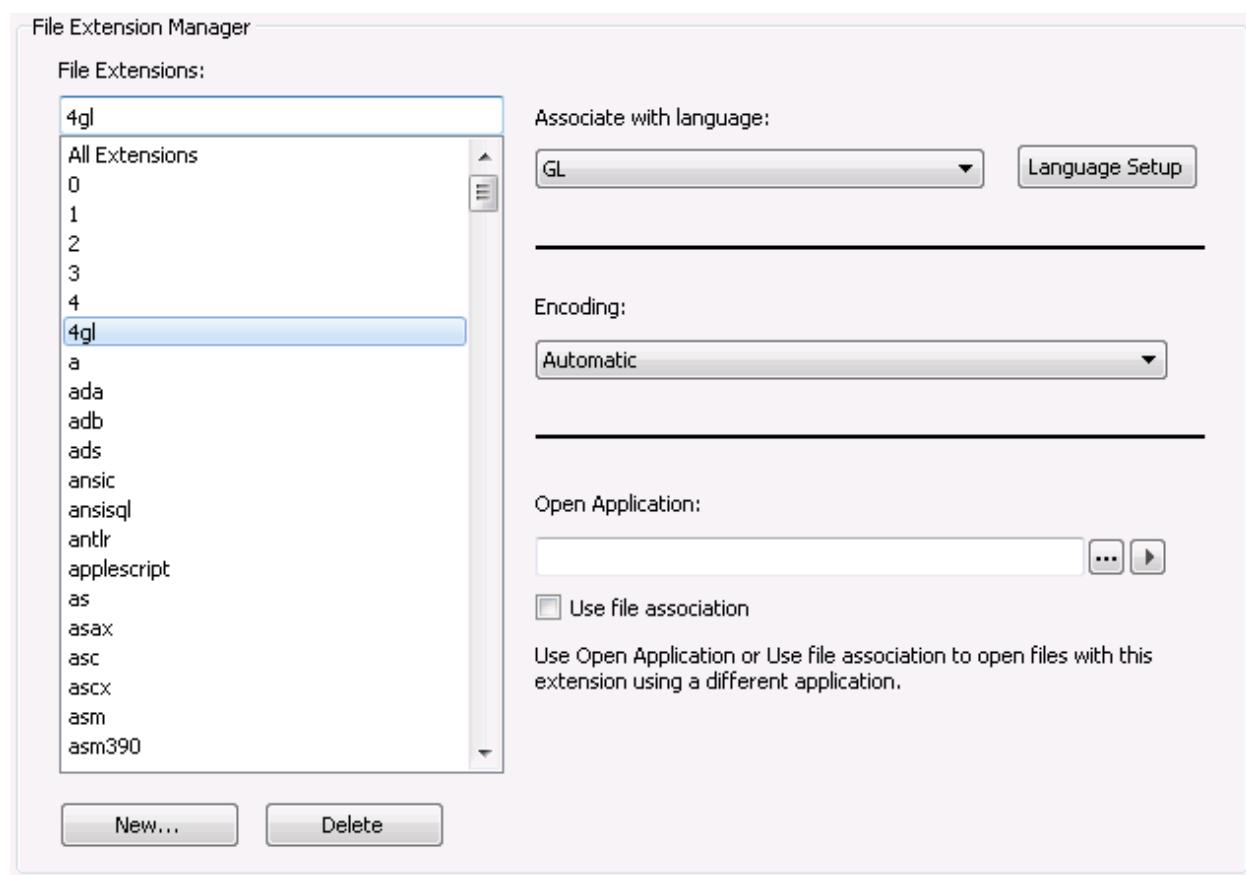
Languages:

The screenshot shows a window titled "Language Manager". On the left is a scrollable list of languages, each preceded by a small green icon. The languages listed are: ActionScript, Ada, ANSI-C, ANTLR, Applescript, Awk, Batch, Binary, Bourne Shell, Bulletin Board Code, C Shell, C#, C/C++, CFML, CFScript, Ch, CICS, Cobol, Config, CSS, DB2, and dBASE. To the right of the list are three buttons: "Add Language...", "Delete Language", and "Settings".

File Extension Manager

The File Extension Manager associates a file extension with a language. You can set the default encoding for files with that extension or specify an application to use to open files of this type. Click the **Language Setup** button to go to the General options for the selected language. See [Language-Specific General Options](#) for more information.

Language Options



Advanced File Mappings

Some files do not have extensions or need their language type determined based on their filename. The Advanced File Mappings dialog allows you specify the language based on an ant-like file pattern.

The screenshot shows the 'Advanced File Mappings' dialog. It features a table titled 'Patterns' with columns for 'Pattern' and 'Language'. Two entries are listed: '**\makefile' mapped to 'Makefile' and '**\Imakefile' mapped to 'Imakefile'. To the right of the table are buttons for managing mappings: 'Edit', 'Add name', 'Add path with name', 'Add path', 'Add pattern', 'Delete', 'Move up', and 'Move down'. A scroll bar is visible at the bottom of the table area.

Pattern	Language
**\makefile	Makefile
**\Imakefile	Imakefile

Patterns

- **Edit** - Edit the selected pattern.

- **Add name** - Add a pattern that matches a name where no path information is necessary.
- **Add path and name** - Add a pattern that matches an absolute filename.
- **Add path** - Add a pattern that matches a path.
- **Add pattern** - Add a pattern on your own.
- **Delete** - Remove the selected pattern from the list.
- **Move up** - Move the selected pattern higher in the list.
- **Move down** - Move the selected pattern lower the list.

All Languages

SlickEdit supports many languages and allows you to configure each one on an individual basis. Should you wish to set a language-specific setting to the same value for each language, you can use the **All Languages Options** (**Tools** → **Options** → **Languages** → **All Languages**). By expanding this category, you can see the same options nodes found underneath the individual languages.

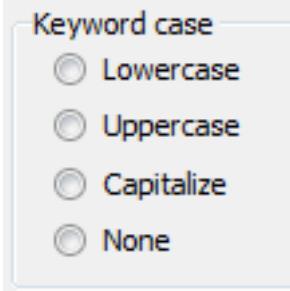
Setting All Languages Options

To set an all language options, simply use the control as you would any other control. Set the value you wish to be used by all languages, then click **OK** or **Apply**. The settings of each language will be updated. Once you set a value, other language options nodes will be updated to reflect that change. For example, if you set **All Languages > Formatting > Tabs** to **+3**, when you navigate to **C/C++ > Indent**, the **Tabs** value will be set to **+3**. You can still set an individual language option. If you then set **Tabs** for C/C++ to **+4** and click **OK**, then the tabs for C/C++ will be set to **+4**, while the tabs for every other language will still be set to **+3**, as specified in the All Languages options.

Initial Settings

When you first view an options node for an individual language, the options show the current settings for that language. However, for **All Languages**, the options shown reveal amalgamated settings for all the languages that SlickEdit supports. If every language has a setting turned off, it will be shown as off under **All Languages**. Additionally, each control has a "neutral" setting, which indicates that all the languages do not share the same setting. This neutral setting appears differently for different controls.

- **Radio buttons** - When all languages do not have the same setting for a radio button set, then none of the radio buttons will be selected.



- **Check boxes** - A check box option will be filled in with a square to indicate that all languages do not have the same value for the setting.



Indent with tabs

- **Text boxes** - Text boxes will be left blank to indicate that all languages do not share the same value.

Tabs:

- **Combo boxes** - When all languages do not have the same value for a combo box setting, the combo box will say **Languages Differ** in the text area.

Completion choice:

Languages Differ

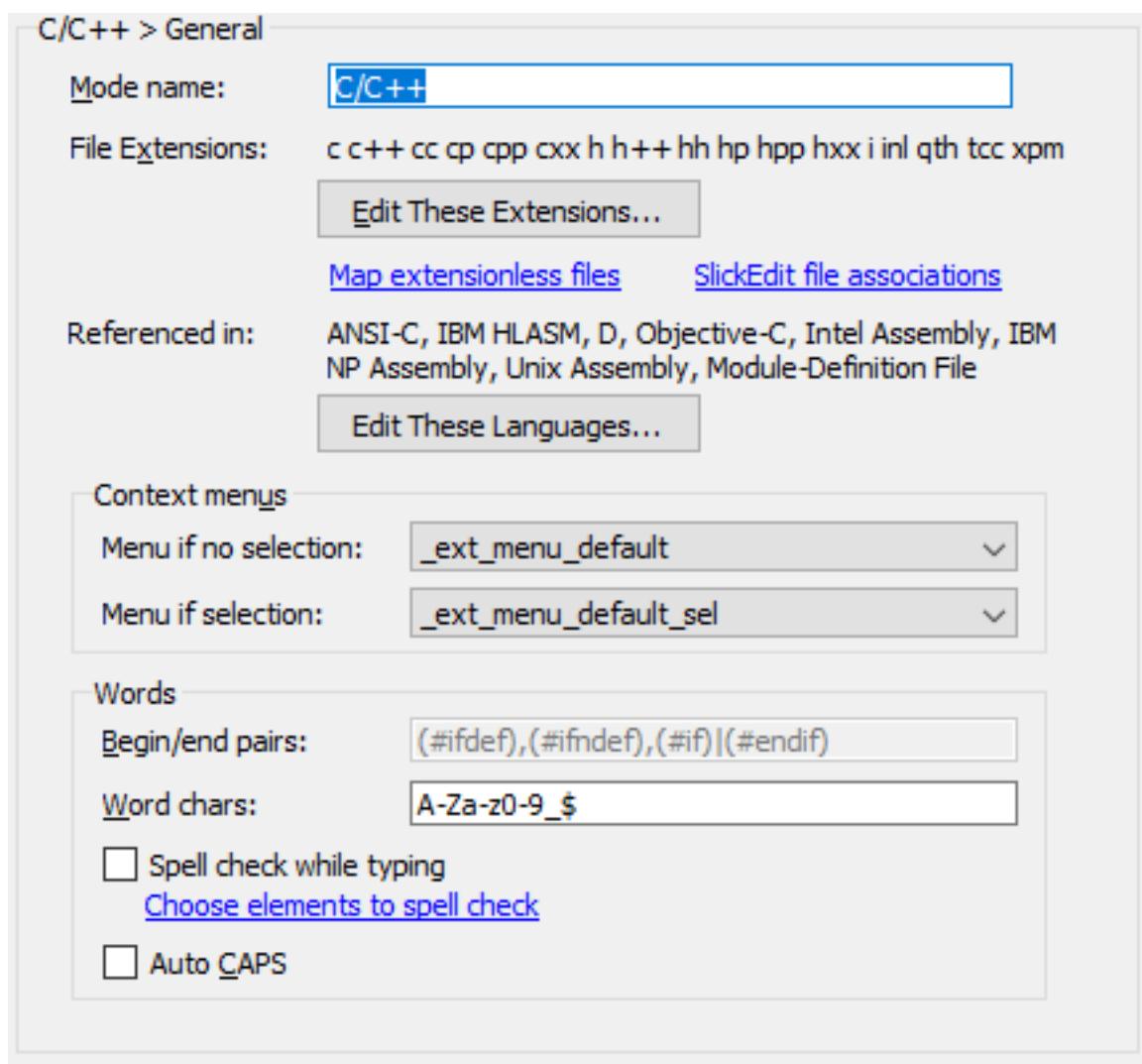


- **Property Sheet Options** - If an option found in a property sheet does not have the same setting for all languages, then the property value will say **Languages Differ**.

Option	Value
Load Options	
Load as Binary	Languages Differ

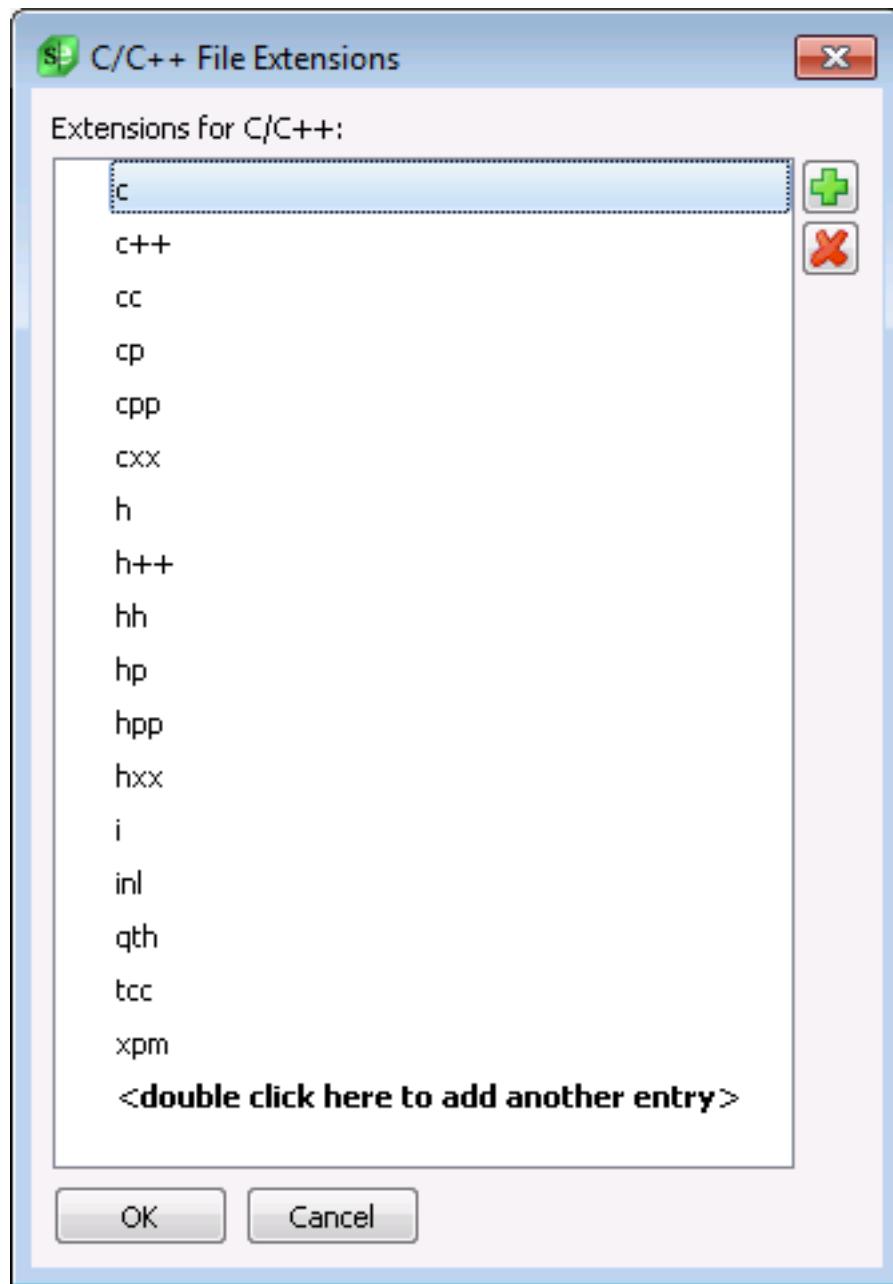
Language-Specific General Options

This option screen shows the mode name and associated file extensions for the selected language, and provides other general options. The settings on this page depend on the selected language. As an example, the C/C++ General options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **General**).

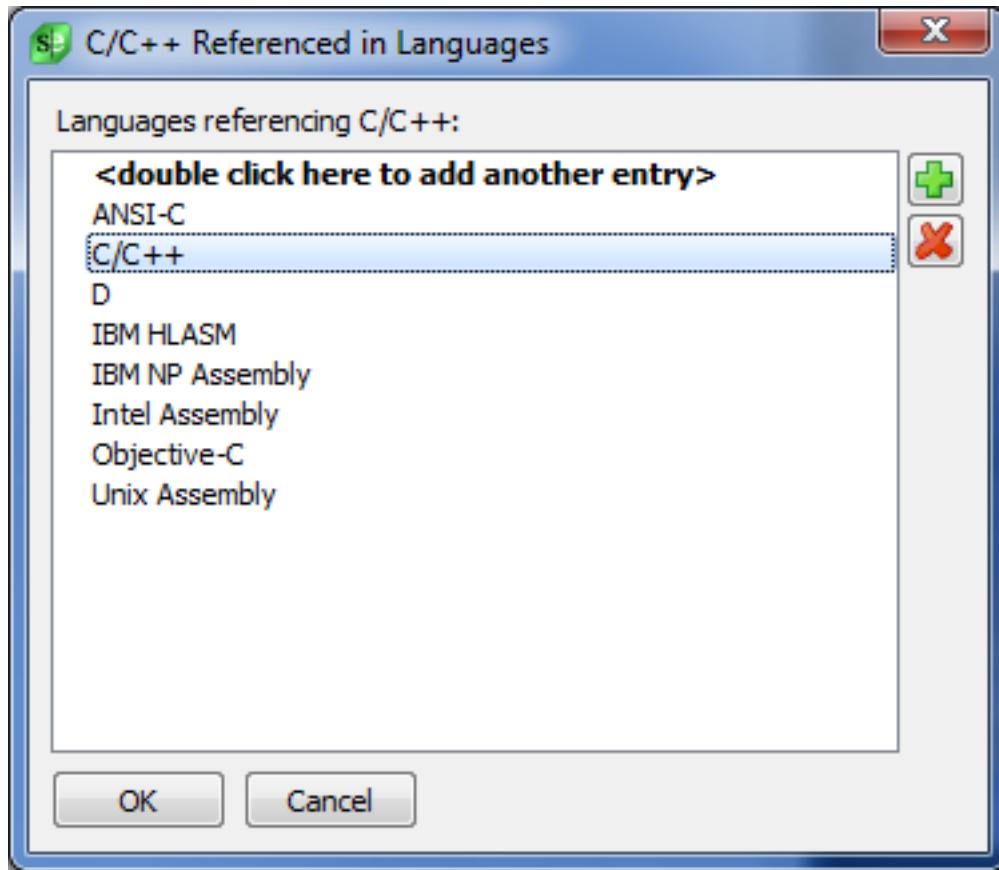


The options are described as follows:

- **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. See [Language Editing Mode](#) for more information.
- **File Extensions** - This area displays a list of file extensions associated with the selected language. See [Managing File Extensions](#) for more information.
To associate file extensions such that they are automatically opened in SlickEdit, see [Setting File Associations](#) (Windows only).
- **Edit These Extensions** - Allows you to add or remove file extensions for the language mode. The language-specific File Extensions dialog is displayed. Click the green **Plus** button to add a new extension and use the red **X** button to delete the selected extension. You can also add an extension by double-clicking where indicated.



- **Referenced in** - This area displays a list of languages which may contain code which can reference symbols in this language. This is used by the tagging and references searching to narrow down search results to files that are related to the originating file and to avoid searching through references in unrelated languages. See [Managing File Extensions](#) for more information.
- **Edit These Languages** - Allows you to add or remove languages from the list of languages which can reference symbols defined in this language mode. The language-specific Referenced in Languages dialog is displayed. Click the green **Plus** button to add a new language and use the red **X** button to delete the selected language. You can also add an language by double-clicking where indicated.



- **Context menus** - These options specify which context menu to display in the editor window based 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.
- **Words** - These options specify how to handle word boundaries, begin/end pairs, spell checking, and auto-capitalization of keywords.
 - **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 unavailable 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** - These are the characters that SlickEdit® uses to recognize a string of text as a word. 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.

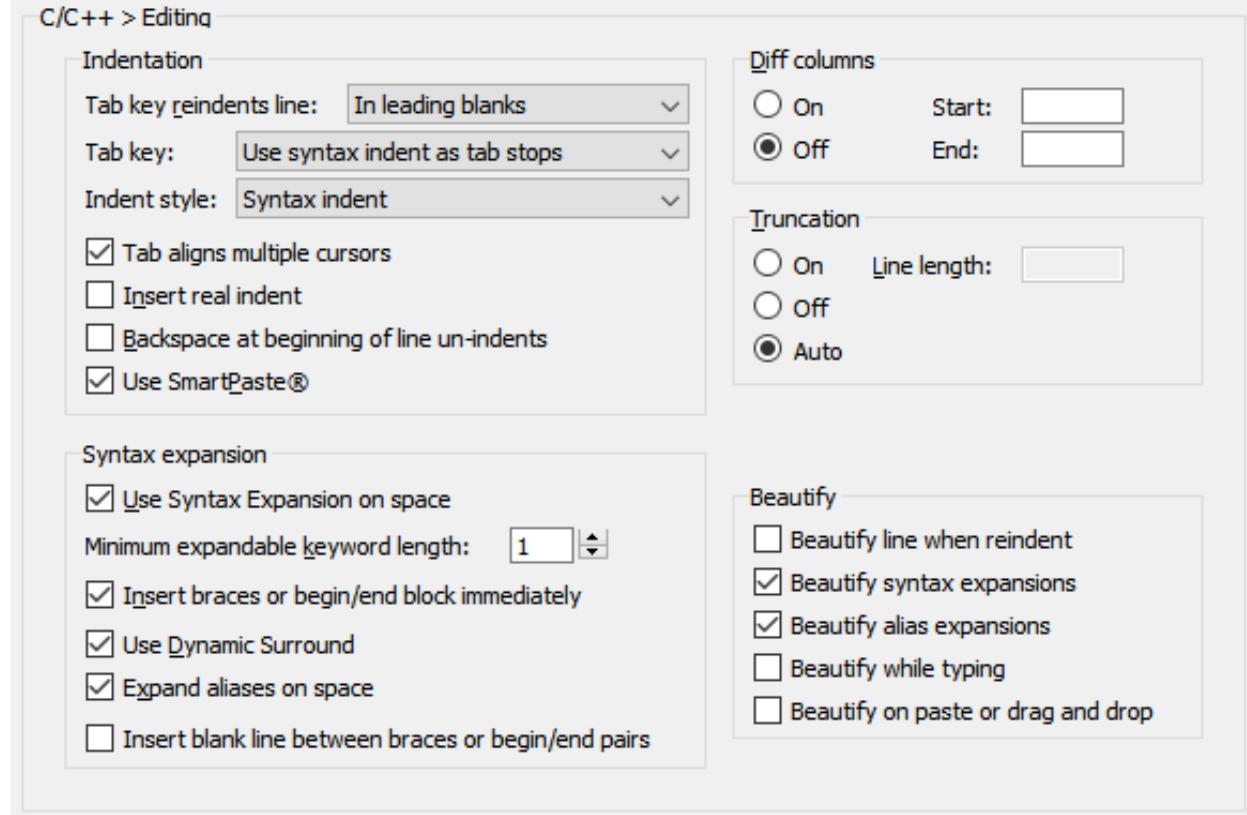
Note

Word chars are not used for tagging operations. To adjust the identifier characters used by Context Tagging®, use the **Identifiers** options on the **Tokens tab** of the Color Coding options page (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Color Coding**).

- **Spell check while typing** - (Pro only) When enabled, words are spell check as you type them. No more than the current page is ever spell checked making this a very efficient implementation. See [Spell Check Options](#) for information on configuration spell checking options.
- **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.
- **Language-Specific Project** - Click this button to set project properties specific to the selected language. See [Defining Language-Specific Projects](#) for more information.

Language-Specific Editing Options

This option screen shows the editing options for the selected language. The settings on this page depend on the selected language. As an example, the C/C++ Editing options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Editing**).



The options are described as follows:

- **Indentation** - These options specify indentation options for the selected language.
 - **Tab key reindents 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 reindents the line only if the cursor position is before the intended indent location and within the leading white space of the line; otherwise, it will insert an additional tab stop.
 - **In leading blanks strict** - Pressing the **Tab** key will reindent the line if the cursor is positioned within the leading white space of the line.
 - **Tab key** - These options specify how the **Tab** key indents. These options have no effect in cases where the line is reindented or beautified. Select from the following settings:
 - **Indent by syntax indent** - When the cursor is within the leading white space, pressing **Tab** indents by the syntax indent amount. For example, if the syntax indent is 4 and current column is 3, the cursor will be indented to column 7. Notice that is not treating the syntax indent amount as if it is tab stops. This is useful if your code has not been consistently indented. When the cursor is not within the leading white space, pressing **Tab** indents to the next syntax indent tab stop. Note that due to proportional fonts and Unicode, tab stops sometimes only work well for leading indent.
 - **Use syntax indent as tab stops** - When the cursor is within the leading white space, pressing **Tab** indents to the next syntax indent tab stop. For example, if the syntax indent is 4 and current column is 3, the next syntax indent tab stop is column 5. This is useful if your code has been consistently indented by the syntax indent amount. When the cursor is not within the leading white space, pressing **Tab** indents to the next syntax indent tab stop. Note that due to proportional fonts and Unicode, tab stops sometimes only work well for leading indent.
 - **Use tab stops and not syntax indent** - Move cursor to next tab stop defined by your tab stop settings. For example, if your tab stops are "5 20 25 80" and current column is 3, the next tab stop is column 5. This can be useful for text files or source languages which are very column oriented (possibly assembly or Cobol). Note that due to proportional fonts and Unicode, tab stops sometimes only work well for leading indent.
 - **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.
 - **Tab key aligns multiple cursors** - When this option is selected, the **Tab** key, when invoked with multiple cursors on unique lines, will align all of the cursors on the right-most tab stop. This is a helpful tool for aligning text in columns. Generally, this functionality is only available when the cursor

is in the middle of a line, in a circumstance where it would not be expected to re-indent the line. This functionality is not supported in some languages which do special processing for the **Tab** key.

- **Insert real indent** - When this option is selected, the **Enter** key inserts real spaces or tabs representing the indent instead of virtual spaces. This option allows the function for the **End** key on the keyboard to place the cursor after blank text where new text can be typed.
- **Backspace at beginning of line un-inds** - When this option is selected and the cursor is located before the first non-blank character, pressing the **Backspace** key unindents 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.
- **Syntax Expansion** - These options specify syntax expansion options for the selected language.
 - **Use Syntax Expansion on space** - Activates the Syntax Expansion feature. When this option is selected, pressing the spacebar after typing a keyword 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.

In addition, for brace-oriented languages, this setting also determines if certain control statements can be expanded as one-line statements (with no brace block) by typing a **semicolon** immediately after a control keyword such as **if** or **for**.

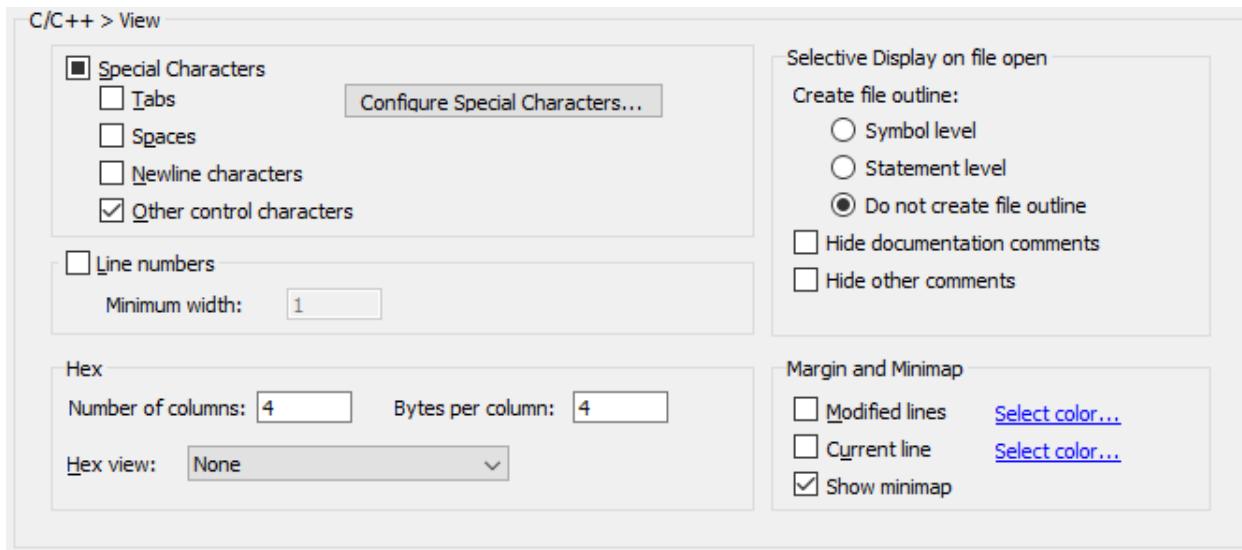
- **Minimum expandable keyword length** - Sets the minimum length for a keyword that will trigger Syntax Expansion. For example, if this is set to **3**, then two-letter keywords such as **if** will not be expanded.
- **Use Dynamic Surround** - Provides the ability to surround a group of statements with a block statement, indented to the correct levels according to your indent settings. 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.
- **Expand aliases on space** - When set to **On**, typing an alias identifier, then pressing space will automatically expand the alias. When set to **Off**, space does not expand aliases automatically. See [Global Aliases](#) for more information about Aliases.
- **Insert blank line between braces** - Specifies whether a blank line should be inserted between braces when a template expands with braces.
- **Diff Columns** - When **On**, diff will ignore changes outside the column range specified. This feature is designed for main frame languages like COBOL.
- **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).

- **Bounds** - This setting is unique to ISPF emulation. It controls column bounds for specific ISPF commands that operate on column ranges. See [Section ISPF Emulation Options](#)/[ISPF Options](#) for more information.
- **Beautify** - These options specify options for beautification while editing the selected language. These options are only available for languages which support beautify.
 - **Beautify line when reindent** - (Pro only) When on, beautifies the line when the **Tab** key reindents the line. This option is only supported by languages that support the beautify while typing feature. This feature is great for beautifying the current line when the beautify while typing feature doesn't get automatically triggered (i.e. you haven't typed a semicolon or other beautifier trigger key).
 - **Beautify syntax expansions** - (Pro only) When enabled, every time a syntax expansion occurs, the snippet of expanded code is run through the beautifier. Enabled by default. Only available for languages which have formatting beautifiers. See [Formatting Beautifiers](#) for more information.
 - **Beautify alias expansions** - (Pro only) When enabled, whenever a language-specific alias is expanded, the expansion is run through the beautifier. Enabled by default. Only available for languages which have formatting beautifiers. See [Formatting Beautifiers](#) for more information.
 - **Beautify while typing** - (Pro only) When enabled, the beautifier will be run on statements as you type them, usually when a statement terminator is encountered. Disabled by default. Only available for languages which have formatting beautifiers. See [Formatting Beautifiers](#) for more information.
 - **Beautify on paste or drag and drop** - (Pro only) Whenever a paste or drag and drop event occurs, the beautifier can run on the newly inserted statements. Only available for languages which have formatting beautifiers. See [Formatting Beautifiers](#) for more information.

Language-Specific View Options

These options control the display of special characters, line numbers, and more. C/C++ View options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **View**).



Special Characters

When this option is selected, view of all types of special characters is enabled for the language. This includes **Tabs**, **Spaces**, **Newline characters**, and **Other control characters** as well as all of the other special characters listed on the **Tools → Options → Appearance → Special Characters** option screen.

Alternately, select the individual options to enable display of the special characters you want to see. Note that you can also toggle display of special characters on a per-document basis with the menu item **View → Special Chars** (or use the `view_specialchars_toggle` command). Viewing of special characters is only available for ASCII files. See [Viewing Special Characters](#) for more information.

Configure Special Characters - Jumps to the **Tools → Options → Appearance → Special Characters** node in the Options dialog, where you can define the visible characters that represent each type of special character. See [Special Character Options](#) for more information.

Line Numbers

When this option is selected, display of line numbers is enabled for the selected language. By default, SlickEdit automatically adjusts the width of the line numbers based on the length of the current file. You can set a fixed width if you prefer.

Note that you can also toggle display of line numbers for a single document with **View → Line Numbers** (or the `view_line_numbers_toggle` command). See [Viewing Line Numbers](#) for more information.

Hex

The following options are available for hex editing

- **Number of columns** - Specifies the number of columns in Hex mode. Has no effect on Line Hex mode.
- **Bytes per column** - Specifies the number of bytes per column in Hex mode. Has no effect on Line Hex mode.

- **Hex View** - This option is used to determine the hex mode when you open a file. This setting is most useful for binary files. After opening a file, you can also enable Hex mode on a per-document basis with the menu items **View → Hex** and **View → Line Hex** (or use the commands **hex** and **linehex**). See [Hex Mode Editing](#) for more information.

Selective Display on file open

The following options are available for creating a selective-display outline of symbols, statements, and/or comments in a file when the file is opened.

- **Create file outline:**
 - **Symbol level** - Create an outline of all the global-level symbols.
 - **Statement level** - Create an outline down to the statement level. Note that this option is only supported for languages that support statement tagging.
 - **Do not create file outline**
- **Hide documentation comments** - Collapse documentation style comments, such as JavaDoc, XMLDoc, or Doxygen comments. This feature only works with documentation comment styles supported by the color coding engine.
- **Hide other comments** - Collapse other multi-line comments.

Margin and Minimap

The following options are available for how to color the margin for modified lines, whether to color the current line, and whether to show the minimap

Modified Lines

When checked, modified and inserted lines are indicated with a color bar in the left margin. Click the **Select color** link to select the colors for each. For more information see [Modified Lines](#).

Current Line

When checked, the current line is highlighted using the selected background and foreground color. To select the colors used, click the **Select colors** link. For more information see [Current Line](#).

Show minimap

When checked, the minimap window is displayed.

Language-Specific Appearance Options

These options control the appearance of files in the selected language. C/C++ Appearance options are shown below (**Tools → Options → Appearance**).

Pro:

Language Options

C/C++ > Appearance

Fonts

	Font name	<input checked="" type="checkbox"/> Fixed Fonts Only	Size
Source Windows:	Default: Default Unicode Font	<input type="button" value="..."/>	Default: 10 <input type="button" value="..."/>
Minimap Windows:	Default: Default Unicode Font	<input type="button" value="..."/>	Default: 1 <input type="button" value="..."/>
Diff Editor Windows:	Default: Default Unicode Font	<input type="button" value="..."/>	Default: 10 <input type="button" value="..."/>

[Configure Default Fonts...](#)

Colors

	Editor Windows	Minimap Windows
Color Profile:	Default: Automatic <input type="button" value="..."/>	Default: <input type="button" value="..."/>
Alternate Profile:	Default: Automatic <input type="button" value="..."/>	Default: <input type="button" value="..."/>
for files matching:	*.h; *.h++; *.hh; *.hp; *.hpp; *.hxx; *.qth	
Configure Color Profiles...		Example: *.h; *.hpp; *.inl

Symbol coloring

Use **bold** for symbol names in definitions and declarations

Highlight unidentified symbols

Color positional keywords

Symbol lookup strategy: [Use strict symbol lookups \(full symbol analysis\)](#)

[Configure Symbol Coloring Rules...](#)

Standard

C/C++ > Appearance

Fonts

	Font name	<input checked="" type="checkbox"/> Fixed Fonts Only	Size
Source Windows:	Default: Default Unicode Font	<input type="button" value="..."/>	Default: 10 <input type="button" value="..."/>
Minimap Windows:	Default: Default Unicode Font	<input type="button" value="..."/>	Default: 1 <input type="button" value="..."/>
Diff Editor Windows:	Default: Default Unicode Font	<input type="button" value="..."/>	Default: 10 <input type="button" value="..."/>

[Configure Default Fonts...](#)

Colors

	Editor Windows	Minimap Windows
Color Profile:	Default: Automatic <input type="button" value="..."/>	Default: <input type="button" value="..."/>
Alternate Profile:	Default: Automatic <input type="button" value="..."/>	Default: <input type="button" value="..."/>
for files matching:	*.h; *.h++; *.hh; *.hp; *.hpp; *.hxx; *.qth	
Configure Color Profiles...		Example: *.h; *.hpp; *.inl

Fonts

Font choices can be chosen on a per-language basis for source windows, the minimap window, and diff editor windows. In each case, the font face, as well as the font size may be overridden individually over corresponding global default setting.

Using this feature, you can select a proportional font for text-centric languages like HTML and let other languages fall back on a default programming oriented fixed-width font. It also lets you select specific programming fonts for specific languages, because in some cases ligatures are more appropriate for some languages than they are for others.

- **Source Windows** - Applies to Editor windows that are displaying content in the selected language mode in either Unicode or SBCS/DBCS.
- **Minimap Windows** - Applies to Editor minimap windows that are displaying content in the selected language mode in either Unicode or SBCS/DBCS.
- **Diff Editor Windows** - Applies to Editor windows used by DIFFzilla® that are displaying content in the selected language mode in either Unicode or SBCS/DBCS.
- **Configure Default Fonts...** - Click this link to open the Font Options. where you can define global defaults for various font options. See [Font Options](#) for more information.

Note

Any of these options can be cleared back to defaults simply by selecting the combo box and hitting delete.

Colors

Color profiles can be chosen on a per-language basis for source windows and minimap windows. Using this feature you can select a different color profile for C++ source files, for example, than you use for SQL source files.

- **Color Profile** - The default color profile to user for the selected language. Press the chooser to preview and select a color profile.
- **Alternate Profile** - For each language, there is a primary and alternate color profile. The alternate color profile is chosen if the file name matches the given list of file extension wildcards. Using this feature, you can configure SlickEdit to select a different color profile for C++ header files than you use for C++ source files.
- **for files matching** - List of file extension wildcards to match for the alternate color profile.
- **Configure Color Profiles...** - Click this link to open the Color Options. where you can define editor color profiles, as well select global defaults. See [Color Options](#) for more information.

Note

Any of the color profile can be cleared back its default simply by selecting the combo box and hitting delete.

Symbol Coloring (Pro only)

Use this to enable Symbol Coloring for the selected language.

- **Symbol coloring** - Enables or disables Symbol Coloring entirely for the language. Disabling Symbol coloring is a good idea if symbol analysis was particularly slow or ineffective for a language, such that symbol coloring was only slowing you down.

Note

Symbol coloring is automatically disabled for HTML and other XML variants. It is also automatically disabled in all modes which do not have any Context Tagging® implemented. Finally, Symbol Coloring is disabled in all embedded language contexts. This means that Symbol Coloring is disabled for all PHP code, since PHP is always embedded in an HTML or XML processing instruction (<?php).

- **Use bold for symbol names in definitions and declarations** - By default, Symbol Coloring will bold the name part of symbol declarations and definitions. This is particularly useful for languages which allow implicit local variable declarations. It is also helpful when the declaration syntax is not always visually distinct from the rest of the code.
- **Highlight unidentified symbols** - Symbol Coloring is able to select the **Symbol not found** rule for symbols which are not found by Context Tagging®. This can serve effectively as a live error checker with respect to spelling and capitalization of symbols.

However, in certain languages, especially scripting languages that allow variables to be declared implicitly, Context Tagging® can be less effective, simply because the code can not be analyzed statically. In this case, you might see an unusually large number of symbols highlighted as unidentified symbols. This can also happen if you do not have Context Tagging® configured correctly for the code and libraries you are working with.

For this reason, this feature is *disabled* by default.

- **Override color for Library symbols and User defined keywords** - When enabled, this option allows identifiers configured in Color Coding as **Library Symbol** or **User Defined Keyword** to have their coloring overridden by Symbol Coloring if the symbol's definition can be found by Context Tagging®.
- **Color positional keywords** - When available and checked, positional keywords (identifiers which can be keywords when used in a specific context) which are detected by the language specific tagging parser will be colored as keywords. This feature is only supported by a few languages, including SQL.
- **Symbol lookup strategy** - This setting allows you to select a symbol lookup strategy appropriate for the selected language.
 - **Use strict symbol lookups (full symbol analysis)** - By default, Symbol Coloring uses this fairly

strict language specific symbol lookups in order to identify symbols.

- **Use relaxed symbol lookups (symbol analysis with relaxed rules)** - In some languages, it is necessary to relax the rules in order to find symbol definitions. This can, for example, be useful in heavily templated or preprocessed C++ code which is too complex for Context Tagging®. Selecting **Use relaxed symbol lookups** instead of the default of **Use strict symbol lookups** will tell Symbol Coloring to revert to a more flexible symbol lookup, ignoring scope and visibility rules, if the strict symbol lookup does not yield results.
- **Use fast, simplistic symbol lookups (symbol name only)** - In large, complex code bases, the strict symbol lookup algorithm may require too much time to be practical to use. Sometimes a more simplistic approach of looking up the symbol based on the symbol's name alone, ignoring context, usage, and scope is adequate.

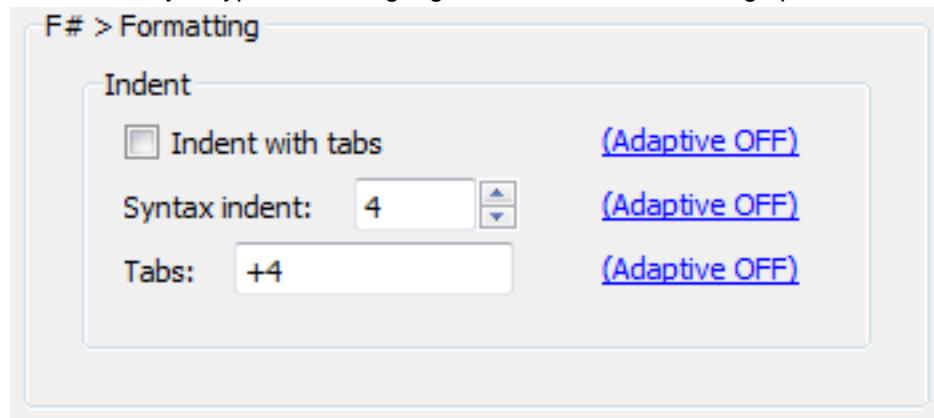
Select this option to enable the fast, simple symbol lookup algorithm. Note that using the simplistic symbol lookup algorithm can drastically decrease the accuracy of Symbol Coloring, especially with respect to detecting misspelled symbols and looking up members of a class or structured type.

- **Configure Symbol Coloring Rules...** - Click here to be redirected to the Symbol Coloring Options to view and modify symbol coloring rules bases. See [Symbol Coloring Options](#) for more information.

Language-Specific Formatting Options

These options let you configure the way SlickEdit® formats code as you type. Depending on the language, you can specify the code formatting templates, how various syntactical elements are treated, when and what code elements are automatically inserted, and more. The formatting options that are available depend on the selected language.

For languages which do not yet have formatting beautifiers, you can still specify how the code will be formatted as you type. Each language has at least the following options:



- **Indent with tabs** - Determines whether **Tab** key, **Enter** key, and paragraph reformat commands indent with spaces or tabs. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting. See [Indenting with Tabs](#) for more information.
- **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. The hyperlink indicates if [Adaptive](#)

Language Options

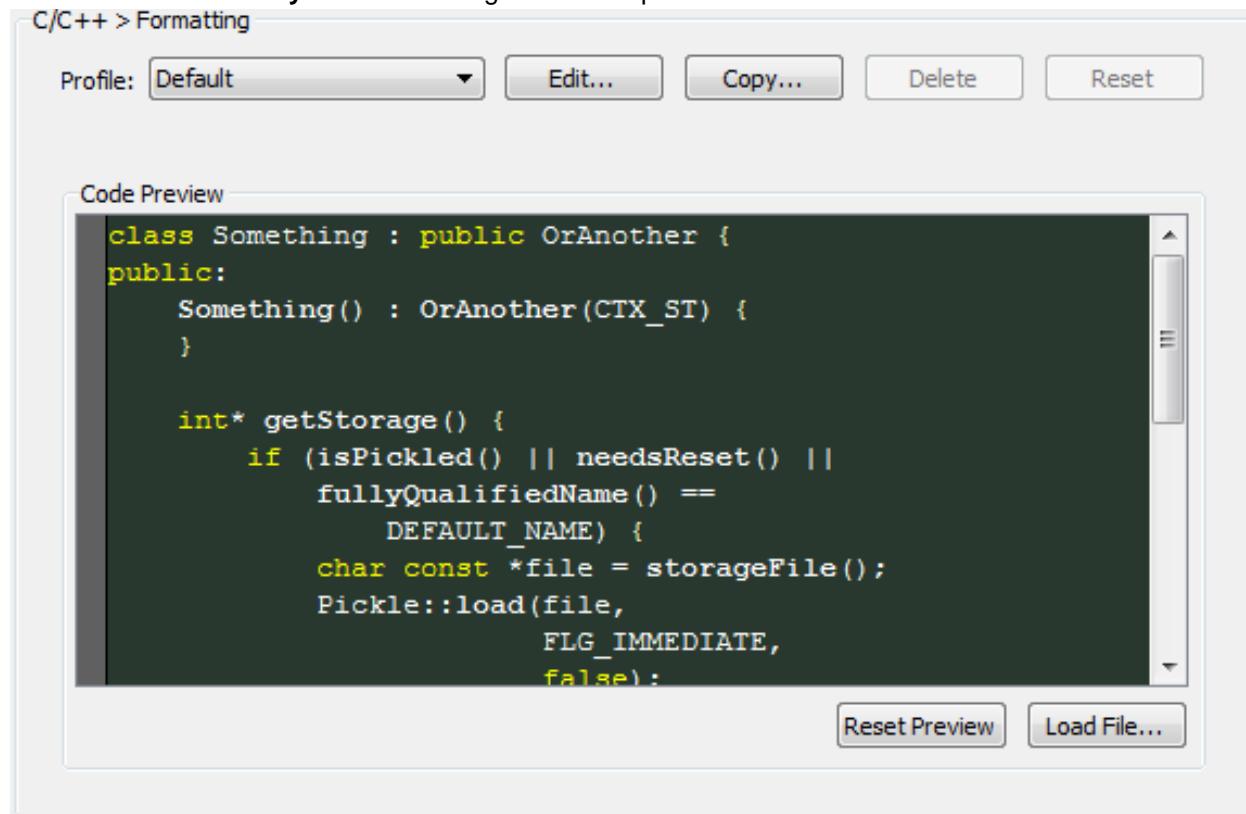
[Formatting](#) is on or off for this setting. See [Syntax Indent](#) for more information.

- **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. The hyperlink indicates if [Adaptive Formatting](#) is on or off for this setting.

Other languages have more advanced options. For more information, see the following section for your language (or the one that most closely relates to your language):

- [Common Formatting Options Interface](#)
- [XML Formatting Options](#)
- [HTML Formatting Options](#)
- [Ada Formatting Options](#)
- [COBOL Formatting Options](#)
- [Pascal Formatting Options](#)
- [PL/I Formatting Options](#)

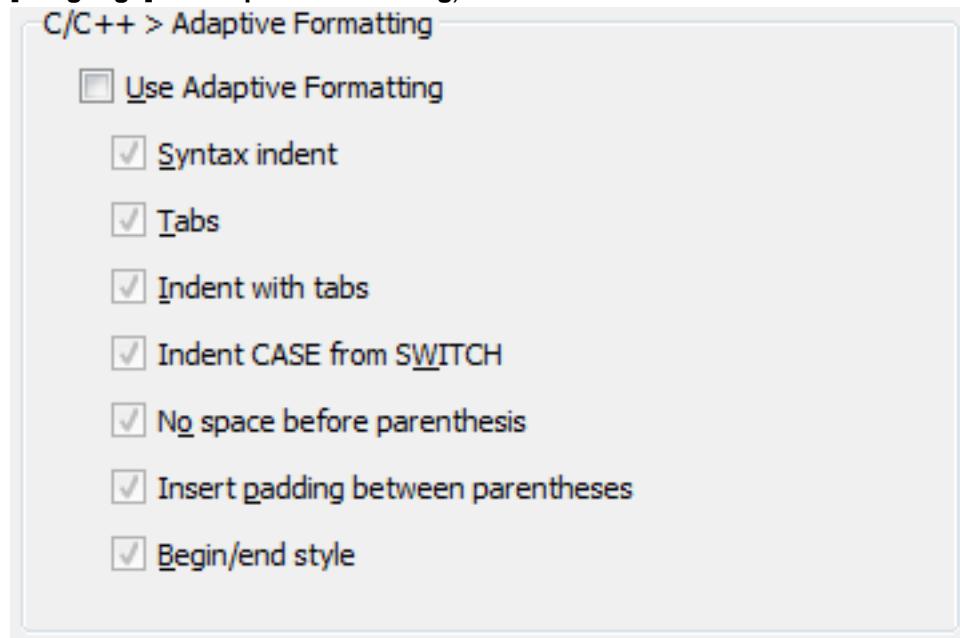
(Pro only) Some languages have beautifiers that handle all formatting settings. Use the beautifier settings to control automatic as-you-type formatting, as well as how SlickEdit® reformats your code when you select **Tools** → **Beautify**. The Formatting Beautifier options are shown below.



For more information, see [Beautifying Code](#).

Language-Specific Adaptive Formatting Options

Adaptive Formatting scans a file for the formatting styles in use and automatically matches those settings for the current editing session. The options on this screen are used to enable/disable Adaptive Formatting and configure the styles that SlickEdit® should recognize for the language. The C/C++ Adaptive Formatting options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **[Language]** → **Adaptive Formatting**).



Select or clear the **Use Adaptive Formatting** option to enable or disable the feature for the selected language. When Adaptive Formatting is enabled, use the subsequent check boxes to select the individual style settings for which SlickEdit should scan. The individual style settings that appear on the Options screen will vary depending on the language. See [Adaptive Formatting](#) for more information.

Language-Specific Comment Options

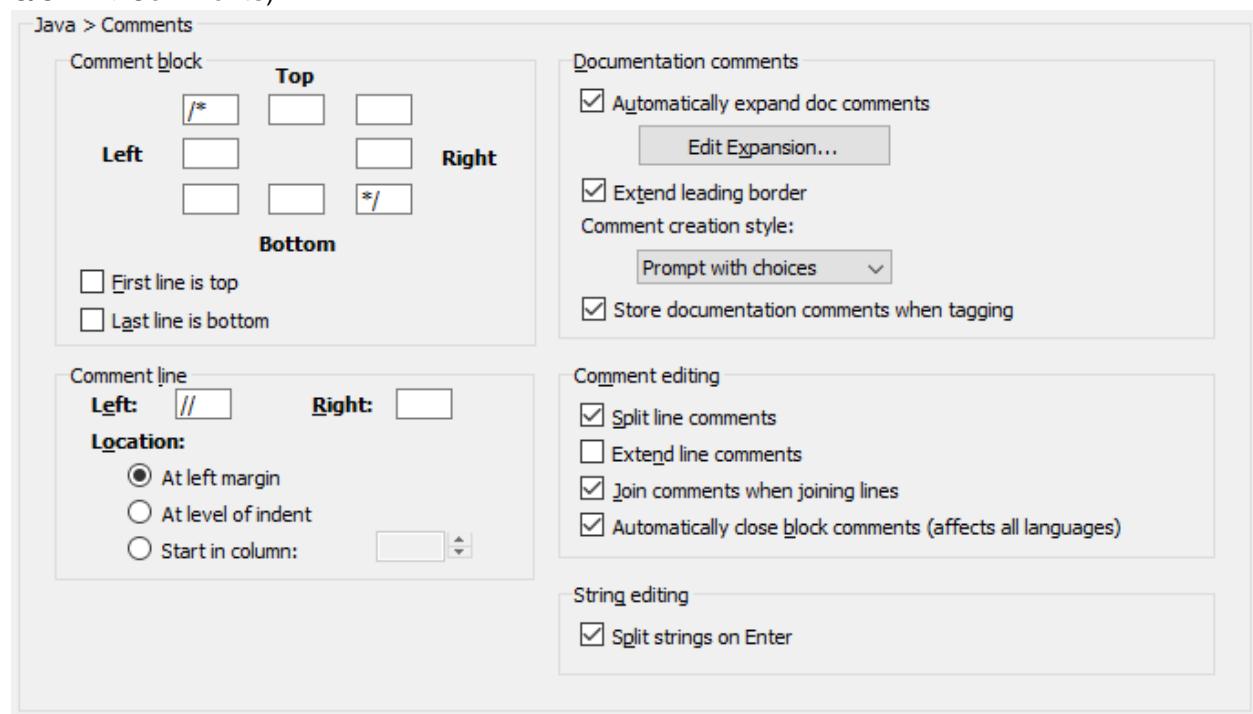
Comment options let you control how block and line comments are created.

To comment out selected lines, select text in the editor and then click **Document** → **Comment Block** or **Document** → **Comment Lines** (**box** and **comment** commands, respectively). These operations use the matching comment style to comment out all text on the lines containing the selection. **Comment Block** surrounds multiple lines with a single block comment. **Comment Lines** comments out each line in the selection with a line comment. See [Commenting](#) for more information.

Note

The settings on this page are used only when inserting block and line comments. To configure which characters are recognized as comments, go to **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Color Coding** and then select the **Comments** tab.

The settings on the Comments screen depend on the selected language. As an example, the C/C++ options for comments are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Comments**).



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:

- **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 //, clicking **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.

Note

In C#, you do not need to press **Enter**, as the skeleton comment is inserted after you type the third slash.

- **Automatically expand XMLDOC comments** - Turn this on to automatically insert a skeleton XMLDOC comment when you type comment start characters directly above a function, class, or variable.
- **Edit expansion** - Click this button to open the Doc Comment Editor, where you can define and edit the templates that are inserted when doc comments are expanded. See [Modifying Doc Comment Templates](#) for more information.
- **Extend leading border** - Put a check in this box to precede each line with the leading characters from the previous line. This is useful if you like to have an asterisk in the first column of your doc comments, for example.
- **Comment creation style** - Use this option to select the kind of documentation comment you prefer to

have generated when you use the **Update Doc Comment** command from the right-click context menu for a function which does not have an existing documentation comment.

- **Prompt with choices** - You will be prompted which style of documentation comment to generate.
- **Do not convert style** - The documentation comment generator will attempt to generate a documentation comment using the existing comment style, even if it is not a documentation comment style.
- **/** Javadoc style** -
- **/*! Doxygen style** -
- **//! Doxygen style** -
- **/// Doxygen style** -
- **/// XMLDOC style** -
- **Store documentation comments when tagging** - When enabled, tagging will scan for documentation comments associated with symbols and store them in the tag file. It will also detect when a symbol has no comments. This speeds up fetching comments displayed in the Preview tool window, Auto-complete, List Symbols, and Function Argument help. This option is enabled by default. Turning off this option has the advantage of making the tag file slightly smaller and forces all the comment extraction to be done using the per-language logic that previous versions of SlickEdit used.

Comment editing

The following options control comment editing behaviors. These options will be unavailable 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.
```

- **Automatically close block comments** - Enables automatic completion of C-style comment block start and end markers. Typing /* on a blank line will auto-complete to /**, with the cursor placed between the two asterisks. This option applies to all languages.

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. ;
```

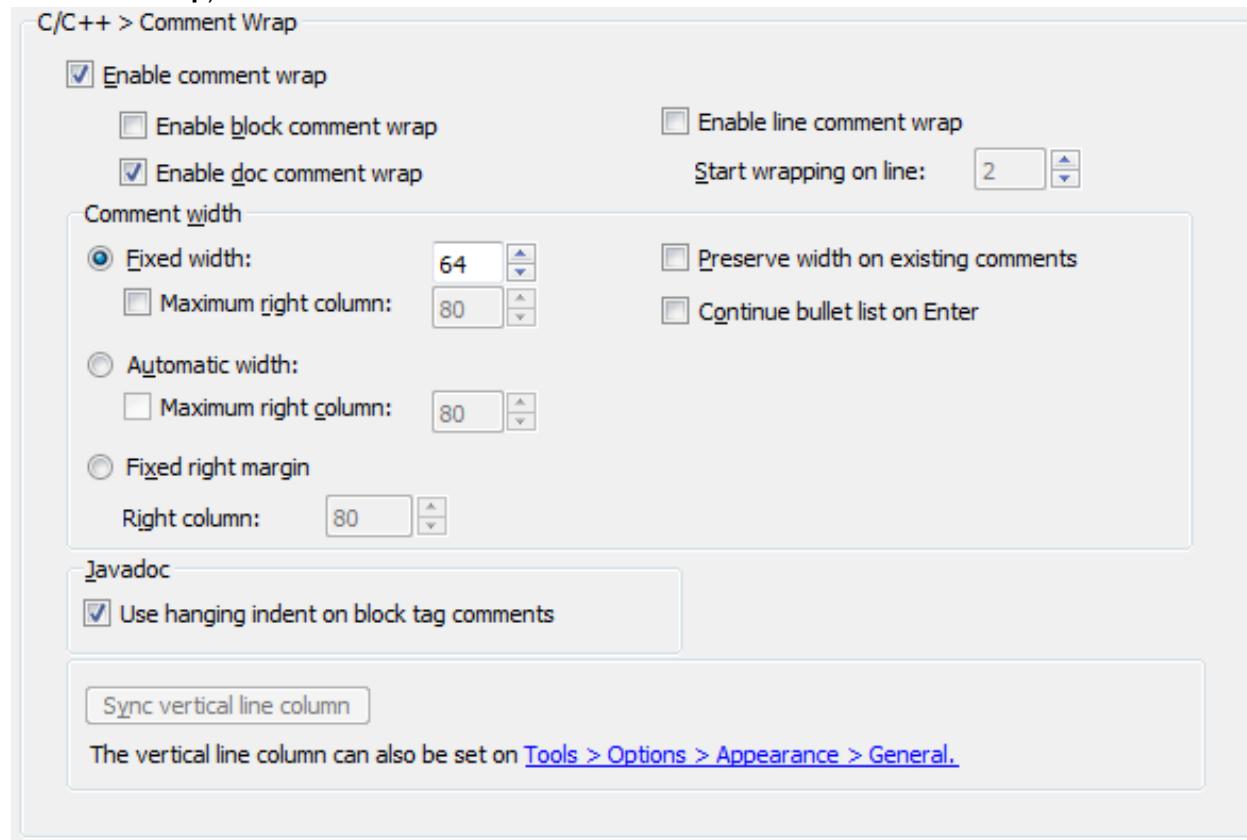
The **Split strings on Enter** feature only supports strings which may not span multiple lines. Not all languages have support for this feature. If you want this feature added to your language, please request it.

Language-Specific Comment Wrap Options

Language Options

Comment Wrap options let you activate wrapping and configure the way block, line, and doc comments are wrapped. See [Comment Wrapping](#) for more information.

The settings on this page depend on the selected language. As an example, the C/C++ Comment Wrap options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Comment Wrap**).



The options are described as follows:

- **Enable comment wrap** - When selected, comments are allowed to be wrapped. You must still specify 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 turned on, 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.

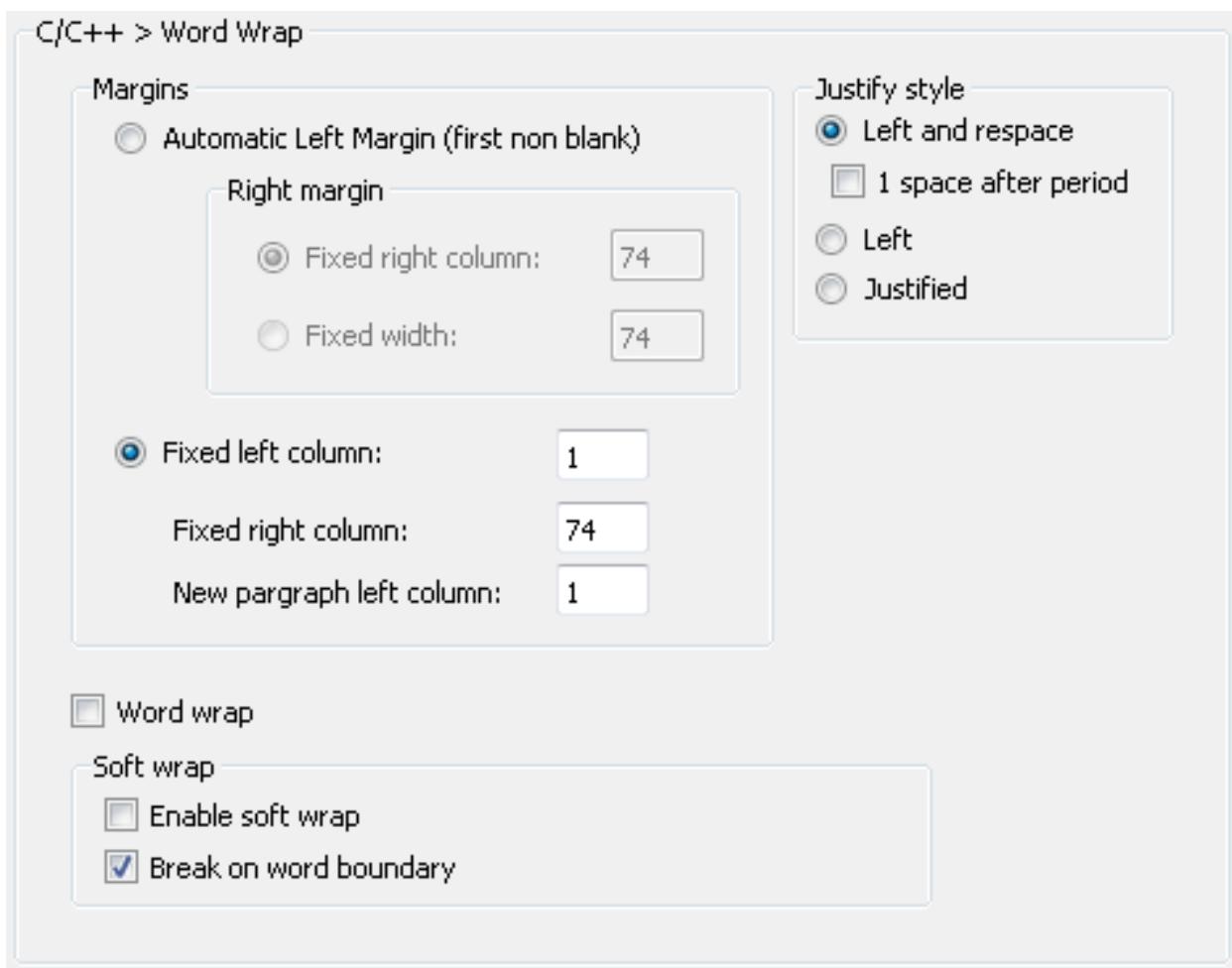
If **Maximum right column** is used, comment lines will be wrapped when they reach the specified column, even if they have not reached the specified fixed width. This is useful if coding standards mandate that text should not exceed a specified column.

- **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). To set the vertical line column to a different value, see [Vertical line columns](#).

Language-Specific Word Wrap Options

These language-specific options let you set margins and the justification style and configure Word Wrap, which keeps the cursor within the specified margins when entering text, moving the cursor, and deleting characters. Note that Word Wrap is intended for plain text only.

The settings on this page depend on the selected language. As an example, the C/C++ Word Wrap options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Word Wrap**).



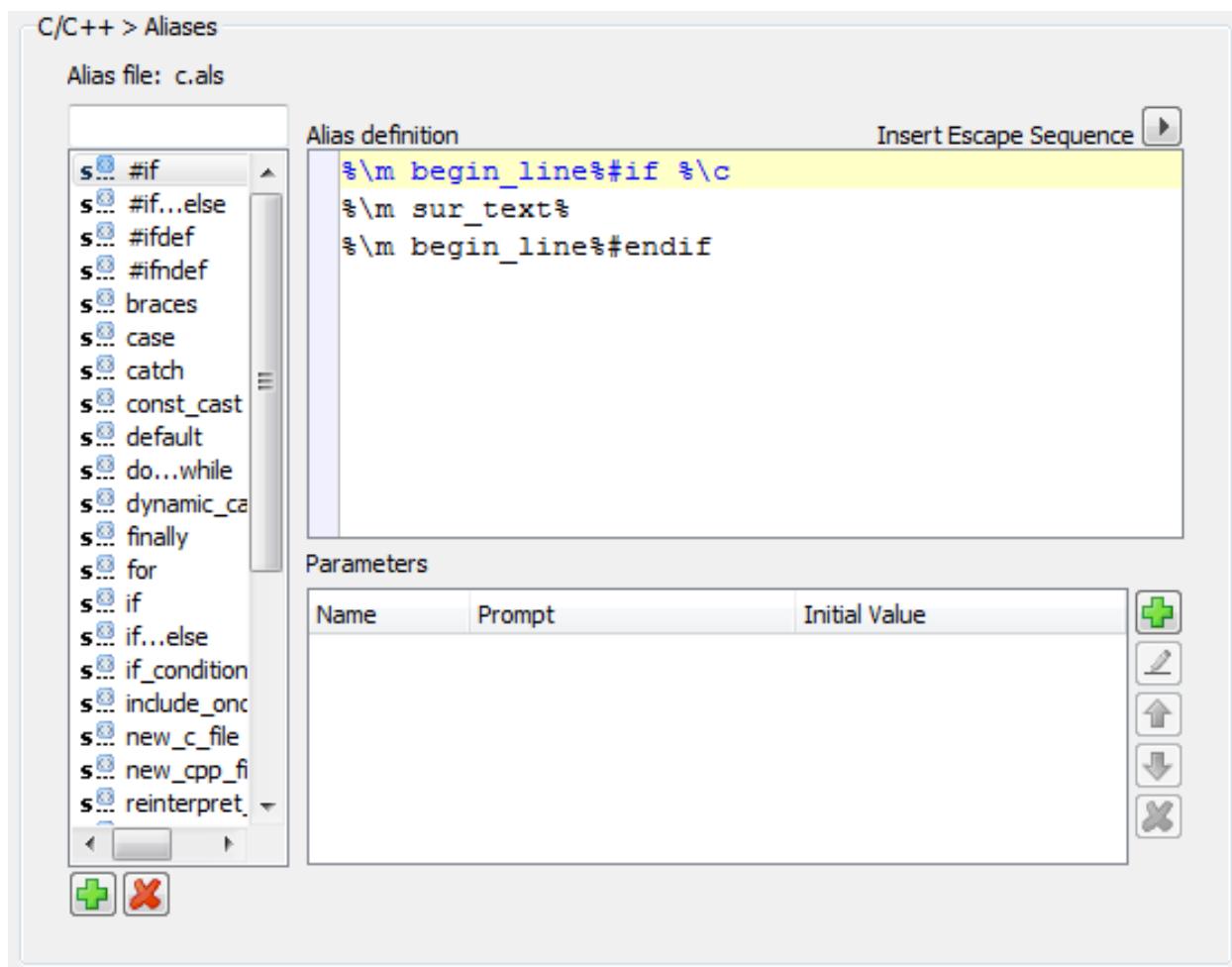
The options are described as follows:

- **Automatic Left Margin** - If selected, the left margin is determined by the first non blank in the line. The right margin may be specified as follows:
 - **Fixed right column** - If selected, lines will break before the specified column.
 - **Fixed width** - If selected, specifies the maximum amount of non blank text allowed on each line.
- **Fixed left column** - If selected, allows you to specify the left margin, right margin, and new paragraph columns.
- **Justify style** - Select from the following justification styles:
 - **Left and respace** - Left justification with space character reformatting. One space is placed between words except after the punctuation characters period, ?, and !, which get two spaces. To have only one space after the period, question mark, and exclamation point punctuation characters, 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 File Options](#) for more information.

- **Justified** - Full justification. Left and right edges of text will align exactly at margins.
- **Partial word wrap** - When on and word wrap while typing is on, a more conservative word wrap approach is taken. This option provides word wrap similar to previous versions of SlickEdit. You may prefer this style of word wrapping if you leave word wrap while typing on for source files. This option only effects word wrap while typing characters, pressing **Backspace**, or pressing **Del**.
- **Word wrap while typing** - This option activates/deactivates Word Wrap. When on, word wrapping within the margins occurs when typing characters, pressing **Backspace**, pressing **Del**, pressing **Left**, or pressing **Right**. Note that Word Wrap is intended for plain text only.
- **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:
 - **Enable soft wrap** - This option activates 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 the line so that words are kept whole. This makes for easier reading, especially in text files.

Language-Specific Alias Options

Aliases are identifiers that you can quickly type, which are then expanded into snippets of text. Language-specific aliases are useful for inserting comment headers, statement and function templates, or any other text that you frequently use. This option screen is used to manage language-specific aliases. As an example, the C/C++ Aliases options are shown below. See also [Language-Specific Aliases](#) for more information.



The name of the file that contains the aliases is displayed next to the label **Alias file**.

The left side of the options screen contains the **alias list**. The box on top of the alias list allows you to search the alias list incrementally as you type, so you can find the alias you want to edit or remove.

The list shows both regular aliases and **Surround With** aliases. Surround With is a feature that lets you surround existing code with text or predefined structures. Alias types are differentiated in the list by icon. See [Surround With](#) for more information about creating and working with Surround With aliases.

The large box on the right is the **alias edit window**. When an alias is selected in the alias list, you can type directly inside this window to define or edit the alias expansion.

Use the **Insert Escape Sequence** button to insert escape sequences into your alias expansion. See [Alias Escape Sequences](#) for a list of available sequences.

The following buttons appear under the alias list and alias edit window:

- **New** - Click this button to create a new alias name to be added to the alias list. After doing this, define the expansion by typing in the alias edit window.
- **Delete** - Deletes the alias that is currently selected in the alias list.

The lower section of the Aliases options page is used to create and manage parameter prompts in aliases. [Parameter Prompting](#) is a feature that lets you insert a parameter inside an alias so that when the alias is expanded, a dialog is displayed, prompting you to input the values.

The parameter list contains a list of the parameters you have created. It is divided into sections that correspond to the fields on the Enter Alias Parameter dialog, which is used to add a new parameter:

- **Param Name** - The name that is used in the alias to identify this parameter.
- **Prompt String** - This string appears as a label on the dialog that prompts for values when the alias is expanded.
- **Initial Value** - (Optional) This text is automatically entered as the initial value for the parameter on the dialog that prompts for values when the alias is expanded.

The Parameters section of the Aliases options page provides the following buttons:

- **Add** - Displays the Enter Alias Parameter dialog, used to add a new parameter for the alias that is currently selected in the alias list. See [Parameter Prompting](#) for more information.
- **Remove** - Deletes the parameter that is selected in the parameter list.
- **Edit** - Displays the Edit Alias Parameter dialog, used to edit the parameter that is selected in the parameter list. See [Parameter Prompting](#) for more information.
- **Up** and **Down** - Use these buttons to change the order of the parameters, moving the selected parameter up or down in the parameter list.

Language-specific aliases can be automatically expanded when you type the alias identifier and press space.

Language-Specific Auto-Complete Options

These options let you configure the behavior of the [Auto-Complete](#) feature. The settings on this page depend on the selected language. As an example, the C/C++ Auto-Complete options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Auto-Complete**).

Language Options

The screenshot shows the 'C/C++ > Auto-Complete' section of the Language Options dialog. It is divided into several panels:

- Enable auto-completion:** A list of checkboxes:
 - History (with a history icon)
 - Symbols
 - Locals
 - Current class
 - Current file
 - Syntax expansion
 - Alias expansion
 - Keywords
 - Word completion
 - Argument completion
- Visual details:** A list of checkboxes:
 - Light bulb
 - Expanded text
 - List of matches
 - Show icons
 - Show categories
 - Show history
 - Show parameters
 - Symbol declaration
 - Show comments
- Auto-Complete Options:** A list of checkboxes:
 - Insert open parenthesis for functions
 - Tab inserts longest unique prefix
 - Tab cycles through choices
 - Space inserts longest unique prefix
 - Space always inserts space
 - Enter always inserts itemBelow these are:
 - Minimum prefix length: (with up/down arrows)
 - Completion choice:
 - Preserve identifier to the right of cursor:
 - List include files after typing #include:
- Strict case-sensitivity options:** A list of checkboxes:
 - For auto-complete
 - For list-members / list-symbols
 - For manual symbol completions
- Subword matching options:** A list of checkboxes:
 - For auto-complete
 - Exclude globals for auto-complete
 - For list-members / list-symbols
 - For manual symbol completions
 - First attempt uses prefix match only
- Subword matching strategy:** A dropdown menu set to "Stone-skipping with subword boundaries".
 - Relax pattern matching order constraints
 - Start matching globals with first char
 - Limit to current workspace
 - Include auto-updated tag files
 - Include compiler tag file
 - Find completions which fix minor typos
- List-Symbols options:** A list of checkboxes:
 - Auto-list members
 - Auto-list compatible values

The options are described as follows:

- **Enable auto-completion** - If selected, activates the Auto-Complete feature. See [Auto-Complete](#) for more information.
- **History** - If selected, SlickEdit will look at auto-completions and manual symbol completions performed in the past and list the most common ones at the top of the list of completions. Reuse of completions is dependant on the scope and context in which the completion is being done. Click on the icon to review and edit the list of completions stored.

Note

Combining sub-word pattern matching and the history feature gives SlickEdit a learning ability with respect to completions. For example, if you have a common symbol such as `endModalGUISession`, and you use sub-word pattern matching to complete it using the abbreviation `mgs`, when you need that same identifier again, auto-complete will look at your history information and suggest `endModalGUISession` as soon as you start typing `m` in a similar context. Taking advantage of this behavior allows you to train auto-complete so you can code faster.

- **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.
- **Locals** - (Pro only) If selected, local variables and parameters will be displayed as symbol completion choices. This functions identically to the **Symbols** setting, except that the results are limited strictly to locals. For performance, **Locals** can be enabled even if **Symbols** is disabled.
- **Current class** - (Pro only) If selected, methods and members in the current class will be displayed as completion choices. This functions identically to the **Symbols** setting, except that the results are limited strictly to members of the current class. **Current class** can be enabled even if **Symbols** is disabled.
- **Current file** - (Pro only) If selected, symbols from the current file will be displayed as completion choices. This functions identically to the **Symbols** setting, except that the results are limited strictly to the current file. For performance, **Current file** can be enabled even if **Symbols** is disabled.
- **Syntax expansion** - If selected, Auto-Complete 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. See [Syntax Expansion](#) for more information.
- **Alias expansion** - If selected, Auto-Complete 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](#).
- **Keywords** - If selected, Auto-Complete will show keyword choices for the word prefix under the cursor, if it matches one or more keywords in the current language, as defined in the language-specific color coding options. For more information on keywords, see [Color Coding Tokens Tab](#).
- **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 the word and line completion features of SlickEdit®. 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. See [Word Completion](#) for more information.
- **Argument completion** - (Pro only) If selected, turns Auto-Complete on in the Build tool window for completing file names and paths.

- **Visual Details** - The Visual details of Auto-Complete system can be customized to your tastes to make it show only the information you require.
 - **Light bulb** - If selected, displays the light bulb as a reminder when Auto-Complete suggestions are available for the current word prefix.
 - **Expanded text** - If selected, shows the rest of the word or statement being completed.
 - **List of matches** - If selected, shows the list of matches underneath the word prefix. Use the key combinations of **Shift+Up** and **Shift+Down** to move the list above or below the current line provided there is enough space to display it there.
 - **Show icons** - If selected, displays symbol icons and folder icons. Turn this feature off to get a more compact list containing only names.
 - **Show categories** - If selected, shows completions in a categorized list for each type. If cleared, all completions will be shown in one flat, sorted list.

Note

Disabling this feature largely negates the value of the Auto-Complete history feature, because your most-common history completions will be mixed in with everything else instead of being in a separate category at the top of the list.

- **Show history** - This option is automatically turned on if **Show categories** is selected. When **Show categories** is not selected, this option controls whether a separate category is displayed for auto-complete history, in which case, there will only be two categories displayed, **History** and **All Completions**. Otherwise, If cleared, all completions will be shown in one flat, sorted list.

Note

Disabling this feature largely negates the value of the Auto-Complete history feature, because your most-common history completions will be mixed in with everything else instead of being in a separate category at the top of the list.

- **Show parameters** - (Pro only) If selected, shows the function parameter signatures for symbol completions. If a function is overloaded, it will show all the overloaded signatures once the list of functions is sufficiently narrowed down. When a specific signature is selected and completed, if enabled, you will be put directly into function argument help for that function signature.
- **Symbol declaration** - (Pro only) If selected, for symbol completions, this will show the symbol declaration as a comment to the right of the symbol completion.
- **Show comments** - (Pro only) If selected, for symbol completions, the comments are displayed for the currently selected symbol in the list displayed by Auto-Complete. 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+PgUp** and **Ctrl+PgDn** to cycle through the comment sets. Click on the blue arrow to jump to the symbol displayed. Use the key

combinations of **Shift+Left** and **Shift+Right** to move the comment to the left or right of the list provided there is enough space to display it there.

- **Auto-Complete Options** - The following Auto-Complete options pertain to how you use the system to select a completion and what happens when you select a completion.
 - **Insert open parenthesis for functions** - 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.
 - **Tab inserts longest unique prefix** - If selected, pressing **Tab** will cause Auto-Complete 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. If this option is not selected, use the similar option for **Space**, or use **Ctrl+Space** when Auto-Complete is displayed to perform symbol completion.
 - **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 cleared, **Tab** will attempt to insert the longest unique prefix (if selected), or insert the selected completion, or cancel Auto-Complete and behave normally if there is no completion selected.
 - **Space inserts longest unique prefix** - If selected, pressing the spacebar when Auto-Complete 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 the similar option for **Tab**, or use **Ctrl+Space** when Auto-Complete is displayed. Note that pressing the spacebar when there is no item selected in the list will simply insert a space.
 - **Space always inserts space** - If selected, pressing the spacebar when Auto-Complete is displayed will insert the current item and a space in the list after the current item. If this option is not selected, pressing the spacebar will only insert the current item with no extra space. Note that pressing the spacebar when there is no item selected in the list will simply insert a space.
 - **Enter always inserts item** - If selected, pressing the Enter key will insert the current item.
 - **Minimum prefix length** - The minimum number of characters the word prefix must contain before auto-completions will be displayed automatically.
 - **Completion choice** - When set to **Automatically choose unique completion**, if Auto-Complete finds exactly one word match, it will automatically select that match for completion. If **Insert current completion in file** is selected, then completions selected from Auto-Complete will replace the current text, modifying the file as you work. Choose **Manually choose completion** to select and insert the completion manually.
 - **Preserve identifier to right of cursor** - When set to **Preserve always**, only the identifier characters before the cursor are replaced with an item selected from an Auto-Complete list, while identifier characters after the cursor are preserved. When this option is set to **Replace entire identifier**, identifier characters following the cursor are replaced with the item selected from an Auto-Complete list. When this option is set to **Preserve for auto list members only**, trailing identifier characters are preserved for auto list members but not when listing symbols on demand by pressing **Alt+Dot** to

invoke the **list-symbols** command.

For example, if List Members is active and the current line is as follows:

```
this->foo<cursor_here>Bar
```

Then if this option is set to **Preserve always** and you choose a symbol named "foodForThought" from the Auto-Complete list, the line will be changed to:

```
this->foodForThought<cursor here>Bar
```

If this option is set to **Replace entire identifier**, doing the same would result in:

```
this->foodForThoughtBar<cursor here>
```

- **List include files after typing #include** - (Pro only) When editing in certain languages that use **#include**, Auto-Complete can generate a list of possible files for you. To view a list of quoted files after typing **#include** followed by a space, set this option to **List quoted files after typing #include**. An empty pair of quotes will be inserted by default. If you prefer to use < and > to specify the include file path, just type < inside the quotes and the **#include** will be converted to that format. To see a list of files after typing " or <, select **List files after typing " or <**. If you do not wish to see a list of possible files, select **Do not list include files**. To access this feature on demand, press **Alt+Dot**.
- **List-symbols options** - (Pro only) The following options apply to List Symbols. See [List Members](#) for more information.
 - **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** to invoke the **list-symbols** command. If you use this feature on demand, and you are not in a member expression, this feature will display a list of all completions available in the current scope, depending on what is enabled. By default you should see locals, current class members, symbols from the current file, global symbols, keywords, syntax expansion, and word completions.
 - **Auto-list compatible values** - If selected, compatible variables are automatically listed after you press the spacebar 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**.
- **Use strict case-sensitivity rules** - The following options allow you to control whether or not Auto-Complete, List Symbols, and Symbol Completion with respect to whether it searches strictly for language-defined exact-case symbol matches, or also for case-insensitive symbol matches.

Note

Enabling this feature can make symbol completion more concise, but it can also make it much less convenient, because you will need to type the symbol case correctly.

- **Use strict case-sensitivity rules** - The strict case-sensitivity options pertain to whether or not symbol completions are searched for using a case-insensitive search, even for case-sensitive languages. When not using the strict option, SlickEdit will first search for exact-case matches, then case-insensitive symbol matches.
- **For auto-complete** - If selected, when auto-complete is automatically displayed, it will attempt to find symbols using an exact-case search.
- **For list-members / list-symbols** - If selected, when auto-list members is displayed, or invoked manually using **Alt+Dot** to invoke the **list_symbols(Pro only)** command, it will attempt to find symbols in the current context matching the identifier under the cursor using an exact-case search. See [List Members](#) for more information.
- **For manual symbol completions** - If selected, when symbol completion is manually invoked using **Ctrl+Space** to invoke the **codehelp_complete(Pro only)** command, it will attempt to find symbols in the current context matching the identifier under the cursor using an exact-case search. See [Completions](#) for more information.
- **Use subword matching rules** - (Pro only) The subword matching options pertain to when and how SlickEdit will attempt to find matches using subword patterns for the symbol under the cursor.

Note

This feature can be rather intensive for large code bases, and will tend to time out before finding *all* of the possible subword matches.

The algorithm is optimized to perform better if the first character of the search pattern matches the first character of the symbol you are trying to complete. For example, if you are trying to complete a symbol named `getLengthOfQueue`, you could potentially use a pattern like `leng`, however, the lookup will be quicker if you use a pattern such as `glq` where the first character matches, thus narrowing the search space.

Note

Subword matching is only enabled when the identifier prefix under the cursor is at least one character longer than the **Minimum prefix length**. See [Minimum prefix length](#) for more information.

- **For auto-complete** - If selected, when auto-complete is automatically displayed, it will attempt to find symbols in the current context matching the identifier under the cursor interpreted as a subword

pattern. Enabling this feature for auto-complete can both impact performance and make auto-complete somewhat less concise because for a given pattern, it might find more matches than you would expect to have.

- **Exclude globals for auto-complete** - For performance, limit subword matching for auto-complete to local variables, the current file, and class members.
- **For list-members / list-symbols** - If selected, when auto-list members is displayed, or invoked manually using **Alt+Dot** to invoke the **list_symbols(Pro only)** command, it will attempt to find symbols in the current context matching the identifier under the cursor interpreted as a subword pattern. See [List Members](#) for more information.
- **For manual symbol completions** - If selected, when symbol completion is manually invoked using **Ctrl+Space** to invoke the **codehelp_complete(Pro only)** command, it will attempt to find symbols in the current context matching the identifier under the cursor interpreted as a subword pattern. See [Completions](#) for more information.
- **First attempt uses prefix match only** - If selected, the first time symbol completion is manually invoked using **Ctrl+Space** to invoke the **codehelp_complete(Pro only)** command, it will only do a quick prefix match. The second and subsequent times you directly invoke symbol completion, it will search for subword pattern matches.

This option exists for performance, because subword matching can be expensive, you might not want it to happen every time. This provides a very convenient way to tell SlickEdit to try harder to find a symbol only when you need to.

- **Subword matching strategy** - The following options are available:

Note

Subword boundaries are defined by camel case transitions and underscore or dash connectors. Leading upper or lower cases prefixes, up to four characters, are also considered as subword boundaries. This exception allows for better handling of Polish notation identifiers and other prefix-based naming conventions.

Example: - The subwords of `MYTwinFallsIdahoTelescope` are M, Y, Twin, Falls, Idaho, and Telescope.

Example: - The subwords of `THE_CAT_IN_THE_HAT` are THE, CAT, IN, THE, and HAT.

- **Stone-skipping with subword boundaries** - Stone-skipping matches the pattern left-to-right skipping characters which do not match. All of the characters in the pattern must be matched. This strategy adds the requirement that when groups of characters are matched, they must start on a subword boundary.

Example: - Given the symbol `TwinFallsIdahoTelescope`, the patterns: falltele matches the symbol, hotel does not match the symbol because it does not start at a word boundary, tfit matches the symbol as an acronym, and twinturbo does not match.

- **Acronyms using subword boundaries** - This strategy allows only the first character at a word boundary to be matched against the pattern. It can be useful, but it precludes matching whole subwords, and is thus the most restrictive of all the strategies.

Example: - Given the symbol TwinFallsIdahoTelescope, the patterns: tfit matches the symbol, hotel does not match the symbol as an acronym, and turbo does not match.

- **Pure stone-skipping** - Stone-skipping matches the pattern left-to-right skipping characters which do not match. All of the characters in the pattern must be matched. Pure stone skipping uses this algorithm with no additional requirements on word boundaries. This is a very general strategy, in fact, all of the other strategies can be characterized as pure stone skipping with additional requirements. Like substring matching, it has the disadvantage of frequently finding matches which are not intended.

Example: - Given the symbol TwinFallsIdahoTelescope, the patterns: falltele matches the symbol, hotel matches the symbol as a substring, hotelscope matches the symbol, tfit matches the symbol as an acronym, twinturbo does not match.

- **Character matching in any order** - Simply determine if the symbol contains all the characters that are in the pattern, in any order, with no word boundaries. If the pattern repeats a character, it must repeat at least as many times in the symbol name in order to match.

Example: - Given the symbol TwinFallsIdahoTelescope, the patterns: twin matches the symbol, twinfall matches the symbol, falltwin matches the symbol, twinfallo matches the symbol, twinfallooo does not match (too many o's) hotel matches the symbol, and twinturbo does not match.

- **Simple substring matching** - Simple substring strategy is self-explanatory, however, despite being easy to understand, it has the disadvantage that it can find matches that are overlap subword boundaries.

Example: - Given the symbol TwinFallsIdahoTelescope, the patterns: hotel matches the symbol, falltele does not match the symbol, and twinturbo does not match.

- **Subword matching** - Subword matching is equivalent to a substring match with the additional requirement that the match starts on a subword boundary.

Example: - Given the symbol TwinFallsIdahoTelescope, the patterns: tele matches the symbol, falltele does not match the symbol because the subword matches are not contiguous, hotel does not match the symbol because it does not start on a subword boundary, and twinturbo does not match.

- **Prefix matching only** - This option is equivalent to turning off subword matching, because Context Tagging with SlickEdit does prefix matching by default.

Example: - Given the symbol TwinFallsIdahoTelescope, the patterns: twin matches the symbol, twinfall matches the symbol, hotel does not match the symbol, and twinturbo does not match.

- **Relax pattern matching order constraints** - Relax pattern matching order constraints to allow for the possibility that the pattern could have one or two subwords or characters out of order.

This option only applies to **Stone-skipping with subword boundaries**, **Acronyms using subword boundaries**, and **Pure stone-skipping**.

Example: - Using the **Stone-skipping with subword boundaries** technique, given the symbol TwinFallsIdahoTelescope, both patterns twinfall and falltwin match the symbol.

- **Start matching globals with first char** - If selected, when searching for all global matches to a subword pattern, start by searching for symbols that begin with the first character of the subword pattern. If this search yields matches, then stop searching.

This option exists for performance. Subword matching can be expensive, especially when you are searching through a large set of global symbols. By narrowing down the search space to just global symbols starting with the first character of the pattern, on average, we can find matches 20 times faster. It also yields slightly more focused matches. The disadvantage of this option is that it can get in the way of finding symbols that *do not* match the first character, but would otherwise be legitimate pattern matches.

When this option is not selected, the first character search strategy is still employed for globals, however, the search will continue to search for all subword pattern matches.

- **Limit to current workspace** - Limit subword pattern matching to the current workspace. This option exists both for performance and to narrow down the results to get more focused and relevant matches.
- **Include auto-updated tag files** - If selected, include the workspace's auto-updated tag files in the subword pattern matching search.
- **Include compiler tag files** - If selected, include compiler tag file in the subword pattern matching search.
- **Find completions which fix minor typos** - When doing symbol pattern matching, also consider looking for symbols and keywords which are corrections to minor typographical errors in the symbol under the cursor, such as fixing transposed characters, inserting a missing character, replacing a single mistyped character, or correcting a repeated character. See [Completions](#) for more information.

It does this by finding the symbols visible in the current context, and matching each one against the symbol under the cursor. If only one unique corrected symbol matches, symbol completion will replace the symbol with the corrected version.

Note

When using **Stone-skipping with subword boundaries**, **Pure stone-skipping**, or **Acronyms using subword boundaries**, you can use the underscore or dash characters (_ or -, respectively), as a tying connector to indicate that the pattern (or acronym) matches need to match adjacent subwords.

For example, the pattern `gsf_to` could be used to match the symbol `GetSnowFallTotal` without matching symbols with letters between the subword matching `f` (`Fall`) and the subword matching `t` (`Total`).

For example, the symbol `GetSouthernFriedTaterTots` would not match the pattern, because `Tater` does not match as an adjacent subword.

This is an advanced technique, but can be helpful when you know exactly the symbol you want and just want to type fewer characters before completing it.

This technique only works for languages where underscore or dash are valid identifier characters.

Language-Specific Auto-Close

These options let you configure Auto-Close for a specific language. Auto-Close inserts matching closing punctuation when opening punctuation is entered. For example, when you type an open parenthesis, Auto-Close automatically inserts the closing parenthesis right next to it.

The settings on this page depend on the selected language. As an example, the C/C++ Auto-Close options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Auto-Close**).

C/C++ > Auto-Close

Enable Auto-Close:

<input checked="" type="checkbox"/> Parenthesis ()	<input type="checkbox"/> Insert padding
<input checked="" type="checkbox"/> Bracket []	<input type="checkbox"/> Insert padding
<input checked="" type="checkbox"/> Angle Bracket <>	<input type="checkbox"/> Insert padding
<input checked="" type="checkbox"/> Double Quote ""	
<input checked="" type="checkbox"/> Single Quote	
<input checked="" type="checkbox"/> Brace {}	

Put closing brace

On same line
 On next line
 After blank line

Quick brace/unbrace statements

[Configure completion \(Enter, Tab\)](#)

[Configure automatic closing of block comments](#)

The following options are available:

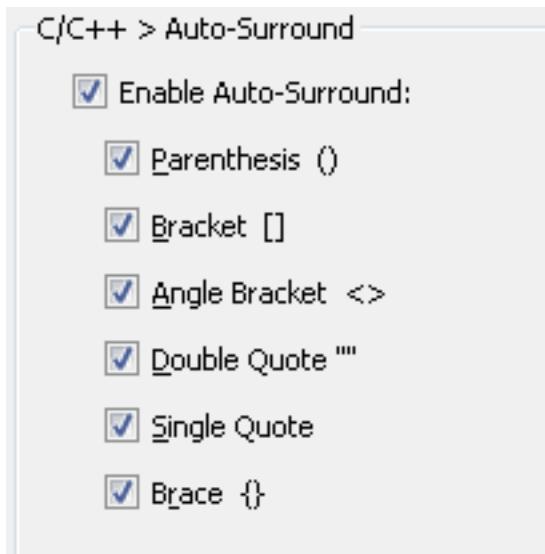
- **Enable Auto-Close** - When set to **On**, the punctuation items selected in the additional checkboxes will Auto-Close. To turn off Auto-Close for all punctuation, set this value to **Off**.
 - **Parenthesis ()** - When set to **On**, Auto-Close will automatically insert a closing parenthesis when an open parenthesis is entered.
 - **Insert padding** - To insert spaces between the parentheses, set this value to **On**.
 - **Bracket []** - When set to **On**, Auto-Close will automatically insert a closing bracket when an open bracket is entered.
 - **Insert padding** - To insert spaces between the brackets, set this value to **On**.
 - **Angle Bracket <>** - When set to **On**, Auto-Close will automatically insert a closing angle bracket when an open angle bracket is entered.
 - **Insert padding** - To insert spaces between the angle brackets, set this value to **On**.
 - **Double Quote ""** - When set to **On**, Auto-Close will automatically insert a closing double quote when an open double quote is entered.

- **Single Quote "** - When set to **On**, Auto-Close will automatically insert a closing single quote when an open single quote is entered.
- **Brace {}** - When set to **On**, Auto-Close will automatically insert a closing curly brace when an open curly brace is entered.
- **Put closing brace** - Specifies where to put the closing brace when auto-closing braces. Possible values are:
 - **On same line** - puts the closing brace on the same line as the opening brace.
 - **On next line** - puts the closing brace on the next line after the opening brace.
 - **After blank line** - inserts a blank line between the opening brace and the closing brace.
- **Quick brace/unbrace statements** - When this feature is enabled, you can convert a single-line statement to a brace-enclosed block and vice versa. For languages without beautifiers, this option is found on the Formatting page.
- **Configure completion (Enter, Tab)** - This link takes you to [Auto-Close](#) so that you can configure completion keys.
- **Configure automatic closing of block comments** - This link takes you to [Language-Specific Comment Options](#) so that you can configure the automatic closing of block comments.

Language-Specific Auto-Surround

These options let you configure Auto-Surround for a specific language. Auto-Surround surrounds the current selection with typed bracketed and quotation pairs. For example, when you type an open parenthesis, Auto-Surround will insert the open parenthesis at the start of the selection and the close parenthesis at the end of the selection.

The settings on this page depend on the selected language. As an example, the C/C++ Auto-Surround options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Auto-Surround**).



The following options are available:

- **Enable Auto-Surround** - When set to **On**, the punctuation items selected in the additional checkboxes will Auto-Surround. To turn off Auto-Surround for all punctuation, set this value to **Off**.
 - **Parenthesis ()** - When set to **On**, Auto-Surround will support parenthesis.
 - **Bracket []** - When set to **On**, Auto-Surround will support square brackets
 - **Angle Bracket <>** - When set to **On**, Auto-Surround will support angle brackets
 - **Double Quote ""** - When set to **On**, Auto-Surround will support double quotes
 - **Single Quote** - When set to **On**, Auto-Surround will support single quotes
 - **Brace {}** - When set to **On**, Auto-Surround will support single quotes

Language-Specific Context Tagging® Options (Pro only)

These options let you configure language-specific settings for Context Tagging (see [Context Tagging Features](#)). Note that global Context Tagging options are located at **Tools** → **Options** → **Editing** → **Context Tagging** (see [Context Tagging® Options](#)).

The settings on this page depend on the selected language. As an example, the C/C++ Context Tagging options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **Context Tagging**).

C/C++ > Context Taqqinq®

Parameter information

- Auto-display parameter information
- Show comments
- Auto-insert matching parameter
- Auto-list compatible parameters
- Pad parentheses
- Insert space after comma
- Insert keyword before parameter when required

Go to Definition

Prioritize navigation to:

Prompt with all choices

When there are multiple choices:

Prompt with all choices

- Prioritize navigation to symbols in the current project
- Ignore forward class declarations
- Use strict case-sensitivity rules
- Attempt to filter out non-matching function overloads (expensive and slow)

Preview and Highlighting

- Show info for symbol under mouse
 - Show comments
 - Evaluate and show return type
- Highlight matching symbols under cursor
- Show preview for symbol under cursor
 - Show comments
 - Evaluate and show return type
- Show statements in the Defs tool window

[Configure list members completion](#)

Parameter Information

The following options control the lookup of parameter information. See [Parameter Information](#) for more details.

- **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 the current argument is highlighted 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+PgUp** and **Ctrl+PgDn** to cycle through the comment sets.
- **Auto-insert matching parameter** - If selected, when 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 **Comma**, **Space**, **Tab**, or a closing 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**. See [Auto List Compatible Parameters](#) for more information.
- **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)**.
- **Insert keyword before parameter when required** - If selected, for languages that require a keyword before function parameters to indicate calling convention (for example, C# **ref** and **out** keywords), the keyword will be inserted automatically when a parameter name is automatically inserted, such as **myfun(ref a, out b, c)**.

Go to Definition

These options control the behavior when you navigate from a symbol to its definition or declaration. You can do this by selecting **Search → Go to Definition** or **Search → Go to Declaration**, selecting **Go to Definition** or selecting **Go to Declaration** from the context menu, pressing **Ctrl+Dot** or **Ctrl+Alt+Dot** in CUA emulation, or by executing the **push_tag** or **push_altag** command from the SlickEdit command line. See [Symbol Navigation](#) for more information.

- **Prioritize navigation to** - Here you can specify if you prefer to navigate directly to a symbol's definition (proc) or declaration (proto). If **Prompt** is selected, the [Select Symbol Dialog](#) is displayed, prompting you for both definitions and declarations.

The **Go to Definition** command, invoked by pressing **Ctrl+Dot**, will honor the navigation settings precisely. However, its counterpart, **Go to Declaration**, invoked by pressing **Ctrl+Alt+Dot**, utilizes the opposite setting for navigation priority. That is, if you specify to prioritize navigation to jump directly to a symbol's definition (proc), **Go to Declaration** will prioritize navigation to jump directly to the symbol's declaration (prototype).

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 back to the original location, where you were before you pressed **Ctrl+Dot**. The same applies for **Go to Declaration**.

invoked by pressing **Ctrl+Alt+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. See [Symbol Navigation](#) for more information about navigating through your code.

- **Prioritize navigation to symbols in the current project** - When this option is enabled, **Go to Definition** will navigate directly to a symbol in the current project if there is a unique match. If there is no unique match within the current project, you will be prompted with all choices, with the matches in the current project, directory and workspace closer to the top of the list. The current project is the project that the current source file belongs to, or the active project.
- **Ignore forward class declarations** - When this option is enabled, **Go to Definition** filters out forward class declarations, and only shows the actual class definitions. Note that **Go to Declaration** does will always show both class definitions and forward declarations.
- **Use strict case-sensitivity rules** - when selected, factors upper/lowercase letters as part of the matching criteria.
- **Attempt to filter out non-matching function overloads (expensive and slow)** - when selected, SlickEdit attempts to filter out functions with the same name but with different function signatures. As stated, this can slow down the matching.
- **For virtual functions, find all subclasses where the method is overridden** - When this option is enabled, **Go to Definition** for a virtual function will attempt to find not just the function in it's apparent class, but also all instances where the function is overridden in derived classes, as well as declarations and definitions found in parent classes.

Preview and Highlighting

- **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.
- **Show comments** - If selected, comments are displayed when mouse-over information is displayed. When a symbol has multiple definitions, and multiple sets of comments, all the comments will be displayed separated by horizontal lines.
- **Evaluate and show return type** - If selected, the symbol's return type is evaluated and fully qualified then displayed along with the symbol declaration. This option is helpful when working with languages that allow you to declare type-inferred variables, for example, using "auto" in C++, or ":=" in Slick-C or Google Go. It is also helpful when the symbol's apparent return type is an imported symbol.

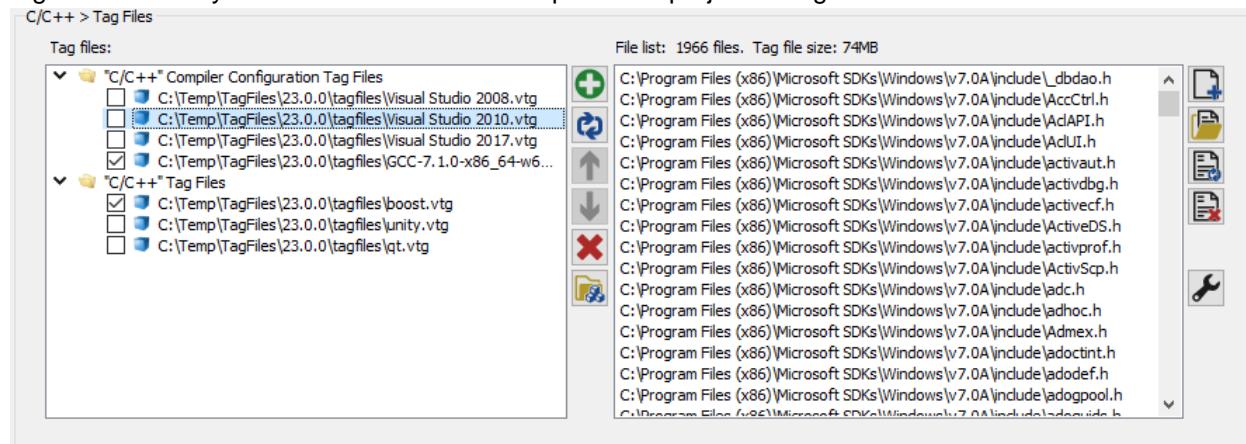
- **Highlight matching symbols under cursor** - When selected, all occurrences of the current symbol under the cursor in the buffer are highlighted. The highlight color is controlled by the **Symbol Highlight** screen element (**Tools** → **Options** → **Appearance** → **Colors**). Advanced configuration options are available. See [Cursor on Symbol Shows All Uses in File](#) for more information.
- **Show preview for symbol under cursor** - When selected, if the Preview tool window is active, the information and code preview for the symbol under the cursor is displayed.
 - **Show comments** - If selected, comments are displayed in the Preview tool window, when a symbol is displayed. This option can be turned off in order to improve performance if you really don't want to see the symbol comments. In addition, you can collapse the comment pane in the Preview tool window so they are not visible.
 - **Evaluate and show return type** - If selected, the symbol's return type is evaluated and fully qualified then displayed along with the symbol comments. This option is helpful when working with languages that allow you to declare type-inferred variables, for example, using "auto" in C++, or ":=" in Slick-C or Google Go. It is also helpful when the symbol's apparent return type is an imported symbol.
- **Show statements in the Defs tool window** - 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 and all other files in the current language mode. This allows you to see a primitive function flowchart or to navigate to a specific statement within a function. Note that statement-level tagging is not supported for all languages.

Language-Specific Tag Files(Pro only)

(Pro only) Displays the Context Tagging® - Tag Files dialog, shown below, to manage all the tag files for the current language mode. This includes both language-specific and compiler-specific tag files, where applicable. For more information, see [Context Tagging - Tag Files Dialog](#). For more information about tag files, see [Building and Managing Tag Files](#).

Language-specific tag files have checkboxes which can be turned off in order to keep a tag file in the list, but ignore it. This makes it easy to de-activate and re-activate a tag file that you need on occasion, but do not want to always use for the current language.

Compiler-specific tag files also have checkboxes, but they are read-only, and used only to indicate which tag file is currently active for the current workspace and project configuration.



Language-Specific Color Coding Options

Color Coding is a feature that displays various portions of code in different colors for improved readability. The recognized syntactic elements (like keywords, comments, strings, etc) are determined by the Color Coding settings defined here. Each syntactic element has a "Type" (short for color element type) which is one of the supported color element types. The actual color displayed for each color element type is determined by the color settings (See [Color Options](#)).

SlickEdit has a very powerful color coding engine which allows any user to add color coding for almost any language. For handling more complex syntactic elements, regular expressions may be used. When the start delimiter is a regular expression, tagged expressions can be used for the end delimiter and embedded language. Also, there is built in support for color coding numbers so most or all number constructs can be added without defining any regular expressions. For convenience, there is an xml literal check box on the Language Tab. This is useful for languages like Scala, Visual Basic, and Action Script which support xml literals.

The settings on the Color Coding options page depend on the selected language. The C/C++ Color Coding options (**Tools → Options → Languages → Application Languages → C/C++ → Color Coding**) are used in the screen shots for this section.

The Color Coding options page contains the following uncategorized options:

- **Profile name** - Select the language color coding profile to use from the **Profile** drop-down list. This sets the active profile for that language.
- **New** - Click this button, located next to **Profile**, to prompt for a profile name to start a new language-specific color coding definition (see [Creating Color Coding for a New Language](#)).
- **Delete** - Click this button, located next to **Profile**, to remove a profile from the list. You can only delete user-created color coding profiles.
- **Reset** - Click this button, located next to **Profile**, to reset a built-in color coding profile to its default settings. You can only reset modified builtin color coding profiles.
- **Import** - Click this button to import color coding profiles from a .cfg.xml or the older .vlx file.
- **Colors** - Click this button to jump to the **Tools → Options → Appearance → Colors** option screen where you can specify the colors used. Click the **Back** button on the Options dialog to return to the Color Coding Language tab. See [Setting Colors for Screen Elements](#) for more information.

Other options are categorized into the following tabs:

- [Color Coding General Tab](#)
- [Color Coding Tokens Tab](#)
 - [Color Coding Settings Tab](#)
 - [Color Coding More Tab](#)
 - [Color Coding Embedded Tab](#)

- [Color Coding Numbers Tab](#)
- [Color Coding Language Tab](#)
- [Color Coding Tags Tab](#)

How To's:

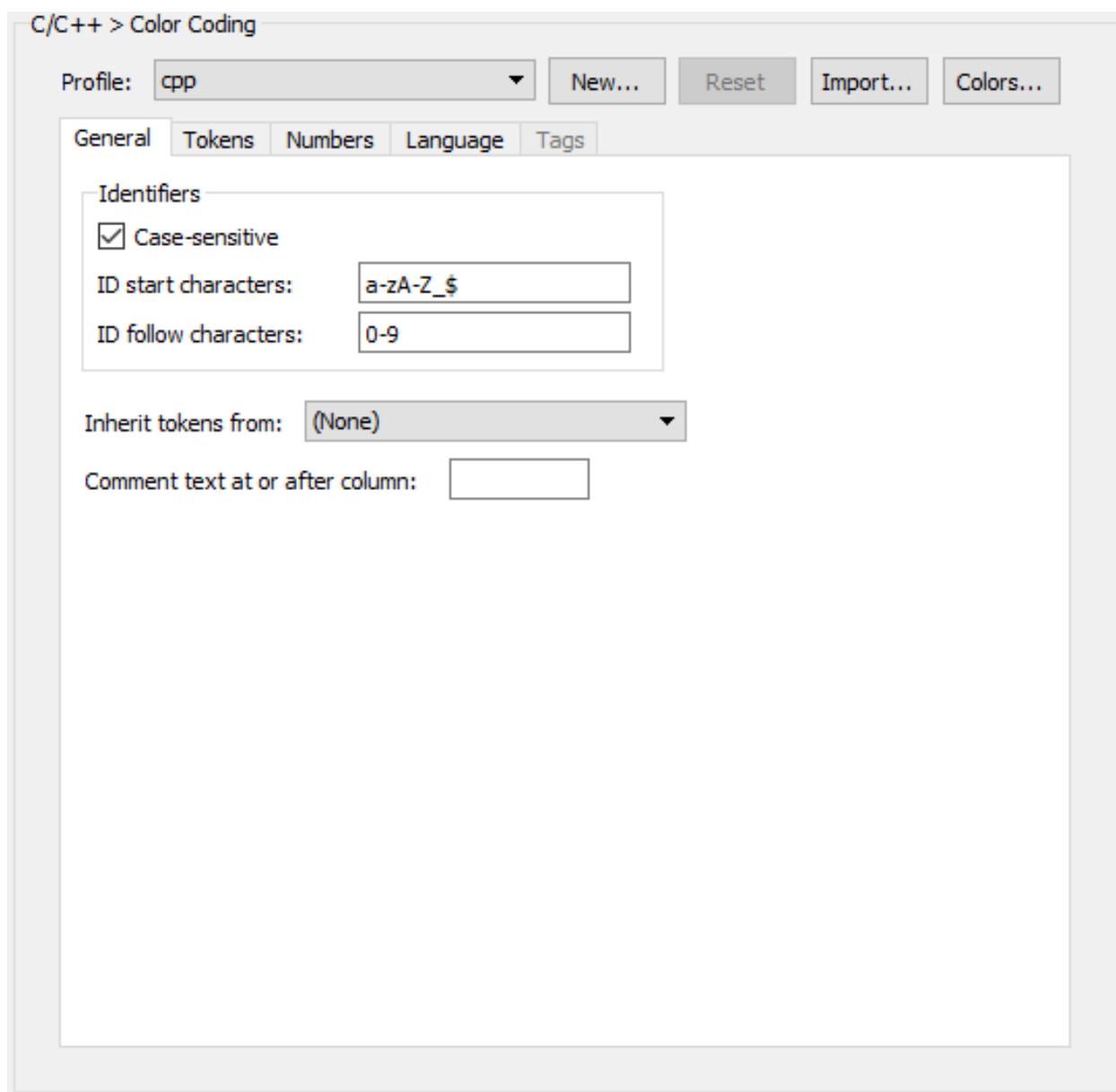
- [Creating Color Coding for a New Language](#)
- [How to add new color coding words \(keywords, library symbols, operators, punctuation etc.\)](#)
- [How to add a line comment](#)
- [How to add a multi-line comment](#)
- [How to add a string](#)
- [How to define color coding for numbers](#)
- [How to add interpolation to a string](#)

If you want to use a regular expression to match the start and/or end delimiter, see [Tips on using regular expressions matching in color coding](#) for more information

Color Coding General Tab

The **General** tab defines the syntax for an identifier and whether identifiers are case sensitive. For most languages, the settings here are sufficient. Recognizing identifiers (and operator identifiers) for Scala required some very complicated regular expressions in addition to some settings here.

Language Options



The following options are available:

- **Identifiers** - All Context Tagging® operations use this set of characters to find identifiers in the code that is being analyzed. Therefore, it is important to set the start and follow characters in a manner that is consistent with the language specification. In most languages, identifiers can contain digits, but they cannot start with them. For example, in C/C++, start characters are "a-zA-Z_\$" and follow characters are "0-9".
 - **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.

- **Inherit tokens from** - Specifies a color coding profile to inherit all tokens from. Inherited tokens are processed before tokens for the current profile. That allows tokens in the current profile to more easily override inherited tokens. Only tokens are inherited and no other settings like Numbers Tab settings or other tab settings.
- **Comment text at or after column** - Specifies that characters at or after the column specified should be colored as a comment. The first column is 1. The column is currently specified in bytes and is intended for some mainframe languages. This field is typically blank which means it has no effect.

Color Coding Tokens Tab

The **Tokens** tab allows you to define the syntactic elements like keywords, strings, and comments etc. Simple syntactic elements like keywords specify a plain text string **Start delimiter** to match and a color element **Type**. Syntactic elements like strings and multi-line comments typically require a **Start delimiter**, **End delimiter**, and color element **Type**. There are many additional options that can be applied to syntactic elements defined here including case sensitivity, regular expression searching, and more.

Language Options

C/C++ > Color Coding

Profile: **cpp**

General **Tokens** **Numbers** **Language** **Tags**

Type	Start	End
Comment	/*	*/
Comment	//	
Doc Comment	/*!	*/
Doc Comment	/**	*/
Doc Comment	//!	
Doc Comment	///	
Keyword	@autoreleasepool	
Keyword	@catch	
Keyword	@class	
Keyword	@defs	

Settings **More** **Embedded**

Type: **Comment** Case sensitive
Start delimiter: **/*** Plain text search
End delimiter: ***/** Plain text search
Color to end of line: **(None)**
End color to end of line: **(None)**
When end delimiter is not on same line: **Color to end across multiple lines**
Two consecutive quotes represent one. Doubles char:
Escape char: Line continuation char:

The buttons to the right of the tree list are the following:

- **Add Words...** - Allows you to add one or more **Start delimiter** words of a specified color element type and case sensitivity at the same level in the tree. Words are specified with a space delimited list of words. Words with spaces can be double quoted. Use the **Add Other** or **Add Sub Item** buttons to add words which contain double quotes. The **Add Other** and **Add Sub Item** buttons also allow you to add simple words like the **Add Words...** button but items can only be added one at a time.
- **Add Other** - Adds a new mostly blank syntactic element at the same level in the tree. The Type is initially set to "Comment" but you can change it to something else.
- **Add Sub Item** - Adds a new mostly blank syntactic element inside the current item in the tree. The Type is initially set to "Comment" but you can change it to something else. It only makes sense to put an element "inside" another when the parent item has a **Start delimiter** and a **End delimiter** or is

colored past the **Start delimiter**. Sub items are only colored when found inside the parent item.

- **Delete** - Deletes the current item in the tree and its children if it has any.
- **Import Word List...** - Allows you to add one or more **Start delimiter** words of a specified color element type and case sensitivity at the same level in the tree. Each line in the file is a space delimited list of words. Words with spaces can be double quoted. Use the **Add Other** or **Add Sub Item** buttons to add words which contain double quotes.

Color Coding Settings Tab

The **Settings** tab provides the most common settings.

This tab contains the following

- **Type** - The primary color element type for the item. This sets the default start delimiter color, continuation color (**Color to end of line**), and the end delimiter color. The start delimiter and end delimiter color can be changed by setting the **Start Color** and/or **End Color** on the More Tab respectively. The default start and end delimiter color for String types is done with some special built-in logic for improved and standardized appearance. There are common cases where only the continuation color is the same color as the **Type** color and the start and end delimiters are colored as keywords. Note that some smart editing features scan the color coding profiles for specific types (like comments that are multi-line).
- **Start delimiter** - Plain text or regular expression string to search for and color. This field is never blank. When this delimiter is not a regular expression and starts with a valid identifier character, the previous character may not be an identifier character. Likewise, when this delimiter is not a regular expression and ends with a valid identifier character, the next character may not be an identifier character. There are rare cases where this automatic extra identifier word break logic isn't what you want. Use a regular expression to avoid automatic identifier word break checking. See [Perl Regular Expressions](#), [Vim Regular Expressions](#) or [SlickEdit® Regular Expressions](#) for information on regular expression syntax. Also see [Tips on using regular expressions matching in color coding](#) for more information
- **End delimiter** - Plain text or regular expression string to search for and color after the **Start delimiter** is found. This field is often blank except for begin/end constructs. When this delimiter is not a regular expression and starts with a valid identifier character, the previous character may not be an identifier character. There are rare cases where this automatic extra identifier word break logic isn't what you want. Use a regular expression to avoid automatic identifier word break checking.

IMPORTANT: When the **Start delimiter** is a regular expression, tagged expressions and escapes are processed in the **End delimiter** even if the end delimiter is not a regular expression. When the **Start delimiter** and **End delimiter** are both regular expressions, things get complicated. First, tagged expressions and escapes are processed. Then the result is compiled as a regular expressions. This means you may need to escape a literal character twice (ex instead of just "\\" you need "\\\""). Note that When the tagged expressions are replaced, special characters are escaped so that the tagged expression replacements are considered literal text. See [Section Replacing with Regular Expressions](#) for information on tagged expressions and special characters in the replace string. Also see [Tips on using regular expressions matching in color coding](#) for more information

- **Color to end of line** - This sets the continuation color or color to end of line color for constructs which color past the **Start delimiter**. When this is set to **(None)** the **Type** color is used.
- **End color to end of line** - When the **End delimiter** is non-blank, this optionally specifies the color to use to color to the end of line. This is useful for begin/end constructs where once the end delimiter is found, the rest of the line is ignored and colored as a comment.
- **Color to end of file** - Only visible when the **End delimiter** is blank. Specifies whether to continue coloring after the **Start delimiter** to the end of file. Color used is determined by the **Color to end of line** if set, otherwise the **Type** is used.
- **When end delimiter is not on same line** - Only visible when **End delimiter** is non-blank. One of the following settings:
 - **Color to end of line** - Color to the end of line when **End delimiter** is not found not found on the same line as the **Start delimiter**.
 - **Color to the end across multiple lines** - Continue coloring until the **End delimiter** is found. Coloring will continue even if the **End delimiter** is not found.
 - **Color start as Other color** - When specified, text is colored between start delimiter and end delimiter only if end delimiter is found on the same line as the start delimiter. Otherwise, the start delimiter is color as Other color.
- **Two consecutive quotes represent one. Doubles char** - Only available when **End delimiter** is non-blank or **Color to end of file** is checked. Specifies that two consecutive quotes (or character specified) represents one. This is often necessary for begin/end string constructs where a correct search for the end quote which is the first character in **End delimiter** requires skipping consecutive quotes.
- **Escape char** - Only available when **End delimiter** is non-blank or **Color to end of file** is checked. Allows you to define the escape character where the next character is skipped so the **End delimiter** can be correctly found. Many String type constructs support escaping with a character like backslash (some languages uses a different escape character).
- **Line continuation char** - Only available for single line constructs. When this character is found at the end of the line, coloring will continue to the next line.

Color Coding More Tab

The **More** tab provides less frequently used settings.

This tab contains the following

- **Nesting allowed** - When checked, allows the nest with start and end to be defined. This is useful for defining constructs like multi-line block comments which support nesting. Set the **Nest with start** and **Nest with end** to specify nested delimiters.
- **Match start delimiter only if first non-blank character in line** - While you could use a regular expression that starts with `^[\t]*` to match beginning of the line followed by blanks, checking this is easier. There is a subtle difference with the **Order** of evaluation. Items with **Match start delimiter only if first non-blank character in line** or **Check for start delimiter first** checked get processed before

items with either of these options checked.

- **Check for start delimiter first** - When selected, checks for the **Start delimiter** before looking for other items. This is typically used only when a **Start delimiter column** is specified.
- **Start color** - Overrides the default start delimiter color. When not set to **(None)**, start delimiter is colored with this color.
- **Start substring colors** - Allows you to specified one or more colors that override parts of the **Start color**.
- **End color** - Overrides the default end delimiter color. When not set to **(None)**, end delimiter is colored with this color.
- **End substring colors** - Allows you to specified one or more colors that override parts of the **Start color**.
- **Start delimiter column** - Specifies the columns in which the **Start delimiter** is considered a match. Specify a begin and end column to set a range of columns. Leave the end column blank, to specify that the start delimiter is recognized anywhere after the start column.
- **End delimiter column** - Specifies the columns in which the **End delimiter** is considered a match. Specify a begin and end column to set a range of columns. Leave the end column blank, to specify that the end delimiter is recognized anywhere after the start column.
- **Order** - This is a signed integer which determines the order of evaluation of items. In more complicated scenarios where there are multiple **Start delimiter** patterns matching the same text, this is used to choose which match gets processed. Lower values are matched first and take precedence. Sometimes a regular expression which is intended to match a longer pattern also exactly matches a shorter duplicate pattern.
- **Repeat after** - Specifies that this item is checked for a match after the parent item (start and end) is matched. The purpose of this option is to make it easier to color code a sequence which repeats over and over where substring coloring doesn't work that well.
- **Enabled** - When checked, this item is applied. When unchecked, has a similar effect to the item being deleted.

Color Coding Embedded Tab

The **Embedded** tab provides options for embedded language support. While handling of embedded XML literals is a built-in, most other embedded language constructs uses these options.

This tab contains the following

- **Embedded profile** - When non-blank, indicates this item is an embedded language even if it doesn't match an existing color coding profile. When the **Start delimiter** is a regular expression, replacements for tagged expressions and escapes just like a typical search and replace will be performed. See [Section Replacing with Regular Expressions](#) for information on tagged expressions and special characters in the replace string.

- **Prefix match embedded profile** - When checked, the prefix of embedded profile name is matched against other color coding profiles (ex "cppEOF" would match "cpp").
- **Embedded end delimiter is token** - This is useful for interpolated strings where finding the **End delimiter** requires tokenizing the text so that tokens like strings which could contain the **End delimiter** are skipped. Note that defining an interpolated string requires the outer string start/end delimiters to be defined and then an inner start/end delimiters (often `\$`` and `) when start delimiter is a SlickEdit regular expression) to be defined where this option is checked. Use **Add Sub Item** to add the sub item for interpolation and set the start/end delimiters. You may need to set the **Nest with start** and **Nest with end** to { and }. Scala interpolated strings require the interpolation to support nested braces where **Nest with start** and **Nest with end** are { and } respectively.
- **Apply multi-line coloring at the end of line** - Only available for multi-line constructs. This option is typically used for here-document constructs where the start should be colored as Other color, then the text until the end of line is colored as if this construct was never hit, and then subsequent lines continue with this construct.
- **End embedded at beginning of line if possible** - When the **Start delimiter** for an embedded language construct starts at the end of a line and the **End delimiter** is the first non-blank in a line, choose this option. That way only lines in between the start and end delimiters are colored in embedded language color.
- **Embedded color style** - One of the following:
 - **Color as embedded** - Switches the background color to embedded.
 - **Don't color as embedded** - Continue to use the current background color which may already be embedded.
 - **Color as embedded only if profile found** - If **Embedded profile** is found, switch to the background to embedded. Otherwise, the current background color which may already be embedded is used.

Color Coding Numbers Tab

The **Numbers** tab provides options for color coding numerical values when working with SlickEdit®.

Language Options

C/C++ > Color Coding

Profile: cpp

General Tokens Numbers Language Tags

Integers may start with a digit [0-9]
 Floating point numbers may start with a digit [0-9]
 1.e4 is valid float (e4 is exponent not data member)
 Floating point may use "D" exponent
 Floating point numbers may start with a decimal point (ex .123)
 Hexadecimal floating point (ex 0x1A.F3p+EF)
 0x1.p4 is valid float (p4 is exponent not data member)
 0x##### Hexadecimal (ex 0xFF)
 0o##### Octal (ex 0o777)
 0b##### Binary (ex 0b1010)
 0d##### Decimal (ex 0d89)
 Verilog base single quote numbers (ex 16'hFFFF 16'd1234)
 No exponent on floating point. Don't allow 1.2e4 but allow 1.2
 Allow hex digits in integers. Used for coloring 1AFFH (Modula-2 and Assembly)
 Color leading sign as part of number

Digit separator char: '

Integer suffixes: ULL UL U LLU LL L I8 I16 I32 I64

Float suffixes: L F

Hex suffixes:

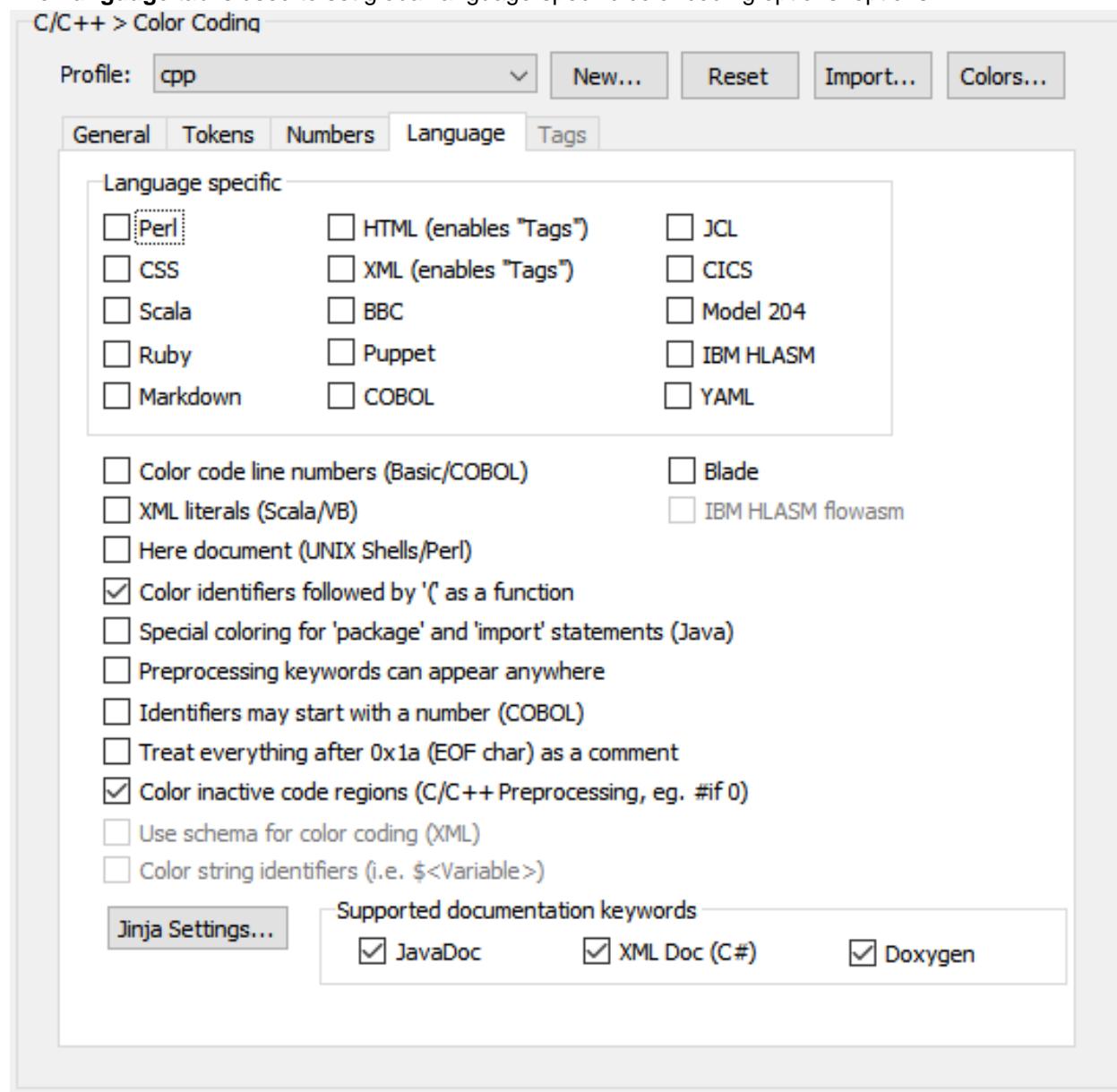
This tab contains the following:

- **Integers may start with a digit [0-9]** - Indicates whether integers starting with a digit (ex. 123) is colored in number color.
- **Floating point numbers may start with a digit [0-9]** - Indicates whether floating point starting with a digit (ex. 1.2) is colored in float point number color.
- **1.e4 is valid float (e4 is exponent and not data member)** - Indicates whether floating point numbers allow the exponent to immediately follow the decimal point (ex 1.e4).
- **Floating point may use "D" exponent** - Indicates whether floating point numbers also allow the exponent to be specified with a "D" (ex 1.4D5).

- **Floating point numbers may start with a decimal point (ex .123)** - Indicates whether floating point numbers may start with a decimal point (ex .123).
- **Hexadecimal floating point (ex 0x1A.F3p+EF)** - Indicates whether floating point hexadecimal numbers are colored in floating point number color. Many languages have adopted this standard syntax for hexadecimal floating point syntax.
- **1.p4 is valid float (p4 is exponent and not data member)** - Indicates whether hexadecimal floating point numbers allow the exponent to immediately follow the decimal point (ex 1.p4).
- **0x##### Hexadecimal (ex 0xFF)** - Indicates whether hexadecimal numbers such as **0x123ABC** is color coded in number color
- **0o##### Octal (ex 0o777)** - Indicates whether octal numbers such as **0o777** is color coded in number color.
- **0b##### Binary (ex 0b1010)** - Indicates whether binary numbers such as **0b1010** is color coded in number color.
- **0d##### Decimal (ex 0d89)** - Indicates whether decimal numbers such as **0d89** is color coded in number color.
- **Verilog base single quote numbers (ex 16'hFFFF 16'd1234)** - Indicates whether Verilog syntax base single quote numbers like **16'hFFFF** and **16'd1234** are color coded in number color.
- **No exponent on floating point. Don't allow 1.2e4 but allow 1.2.** - When checked, indicates that floating point numbers do not have an exponent.
- **Allow hex digits in integers. Used for coloring 1AFFFH (Module-2 and Assembly)** - This option is useful for coloring hexadecimal numbers which start with a digit (no prefix characters) and may contain hexadecimal digits. Typically there is a suffix character like 'H' added to the **Hex suffixes** text box.
- **Color leading sign as part of number** - When on, the leading sign (ex +123 or -123) is colored as part of the number.
- **Digit separator char** - Specifies a single character which is allowed between digits. For example, C++ supports a single quote character (ex 123'000'000). Perl and many other languages support an underscore (123_000_000).
- **Integer suffixes** - Space delimited list of supported integer suffixes. By default, all suffixes are case insensitive. Prefix the suffix with "\c" to specify a case sensitive suffix. If your prefix starts with a backslash, uses two backslashes (ex "\\").
- **Float suffixes** - Space delimited list of supported floating point suffixes. By default, all suffixes are case insensitive. Prefix the suffix with "\c" to specify a case sensitive suffix. If your prefix starts with a backslash, uses two backslashes (ex "\\").
- **Hex suffixes** - Space delimited list of supported hexadecimal integer suffixes. By default, all suffixes are case insensitive. Prefix the suffix with "\c" to specify a case sensitive suffix. If your prefix starts with a backslash, uses two backslashes (ex "\\").

Color Coding Language Tab

The **Language** tab is used to set global language-specific color coding options. options.



This tab contains the following:

- **Language specific** - Selecting some of the language specific options simply adds some extra table entries you could have defined yourself. Others like HTML and XML apply some built-in changes that can't be done as table entries. You may be able to use one of these language-specific settings for another language, but there's no guarantee it will work. Typically only one of these options can be checked at a time.
- **Color Code Line Numbers (Basic/COBOL)** - When selected, indicates that leading line numbers should be color-coded in line number color.

- **XML literals (Scala/VB)** - When selected, indicates XML literals should be color coded as embedded XML.
- **Here Document (UNIX Shells/Perl)** - Adds support for generic **HERE** documents similar to Perl syntax but primarily for backward compatible with previous versions of SlickEdit. Note that the Perl color coding definition no longer uses this option. Instead, more precise Perl specific table items are defined.

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

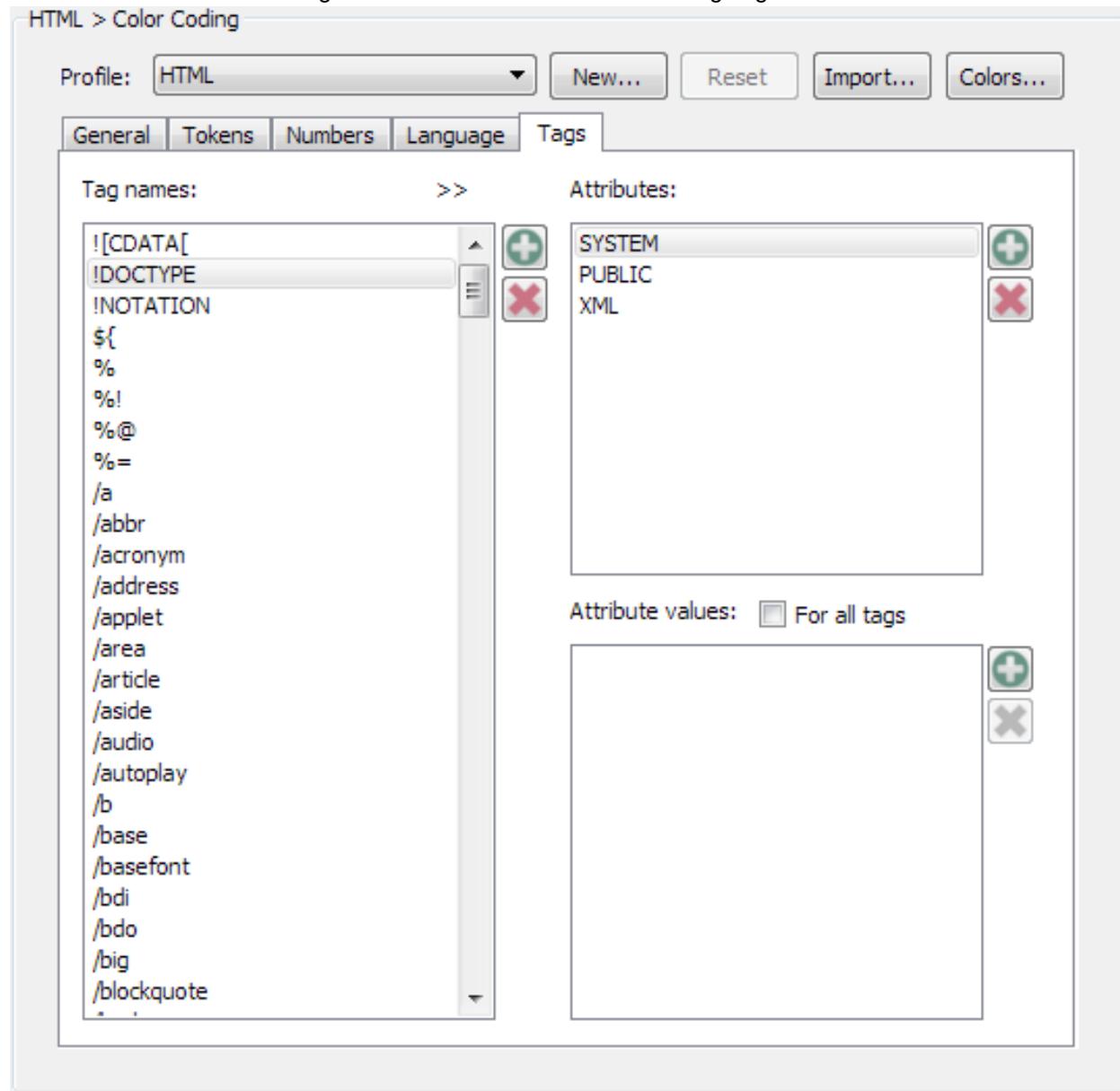
Unknown languages are color-coded in string color. Embedded language colors are user-definable.

- **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 selected, the Java syntax package and import statements are supported. This option is forced on for the Java color coding profile. 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 selected, preprocessing keywords are color-coded even if they are not only preceded by white space.
- **Identifiers may start with a number (COBOL)** - When selected, identifiers may start with one or more decimal digits. By default, leading decimal digits indicate a number.
- **Treat everything after 0x1a as comments (end of file)** - Historically, DOS used 0x1a to mark the end of the file. When checked, SlickEdit will treat all characters after 0x1a as comments.
- **Color inactive code regions (C/C++ Preprocessing, eg. #if 0)** - When checked, uses a single color for inactive code regions, instead of applying normal color coding.
- **Use schema for color coding (XML)** - When opening an XML file, checks for a schema to color elements and attributes. If the schema is remote and inaccessible, this can cause delays.
- **Color string identifiers (i.e. \$<Variable>)** - When on, color \$<Variable> as a different color. Also, effects some more complex expressions (i.e. PHP -- \$a->member). This option is only enabled for PHP, Scala, Dart, Kotlin, Perl, and PowerShell.
- **Blade** - When on, turns on Laravel Blade color coding.
- **IBM HLASM flowasm** - When on, enabled flowasm color coding support for IBM HLASM.
- **Jinja Settings...** - Settings for adding Jinja color coding.

- **Supported documentation keywords** - Indicates the supported documentation comment types (JavaDoc, XML Doc, and/or Doxygen) which should be supported in constructs with the color element **Type** set to **Doc Comment**. For example, when **JavaDoc** is checked, "@param" is colored in Doc Keyword color.

Color Coding Tags Tab

The **Tags** tab is used to set color-coding attributes when working with tagged-based languages such as HTML and XML. The following screen shot shows the Color Coding Tags tab for HTML:



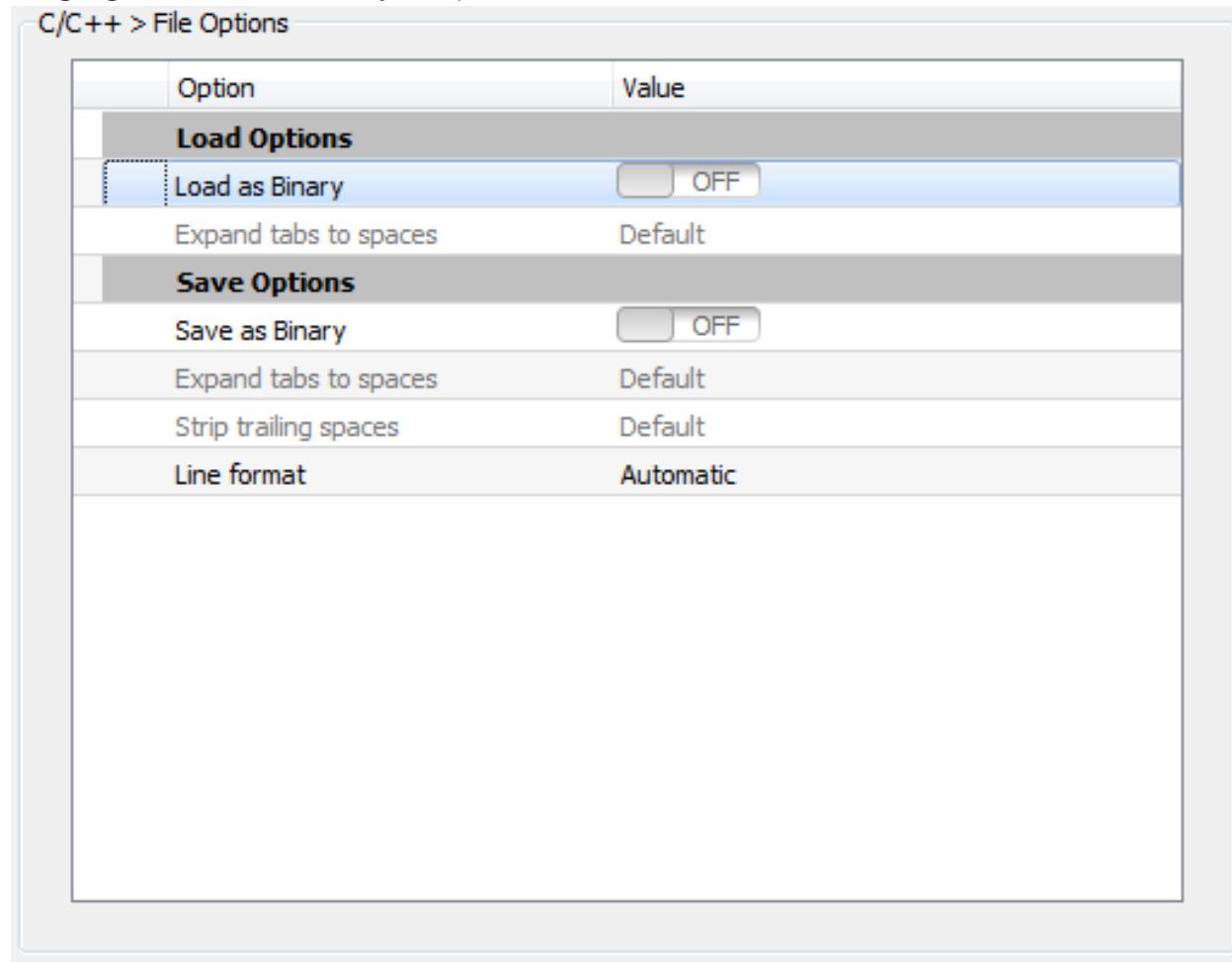
It contains the following options:

- **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.

Language-Specific File Options

These options are used to specify load and save options for files on a language-specific basis. As an example, the C/C++ file options are shown below (**Tools** → **Options** → **Languages** → **Application Languages** → **C/C++** → **File Options**).



The options are described as follows:

- **Load Options:**
 - **Load as Binary** - When set to **On**, files are loaded without any translations (like changing tabs to spaces). This setting has precedence over all global options, as well as all language-specific options.

- **Expand tabs to spaces** - When set to **Default**, SlickEdit® uses the setting for the global file load option, **Expand tabs to spaces** (**Tools** → **Options** → **File Options** → **Load**). When set to **On**, SlickEdit always loads files with tabs expanded to spaces. When set to **Off**, tabs are always left unexpanded.
- **Save Options:**
 - **Save as Binary** - When set to **On**, files are saved without any translations, exactly byte-for-byte as they appear in the buffer. This setting has precedence over all global options, as well as all language-specific options.
 - **Expand tabs to spaces** - When set to **Default**, SlickEdit uses the setting for the global file save option, **Expand tabs to spaces** (**Tools** → **Options** → **File Options** → **Save**). When set to **Expand all tabs to spaces**, SlickEdit all tabs in a file will be expanded to spaces on save. When set to **Do not expand tabs to spaces**, tabs are always left unexpanded. You can also set this value to **Expand tabs to spaces only on modified lines** to expand tabs only on lines that have been modified or inserted.
 - **Strip trailing spaces** - Specifies if and when to remove trailing spaces from the ends of lines. When set to **Default**, SlickEdit uses the setting for the global file save option, **Strip trailing spaces** (**Tools** → **Options** → **File Options** → **Save**). When set to **Strip all trailing spaces**, trailing spaces at the end of lines are stripped. When set to **Do not strip trailing spaces**, spaces at the end of lines are always left. When set to **Strip trailing spaces only from modified lines**, trailing spaces at the end of lines are stripped only from modified or inserted lines.
 - **Line format** - Specifies how end of line characters are translated when a file is saved. When **Automatic** is set, the line breaks are saved automatically in the file format appropriate to the context in which you are working with no changes to the end of line characters. However, you can specify 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/macOS(LF)**.

Note

- The Save As dialog also allows the translation of the line end characters for the current file. See [Save As Dialog](#).
- Classic Mac line endings are a single carriage return (ASCII 13).

Language-Specific Live Error Profiles

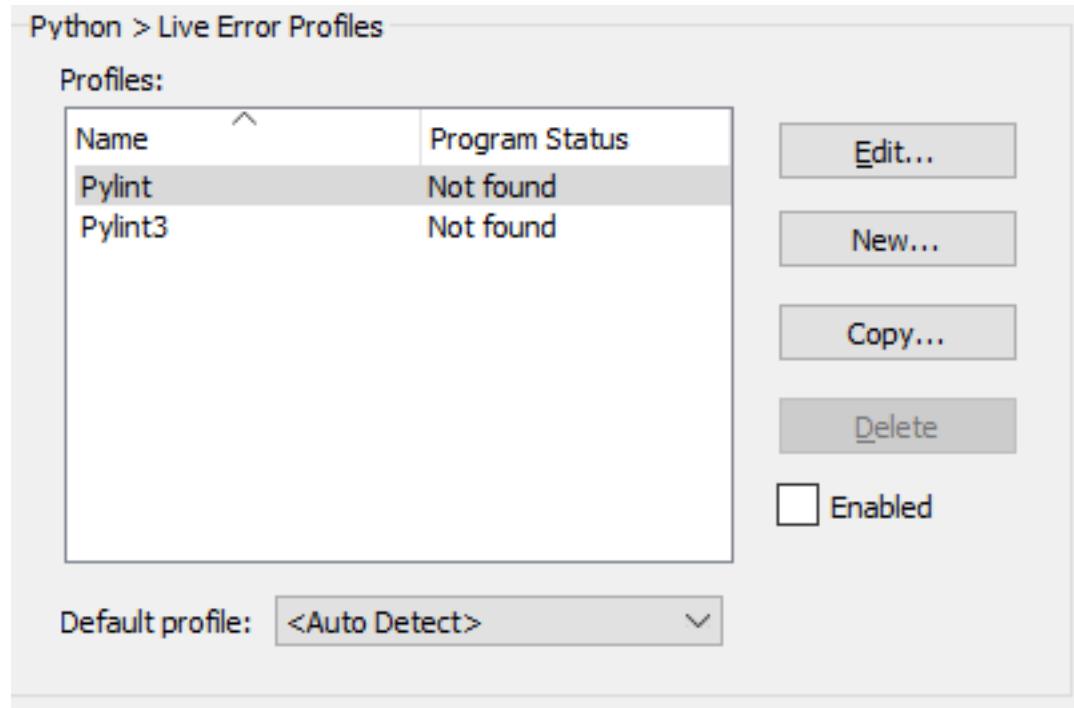
These options are used to define profiles that describe how to run a syntax checking program, and to define the default profile to use for the language.

Note

For the Java language, there is no “Live Errors Profiles” panel, as the legacy Live Errors

implementation has to be more Java centric to get good performance. For details on how to enable Live Errors for Java, see [Java Live Errors](#).

As an example the Python error profile options are shown below (**Tools** → **Options** → **Languages** → **Scripting Languages** → **Python** → **Live Error Profiles**).



- **Edit...** - Edits the selected profile. See [Live Error Profile Dialog](#) for details.
- **New...** - Creates a new blank profile.
- **Copy...** - Copies and renames an existing profile. Useful if you want to just tweak an existing profile without changing the original.
- **Delete** - Deletes a profile permanently. Only user-created profiles can be deleted, the profiles shipped with the editor can not be deleted.
- **Default profile:** - Allows you to pick which profile is used by default for the source language. There is a **<Auto Detect>** selection that will pick the first profile that references a syntax checking program that is installed on your system.

Live Error Profile Dialog

The Live Error system depends on external checking programs producing the error messages shown to the user. The editor runs these programs in the background for the file being edited automatically.

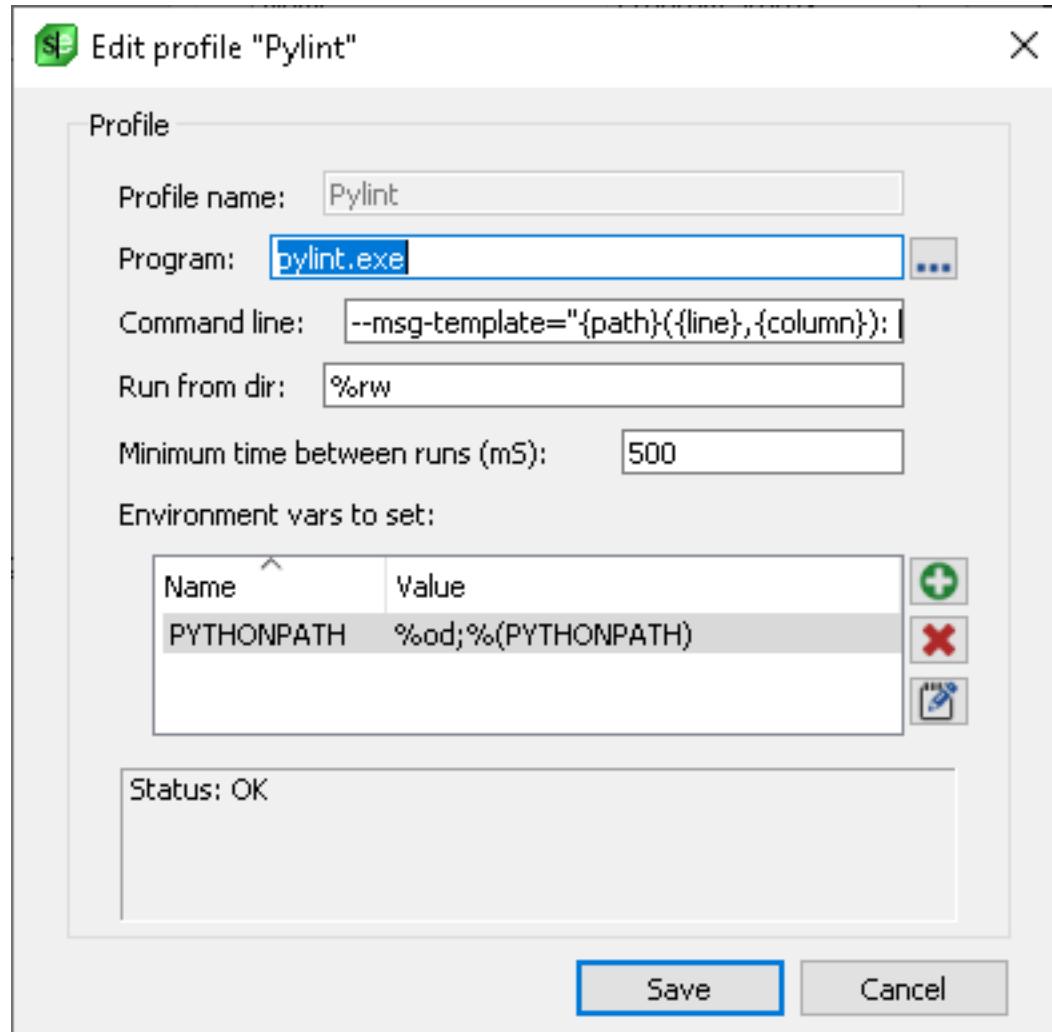
It is expected that a checker program:

- Takes an argument for the file to be checked

- Produces error messages with file and line number references on the standard output or the standard error streams.
- Runs fairly quickly. Feedback that takes more than a few seconds will quickly lag behind your typing.

The editor recognizes the errors in the output of the checker using the same error parsing used for recognizing errors in builds. If the output of the checker program is not recognized, you can update the error parsing configuration. See [Configuring Error Parsing](#) for more details.

The profile editor dialog:



- **Program**- The path to the syntax checking executable, or the bare name of the executable if the checker is on the PATH.
- **Command line** - the command line to be passed to the program where "%f" is a placeholder for the path of the source file being checked. In addition, this field recognizes the escape sequences defined in [Escape Sequences for Build Commands](#).
- Run from dir - directory the checker program should be run from. In most cases the default of "%rw" (workspace directory) should be fine. In addition, this field recognizes the escape sequences defined in

[Escape Sequences for Build Commands](#)

- **Minimum time between runs** - the minimum time between separate runs of the checker in mS. If a checker is resource intensive, this allows you control over how often it can run.
- **Environment vars to set** - allows you to set or modify environment variables for tools use environment variable parameters. Environment variable values recognize the escape sequences defined in [Escape Sequences for Build Commands](#).

For example, you could set "PYTHONPATH" to "%rw/3rdparty:%(PYTHONPATH)" to extend the existing "PYTHONPATH" variable to include the 3rdparty directory that's located in the same directory as your workspace.

Additional Escape Sequences

For the fields that recognize escape sequences, there are two additional escape sequences that are specific to Live Errors.

Sequence	Expands to
%OF	The full path name of the source file in your current buffer that is being checked. This will not be the same as the path passed to the external tool via the %f escape.
%OD	The directory of the source file in your current buffer that is being checked. This is useful for tooling where extra include paths or other details need to be based off the location of the source file being edited.

Language-Specific Compiler Properties (Pro only)

These settings are used to configure your compiler so that SlickEdit® can correctly perform full preprocessing, parsing, symbol analysis, and cross-referencing. The fields and options on this page depend on the selected language.

Note

These language-specific compiler settings are also available in a dialog interface. The fields and options are identical, so you can use the interface you prefer. To access the dialog, from the main menu, click **Project → Project Properties**. Select the **Compile/Link** tab, then click the **Ellipsis** button to the right of the **Compiler** combo box.

The following fields and options are common to all languages on the Compiler Properties interface:

- **Compiler Name** - Contains a list of compilers. Names in this list are the names specified when you click **Add**.
- **Add** - Used to add a new compiler name to the list. After adding the name, you will need to configure the compiler using the configuration settings on the lower-half of the interface.
- **Delete** - Deletes the selected compiler and its associated configuration. Does not delete files from disk.
- **Copy** - Used to add a new compiler configuration by copying the selected compiler's configuration. You will be prompted for a new compiler name.
- **Set Default** - Specifies that the selected compiler should be used as the default. The current default is displayed under the **Compiler Name** field.
- **Build Tag File** - Used to build tag files 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.
- **Compiler Configuration** - The lower half of the Compiler Properties interface is used to configure the selected compiler. In C/C++, you specify the header file and include directories. In Java, you specify the root JDK installation and system libraries.

For more information, see [C/C++ Compiler Settings](#) or [Java Compiler Properties Dialog](#).

File Options

These global file options (**Tools** → **Options** → **File Options**) are used to make settings regarding file operations such as loading and saving. For more information about working with files, see the chapter [Workspaces, Projects, and Files](#).

File option categories are:

- [Open File Options](#)
- [Load File Options](#)
- [Save File Options](#)
- [Backup File Options](#)
- [AutoSave File Options](#)
- [Files of Type Filter Options](#)
- [Associate File Types Options](#)
- [History Options](#)

Open File Options

File Options

Open file options are shown below (**Tools** → **Options** → **File Options** → **Open**). For more information about loading and opening files, see [Opening Files](#).

File Options

Open		
	Option	Value
	Zip Extensions	.zip .jar .aar .xlsx .docx .jmod
	Set filename when creating new file with no name	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Edit "A B C" start on file A	<input type="radio"/> OFF
	Open files using	Open Tool Window
	Decompress .gz files on open	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Decompress .xz files on open	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Decompress .bz2 files on open	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Check for inconsistent line endings on open	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Excludes for inconsistent line ending checking	<Binary Files>;user.cfg.xml
	e/edit command Smart Open workspace files	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	e/edit command Smart Open open documents	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	e/edit command Smart Open files in same directory	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Set current directory when switching buffers	<input type="radio"/> OFF
	Add opened files to Recent Files	<input type="radio"/> OFF
	Prefer short filenames	<input type="radio"/> OFF
Open Tool Window		
	Dismiss on select	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Clear text box on Enter	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Sync current directory	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Show all folders in directory panel	<input type="radio"/> OFF
	Change current directory with single-click	<input type="radio"/> OFF
	Show folders in file list	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	*.ext<Enter> in File name	Open currently selected file
	<Tab> in File name	Move focus to file list
	Show/match workspace and project files	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Show/match open files	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Show/match files in history	<input type="radio"/> OFF
	Show/match files in current file directory	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	*.ext wildcard name match style	Current directory exact matching
	Other wildcard name match style (file*.cpp)	Recursive exact matching
	Non wildcard name match style (file.cpp)	Recursive contains matching
	Auto-size columns	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Show relative paths	<input checked="" type="radio"/> ON <input type="radio"/> OFF
	Maximum number of files to list	5000
	Open tool window orientation	Auto (Default)
Project Properties Dialog		
	Show files in Add Source Files dialog	Hide files already in the project if supported by native dialog

The options are described as follows:

- **Zip Extensions** - Specifies the file extensions which SlickEdit should consider as Zip format files.
- **Set filename when creating new file with no name** - When on, new files created without specifying a name get a filename which includes the date. The AutoSave directory is used for the path.
- **Edit 'A B C' start on file A** - When set to **On**, the first file opened becomes the active buffer.
- **Open files using** - Specifies which style of open dialog you wish to use to open files. Select one of the following values:
 - **Browse files with Open Dialog** - Uses the traditional open dialog to open files.
 - **Smart Open** - Uses the Open Tool Window to open files.
- **Decompress .gz files on open** - When set to **On**, opening a .gz or .Z file will decompress it and open the file it contains instead of displaying the binary data.
- **Decompress .xz files on open** - When set to **On**, opening a .xz file will decompress it and open the file it contains instead of displaying the binary data.
- **Decompress .bz2 files on open** - When set to **On**, opening a .bz2 file will decompress it and open the file it contains instead of displaying the binary data.
- **Check for inconsistent line endings on open** - When set to **On**, opened files will be checked for inconsistent line endings. If the file has inconsistent line endings, you can change the line endings. By default, files in the Binary mode are skipped. Also, files larger than the 'Turn off check for inconsistent line endings when editing file larger than (KB)' are not checked either. Use the `check_line_endings` command (**Edit → Other → Check Line Endings**) to check any file.
- **Excludes for inconsistent line ending checking** - Semicolon delimited list of excludes (ant-like wildcard expressions). Filenames which match these excludes do not get checked for inconsistent line endings on open. Ex. `*.exe;*.dll;temp;/<Binary Files>`
- **e/edit command Smart Open** - Specifies the completion behavior when using the **e** or **edit** commands.
 - **e/edit command Smart Open workspace files** - When using the edit command to open files, specifies that Smart Open will help complete file names in the current workspace.
 - **e/edit command Smart Open documents** - When using the edit command to open files, specifies that Smart Open will help complete file names of other files current open in the editor.
 - **e/edit command Smart Open files in same directory** - When using the edit command to open files, specifies that Smart Open will help complete file names in the same directory as the current file.
- **Set current directory when switching buffers** - When set to **On**, sets the current directory to the location of the selected file each time you change buffers.
- **Add opened files to Recent Files** - When set to **On**, files opened are added to the Windows Recent Files list.

- **Open Tool Window** - The following options are available:
 - **Dismiss on select** - When set to **On**, the Open Tool Window is closed or auto-hidden after a file is selected and opened. When set to **Off**, the tool window remains open and visible.
 - **Clear text box on Enter** - When set to **On**, the **File name** text box will be cleared when Enter is pressed. When set to **Off**, characters in the text box will remain.
 - **Sync current directory** - When set to **On**, changing the current directory outside of the Open Tool Window will also change the current directory within the tool window. Changing the current directory within the Open Tool Window also changes the application's current directory. When set to **Off**, The Open Tool Window maintains its own current directory, independent from the rest of the application.
 - **Show all folders in directory panel** - When set to **On**, displays all folders which are siblings to folders along the path in the directory panel, rather than focusing only on the current path.
 - **Change current directory with single-click** - When set to **On**, a single mouse click in the directory explorer pane will change the current directory and refresh the files panel. When set to **Off**, this behavior is triggered by a double-click.
 - **Show folders in file list** - When set to **On**, files and folders are shown in the top pane of the Open tool window. This allows patterns in the **File name** field to match folders as well as files.
 - ***.ext<Enter> in File name** - Set to one of the following options:
 - **Open currently selected file** - Pressing <Enter> will open the currently selected file.
 - **Set Files of type** - Pressing <Enter> will set the Files of type to the extension list specified.
 - **<Tab> in File name** - Set to one of the following options:
 - **Move focus to file list** - Pressing <Tab> will set focus to the file list control.
 - **Complete selected filename** - Pressing <Tab> will complete selected filename.
 - **Show/match workspace and project files** - When set to **On**, files from the current project and workspace are shown when their file names match the file name filter text. When set to **Off**, these files are never shown in the list of matched file names.
 - **Show/match open files** - When set to **On**, open buffers are shown when their file names match the file name filter text. When set to **Off**, these files are never shown in the list of matched file names.
 - **Show/match files in history** - When set to **On**, files from the recently opened file history are shown when their file names match the file name filter text. When set to **Off**, these files are never shown in the list of matched file names.
 - **Show/match files in current file directory** - When set to **On**, files in the same directory as the current file are shown when their file names match the file name filter text. When set to **Off**, these files are never shown in the list of matched file names.
 - **Show indicator for read-only files** - When set to **On**, files that are read-only on disk will display an overlay to indicate that the file is read-only. Note, for performance, this feature only checks the read-

only attribute on Windows, and simple user/group/other permissions on Unix. It does not check ACL lists or other more sophisticated access control mechanisms.

- **Show indicator for writable files** - When set to **On**, files that are writable (not read-only) on disk will display an overlay to indicate that the file is writable. Note, for performance, this feature only checks the read-only attribute on Windows, and simple user/group/other permissions on Unix. It does not check ACL lists or other more sophisticated access control mechanisms.
- ***.ext wildcard name match style** - Set to one of the following options:
 - **Current directory contains matching** - When selected, lists files in the current directory which contain the wildcard specified anywhere in the string. For example, ".c" will match a file called "main.cpp".
 - **Current directory exact matching** - When selected, list files in the current directory which match the wildcard specified. For example, ".c" will match a file called "main.c" but not a file called "main.cpp".
 - **Recursive contains matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which contain the wildcard specified anywhere in the string. For example, ".c" will match a file called "main.cpp". Searching of files on disk is always limited to the current directory.
 - **Recursive exact matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which match the wildcard specified. For example, ".c" will match a file called "main.c" but not a file called "main.cpp". Searching of files on disk is always limited to the current directory.
- **Other wildcard name match style** - Set to one of the following options:
 - **Current directory contains matching** - When selected, lists files in the current directory which contain the wildcard specified anywhere in the string. For example, "n*.c" will match a file called "main.cpp".
 - **Current directory exact matching** - When selected, list files in the current directory which match the wildcard specified. For example, "m*.c" will match a file called "main.c" but not a file called "main.cpp".
 - **Current directory prefix matching** - When selected, list files in the current directory which match the wildcard prefix specified. For example, "m*.c" will match a file called "main.c" and also a file called "main.cpp".
 - **Recursive contains matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which contain the wildcard specified anywhere in the string. For example, "m*.c" will match a file called "main.cpp". Searching of files on disk is always limited to the current directory.
 - **Recursive exact matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which match the wildcard specified. For example, "m*.c" will match a file called "main.c" but not a file called "main.cpp". Searching of files on disk is always limited to

the current directory.

- **Recursive prefix matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which match the wildcard prefix specified. For example, "m*.c" will match a file called "main.c" and also a file called "main.cpp". Searching of files on disk is always limited to the current directory.
- **Non wildcard name match style** - Set to one of the following options:
 - **Current directory contains matching** - When selected, lists files in the current directory which contain the wildcard specified anywhere in the string. For example, "n.c" will match a file called "main.cpp".
 - **Current directory exact matching** - When selected, list files in the current directory which match the wildcard specified. For example, "main.c" will match a file called "main.c" but not a file called "main.cpp".
 - **Current directory prefix matching** - When selected, list files in the current directory which match the wildcard prefix specified. For example, "main.c" will match a file called "main.cpp" but will not match a file called "grepmain.cpp".
 - **Recursive contains matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which contain the wildcard specified anywhere in the string. For example, "main.c" will match a file called "main.cpp" and also a file called "grepmain.cpp". Searching of files on disk is always limited to the current directory. Recursive searching of files (workspace, open files, history) ignores the current directory.
 - **Recursive exact matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which match the wildcard specified. For example, "main.c" will match a file called "main.c" but not a file called "main.cpp". Searching of files on disk is always limited to the current directory. Recursive searching of files (workspace, open files, history) ignores the current directory.
 - **Recursive prefix matching** - When selected, lists files from configured **Show/match files** options (workspace, open files, history) which match the wildcard prefix specified. For example, "main.c" will match a file called "main.cpp" but will not match a file called "grepmain.cpp". Searching of files on disk is always limited to the current directory. Recursive searching of files (workspace, open files, history) ignores the current directory.
- **Auto-size columns** - When **On** the first column will be auto-sized in an attempt to make the longest item fit. Otherwise, the previous column size is used.
- **Show relative paths** - When **On** paths are displayed relative to the project.
- **Maximum number of files to list** - To improve performance, only a limited number of files are shown in the upper panel of the Open tool window. When filtering the file list based on a search string, this limit is not used. The limit also applies to the Workspace tab of the Files tool window.
- **Open tool window orientation** - Specifies whether the Open tool window should be oriented

vertically (with the directory tree under the files list) or horizontally (with the directory tree next to the files list). Select **Auto (Default)** to have the orientation selected automatically based on the tool window's current size.

- **Show files in Add Source Files dialog** - Set to one of the following options:
 - **Hide files already in the project if supported by native dialog** - When selected, files already in the project are hidden when using the Add Source Files dialog as long as the native open dialog supports it.
 - **Show files already in the project** - When selected, files already in the project are always displayed.
- **Project Properties dialog** - The following options are available:
 - **Show files in Add Source Files dialog** - Specifies whether or not to hide files that are already in the project when adding source files to a project using the Add Source Files dialog (see [How to Add or Remove Files From a Project](#)). There are two options for hiding files. The default is to only hide files already in the project if that kind of filtering is supported by the native open dialog. Use the second option if the convenience of filtering out files that are already in the project is more important than using the operating system's native open file dialog.

Tip

Turning this option off (**Show files already in the project**), can improve the performance of the Add Source Files dialog for large projects.

Load File Options

Load file options are shown below (**Tools** → **Options** → **File Options** → **Load**). For more information about loading and opening files, see [Opening Files](#).

File Options

Load	
Option	Value
Fast line count on partial load	<input type="checkbox"/> OFF
Show EOF character	<input checked="" type="checkbox"/> ON
Expand tabs to spaces	<input type="checkbox"/> OFF
File locking	<input type="checkbox"/> OFF
Reinsert after current	<input checked="" type="checkbox"/> ON
Wrap line length	4000
Use undo	<input checked="" type="checkbox"/> ON
Max undo steps	32000
Prompt to undo past last save	<input checked="" type="checkbox"/> ON
Save/restore file position	<input checked="" type="checkbox"/> ON
Max files	1000
Encoding	Auto Unicode2
Load entire file	
Load entire file	<input checked="" type="checkbox"/> ON
Load partially for large files (use file locking)	<input checked="" type="checkbox"/> ON
Load partially when files are larger than (KB)	8000
Count number of lines	<input type="checkbox"/> OFF
Truncate file at EOF	<input type="checkbox"/> OFF
Auto reload	
Auto reload	<input checked="" type="checkbox"/> ON
Suppress prompt unless modified	<input type="checkbox"/> OFF
Compare file contents before auto reload	<input checked="" type="checkbox"/> ON
Size limit for comparing contents (KB)	2000
Auto read only	<input checked="" type="checkbox"/> ON
Fast auto read only	<input checked="" type="checkbox"/> ON
Reload on switch buffer	<input checked="" type="checkbox"/> ON
Auto reload current file only	<input type="checkbox"/> OFF
Auto reload all files if current file changed	<input type="checkbox"/> OFF
Auto reload timeout (ms)	5000
Show auto reload timeout notifications	<input checked="" type="checkbox"/> ON

The options are described as follows:

- **Fast line count on partial load** - When set to **On**, SlickEdit® counts 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 **edit** command to open files, use the switch **+LF** to control this option (see [Command Line Switches](#)).

- **Show EOF character** - When set to **On**, the EOF character is not removed when a file is loaded. If you are using the **edit** command to open files, use the switch **+LE** to control this option (see [Command Line Switches](#)).
- **Expand tabs to spaces** - When set to **On**, the entire contents of 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** → **Languages** → **[Language Category]** → **[Language]** → **Indent**, and 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 **edit** command to open files, use the switch **+E** to specify this option (see [Command Line Switches](#)).
- **File locking** - When set to **On**, 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 **edit** command to open files, use the switch **+N** to specify this option (see [Command Line Switches](#)).
- **Reinsert after current** - When set to **On**, SlickEdit switches back to the previous buffer with the **prev_buffer** command. If you are using the **edit** command to open files, use the switch **+BP** to specify this option (see [Command Line Switches](#)).
- **Wrap line length** - Specifies the number of characters at which long lines should be wrapped when a file is opened. This option improves editing performance and is particularly useful for editing very large, single-line XML files.
- **Use undo** - When set to **On**, modifications to buffers may be undone (**Ctrl+Z** or **Edit** → **Undo**).
- **Max undo steps** - Specifies the maximum number of steps that are stored for undo operations when **Use undo** is enabled. Cursor motion can be undone but is not counted as a step. If you are using the **edit** command to open files, use the switch **+U** to specify this option (see [Command Line Switches](#)). For example, **+U:32000** turns on undo and specifies a 32,000-step max.
- **Prompt to undo past last save** - When set to **On**, when you undo all the changes up to the last save, you will be prompted whether you want to continue undoing changes.
- **Save/restore file position** - When set to **On**, the cursor position in files is preserved on close and restored on open.
- **Max files** - Specifies the maximum number of recently closed cursor positions to save when **Save/restore file position** is enabled.
- **Line endings for new files** - Specifies the line endings used when a new file is created.
- **Encoding** - Unicode support required. Specifies the global (non-extension specific) file encoding. This

setting is overridden if a extension-specific encoding is defined on the File Extension Manager for the selected language (see [Managing File Extensions](#)). 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, csproj, vbproj) that are XML files but that default to active code page encoding and not UTF-8, like XML. See [Encoding](#) for more information.

- **Load entire file** - The following options are available:

- **Load entire file** - When set to **On**, the entire contents of opened files are read into memory, however, 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 **Off**, Auto Reload does not work until the file is saved. If you are using the **edit** command to open files, use the switch **+L** to specify this option (see [Command Line Switches](#)).
- **Load partially for large files** - When set to **On**, the editor only partially loads files larger than the size specified in **Load partially when files are larger than**, in order to conserve 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 enabled.
- **Load partially when files are larger than** - Specifies the size limit for files to be opened completely in the editor. If the size of the file being loaded is greater than this value, only a portion of the file is read into memory.
- **Count number of lines** - When set to **On**, the entire contents of 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** setting will have the same affect as this setting 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 **edit** command to open files, use the switch **+LC** to specify this option (see [Command Line Switches](#)).
- **Truncate file at EOF** - When set to **On**, 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 (End of File) 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 **edit** command to open files, use the switch **+LZ** to specify this option (see [Command Line Switches](#)).

- **Auto reload** - The following options are available:

- **Auto reload** - When set to **On**, SlickEdit detects when files being edited have been modified by other applications and prompts to replace the files with the new copies on disk. If there are files with unsaved changes, the user can select among them to compare them to the files on disk before choosing to reload or merge changes. Open files that have been deleted from disk are also detected and the user can select which to resave.
- **Suppress prompt unless modified** - When set to **On**, files that have been changed on disk are automatically reloaded unless the file has been modified in SlickEdit.

- **Compare file contents before auto reload** - When set to **On**, if auto reload detects that the current file has been changed on disk, SlickEdit will also compare the contents of the file on disk with the version in memory to determine if it really needs to be reloaded. This option is useful when you have a file system which does not report modification dates correctly.
- **Size limit for comparing contents (KB)** - Specifies the size limit below which files are compared to determine whether they need to be reloaded.
- **Auto read only** - When set to **On**, SlickEdit detects when other applications change the read-only attribute of a file, and automatically changes the permissions of the file being edited to match.
- **Fast auto read only** - (Windows only) When set to **On**, this option speeds up the **Auto read only** feature by only checking the attribute on disk (not opening every file). This option is controlled by the configuration variable `def_fast_auto_READONLY` (see [Setting/Changing Configuration Variables](#)).
- **Reload on switch buffer** - When set to **On**, SlickEdit detects if a file has been modified by another application when you switch buffers to view the file in the active editor window. When **Off**, the default check is still performed when you switch from another application.
- **Auto reload current file only** - When set to **On**, SlickEdit will check if the current open file has been modified by another application when you switch buffers switch from another application. Otherwise, all files are checked for reload. Enabling this option can improve performance when switching from another application back to SlickEdit.
- **Auto reload all files if current file changed** - When set to **On**, if auto reload detects that the current file has been changed on disk or deleted, SlickEdit will then check if any other open files have been changed on disk or deleted.
- **Auto reload timeout (ms)** - Specifies the amount of time to wait for file information before skipping auto reload.
- **Show auto reload timeout notifications** - When set to **On**, the user will be given a message when auto reload is skipped for a file.

Save File Options

Save file options are shown below (**Tools** → **Options** → **File Options** → **Save**).

Save	
Option	Value
Append EOF character	<input type="checkbox"/> OFF
Remove EOF character	<input type="checkbox"/> OFF
Expand tabs to spaces	Do not expand tabs to spaces
Strip trailing spaces	Do not strip trailing spaces
Save files on loss of focus	<input type="checkbox"/> OFF
Reset modified lines	<input type="checkbox"/> OFF
Add file to project upon Save As	<input type="checkbox"/> OFF
Save all prompts to name unnamed files	<input type="checkbox"/> OFF

The options are described as follows:

- **Append EOF character** - When set to **On**, an EOF (End of File) character is appended to the end of DOS files when the buffer is saved. Has no effect on UNIX, Mac, or binary files. If you are using the **save** command to save files, use the switch **+Z** to specify this option (see [Command Line Switches](#)).
- **Remove EOF character** - When set to **On**, the EOF (End of File) character is removed from the end of DOS files when the buffer is saved. Has no effect on UNIX, Mac, or binary files. If you are using the **save** command to save files, use the switch **+ZR** to specify this option (see [Command Line Switches](#)).
- **Expand tabs to spaces** - When set to **Expand all tabs to spaces**, SlickEdit all tabs in a file will be expanded to spaces on save. When set to **Do not expand tabs to spaces**, tabs are always left unexpanded. You can also set this value to **Expand tabs to spaces only on modified lines** to expand tabs only on lines that have been modified or inserted..
- **Strip trailing spaces** - select one of the following values:
 - **Do not strip trailing spaces** - Leaves all trailing spaces as they are in the file.
 - **Strip all trailing spaces** - Trailing spaces at the end of lines 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 (see [Command Line Switches](#)).
 - **Strip trailing spaces only from modified lines** - Strips trailing spaces at the end of modified or inserted lines only are stripped when the buffer is saved. If you are using the **save** command to save files, use the switch **+SM** to specify this option (see [Command Line Switches](#)).
- **Save files on loss of focus** - When set to **On**, all modified files are saved when you switch to another application.
- **Reset modified lines** - When set to **On**, 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 (see [Command Line Switches](#)). For more information on viewing modified lines, see [Modified Lines](#).
- **Add file to project upon Save As** - This option controls the default value of the **Add to project** option

on the Save As dialog. If you are using the **save** command to save files, use the switch **+P** to specify this option (see [Command Line Switches](#)).

- **Save all prompts to name unnamed files** - Determines whether the `save_all` command requires unnamed files to be named before being saved. When off, a temp name is automatically created and used. This option effects the "Save All" button in the List Modified Buffers dialog when it's displayed before exiting the application or closing a workspace. This does not affect untitled files that have already been given a filename.

Backup File Options

Backup file options are shown below (**Tools** → **Options** → **File Options** → **Backup**). For more information about backing up files, see [File Backups](#).

Backup	
Option	Value
Make backup files	Backup history on save
Backup directory path	
Number of backups to keep for each file	400
Limit size of backup	<input checked="" type="button"/> ON
Maximum size to backup (KB)	20000
Maximum size of individual deltas in archive file:	10
Minimum size for fast delta creation (KB)	1000
Use timeout	<input checked="" type="button"/> ON
Timeout (Seconds)	3
Exclusions	

The options are described as follows:

- **Make backup files** - select one of the following values:
 - **Backup history on save** - (Not in Community edition)Creates a new version each time you save a file. This produces a more fine grained version history than what is available in version control, bridging the gap between checkins. Note that disk space requirements are reduced by storing differences and not whole files unless it takes too long to determine the differences.
 - **Backup file on save ([Backup location option])** - Creates a single backup the first time in an editing session that you save a file. This is useful to preserve the state of a file prior to working on it. See ([Backup location option]) below for information on backup location options.
 - **Off** - No backups are created.
- **([Backup location option])** - When a **Backup file on first save** option is selected, you can specify

where backup files are stored. Select from the following:

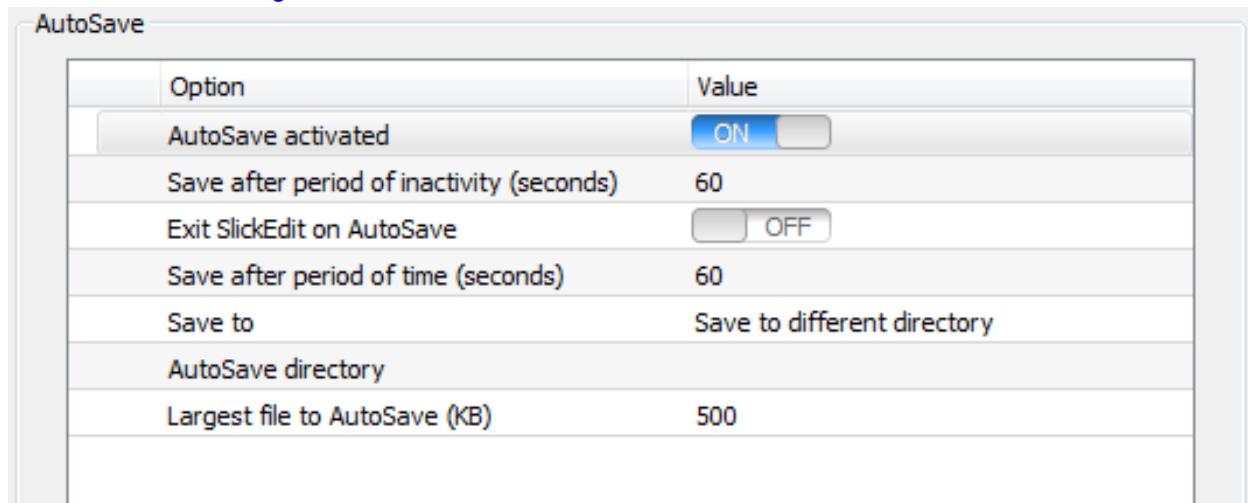
- **(global directory)** - This option places backup files in a single directory. The default backup directory is your unversioned configuration directory (Windows: ... \My Documents\My SlickEdit Config\backup, UNIX: \$HOME/.slickedit/backup, Mac: \$HOME/Library/Application Support/SlickEdit/backup). The backup file gets the same name as the destination file. For example, given the destination file c:\project\test.c (UNIX or Mac: /project/test.c), the backup file will be \$SLICKEDITCONFIG\backup\test.c (UNIX or Mac: \$SLICKEDITCONFIG/backup/test.c). If you are using the **save** command to save files, use the switch **+D** to specify this option (see [Command Line Switches](#)).
- **(global nested directories)** - This option places backup files into a directory derived from concatenating a backup directory with the path and name of the destination file. The default backup directory is your unversioned configuration directory (Windows: ... \My Documents\My SlickEdit Config\backup, UNIX: \$HOME/.slickedit/backup, Mac: \$HOME/Library/Application Support/SlickEdit/backup). For example, given the destination file c:\project\test.c, the backup file will be \$SLICKEDITCONFIG\backup\project\test.c (UNIX or Mac: \$SLICKEDITCONFIG/backup/project/test.c). If you are using the **save** command to save files, use the switch **-D** to specify this option (see [Command Line Switches](#)).
- **(same directory as *.BAK)** - This option places backup files in the same directory as the destination file but the extension is changed to .bak. If you are using the **save** command to save files, use the switch **+DB** to specify this option (see [Command Line Switches](#)).
- **(child backup directory)** - This option places backup files in a directory off the same directory as the destination file. For example, given the destination file c:\project\test.c (UNIX or Mac: /project/test.c), the backup file will be c:\project\backup\test.c (UNIX or Mac: /project/backup/test.c). If you are using the **save** command to save files, use the switch **+DK** to specify this option (see [Command Line Switches](#)).
- **Backup directory path** - Specifies the path to the directory (including the drive, if you wish) in which to place backup files when **Make backup files** is enabled and one of the "Global" **Backup directory options** is selected. Sets the VSLICKBACKUP environment variable. Press **Delete** to clear this field, specifying the default.
- **Number of backups to keep for each file** - Specifies the number of backups to store for each file.
- **Limit size of backup** - When set to **On**, SlickEdit® limits backups to the size specified in the option **Maximum size to backup**.
- **Maximum size to backup** - Specifies the maximum size, in kilobytes, for files in order to be backed up when **Limit size of backup** is enabled.
- **Maximum size of in-line deltas in archive files (KB)** - Approximate maximum size of an in-lined delta. Once the delta is larger than this amount, it is written to an external file.
- **Minimum size for fast delta creation (KB)** - Minimum size of file to use faster delta creation algorithm. Improves speed of creating backup deltas when saving a file.
- **Use timeout** - When set to **On**, SlickEdit® stops comparing after the timeout and stores a backup of

the entire file.

- **Timeout (s)** - Specifies the maximum time, in seconds, to wait for a compare operation before storing entire version of file.
- **Exclusions** - Specifies a list of semicolon delimited ant-like file specifications that should not be backed up using backup history. For example **path1/*.bak** will exclude files under a directory named **path1** and files with extension **.bak**. For more examples, see [Exclusion Examples](#).

AutoSave File Options

AutoSave file options are shown below (**Tools** → **Options** → **File Options** → **AutoSave**). For more information, see [Saving Files](#).



Option	Value
AutoSave activated	<input checked="" type="button"/> ON
Save after period of inactivity (seconds)	60
Exit SlickEdit on AutoSave	<input type="button"/> OFF
Save after period of time (seconds)	60
Save to	Save to different directory
AutoSave directory	
Largest file to AutoSave (KB)	500

The options are described as follows:

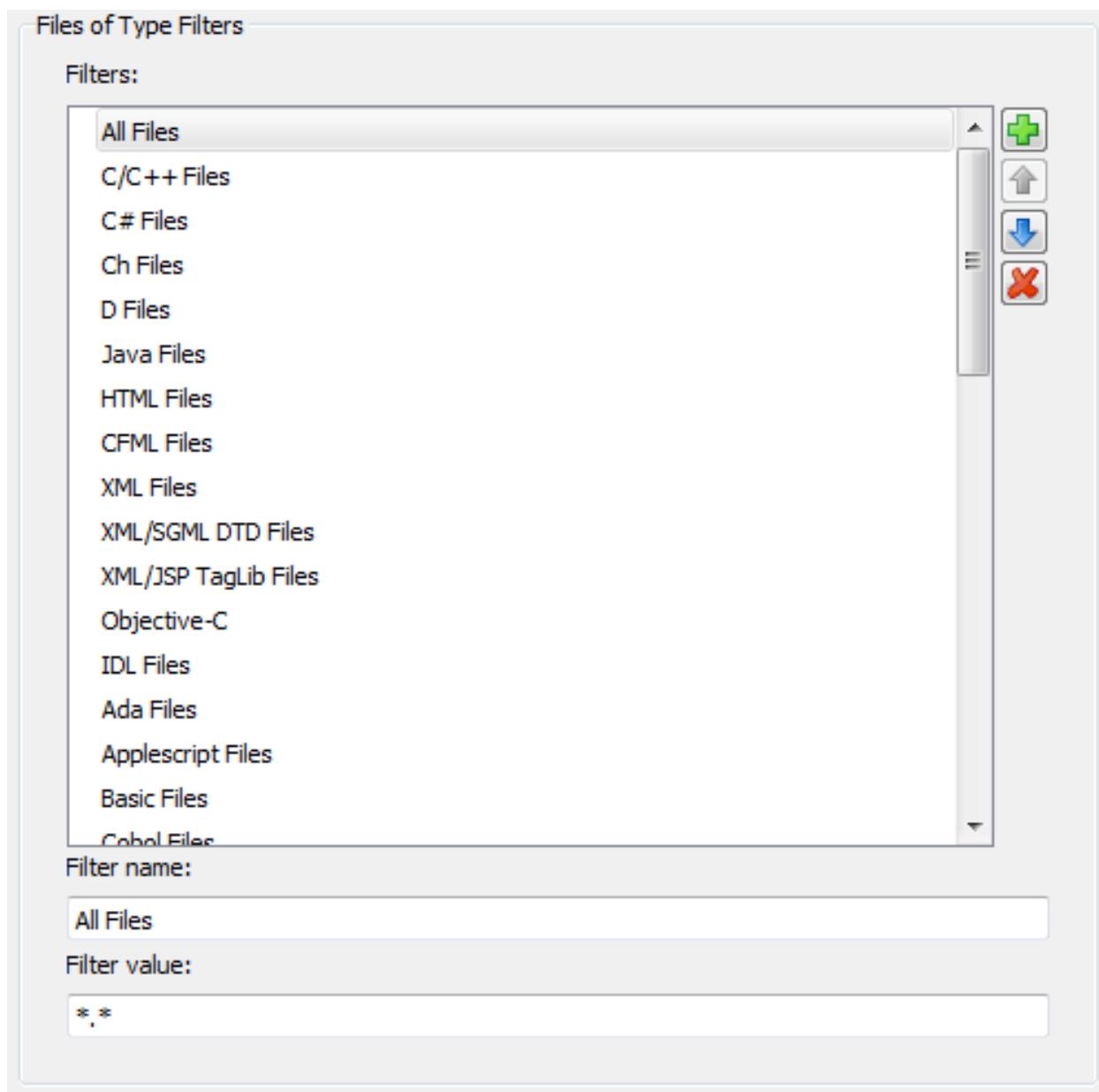
- **AutoSave activated** - When set to **On**, AutoSave is enabled, which prevents you from losing data when an abnormal editor exit occurs (possibly from a power loss). AutoSave creates temporary files in the specified AutoSave directory. Temporary files are only created for modified files and are replaced or deleted when AutoSave runs subsequently. AutoSave files are deleted when they are no longer needed. When AutoSave occurs, the Auto Restore file (vrestore.slk) is written in such a way that SlickEdit will know it is auto-restoring from an AutoSave state and knows the temp names/file names.
- **Save after period of inactivity** - Specifies the amount of idle time, in seconds, after which modified files are saved. Set this value to **0** if you do not want this option ignored.
- **Exit SlickEdit on AutoSave** - When set to **On**, the SlickEdit® application closes after an AutoSave, if AutoSave is enabled.
- **Save after period of time** - Specifies the amount of time, in seconds, after which modified files should be saved. Set this value to **0** if you want this option ignored.
- **Save to** - Specifies save options for AutoSave temporary files. Select from:
 - **Save to different directory** - This option places all AutoSave files in the directory specified by the **AutoSave directory**. Use this option to clean up or find all of the AutoSave files if an abnormal editor

exit occurs.

- **Same name, different extension** - Named files are auto-saved in the same directory as the file that is being auto-saved, but with a different extension. The third character of the extension is replaced with a ~ (tilde) 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. Unnamed files are auto-saved in the directory specified by the **AutoSave directory** field.
- **Same name** - Modified named files are saved back to the original filename (like a normal save). Unnamed files are auto-saved in the directory specified by the **AutoSave directory**.
- **AutoSave directory** - Specifies the directory to use for AutoSave temporary files. If this field is blank, `<configuration_directory>\autosave` is used. To find the location of your configuration directory, see **Help → About SlickEdit**. Press **Delete** to clear this field, specifying the default. Temporary files for unnamed files are saved to this location.
- **Largest file to AutoSave** - Specifies the maximum size, in kilobytes, a file is allowed to have in order to be automatically saved. To have all files auto-saved, set this value to **0**.

Files of Type Filter Options

The Files of Type Filter options are shown below (**Tools → Options → File Options → Files of Type Filters**). They are used to specify the list file filters for the Open, Save Copy As, and Save As dialogs. Each filter defines a set of related file types that are used together.



The order of the filters specifies the order they will appear in the Open, Save Copy As, and Save As dialogs. The first file filter is used to initialize the file list. Use the **Up** and **Down** arrow buttons to change the order. Click the **Delete** button to delete a selected filter from the list.

To add a new filter, click the **Add** button and enter the new filter name. Then set the value using the **Filter value** box on the options screen. Separate each filter with a comma. Place file patterns in parentheses and separate them with a semicolon. Some example filters are:

- Basic Files (*.bas), All Files (*.*)
- C/C++ Files (*.cpp;*.cxx;*.c;*.h), All Files (*.*)

Associate File Types Options

Use these options (**Tools** → **Options** → **File Options** → **Associate File Types**) to set up file associations. Files that are associated run in SlickEdit® when you open them from Windows Explorer. The options, shown below, can also be displayed with the **assocft** command. See [Setting File Associations](#) for more information.

Note

This feature is available for Windows platforms only.

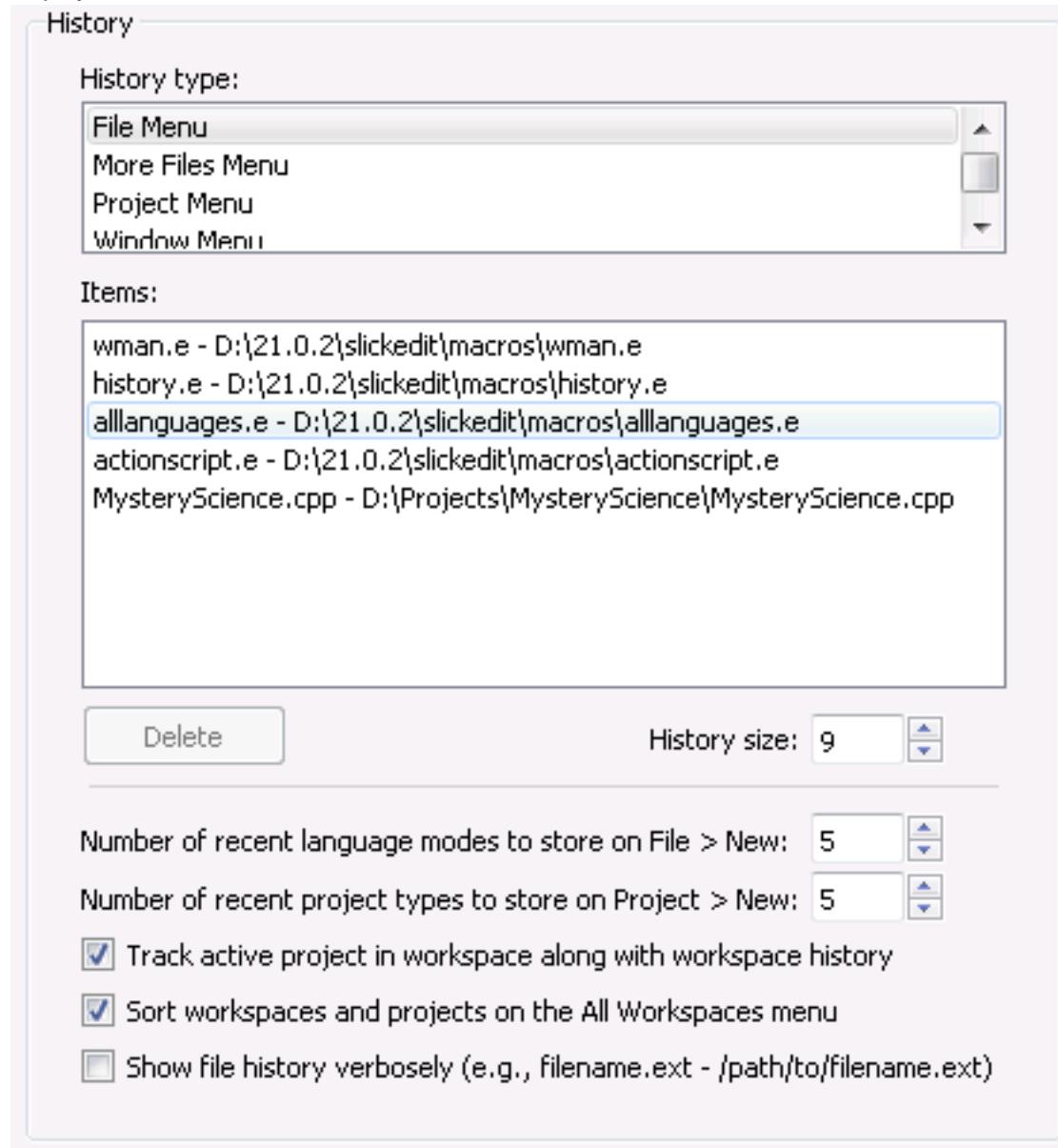
Associate File Types

File Types: View: List extensions by language

The screenshot shows the 'Associate File Types' dialog box. At the top, there's a title bar with the window title and a 'File Types:' label. To the right of the label is a dropdown menu set to 'List extensions by language'. Below this is a large list area containing a tree view of file types. The root node is 'Language Mode > File Extension'. Underneath it, several categories are expanded, such as 'Applescript', 'Awk', 'BibTeX', 'Binary', 'Bourne Shell', 'Bulletin Board Code', 'C Shell', 'C#', 'C/C++', 'CFML', and 'CFScript'. Each category has a list of file extensions under it, with checkboxes indicating their status. For example, under 'C/C++', extensions like 'c', 'c++', 'cc', 'cp', 'cpp', 'cppm', 'cxx', 'h', 'h++', 'hh', 'hpp', 'hxx', 'i', 'inl', 'ixx', 'qth', 'sig', 'tcc', and 'xpm' are listed. The 'hpp' extension is currently selected, highlighted with a blue background. At the bottom of the list, there's a summary section titled 'Current associations' containing a long list of file extensions separated by spaces: ada adb ads ansic as awk bash bourneshell c c++ cc cfc cfm cfml dj djc djs djx cp cpp ppm cs csh csx cxx h h++ hh hpp hxx i inl ixx qth sig tcc xpm.

History Options

These options (**Tools** → **Options** → **File Options** → **History**) let you view a list of recently opened files, projects, and windows, in addition to allowing you to modify options pertaining to how the history is displayed.



You can see the history of items recently opened from the File or Project menu. These are listed in the **History type** box. When you select a menu, the **Items** box updates to show the latest files that were opened from that menu. Click **Delete** to remove selected items from the history list.

You can also specify how many items should appear in each history by changing the value of the **History size**. This value will control the maximum number of items that will appear in the history selected in the **History type** list.

You can also control the number of recently used items appearing on the New File and New Project dialogs, as well as options for how the file history should be displayed and how project history should be

tracked. These options are as follows:

- **Number of recent language modes to store on File > New** - Specifies the number of recently used language editing modes to display at the top of the **Document Mode** list.
- **Number of recent project types to store on Project > New** - Specifies the number of recently used project types to display at the top of the **Project type** list.
- **Track active project in workspace along with workspace history** - When set to On, track the current active project history along with the workspace history on the **Project** menu. This is useful for workspaces that contain multiple projects to make it easy to switch between the projects used most frequently.
- **Track active project in workspace along with workspace history** - When set to On, sort all the workspace and project names under the **All Workspaces** menu. When set to Off, display the menu in the order it was arranged using **Organize All Workspace** dialog. The Move Up and Move Down buttons on the **Organize All Workspaces** dialog are only enabled if this option is Off. See [Organize All Workspaces](#) for more information.
- **Show file name twice** - When set to On, file history is displayed on menus with the filename and the full path to the file, redundantly including the file name in the path, rather than just showing the path part to save space. This setting applies to the history items on the **File**, **Project**, and **Window** menus.
Path length - When file history is displayed on menus, to conserve space if possible, the file path is abbreviated in length to this number of characters.

Application Options

Application options (**Tools** → **Options** → **Application Options**) pertain to the SlickEdit® application. You can specify what parts of SlickEdit should be restored on startup, the amount of virtual memory to use, and more.

Application option categories are:

- [General Application Options](#)
- [Auto Restore Options](#)
- [Virtual Memory Options](#)
- [Exit Options](#)
- [Notification Options](#)
- [Product Improvement Program Options](#)

Directory Project Options

Directory Project Options are shown below. They let you specify how SlickEdit behaves when invoked with a directory as an argument and when a directory is drag/dropped onto SlickEdit

- **Activate Open tool window** - One of the following settings:
 - **Don't activate tool window**- No tool window is activated when SlickEdit is invoked with a directory or a directory is drag/dropped.
 - **Open tool window**- Activate the Open tool window when SlickEdit is invoked with a directory or a directory is drag/dropped.
 - **Projects tool window**- Activate the Projects tool window when SlickEdit is invoked with a directory or a directory is drag/dropped.
- **Include filespecs** - Sets the default Add Tree **Include filespecs** (see [Add Tree](#)). Used when SlickEdit is invoked with a directory or a directory is drag/dropped.
- **Exclude filespecs** - Sets the default Add Tree **Exclude filespecs** (see [Add Tree](#)). Used when SlickEdit is invoked with a directory or a directory is drag/dropped.
- **Recursive** - Sets the default Add Tree option (see [Add Tree](#)) to **Recursive**. Used when SlickEdit is invoked with a directory or a directory is drag/dropped.
- **Create subfolders** - Sets the default Add Tree option (see [Add Tree](#)) to **Create subfolders**. Used when SlickEdit is invoked with a directory or a directory is drag/dropped.
- **Add as wildcard** - Sets the default Add Tree option (see [Add Tree](#)) to **Add as wildcard**. Used when SlickEdit is invoked with a directory or a directory is drag/dropped.
- **Create parent directory folder** - Sets the default Add Tree (see [Add Tree](#)) option to **Create parent directory folder**. Used when SlickEdit is invoked with a directory or a directory is drag/dropped.
- **Don't prompt** - By default, the Project New dialog or Add Tree dialog is displayed when SlickEdit is invoked with a directory or a directory is drag/dropped. Check this option if you want to use your default Add Tree settings.
- **Show new workspace dialog** - When set to **On**, the new workspace dialog will be shown so that you can immediately create a workspace for the directory SlickEdit was invoked with.

Auto Restore Options

Auto Restore options are shown below (**Tools** → **Options** → **Application Options** → **Auto Restore**). They let you specify the elements of your SlickEdit® environment that are restored when you switch workspaces or close and re-open SlickEdit. See [Restoring Settings on Startup](#) for more information about these options.

Application Options

Option	Value
Auto restore files	ON <input checked="" type="radio"/>
Auto restore clipboards	OFF <input type="radio"/>
Auto restore working directory	ON <input checked="" type="radio"/>
Auto restore build window	OFF <input type="radio"/>
Auto restore workspace	ON <input checked="" type="radio"/>
Auto restore workspace files	ON <input checked="" type="radio"/>
Auto restore line modify	OFF <input type="radio"/>
Auto restore selective display	ON <input checked="" type="radio"/>
Auto restore symbol browser tree	OFF <input type="radio"/>
Auto restore projects tree	ON <input checked="" type="radio"/>
Auto restore supported options per monitor configuration	ON <input checked="" type="radio"/>

The options are described as follows:

- **Auto restore files** - When set to **On**, files and editor windows that were open in your last edit session are restored when you start SlickEdit.
- **Auto restore clipboards** - When set to **On**, clipboards are preserved and restored across editing sessions.
- **Auto restore working directory** - When set to **On**, the working directory is preserved and restored across editing sessions.
- **Auto restore build window** - When set to **On**, the concurrent process buffer is preserved and restored across editing sessions.
- **Auto restore workspace** - When set to **On**, the open workspace is preserved and restored across editing sessions. When set to **Off**, the editor opens with no workspace open.
- **Auto restore workspace files** - When set to **On**, files and windows that were open in the workspace previously are restored when you switch between workspaces. See [Workspaces and Projects](#) for more information.
- **Auto restore line modify** - When set to **On**, line modification flags are saved and restored when you save and open files, respectively. Line modification flags for the last 200 files are saved. This option works best when the language-specific Color Coding option **Modified lines** is enabled (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **View**).
- **Auto restore selective display** - When set to **On**, [Selective Display](#) is saved and restored when saving and opening files, respectively. Selective Display for the last 200 files is saved.
- **Auto restore symbol browser tree** - When set to **On**, the symbol browser tree in the [Symbols Tool Window](#) is restored across edit sessions. The currently selected position is always restored regardless of this setting.
- **Auto restore projects tree** - When set to **On**, the projects tree in the [Projects Tool Window](#) is restored across edit sessions. This setting can impact startup performance for very large wildcard projects.

- **Auto restore supported options per monitor configuration** - When set to **On**, auto restore of window and tool window/toolbar layouts is per monitor configuration. Dialog position and size options support per monitor configuration. This can be very useful if you add/remove monitors from your machine.

Virtual Memory Options

Virtual memory options are shown below (**Tools** → **Options** → **Application Options** → **Virtual Memory**).

Note

You must restart SlickEdit® for these settings to take effect.

Virtual Memory	
Option	Value
Spill file path	
Buffer cache size (MB)	2
Tag file cache size (MB)	64
Tag file cache maximum (MB)	512
Maximum number of memory allocators	2

The options are described as follows:

- **Spill file path** - Specifies the directory for spill and temporary files. 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. Press **Delete** to clear this field, specifying the default.
- **Buffer cache size (MB)** - Specifies the maximum amount of memory, in megabytes, used to store text buffer data. A value that is less than zero specifies all available memory. The buffer cache size must be approximately twice the size as the file on disk in order for the entire file to be cached in memory. For example, if a file is 100 megabytes, the buffer cache size would need to be 200 megabytes for the entire file to be cached in memory. There is only one buffer cache for all files being edited.

Caution

If the operating system starts the swapping process memory before the cache is full, performance might be degraded. In practice, it's best to make sure the total memory used by SlickEdit® is no more than about one third of available memory.

- **Tag file cache size (MB)** - (Pro only) Specifies the cache size, in megabytes, for tag files. Tagging

performance can be improved by adjusting this setting 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 as the **Tag file cache size** option under **Tools → Options → Editing → Context Tagging**. For more information about tagging, see [Building and Managing Tag Files](#).

- **Tag file cache maximum (MB)** - (Pro only) Specifies the maximum cache size, in megabytes, for tag files. The tag file cache size can be dynamically adjusted as high as this amount depending on the amount of available memory on your machine at the time SlickEdit is started.

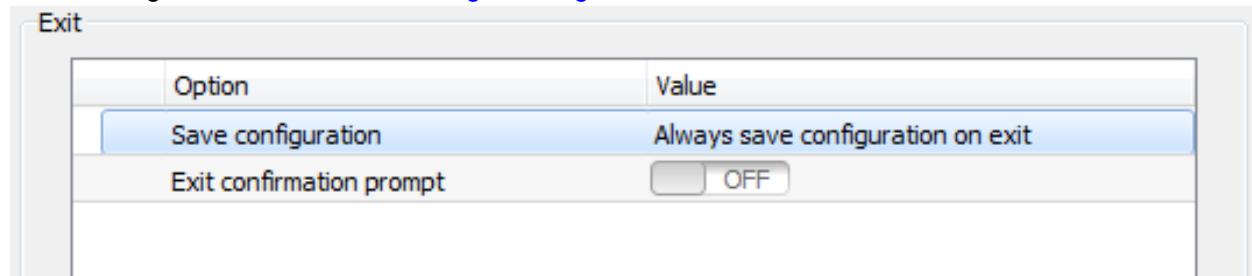
Note that this is the same as the **Tag file cache maximum** option under **Tools → Options → Editing → Context Tagging**. For more information about tagging, see [Building and Managing Tag Files](#).

- **Maximum number of memory allocators** - Specifies the maximum number of memory allocators. Making this value smaller will reduce SlickEdit's memory usage but will also reduce threading performance due to increased contention when allocating memory. This primarily effects background tagging performance. It's easier to test for memory leaks when this option is set to 2.

Tagging Performance	Recommended Setting
Minimum	8 MB
Default	64 MB
Ideal	Sum of tag file sizes
Maximum	25% of physical system memory

Exit Options

Exit options are shown below (**Tools → Options → Application Options → Exit**). For more information about exiting the editor, see also [Exiting the Program](#).



The options are described as follows:

- **Save configuration** - Specifies whether or not configuration changes are made immediately, or upon exit, and whether or not a confirmation prompt is displayed. If set to **Save configuration immediately**,

Application Options

configuration changes will be saved immediately after changes are detected.

- **Exit confirmation prompt** - When set to **On**, SlickEdit always displays a confirmation prompt prior to exiting the application.

Notification Options

The Notifications system informs you when SlickEdit performs automatic actions. These actions are divided into three groups: background processes, feature notifications, and informational messages. Background processes include features like background tagging, which run while you work. Feature notifications provide information about features which may insert more text in the buffer than you have typed or format your code differently than your settings. For more information see [Feature Notifications](#). Informational messages include warnings, product update notifications, and debugging status messages.

The screenshot shows the 'Notifications' section of the SlickEdit application options. It includes sections for 'Background Processes' and 'Feature Notifications'. Under 'Background Processes', there are two checkboxes: 'Hide icon' and 'Disable all pop-ups'. Under 'Feature Notifications', there are also two checkboxes: 'Hide icon' and 'Disable all pop-ups'. Below these is a radio button group: 'Turn all notifications to:' with 'Dialog' selected, and 'Set notification method by feature.' which is selected. Further down, there are dropdown menus for 'Notification Type' (set to 'Feature Notifications: Auto-Close Visited File') and 'Notification level' (set to 'Status line icon, no pop-up'). A checked checkbox 'Log in Notifications tool window' is present. A note below it states: 'When a file is opened as a result of a symbol navigation or search operation, but not modified, it can be closed automatically after navigating away from it.' At the bottom, there are links for 'Auto-Close Visited File Options' and 'Auto-Close Visited File Help'. A note at the bottom of the dialog box reads: 'SlickEdit has many helpful features that assist you while editing to improve efficiency and productivity. However, since these features often alter text while you type, users unfamiliar with the feature may be surprised or confused. To inform you when a feature has made a change in your file, a notification is displayed. The Notification Options will help you to understand and customize editing features, as well as which notifications are displayed for each one.'

The Notifications options screen is shown above. From this form, you can set how you want to be notified about different events.

All background processes use a status bar icon and popup to notify you that they are running. If you wish to disable just the popup or even both the icon and the popup, you can do so using the checkboxes in the **Background Processes** group.

Feature Notifications and information messages offer more options to determine how you wish to be notified about what automatic events happen within the editor. You can turn all notifications to the same value or set them individually by feature. If you want to be notified of all features by a message on the status line, select **Turn all notifications to** and select **Status line message**. If you want to be notified about Adaptive Formatting with a dialog but only want notifications about Syntax Expansion to appear in the status line, select **Set notification level by feature**, choose the appropriate notification type from the drop-down list and then set the level in the **Notification level** drop-down list. You can also set which notifications appear in the **Notifications** tool window by checking the **Log in Notification tool window** checkbox.

The Notifications options screen contains the following settings:

- **Turn all notifications to** - choose a notification level for all features. Note that choosing to have all notifications display a dialog can create a lot of disruption.
- **Set notification level by feature** - allows you to set the notification level differently for each notification type. Since some features are more surprising than others, you may wish to have a more intrusive notification.

The following options are only available if you have selected **Set notification level by feature**.

- **Notification type** - select the notification type for which you want to set the notification level.
- **Notification level** - sets the kind of notification for the selected feature. You can choose from the following:
 - **Dialog** - displays a dialog notification. This requires you to click a button to continue. This is used for the most surprising features.
 - **Status line icon with pop-up** - activates the status line icon and pops up a message. This option is not available when the status line icon or pop-ups have been disabled.
 - **Status line icon without pop-up** - activates the status line icon, but does not pop up a message. This option is not available when the status line icon has been disabled.
 - **Message line** - a short text message identifying the feature is displayed on the SlickEdit message line at the bottom of the application window.
 - **None** - suppresses all notifications.
- **Log in Notification tool window** - By checking this, then all notifications of this type will be listed in the **Notifications** tool window.
- There are two links at the bottom of this group that allow you to navigate to the options screen for the

selected feature or view help about the feature.

Product Improvement Program Options

You can opt in or out of the SlickEdit Product Improvement Program using the options found at (**Tools** → **Options** → **Application Options** → **Product Improvement Program**). These options are shown below.

Product Improvement Program



The SlickEdit Product Improvement Program collects data on how you use the application. Usage information is then sent back to SlickEdit so that we may be able to determine how to best improve the user experience. All information is completely anonymous.

The program will NOT gather any machine names, file names, project names, workspace names, or send in any source code. It does gather information on file types, project types, tool windows used, and options settings.

[Product Improvement Program Info](#)

Participate in the Product Improvement Program

To participate in the program, check the **Participate in the Product Improvement Program** checkbox. The options page contains information about the program, as well as a link to additional information found on SlickEdit's website. For more information about the Product Improvement Program, see [Product Improvement Program](#).

Network & Internet Options

Network and Internet options (**Tools** → **Options** → **Network & Internet Options**) are used to configure the IP setting, FTP connection profiles and options, proxy settings, and more.

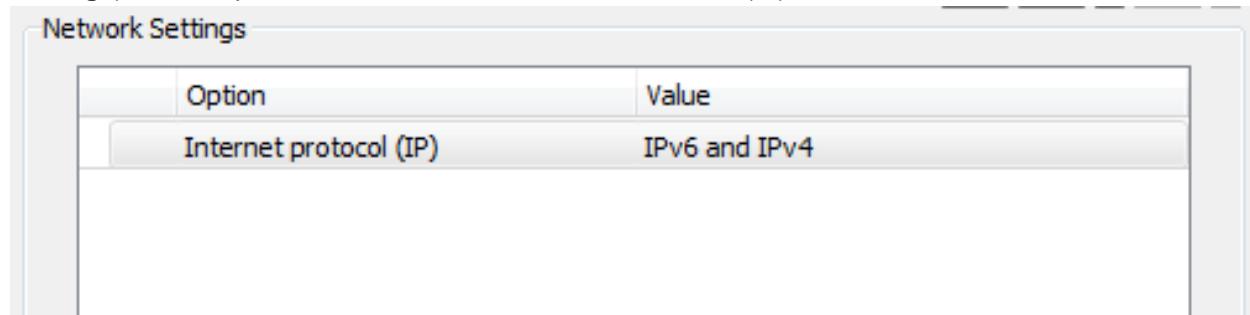
Network and Internet option categories are:

- [Network Settings](#)
- [FTP Default Options](#)
- [URL Mapping Options](#)
- [URI Scheme Options](#)
- [Proxy Settings](#)

- [Web Browser Setup Options](#)

Network Settings

Network settings are shown below (**Tools** → **Options** → **Network & Internet Options** → **Network Settings**). These options are used to set the Internet Protocol (IP) version.



Option	Value
Internet protocol (IP)	IPv6 and IPv4

The option is described as follows:

- **Internet Protocol (IP)** - The IP setting affects how addresses are chosen when connecting to a host. Features that use this setting include [FTP](#), [SFTP](#), and [Opening URLs](#). The options are mutually exclusive: Select **IPv6** and **IPv4** (the default) to automatically select the address when connecting to a host. Select **IPv4 only** to force IPv4 address connections or **IPv6only** to force IPv6 address connections.

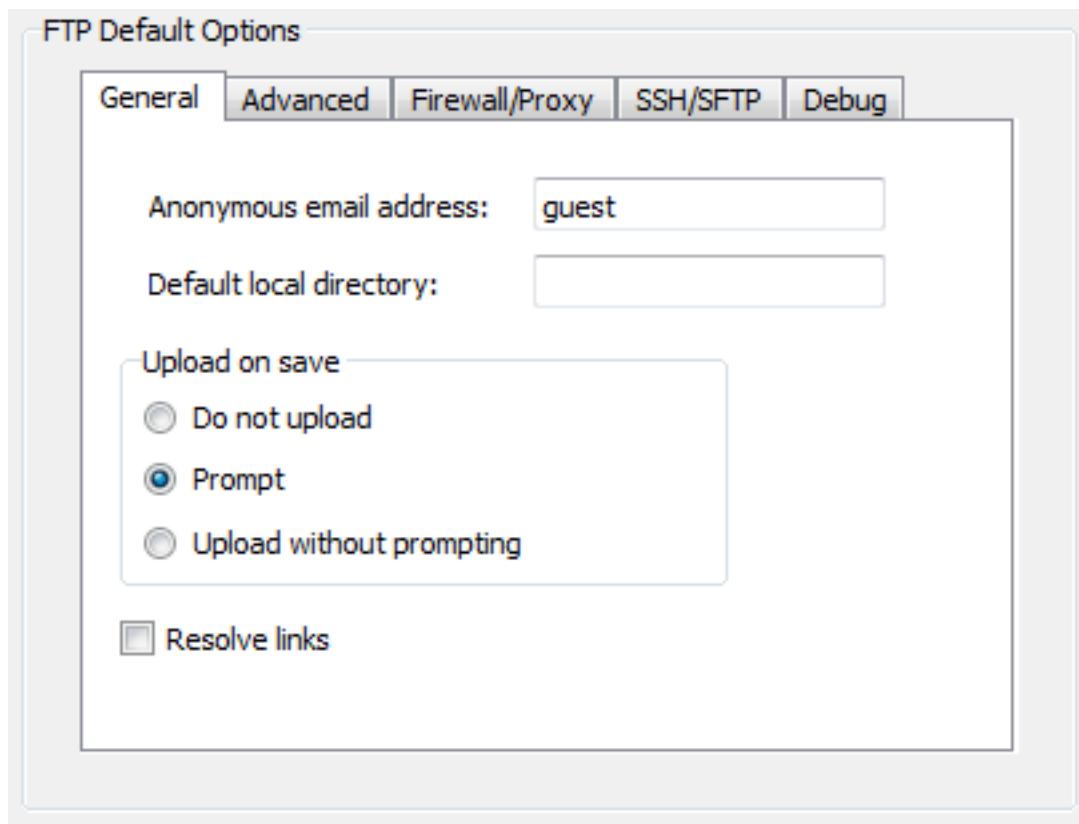
FTP Default Options

Default FTP options are shown below (**Tools** → **Options** → **Network & Internet Options** → **FTP Default Options**). This Options screen is also displayed when you click **File** → **FTP** → **Default Options** or when you click the **Default Options** button on the FTP Profile Manager. Options include the ability to set the default local directory, specify preferences such as the default time-out and port information, enable firewall/proxy support, and configure SSH information. See [FTP](#) for more information.

The options are categorized into the following tabs:

- [FTP Default Options General Tab](#)
- [FTP Default Options Advanced Tab](#)
- [FTP Default Options Firewall/Proxy Tab](#)
- [FTP Default Options SSH/SFTP Tab](#)
- [FTP Default Options Debug Tab](#)

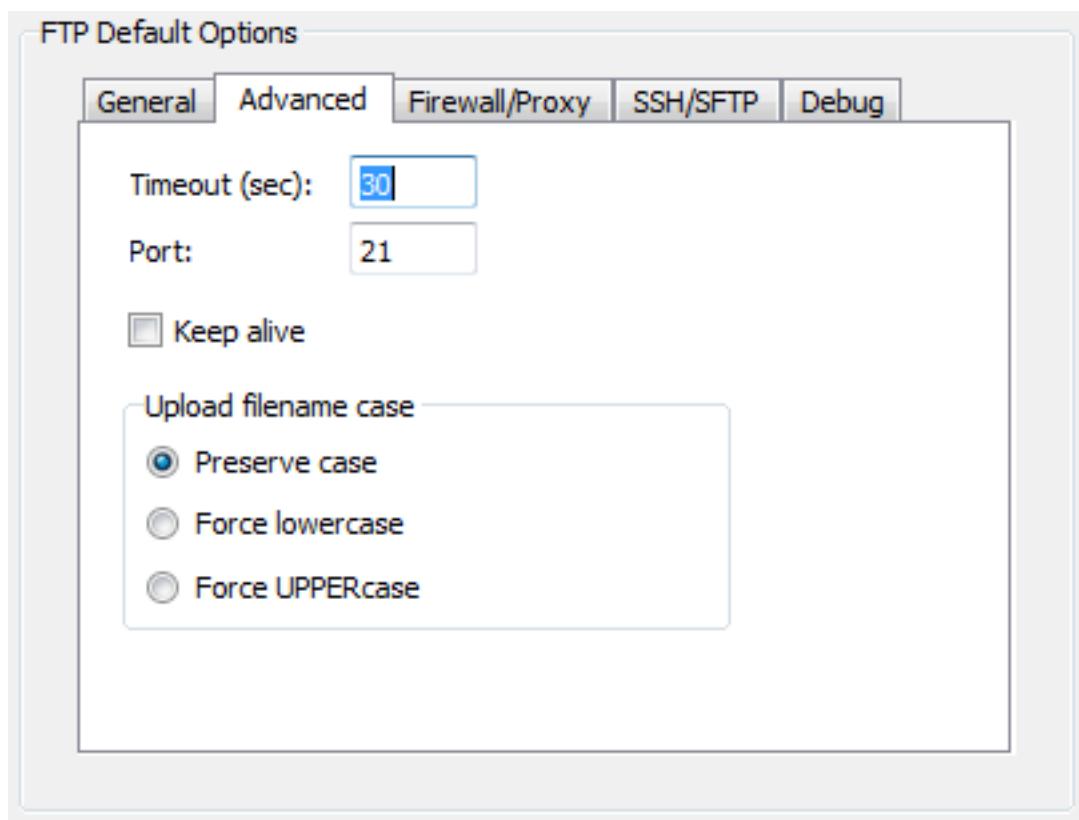
FTP Default Options General Tab



The General tab contains the following:

- **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.
- **Upload on save** - Select from the following:
 - **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.

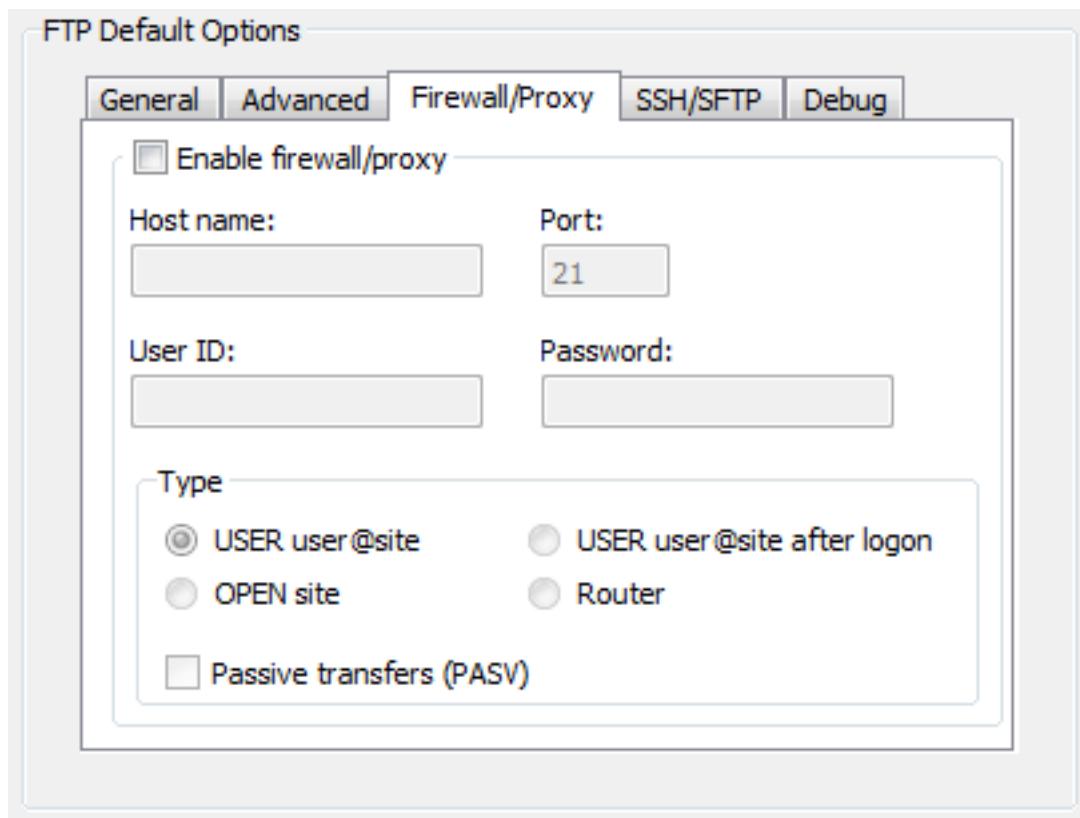
FTP Default Options Advanced Tab



The Advanced tab contains the following:

- **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.
- **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 file name based on the local file name.

FTP Default Options Firewall/Proxy Tab



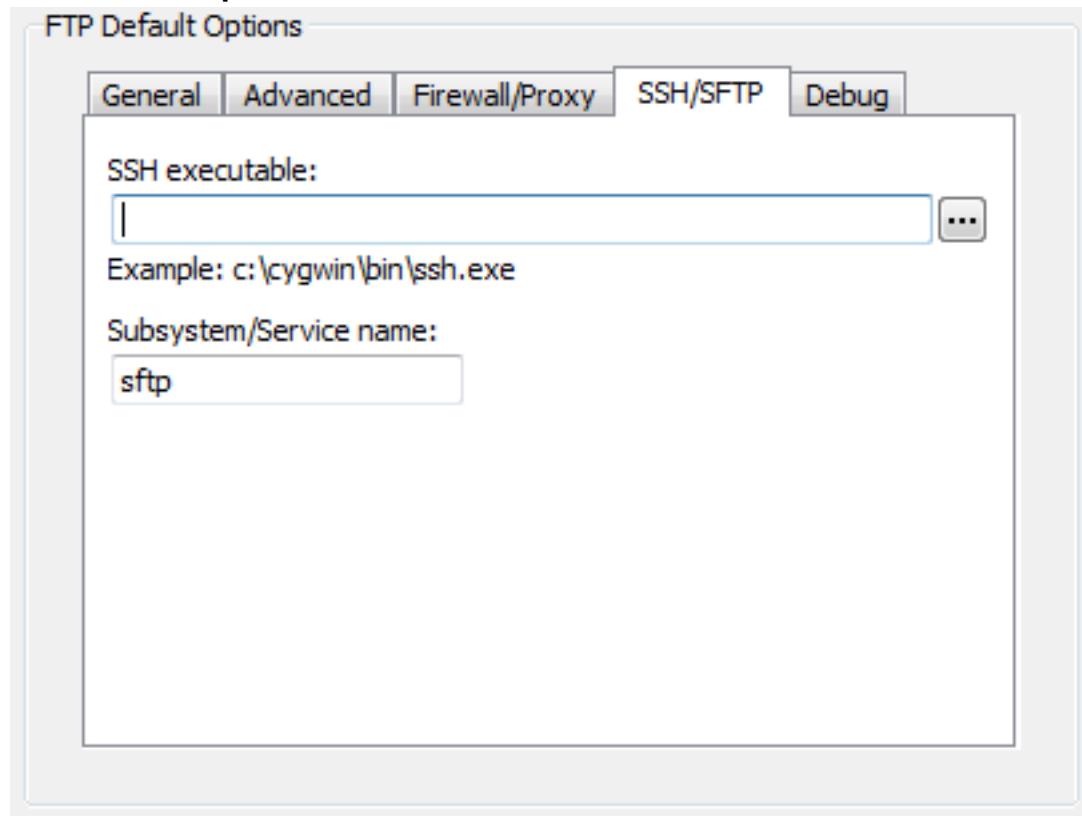
The Firewall/Proxy tab contains the following:

- **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.
- **Type** - Select from the following:
 - **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.
 - **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.
 - **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.
 - **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®.

FTP Default Options SSH/SFTP Tab



The SSH/SFTP tab contains the following options:

- **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) 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**.

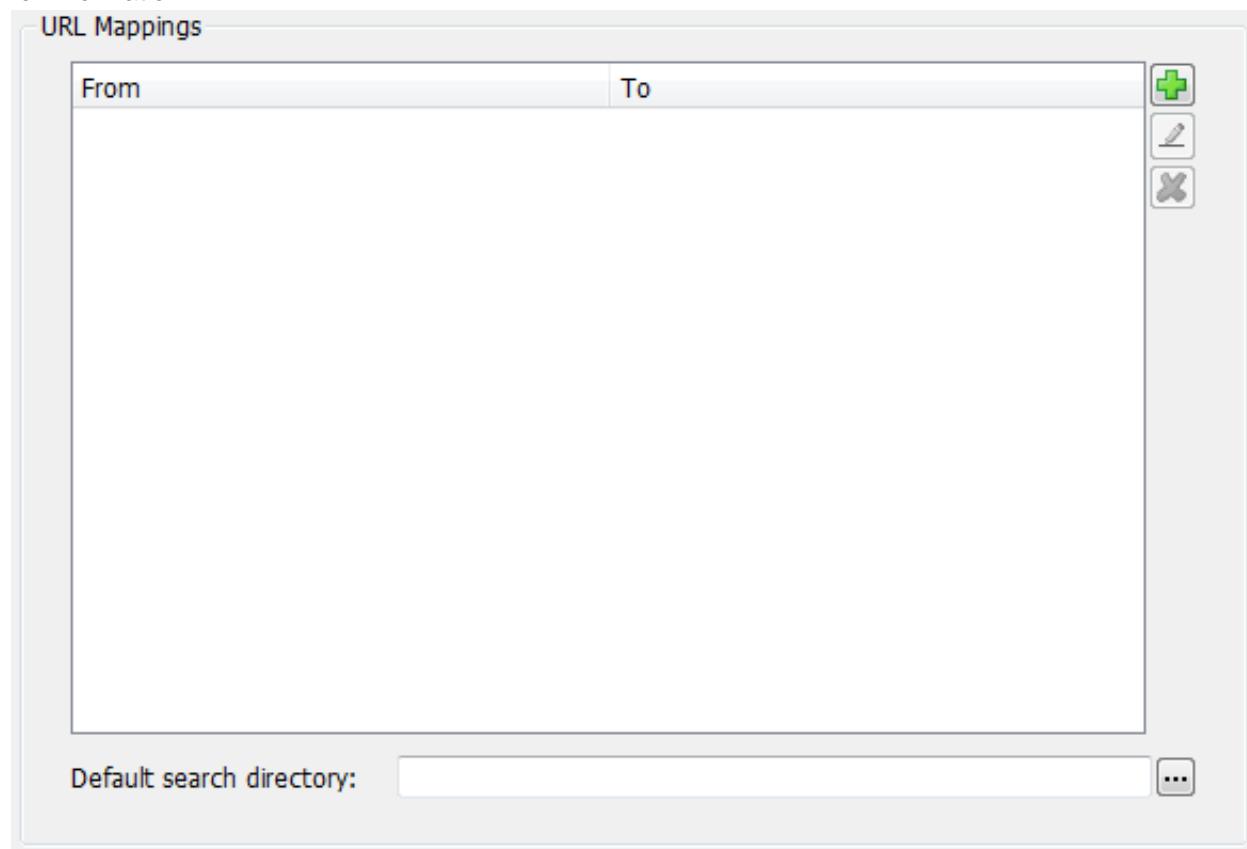
FTP Default Options Debug Tab

This tab is used by SlickEdit® Product Support to debug customer FTP/SFTP issues.

URL Mapping Options

URL Mapping options are shown below (**Tools → Options → Network & Internet Options → URL**

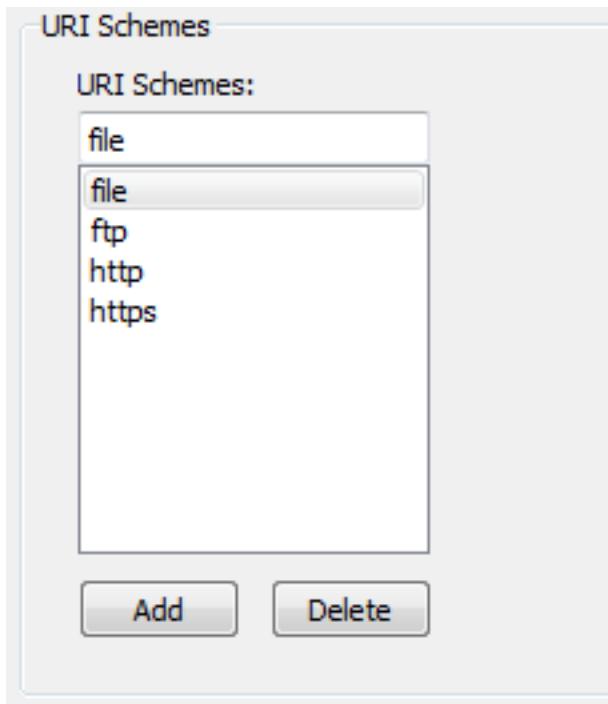
Mappings). These options let you map a URL path or file to a local or remote path or file, so you can work offline or from a test location that contains DTDs referenced by your XML files. See [URL Mapping](#) for information.



To add a new URL mapping, click the **Add** button (or click **<add>** in the **From** column) and type the URL that will be mapped to a different location. Then in the corresponding field in the **To** column, type the location to use for this URL. To delete the selected mapping, click **Delete**.

URI Scheme Options

URI Scheme options are shown below (**Tools** → **Options** → **Network & Internet Options** → **URI Schemes**). These options let you specify URL types that SlickEdit should recognize as hyperlinks in the editor. See [Navigating to URLs](#) for information.

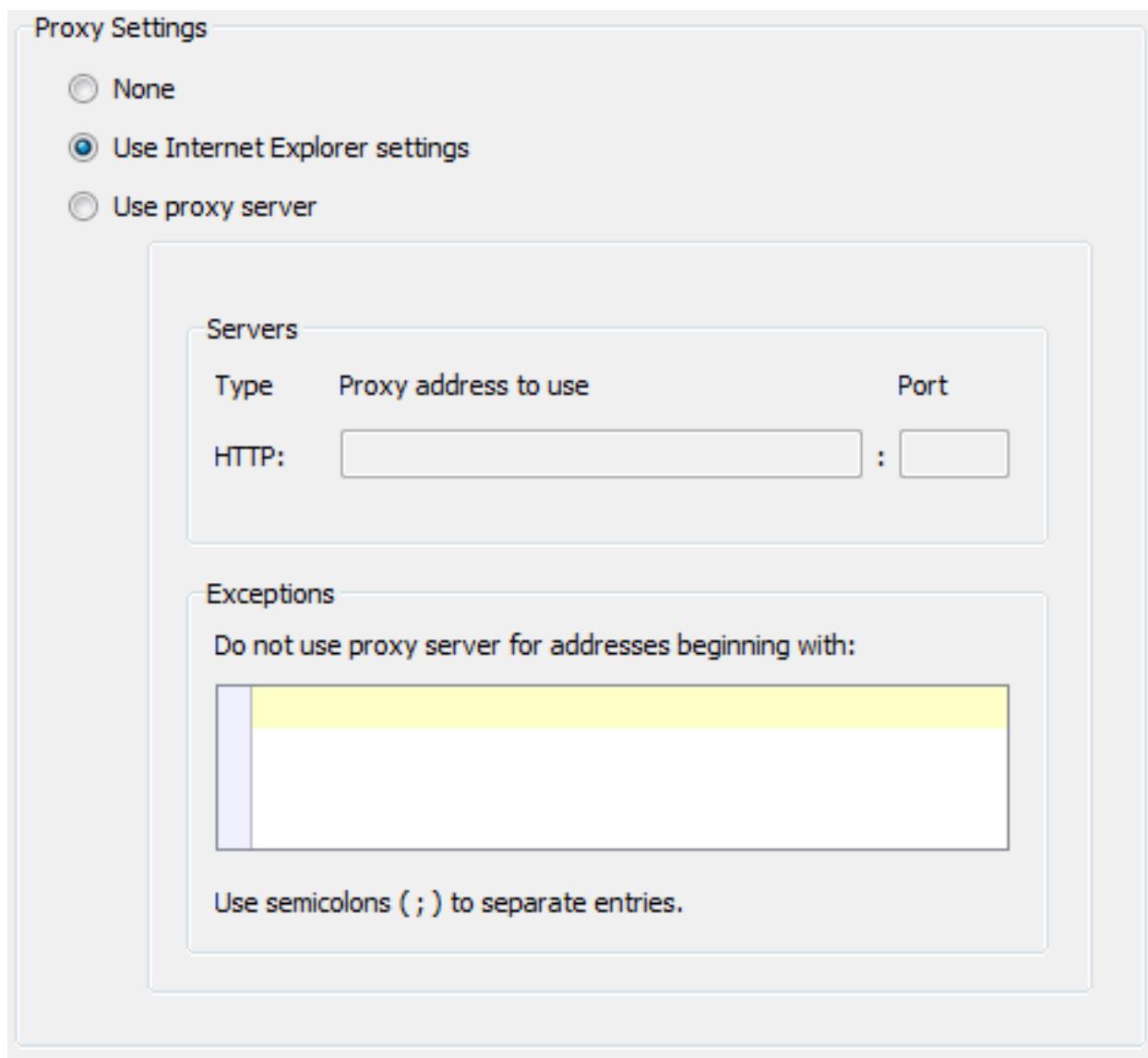


To add a new URI scheme, click **Add**, and type the scheme you want to add. Do not include the "://". For example, if you want to be able to click on e-mail addresses in the editor, you could add a `mailto` scheme. After adding a scheme, newly recognized URLs are underlined in the edit window as soon as it regains focus.

To delete the selected scheme, so that those URLs are not hyperlinked, click **Delete**.

Proxy Setting Options

Proxy options are shown below (**Tools** → **Options** → **Network & Internet Options** → **Proxy Settings**). These options allow you to configure a proxy server to use when SlickEdit® needs to use an Internet connection. Internet Explorer settings are used by default.

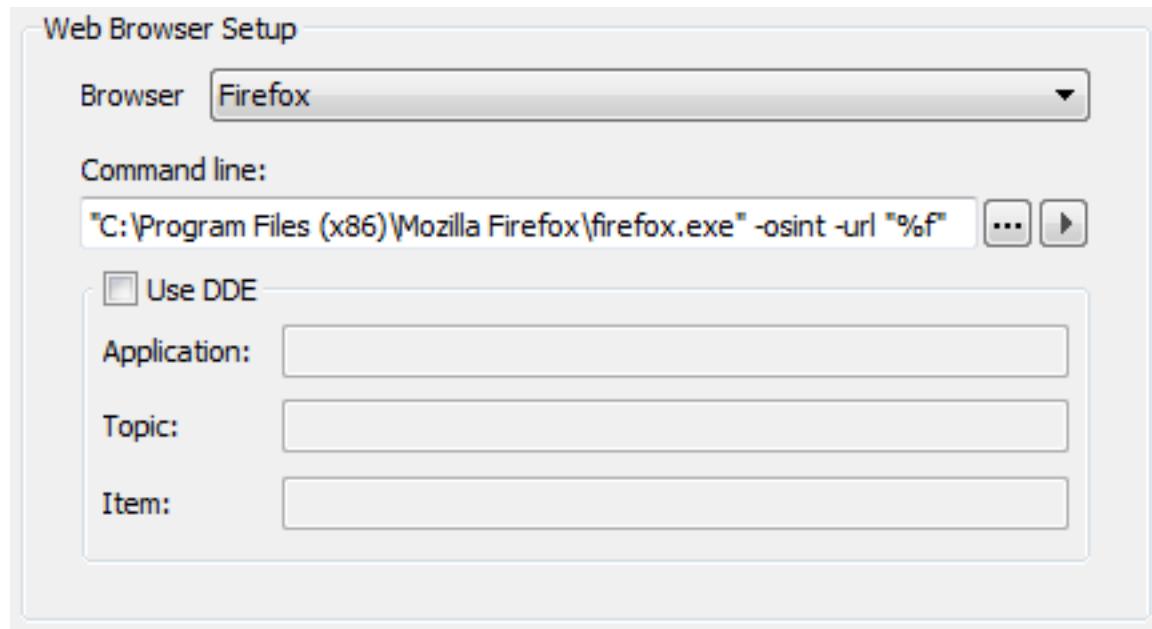


The following options are mutually exclusive:

- **None** - Specifies that no proxy server is used.
- **Use Internet Explorer settings** - If selected, Internet Explorer settings will be used, and the remaining options and fields on the page are inapplicable and therefore dimmed.
- **Use proxy server** - If selected, the remaining options and fields are applicable and 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 (;).

Web Browser Setup Options

Web browser setup options are shown below (**Tools** → **Options** → **Network & Internet Options** → **Web Browser Setup**). Use these options to specify the browser to use when SlickEdit® needs to launch one. Selecting a preferred browser automatically sets the defaults for the other options on this form.



The following options 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 platforms - Your Web browser is automatically detected.
 - UNIX and macOS platforms - You need to specify which Web browser you are using. In addition, you need to give the full path to the program executable.
- **Command line** - 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.

Search Engines Options

Search Engines options are shown below (**Tools** → **Options** → **Network & Internet Options** → **Search Engines**). Choose the default search engine and/or add new search engines. The default search engine is used for **Internet search** actions in API Help

Tool Options

The screenshot shows a dialog box titled "Search Engines". It contains a table with two columns: "Search engine" and "URL". The table lists four search engines: Google, DuckDuckGo, Yahoo!, and Bing, each with its corresponding URL. To the right of the table are four buttons: "Add...", "Edit...", "Delete", and "Set Default".

Search engine	URL
Google	http://www.google.com/search?ie=UTF-8&q=%s
DuckDuckGo	http://www.duckduckgo.com/?q=%s
Yahoo!	https://search.yahoo.com/search?ei=UTF-8&p=%s
Bing	https://www.bing.com/search?q=%s

The following options are available:

- **Add...** - Adds a new search engine.
- **Edit...** - Allows you to modify the settings for the selected search engine.
- **Delete** - Deletes the selected search engine. Built-in search engines can't be deleted.
- **Set Default** - Sets the default search engine.

Tool Options

Options for tools (**Tools** → **Options** → **Tools**) pertain to tools such as Spell Check, and utilities supported by SlickEdit®, such as version control (CVS, Subversion, etc.).

Tools option categories are:

- [Spell Check Options](#)
- [Version Control Setup Options](#)
- [Configure Error Parsing](#)

Spell Check Options

Spell Check options are shown below (**Tools** → **Options** → **Tools** → **Spell Check**). These settings control the behavior of Spell Check in the editor (**Tools** → **Spell Check**). You can also access these options from the main menu item **Tools** → **Spell Check** → **Spell Options**, or by using the **spell_options** command.

Spell Check	
Option	Value
User list 1	User1
User list 2	User2
Ignore UPPERCASE words	<input checked="" type="checkbox"/> ON
Ignore URLs	<input checked="" type="checkbox"/> ON
Detect repeated words	<input checked="" type="checkbox"/> ON

The Spell Check Options screen contains the following:

- **User list 1 or 2** - When a word is not found during a spell check and you add the word to **User list 1** or **2**, the word is added to the associated profile specified in these fields. User modified spelling profiles are stored in `user.cfg.xml`. You may create as many spelling profiles as you want. All spelling profiles are combined when SlickEdit is invoked.
- **Ignore all UPPERCASE words** - When set to **On**, all words in uppercase are ignored during a spell check operation. This applies to all spell check operations.
- **Ignore URLs** - When set to **On**, all words in URLs are ignored. This applies to all spell check operations.
- **Detect repeated words** - When set to **On**, words that are repeated twice in a row are detected during a spell check operation. Spell checking while typing does not support this option.

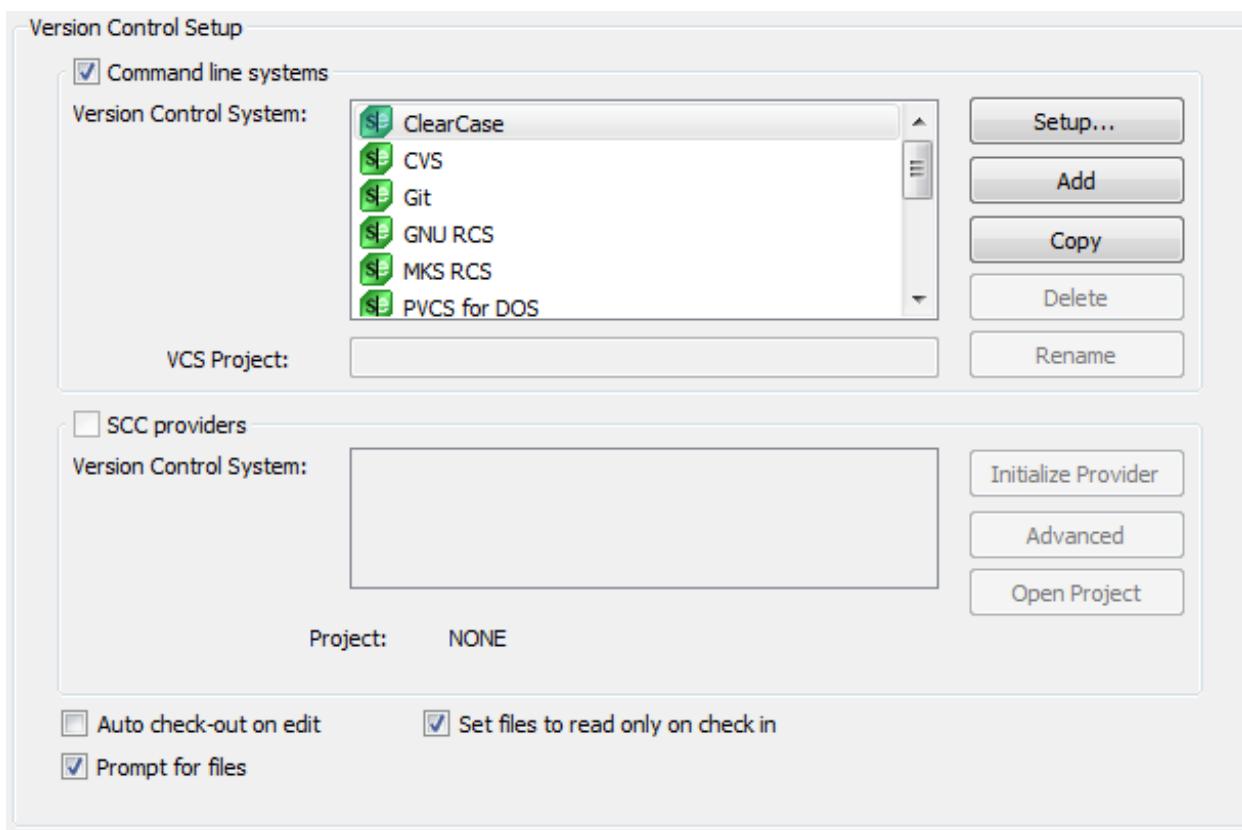
Version Control (Pro only)

SlickEdit provides seamless integration with several version control systems. Version Control configuration options are found at (**Tools** → **Options** → **Tools** → **Version Control**).

Version Control Setup Options (Pro only)

Version Control setup options are shown below (**Tools** → **Options** → **Tools** → **Version Control** → **Version Control Setup**). Use these options to enable version control support and to specify configuration information and preferences. For more information, see [Version Control](#).

Tool Options



This screen contains the following:

- **Command line systems support** - The following items are used to manage your command line version control systems.
 - **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** - 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 takes you to the section of the options dialog that configures the version control system you have selected.
 - **Add** - 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. The newly-added version control system will be selected in the list.
 - **Copy** - To add a new command line version control system with the same settings as an existing system, click the **Copy** button and you will be prompted to fill in a name for the new version control system. The new version control system will be selected in the list.
 - **Delete** - To remove the support of a command line version control system, select the system in the list and then click this button. This action is only available for version control systems you previously added, not systems with built-in SlickEdit support.
 - **Rename** - To rename a command line version control system, select the system and then click the

Rename button. You will be prompted for the new name. This action is only available for version control systems you previously added, not systems with built-in SlickEdit support.

- **SCC providers** - This section is for SCC version control systems that are registered.
- **Other**
 - **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 Providers

SlickEdit provides the capability for integration with several different Version Control Providers. Each provider is customizable on an individual basis, since your settings for one provider may be different from another.

Use the Version Control Providers section of the Options dialog (**Tools** → **Options** → **Tools** → **Version Control** → **Version Control Providers**) to control the behavior of SlickEdit for specific version control systems. Each provider has its own node, some of which expand to show multiple options pages. If you add your own version control provider, it will also be added to the Options dialog under this node. Use [Version Control Setup Options](#) to add or remove your own custom provider.

Most command line version control systems, including user-defined systems, use the options forms shown below. However, CVS, Git, and Subversion have their own specific set of options. For more information about these options, see [CVS Options](#), [Git Options](#), and [Subversion Options](#).

Version Control Commands Setup Dialog

You can define what commands are used to run specific version control actions. To configure these commands, go to **Tools** → **Options** → **Tools** → **Version Control** → **Version Control Providers**, then expand your provider of choice, then select **Commands**. These options are available for all command line systems except CVS and Subversion. As an example, the ClearCase Commands dialog is shown below.



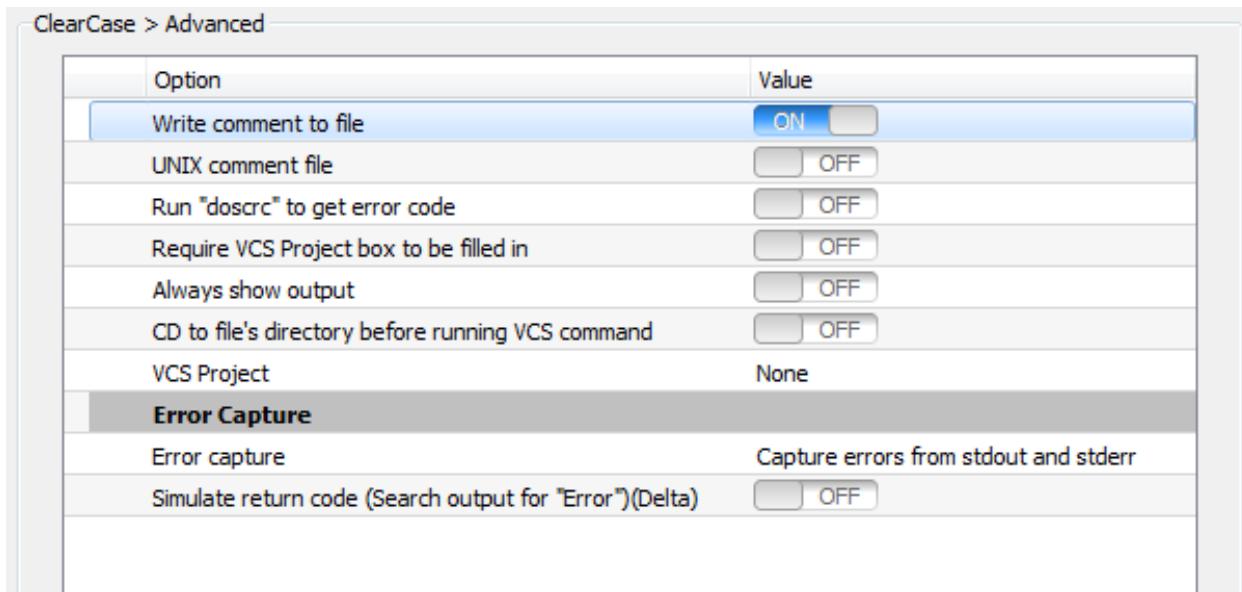
The following items appear on this setup dialog:

- **Version Control Command** - This is a list of commands that can be run from the editor. To edit how a command is performed, select the command in this list and edit the **Command** text box described below.
- **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.

Version Control Advanced Settings

You can access advanced setup options for each version control system that supports them. From the Version Control Setup options (**Tools** → **Options** → **Tools** → **Version Control**), expand the Version Control Providers node and select the version control system you wish to set up. Click the **Advanced** options node to access the advanced options. The options are similar for each version control system. For example, below is a screen capture of the ClearCase Advanced Setup dialog. For more information about working with Version Control, see [Version Control](#).

Tool Options



The following options are available:

- **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** - Used to help get the return code, especially from 16-bit applications. Has no adverse affect, so it is best to leave this on under 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.
- **VCS Project** - Specifies VCS Project behavior. Select one of the following:
 - **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 (RCS)** - 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 **sstree** and **ssonedir** because it requires the **locate** command to execute for each file.
- **None** - Select this style if none of the others apply.
- **Error Capture**
 - **Error Capture** - Specifies how error output from the version control system should be capture. The choices are as follows:**Error Capture** - Specifies how error output from the version control system should be capture. The choices are as follows:
 - **Capture errors from stdout and stderr** - If the version control executable returns non-zero (indicating an error), display output from both stdout and stderr. Most command line version control systems will behave this way.
 - **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 file name used is the default file name 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.
 - **Internal command lookup** - Check to see if the command specified is a Slick-C® command, otherwise run as an OS command.
 - **Simulate return code (Search output for "Error") (Delta)** - Searches for Error or Warning. Displays output from the version control system if either are found.

Note

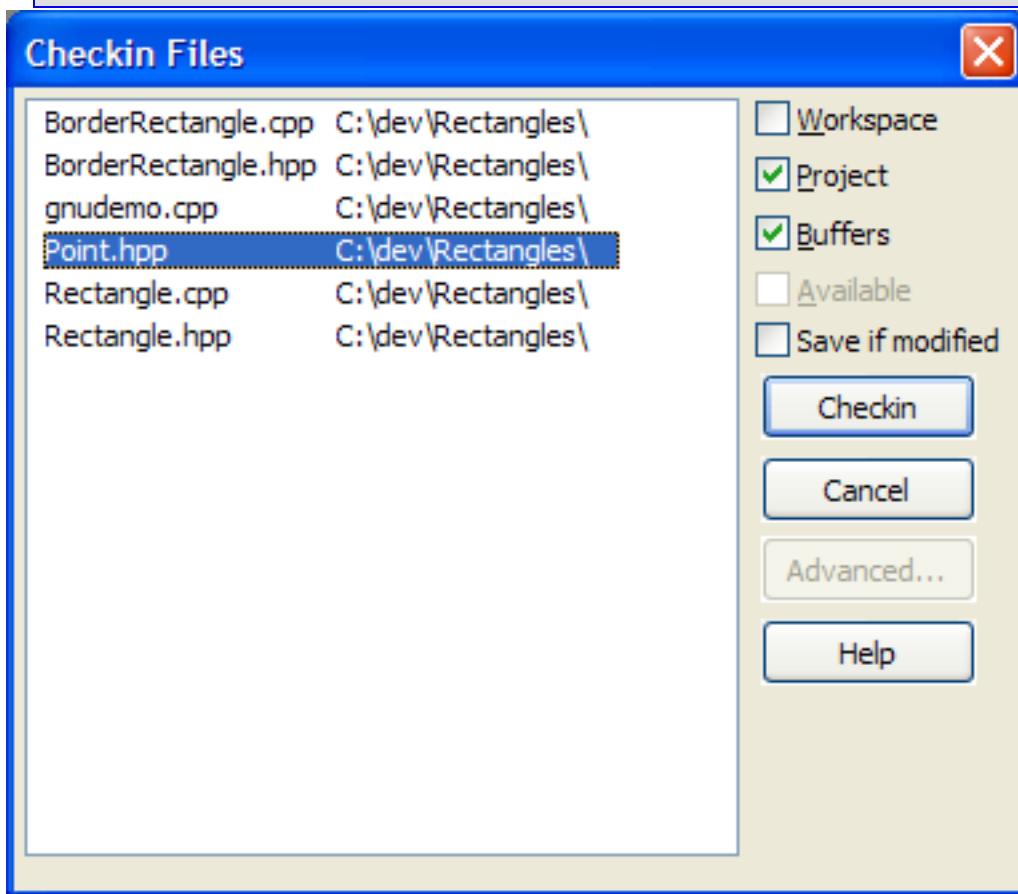
If you are using Source Safe for Windows, use the SCC interface.

Checkin/Checkout Files Dialog

When the option **Prompt for files** is checked in the Version Control Setup options, 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.

Configure Error Parsing (Pro only)

Options used to configure error parsing are shown below (**Tools** → **Options** → **Tools** → **Configure Error Parsing**). They can also be accessed by clicking **Build** → **Configure Error Parsing** from the main menu, or by using the **configure_error_regex** command. See [Parsing Errors with Regular Expressions](#) for information about these features.

Configure Error Parsing

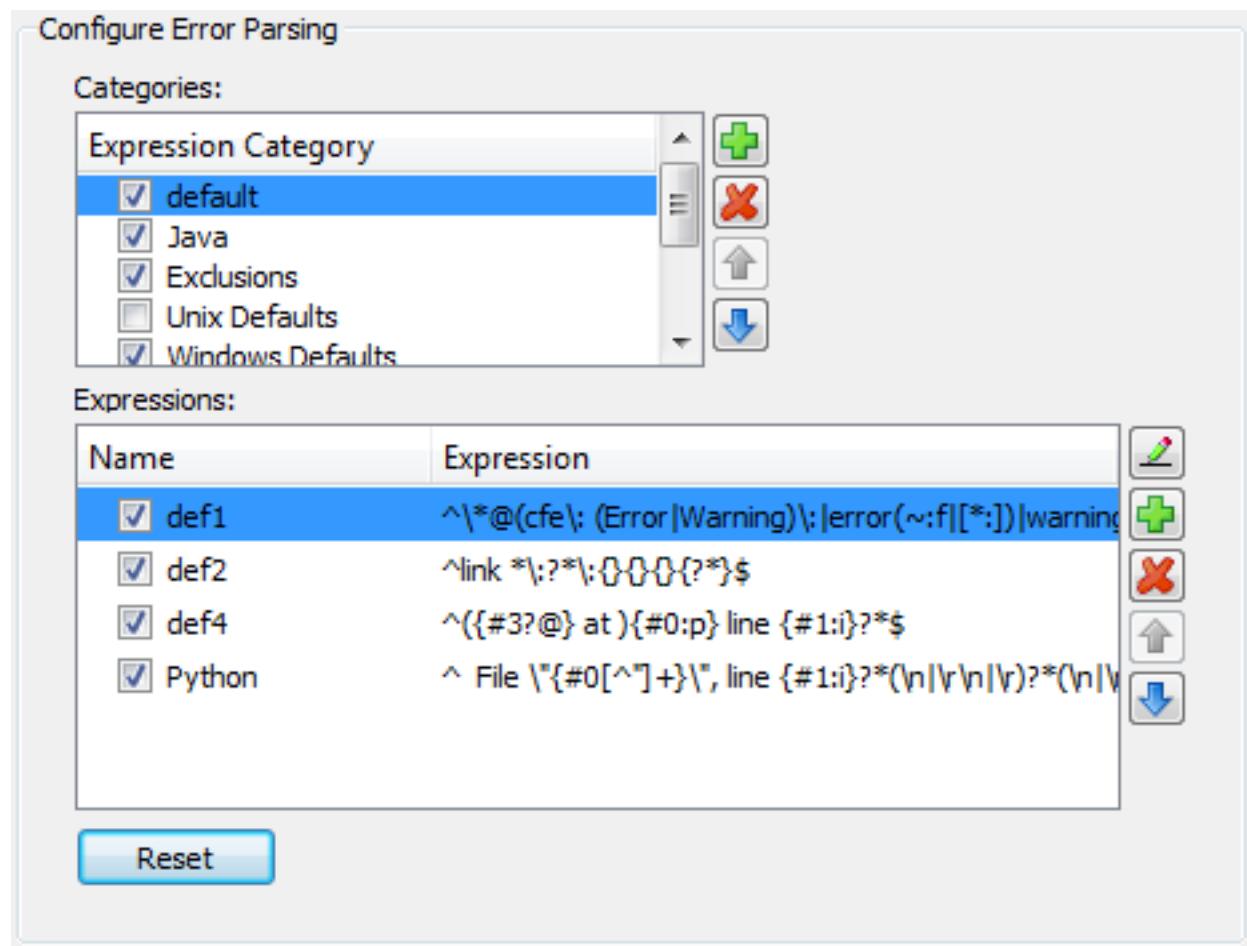
Categories:

Expression Category
<input checked="" type="checkbox"/> default
<input checked="" type="checkbox"/> Java
<input checked="" type="checkbox"/> Exclusions
<input type="checkbox"/> Unix Defaults
<input checked="" type="checkbox"/> Windows Defaults

Expressions:

Name	Expression
<input checked="" type="checkbox"/> def1	<code>^@\{cfe\}; (Error Warning)\}; error(~:f [*]) warning</code>
<input checked="" type="checkbox"/> def2	<code>^link *\;?*\{\-\-\-\}\?*\\$</code>
<input checked="" type="checkbox"/> def4	<code>^(\#\{@\} at)\{#0:p\} line \{#1:i\}?*\$</code>
<input checked="" type="checkbox"/> Python	<code>^ File \"\#\{#0[^"]+\}\", line \{#1:i\}?*(\n \r\n \r)?*(\n \r</code>

Reset



CTags Tagging Options (Standard only)

CTags Tagging options are shown below (**Tools** → **Options** → **Tools** → **CTags Tagging**). These settings control the CTags(1) executable used for building 'tags' files for SlickEdit.

The ctags program generates an index (or "tag") file for a variety of language objects found in a set of file(s). This tag file allows these items to be quickly and easily located by a text editor or other utility. A "tag" signifies a language object for which an index entry is available (or, alternatively, the index entry created for that object).

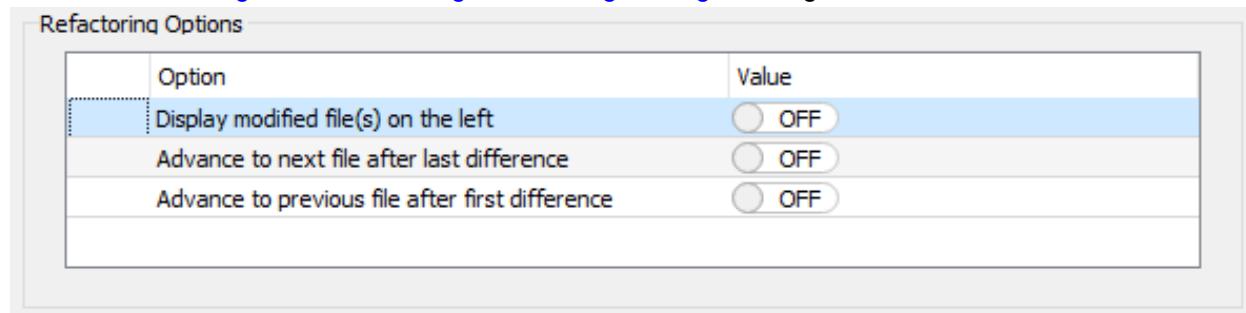
The CTags Tagging Options screen contains the following:

- **CTags executable** - Specifies the path to the CTags executable that you wish to use.
- **CTags options** - Specifies the options to pass to CTags when building ctags-based tag files. Warning, '-e' for etags mode (exuberant ctags), '-x' for generating a cross-reference, and any other option that causes the output to deviate from standard ctags output are not supported.

See [Building CTags Based Tag Files](#) for information.

Refactoring Options (Pro only)

Refactoring options are shown below for [Quick Refactoring](#). These settings control the behavior of the [Section_Refactoring_results](#)[Reviewing_Refactoring_Changes](#) dialog.



- **Display modified file(s) on the left** - When set to **On**, the refactored file will be displayed on the left hand side of the **Refactoring Results** dialog.
- **Advance to next file after last difference** - When set to **On**, hitting **Next Diff** when on the last difference in a file in the **Refactoring Results** dialog will automatically proceed to review the next file in the list.
- **Advance to previous file after first difference** - When set to **On**, hitting **Prev Diff** when on the first difference in a file in the **Refactoring Results** dialog will automatically proceed to review the previous file in the list."/>

Options History

The **Options History** node in the Options dialog (**Tools** → **Options** → **Options History**) is used to see changed options. From the drop-down, select **Anytime** to see all options that have been changed from the default values since the editor was installed, or, choose to see only those options that were changed

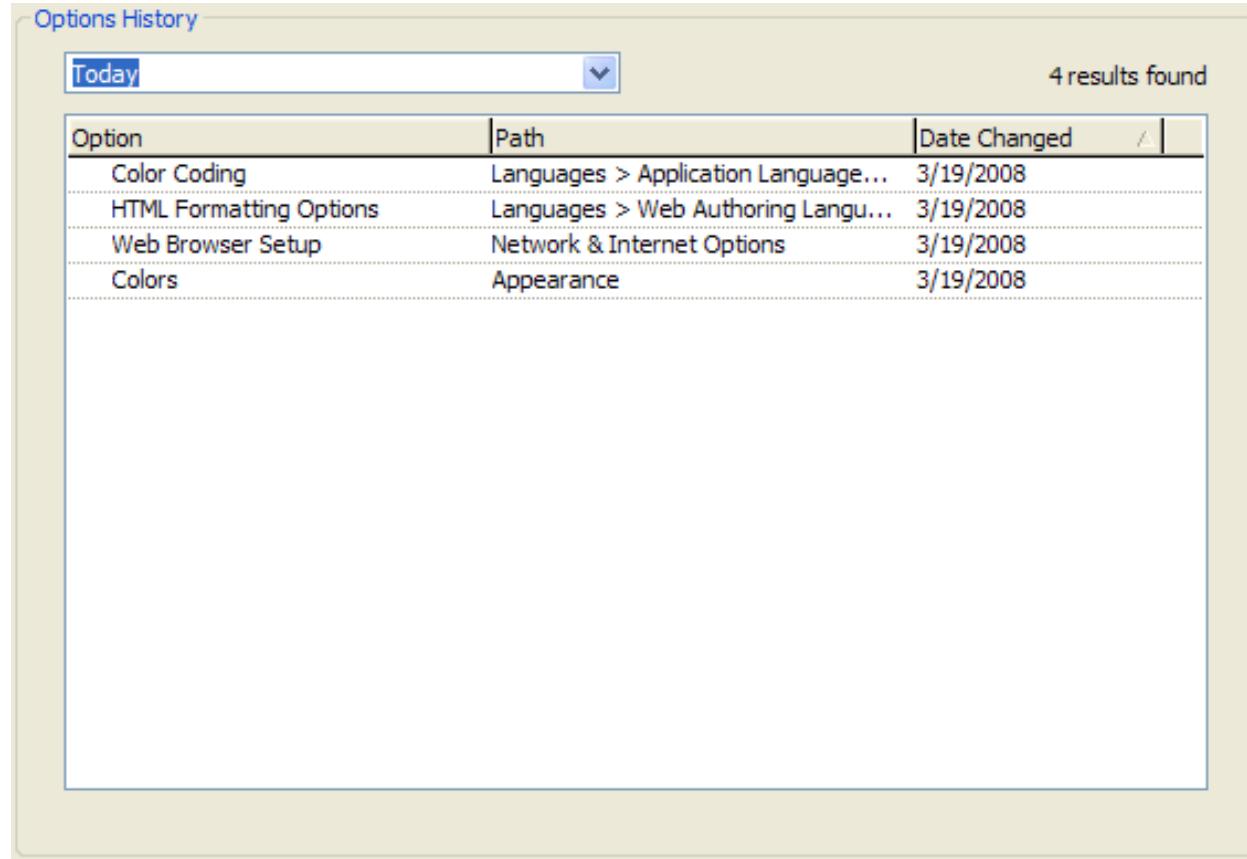
Export/Import Options

today, yesterday, or within the last week or month. Only the most recent date is shown for options that have been changed more than once.

Note

Options History only shows changes that were made through the Options dialog. Changes made by setting configuration variables, in macro code, or in other dialogs are not viewable in the Options History.

The results are displayed in the Options dialog in the results window, as shown in the following screen.



The screenshot shows a results window titled "Options History". At the top left is a dropdown menu set to "Today". At the top right, it says "4 results found". Below is a table with three columns: "Option", "Path", and "Date Changed". The data is as follows:

Option	Path	Date Changed
Color Coding	Languages > Application Language...	3/19/2008
HTML Formatting Options	Languages > Web Authoring Langu...	3/19/2008
Web Browser Setup	Network & Internet Options	3/19/2008
Colors	Appearance	3/19/2008

The number of results returned is displayed at the top right of the results window. The results are divided into columns showing the name of the option, the path to the option in the option tree, and the date it was last changed. Click on the column header to sort by any column. Double-click on an option to display that option panel in the dialog. For options changed on forms embedded in the Options dialog, the results show only the name of the form. For example, if you made a change to a color under **Tools → Options → Appearance → Colors** on the Options dialog, the Options History results show "Colors" as the name of the option and "Appearance" as the path.

Export/Import Options

To export options, select **Tools → Options → Export/Import Options**. You can export all your options at once or you can export a designated group of them. To export all options click the **Export All Options**

button. To export a particular set of options click the **Setup Export Groups** button. To import already exported options, click **Import Options**. See below for details of each operation.

Note

Moving options to a machine with a different operating system is allowed, but not supported. The same applies to exporting options from a one version of SlickEdit and then imported into another. While these operations may work for some options, we cannot predict when this will cause a problem.

Export/Import Options

Export Options

Exporting options is one way to back up your SlickEdit settings. You can also share your settings with other SlickEdit users with this feature. Use Export Groups to pick and choose which options you want to export.

[Setup Export Groups...](#)

[Export All Options...](#)

Import Options

You can import the settings of other users using the Import Options feature. You can also restore settings from defaults or items that you backed up earlier using the Export Options feature. Once you pick a package to import, you can preview the settings within and choose which ones to import.

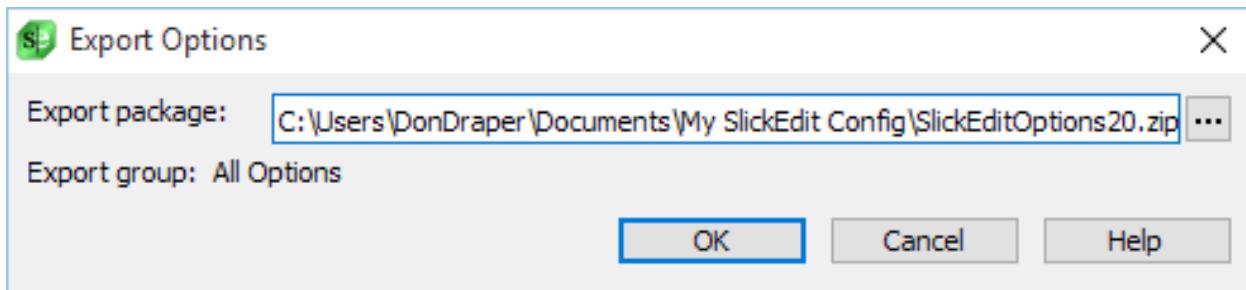
[Import Options...](#)

[Restore Default Options...](#)

Export All Options

To Export all of your options, click the **Export All Options** button. You will be asked to select a file where you want to save your export package. Exports are saved in packages with the extension ".zip." Once you have selected a file, click **OK** to begin the export. If there are any errors, you will be notified with a message. Individual errors will also be listed in the Message List.

Export/Import Options

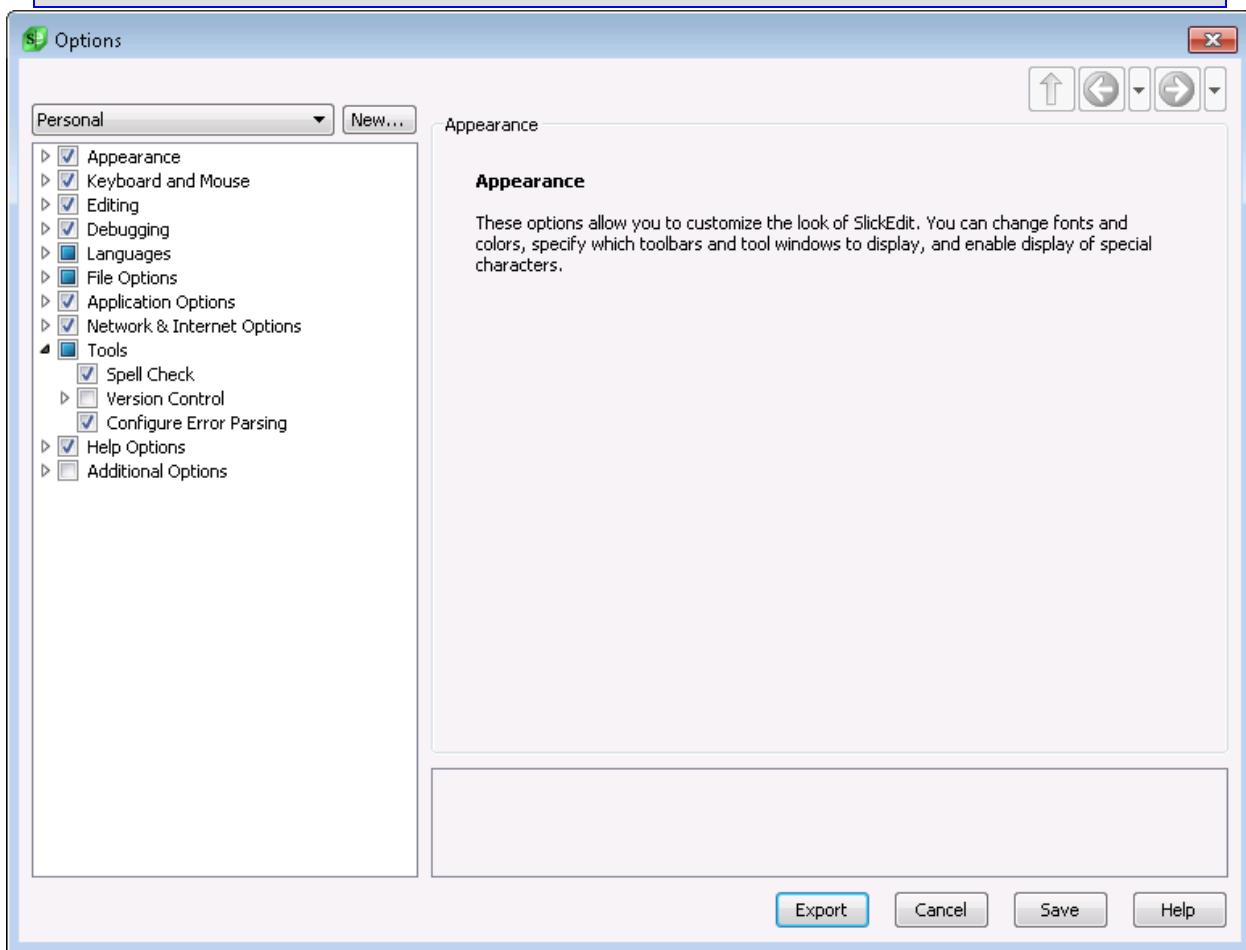


Setup Export Groups

Export Groups allow you to export a specific set of options. Two default export groups have been created: Team and Personal. The **Team Export Group** contains settings which might be shared across a programming team, such as coding styles. The **Personal Export Group** contains settings which control how the editor looks and behaves. You can change these groups or create new ones. To view or use an export group, select the group from the combo box found at the top left corner of the **Export Groups Editor** dialog.

Note

The Standard and Community editions do not have all of the export groups listed below.



Export/Import Options

Add or remove options to include this group by checking or unchecking nodes in the hierarchy. To include all of the items in subtree, put a check in the box next to a parent. If you select specific items in a subtree, the parent node will be filled in with gray.

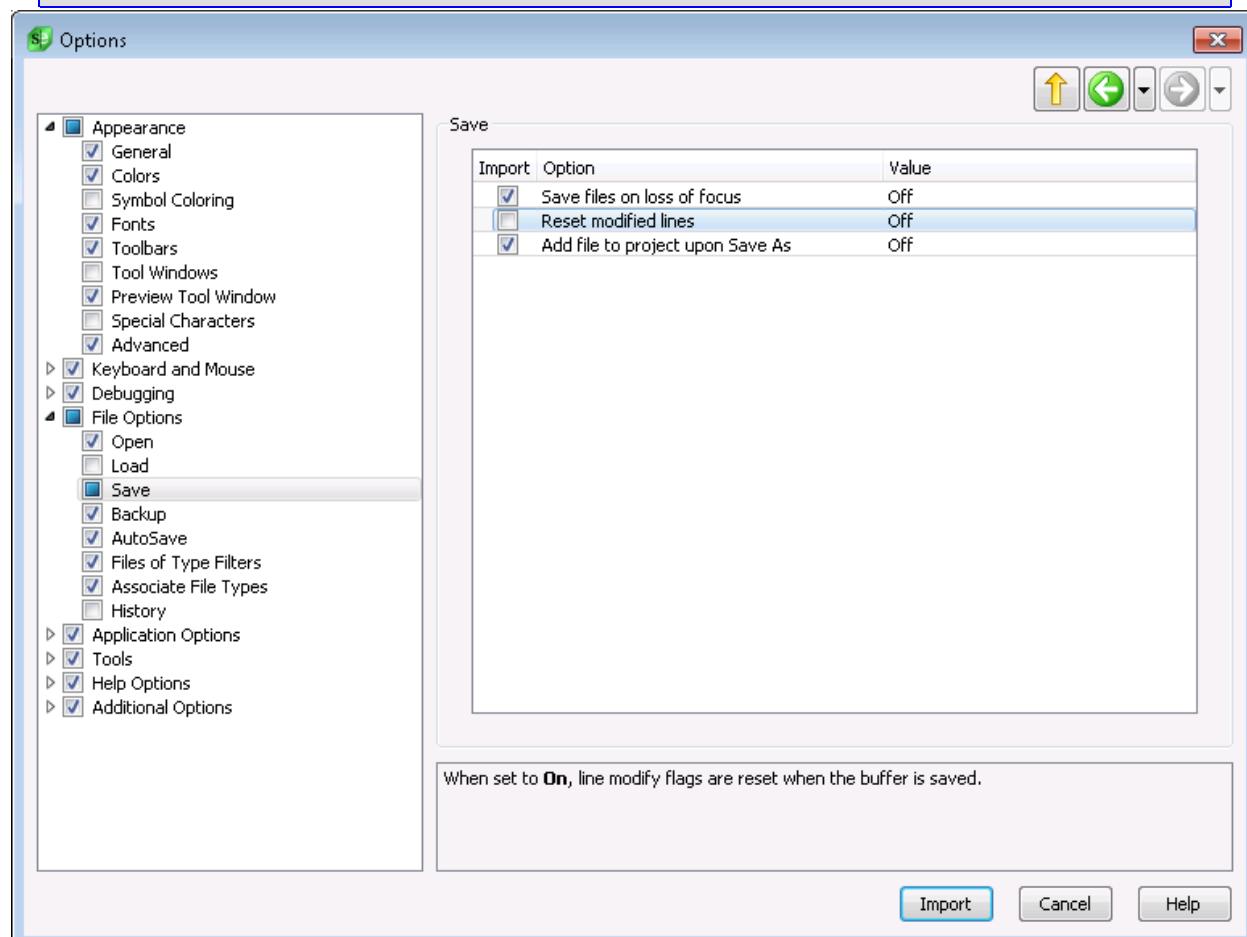
To create a new export group, click the **New** button next to the combo box containing the list of Export Groups. You can either create a blank group or copy an existing group. Then you can drill down to the property level to select which options to export. You can also select whole categories at a time if you wish. Once you are finished modifying your Export Groups, click "Save" to save your changes. You can also export the current group by clicking **Export**.

Importing Options

You can import options by clicking the **Import Options** button on **Tools → Options → Export/Import Options**. Then navigate to the location of the export package file.

Note

The Standard and Community editions do not have all of the export groups listed below.



Once the export package is read, a tree of the options within the package is displayed. You can choose which options to import. Once you have made your selections, click **Import**. The options will be imported

and set. If there are any errors, you will be notified with a message. Individual errors will also be listed in the Message List.

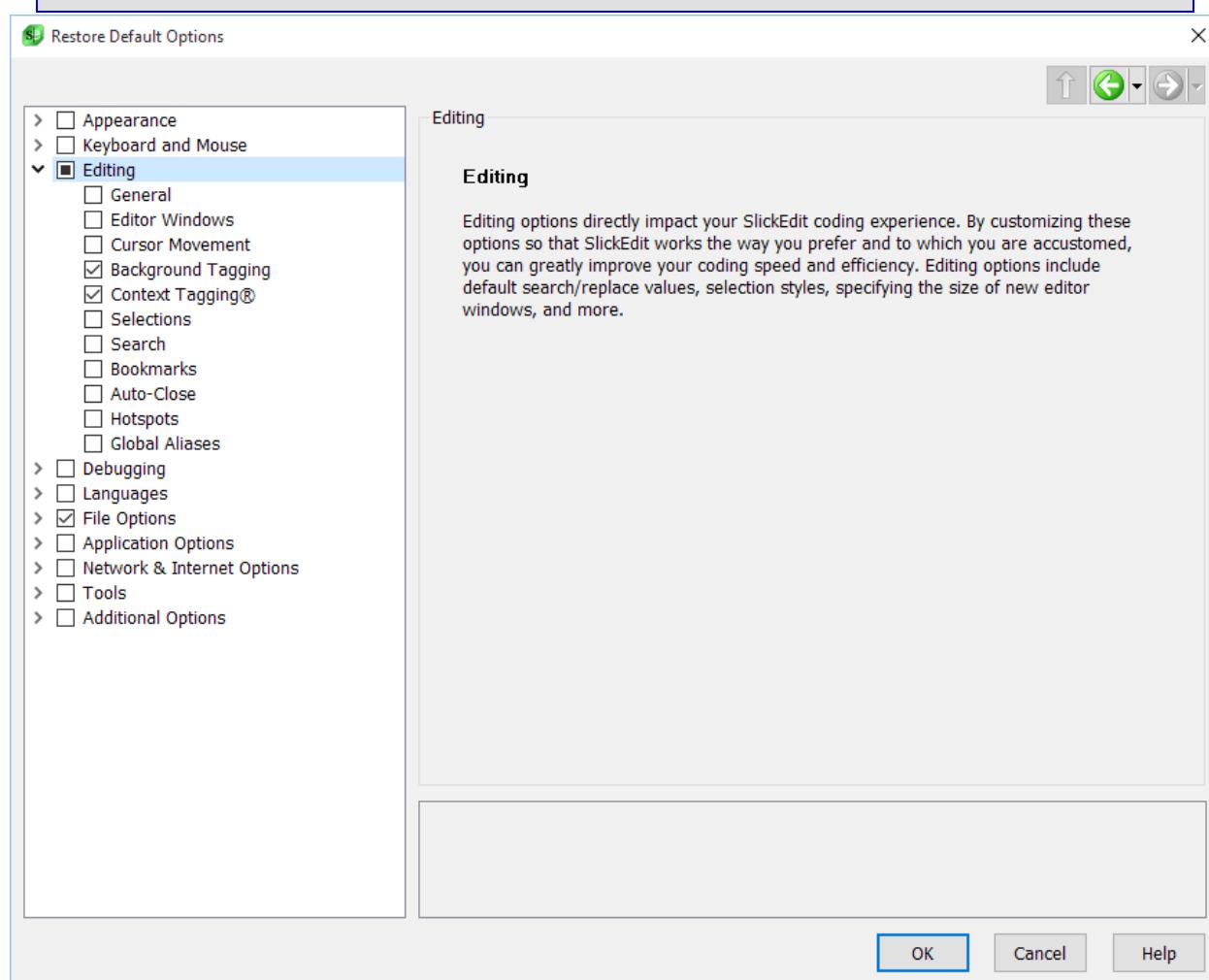
Restoring Default Options

You can restore options to the default settings by clicking the **Restore Default Options** button on **Tools** → **Options** → **Export/Import Options**.

Once the default export package is read, a tree of the options within the package is displayed. You can choose which options to restore. Once you have made your selections, click **Import**. The options will be restored and set. If there are any errors, you will be notified with a message. Individual errors will also be listed in the Message List.

Note

The Standard and Community editions do not have all of the export groups listed below.



Additional Options

Some settings are not configurable through the options dialog but are still available for export/import. These options can be found under the **Additional Options** node in the Setup Export Groups dialog. They are also included automatically if you export all options.

The options available for export/import under **Additional Options** are:

- **Menu Customizations** - Exports any changes made to menus by adding, removing, or modifying individual menu items. For more information about customizing menus, see [Menus](#).
- **Toolbar Customizations** - Exports customizations made to any of the Toolbars, including added or removed buttons and changes to button behavior. For more information about customizing Toolbars, see [Toolbar Options](#).
- **Toolbar and Tool Window Layout** - Exports changes made to the layout of tool windows and toolbars.
- **User-Created Forms** - Exports any forms created by the user.
- **User-Created Menus** - Exports any menus created by the user.
- **User-Created Toolbars** - Exports any toolbars created by the user.
- **User-Recorded Macros** - Exports any macros recorded by the user such that these macros can be shared with other users. To find out about recording macros, see [Recorded Macros](#).
- **Beautifier Settings** - Code beautifiers are used to reformat existing code and rely on formatting preferences set by the user. For more information about beautifiers, see [Beautifying Code](#).
- **Code Templates** - You can export and import your code templates, which are used to automate the creation of common code elements. For information about creating and using templates, see [Code Templates](#).
- **Project Types** - Project types allow you to create a template of a project by setting up directories, build tools, compiler properties and more. You can then export and import these types. For more information, see [Creating Custom Project Types](#).

Configuration Backup

Whenever you import any options, certain configuration files are backed up so that you can restore your application should the imported options cause any problems. To restore your configuration after an options import, do the following:

1. Close the application.
2. Locate your configuration directory. For more information about your configuration directory and how to find it, see [Configuration Directory Location](#). Make sure you can find the backed-up configuration files. They will be named `vslick.sta.bak`, and `vusrobs.e.bak` (UNIX: `vslick.sta.bak`, `vunxobj.e.bak`).
3. Remove the existing configuration files, named `vslick.sta`, and `vusrobs.e` (UNIX: `vslick.sta`, `vunxobj.e`). Also remove the loaded macro files `vusrobs.ex` (`vunxobj.ex`).

4. Rename the backed-up configuration files by removing the .bak extension.
5. Run the application. Your configuration should be back to where it was before the options import.

Window

This section describes items on the **Window** menu and associated dialogs and tool windows. For more information about working with editor windows, see [Files, 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
Tile	Tiles editor windows.	<code>tile_windows</code>
Tile Horizontal	Tiles editor windows horizontally when there are three or less windows.	<code>tile_windows h</code>
Arrange Icons	Rearranges iconized windows.	<code>arrange_icons</code>
Next	Switches to next window.	<code>next_window</code>
Previous	Switches to previous window.	<code>prev_window</code>
Close	Closes the current window.	<code>close_window</code>
Font	Displays the Window Font dialog, which allows you to set/view fonts for the current editor window or all windows. See Window Font Dialog .	<code>wfont</code>
Color	Displays the Window Color dialog, which allows you to select a color profile for the current editor window or all windows. See Window Color dialog box .	<code>window_color</code>
Split Horizontally	Splits the current window horizontally in half.	<code>hsplit_window</code>
Split Vertically	Splits the current window vertically in half.	<code>vsplit_window</code>
Zoom Toggle (Show/Hide Document Tabs)	Zooms or unzooms the current window. Document Tabs are	<code>zoom_window</code>

Window Menu Item	Description	Command
	hidden when zoomed.	
One Window	Zooms the current window and deletes all other windows.	one_window
Duplicate	Creates another window linked to the current buffer.	duplicate_window
Link Window	Displays the Link Window dialog, which allows you to select a buffer to display in the current editor window. See Link Window Dialog .	link_window

Window Dialogs and Tool Windows

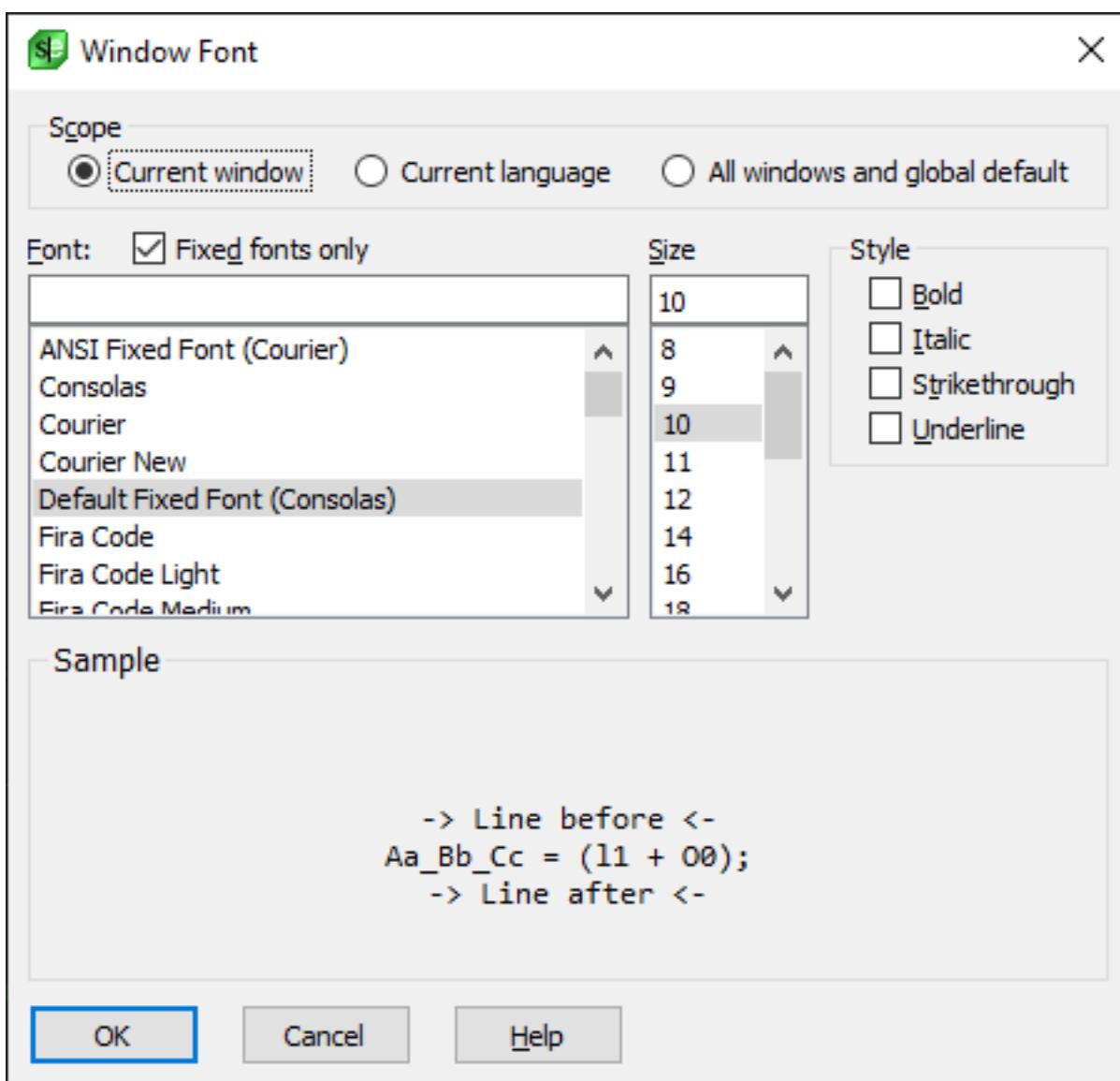
This section describes the dialogs and tool windows that are associated with the **Window** menu items.

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 [Fonts](#).

To access the Window Font dialog, click **Window** → **Font**, or use the **wfont** command.

Window Dialogs and Tool Windows



The following options and settings are available:

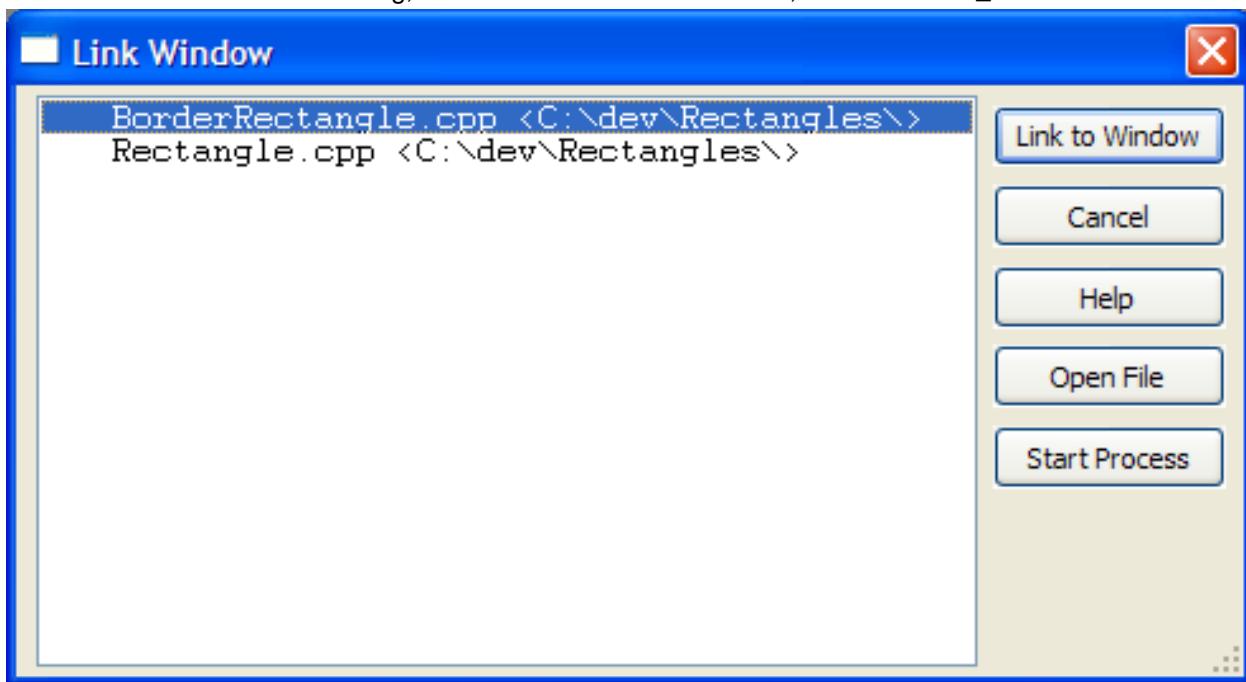
- **Scope** - Specifies the editor windows to affect.
 - **Current window** - Affects the current editor window only.
 - **All windows and Default** - Affects all open editor windows and all newly-created editor windows.
- **Font** - Displays a selection list of the fonts installed on your computer.
- **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 [Files, Buffers, and Editor Windows](#).

To access the Link Window dialog, click **Window** → **Link Window**, or use the **link_window** command.



The buttons are described as follows:

- **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 [The Help System](#) and [Product Support](#).

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. See The Help System .	help -contents
Index	Displays the Help system open to the Index, where you can search for index items. See The Help System .	help -index
Search	Displays the Help system open to the Search tab, where you can search for any item. See The Help System .	help -search
New Features	Displays the Help system open to the New Features and Enhancements section. See New Features and Enhancements .	help new features
Cool Features	Displays the Cool Features dialog, which shows SlickEdit® feature tips. See Help Dialogs and Tool Windows .	cool_features
Quick Start	Displays the SlickEdit Quick Start documentation. See Quick Start .	help Quick Start
Keys Help	Displays the Help system open to the emulation key binding reference tables for the current emulation. See Emulation Tables .	help key bindings
What Is Key	Used to discover the command	what_is

Help Menu

Help Menu Item	Description	Command
	associated with a key binding. Opens the command line, prompting with the text What is key . See Determining the Command of a Key Binding .	
Where Is Command	Used to discover the key binding associated with a command. Opens the command line, prompting with the text Where is command . See Determining the Key Binding of a Command .	where_is
Macro Functions by Category	Displays the Help system open to this topic, which shows a categorized list of macro functions.	help macro functions by category
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.	goto_faq
License Manager	Displays the SlickEdit License Manager for managing licenses. See Licensing .	lmw 1
Product Updates	Displays the Product Updates menu, from which you can install updates and hot fixes. See Help Menu .	N/A
Register Product	Displays the Register dialog, from which you can begin the SlickEdit on-line registration process.	online_registration
SlickEdit Support Web Site	Invokes a Web browser and opens the SlickEdit Support Web page. See Product Support .	goto_slickedit
Contact Product Support	Used to invoke a Web browser and opens a form on the SlickEdit Web site that you can use to contact Product Support. See Contact Product Support .	do_webmail_support

Help Menu

Help Menu Item	Description	Command
	Product Support.	
Check Maintenance	Invokes a Web browser and opens a SlickEdit Web page that shows the status of your Maintenance and Support Agreement.	check_maintenance
SlickEdit Forum	Invokes a Web browser and opens the SlickEdit Forum Web page.	check_maintenance
About SlickEdit	Displays a property sheet containing information about your product, such as serial and version numbers, as well as release notes, copyright notices, the license agreement, and contact information.	version

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.	upcheck_display
Options	Displays the Update Manager Options dialog, used to set the frequency of automatic checking of new updates. See Update Manager Options Dialog .	upcheck_options
Load Hot Fix	Displays an Open-style dialog, to begin the process of installing a hot fix. See Applying Hot Fixes .	load_hotfix
List Installed Fixes	Displays a summary sheet of hot fixes that are installed on your computer. See Applying Hot Fixes .	list_hotfixes

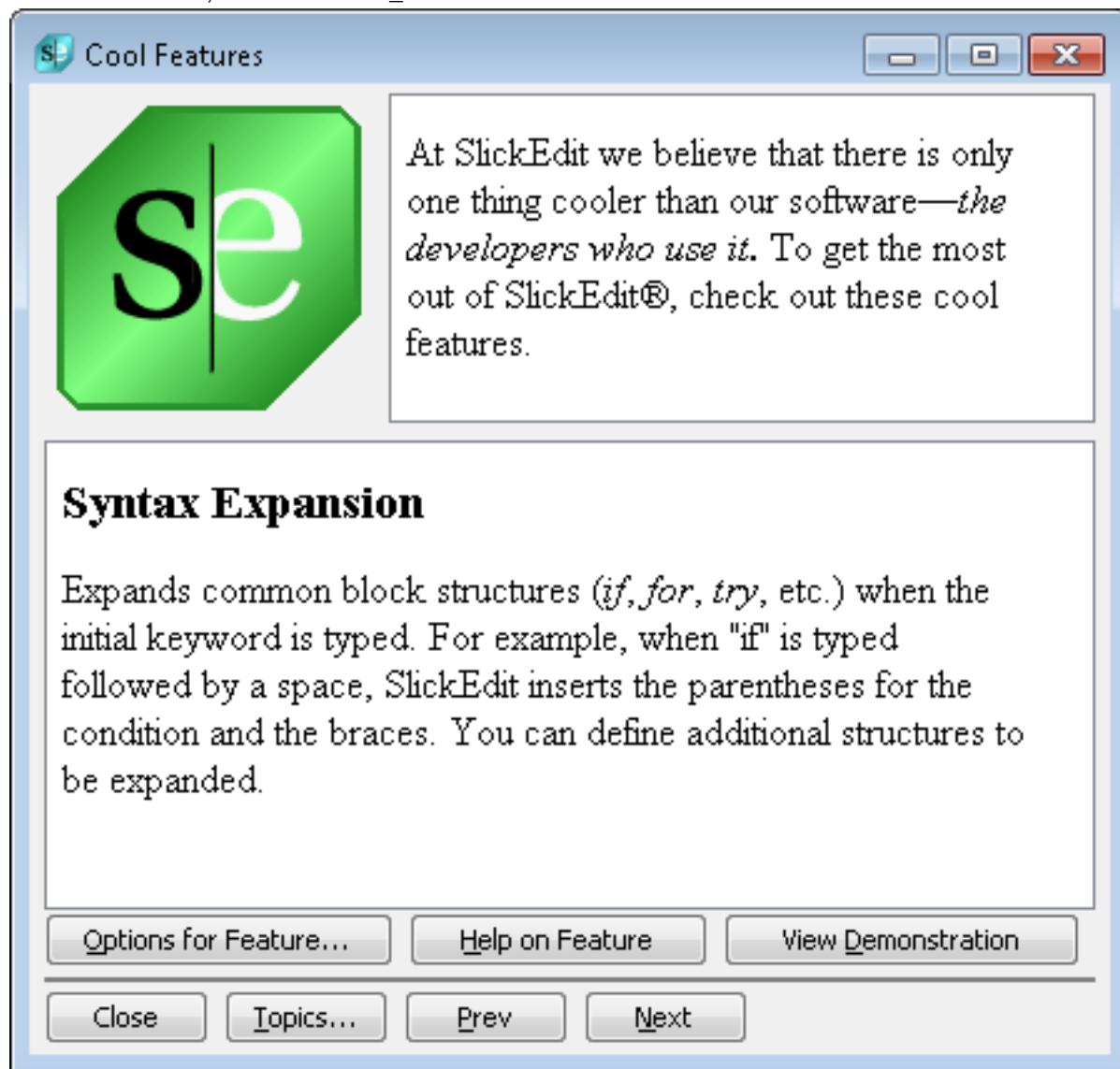
Product Updates Menu Item	Description	Command
---------------------------	-------------	---------

Help 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 automatically after the product installation has completed, and by default, each time the editor is started. To access the dialog at any time, from the main menu, click **Help** → **Cool Features**, or use the **cool_features** command on the SlickEdit® command line.



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. The previously viewed topic is remembered and displayed the next time the dialog is invoked.
- **Prev** - Scrolls to the previous Cool Feature.
- **Next** - Scrolls to the next Cool Feature.
- **Show on startup** - If selected, prevents the Cool Features dialog from appearing each time the editor is started.

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 Setting Options](#).

Appendix

This chapter contains reference information about encodings, emulations, and configuring SlickEdit.

Tutorials

Tutorials in this section:

- [Hello World Tutorial \(C/C++\)](#)
- [Hello World Tutorial \(C#\)](#)
- [Hello World Tutorial \(Java\)](#)
- [Vim Tutorial](#)
- [Creating and Distributing Custom Toolbars](#)

Hello World Tutorial (C/C++) (Pro only)

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.

Create the Project Using the GNU C/C++ Wizard

1. From the main menu, click **Project** → **New**.
 2. On the **Project** tab, click to expand **C/C++**, then click **GNU C/C++ Wizard**.
 3. Specify the **Project name** as **HelloWorld**. Change the location if you want.
 4. Click **OK** on New project dialog.
 5. Select the **Project Type** as **Executable** and the **Source Type** as **C++**.
 6. Click **Next**.
 7. For the **Application Type**, select A "Hello World" application.
 8. Click **Next**.
 9. Select **Build without a makefile**.
- 1 Click **Finish**. A dialog is displayed, containing information about the new project. Click **OK**. The wizard 0 constructs a workspace by the name of `HelloWorld.vpw`, a project called `HelloWorld.vpj`, and a program file called `HelloWorld.cpp`.

Build the Project

To build this project, from the main menu, select **Build** → **Build**.

Run the Program

To run the program, from the main menu, click **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 (C#) (Pro only)

This tutorial describes how to build a simple C# console application with SlickEdit, no Visual Studio required. It assumes you have the .NET Framework 2.0 and the C# compiler (`Csc.exe`) installed under `%WINDIR%\Microsoft .NET\Framework\v2.0.50727`. The Windows SDK (v6.0 or later) or the full .NET Framework SDK is required if you want to interact with the managed code debugger (`mdbg.exe`).

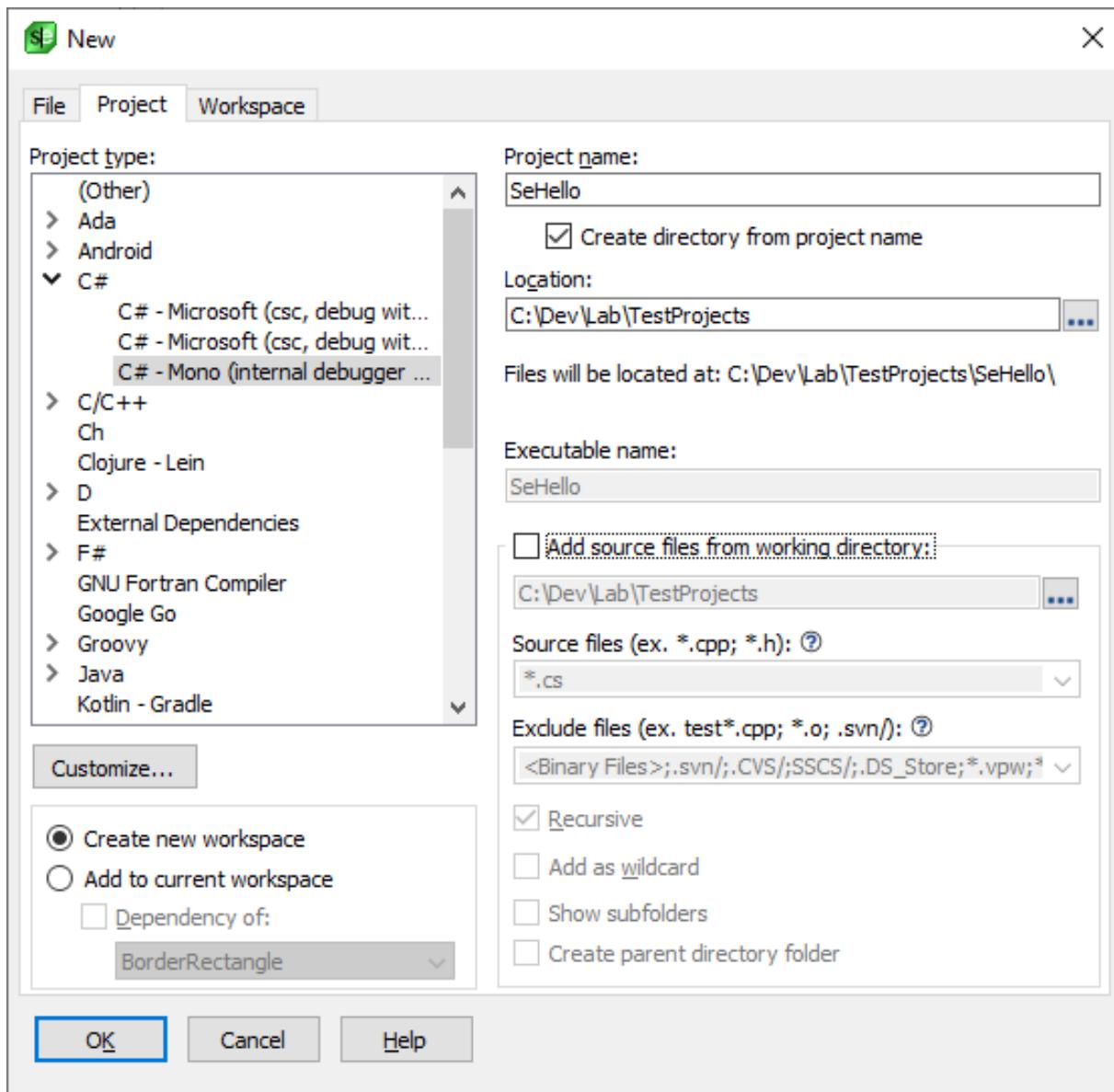
Creating the Starter Project

Project Setup

Use the following steps to set up the starter project:

1. From the main menu, click **Project → New**.
2. Select the **(Other)** project type.
3. In the **Project name** box, type **SeHello**.
4. Select the **Create project directory from project name** option.
5. Change the **Location** to the path where you want to store the project.

Hello World Tutorial (C#) (Pro only)



6. Click **OK**.

7. Close the Project Properties dialog that automatically appears.

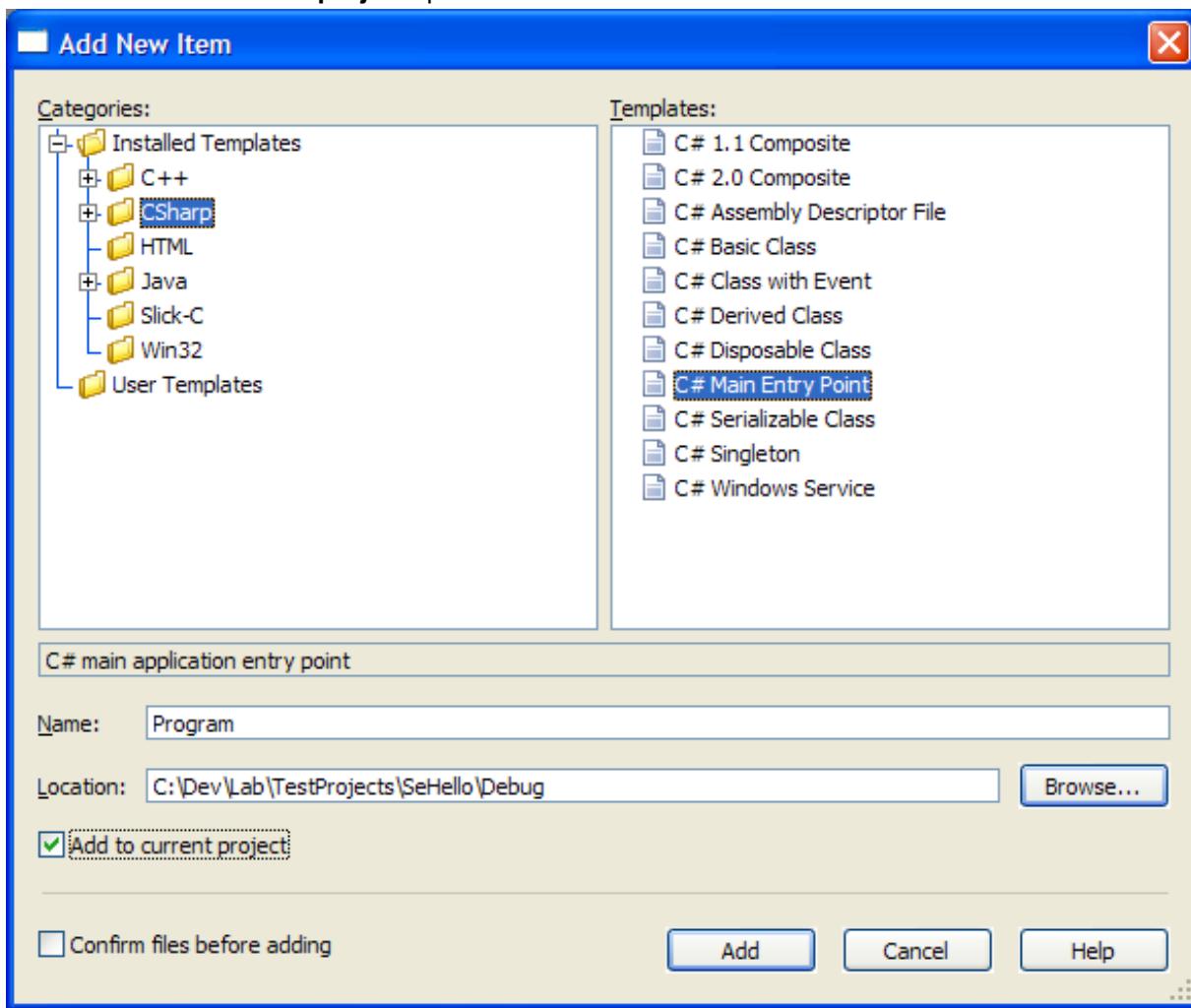
Create and Add a File to the Project

Next, use the following steps to create and add a file to the project:

1. From the main menu, click **File → New Item from Template**.
2. In the Categories list, under **Installed Templates**, select **CSharp**.
3. In the Templates list, select **C# Main Entry Point**.
4. In the **Name** box, select the default or change the name based on your preference.

Hello World Tutorial (C#) (Pro only)

5. Select the **Add to current project** option.



6. Click **Add**.
7. Click **OK** on the Parameter Entry dialog.
8. The new file opens in the editor and also appears in the Source Files project folder. Use the Projects tool window to see project folders and their contents.
9. Add the following code inside **Main()**:

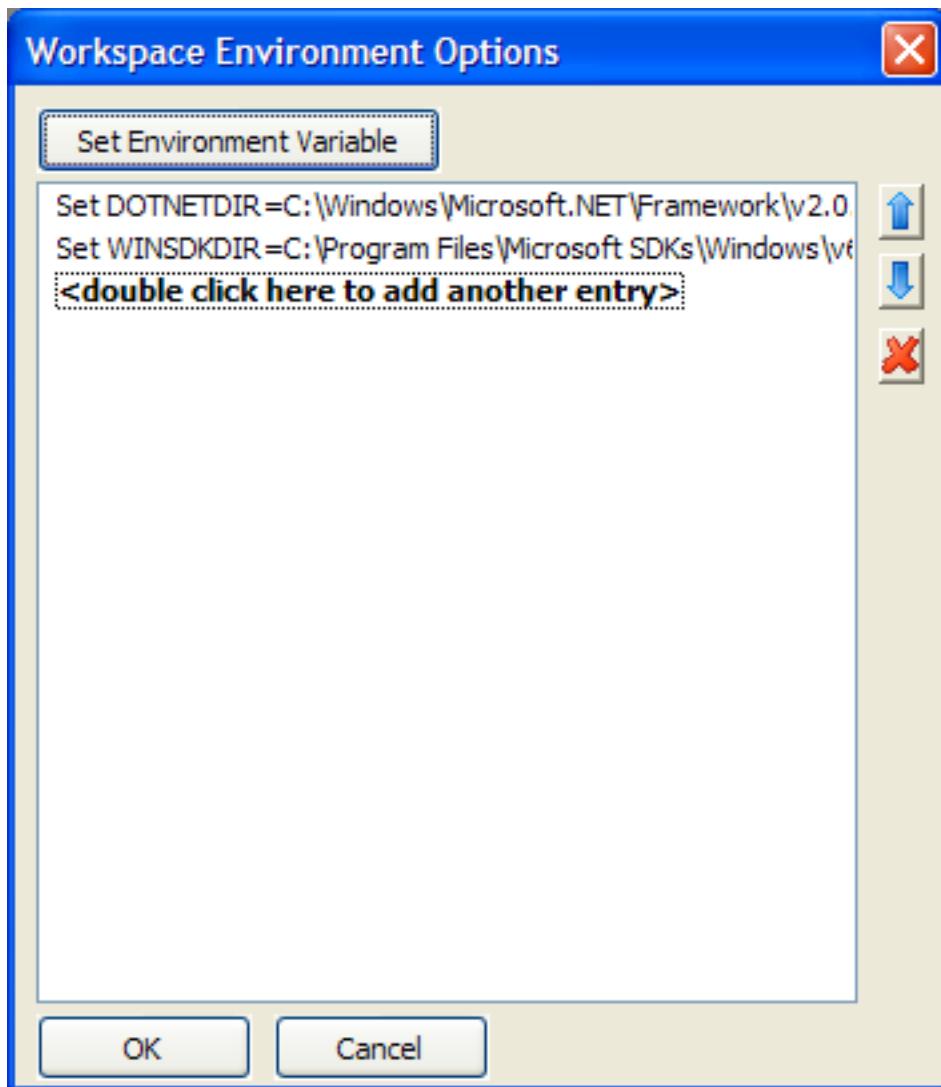
```
#if DEBUG
Console.WriteLine("Hello World - Debug");
#else
Console.WriteLine("Hello World - Release");
#endif
```

- 1 Save and close the file.
- 0.

Set Environment Variables

Use the steps below to set variables for the workspace. A SlickEdit workspace (.vpw) is equivalent to a solution (.sln) in Visual Studio.

1. From the main menu, click **Project → Workspace Properties**.
2. Click the **Environment** button.
3. Click **Set Environment Variable**.
4. In the Name box, type **DOTNETDIR**.
5. In the **Value** box, specify your .NET framework that you want to use for the C# compiler, for example, C:\Windows\Microsoft.NET\Framework\v2.0.50727. To use an existing environment variable in the definition of the workspace environment, use the %(VARIABLE) syntax, not the VARIABLE% syntax, for example, %(WINDIR)\Microsoft.NET\Framework\v2.0.50727. Be sure to select environment variable names that do not already exist.
6. Optionally, create another variable named **WINSDKDIR**, and point it to the Windows SDK or .NET Framework SDK, for example, C:\Program Files\Microsoft SDKs\Windows\v6.0. This should be a directory where the mdbg.exe managed code debugger can be found.



7. Click **OK**.
8. Close and reopen the workspace to set the environment variables.

Setting Up the Release Build

Once you've set the environment variables, complete the following steps to configure the commands for building a release version:

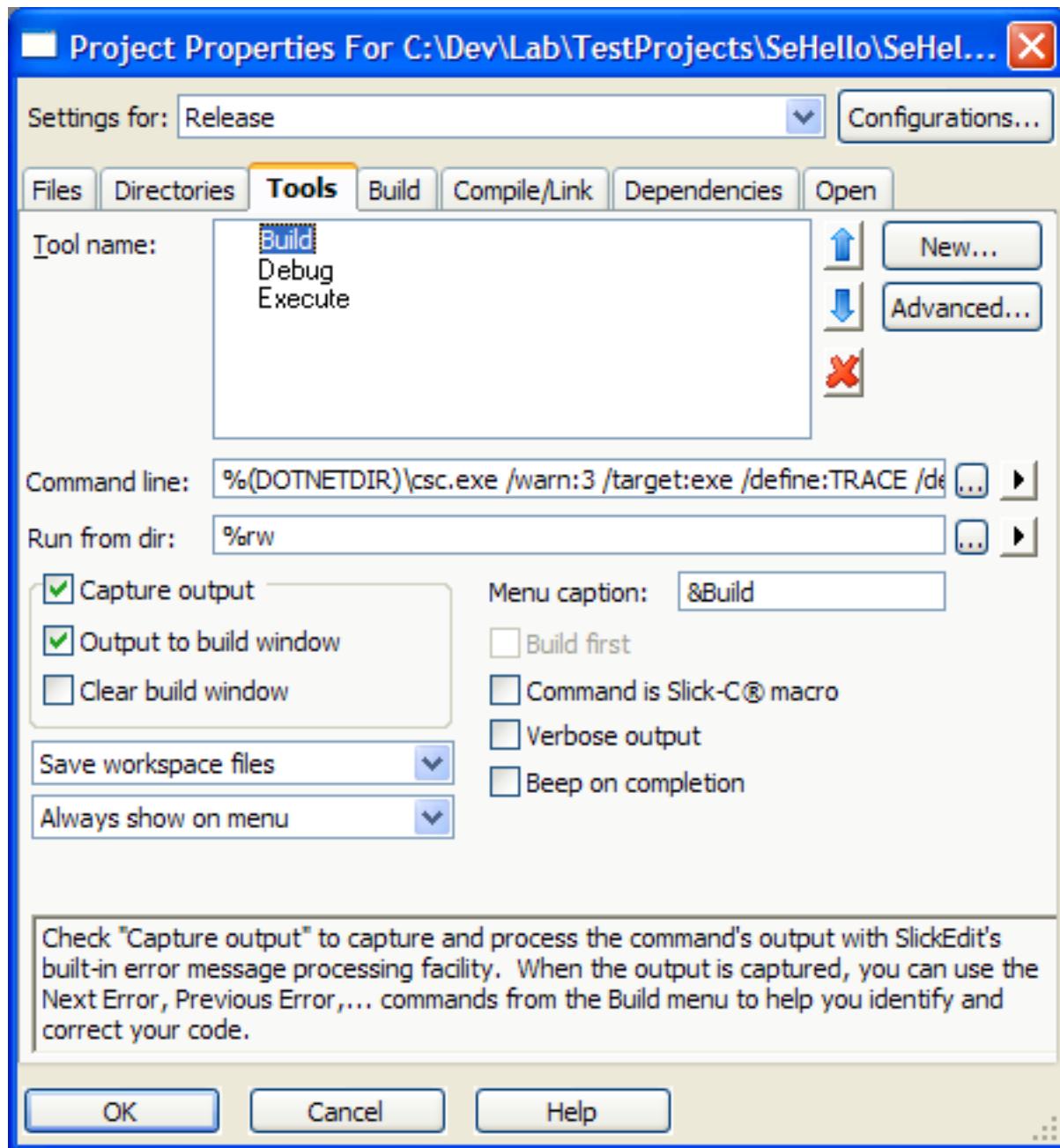
1. From the main menu, click **Project → Properties**.
2. In the **Settings for** box, select **Release**.
3. Click to display the **Tools tab**.
4. In the **Tool name** box, remove both **Compile** and **Rebuild** by selecting them and clicking the red X button. The remaining tools are Build, Debug, and Execute.

5. In the **Tool name** box, select the **Build** tool.
6. In the **Command line** box, type the following, replacing `SeHello.exe` with your own executable name:

```
% (DOTNETDIR)\csc.exe /warn:3 /target:exe /define:TRACE /debug-
/out:%bdSeHello.exe %{*.cs}
```

The `%bd` in the above command line is a variable that represents the build output directory, in this case **Release**. The `{*.cs}` construct specifies all project files that end with the `.cs` extension.

7. Make sure the **Run from dir** box contains `%rw`, which specifies the build should run from the project's working directory.
8. Make sure the **Capture output** and **Output to build window** options are selected, so the output is captured and displayed in the Build tool window.



9. Back in the **Tool name** box, select the **Execute** tool.

1 Change the command line to "**SeHello.exe**" (including quotes), replacing **SeHello.exe** with the name of the executable specified in Step 6 above.

1 In the **Run from dir** box, change the option to **%bd** (which represents the build output directory).

1.

1 Make sure the **Capture output** and **Output to build window** options are selected.

2.

1 Click **OK** to save your settings and close the Project Properties dialog.

3.

You can now execute a build and see the output of the C# compiler by clicking **Build** → **Build** from the main menu. Once successful, you can run the console program and display the "Hello World-Release" results by clicking **Project** → **Execute** from the main menu.

Setting Up the Debug Build

To configure commands for a debug build, complete the following steps:

1. From the main menu, click **Project** → **Project Properties**.
2. Click the **Configurations** button.
3. Click **New**.
4. In the **New config name** box, type **Debug**.
5. In the **Copy settings from** box, make sure **Release** is selected.
6. Click **OK** to create the Debug config, and dismiss all of the dialogs.
7. On the Project Properties dialog, click to display the **Tools tab**.
8. In the **Settings for** box, select **Debug**.
9. In the **Tool name** box, select the **Build** tool.

1 In the Command line box, type the following, replacing `SeHello.exe` with your own executable name:
0.
`% (DOTNETDIR)\csc.exe /warn:3 /target:exe /define:DEBUG;TRACE /debug+
/out:%bdSeHello.exe %{*.cs}`

Adding the DEBUG define and changing `/debug-` to `/debug+` are the only changes between the Release and Debug configuration command lines.

- 1 In the **Tool name** box, select the **Execute** tool.
 1. Set up the **Execute** tool the same as you did in Step 9 for the Release configuration (see [Setting Up the Release Build](#)).
 - 1 Click **OK** to save your settings and close the Project Properties dialog.
 - 3.
 - 1 From the main menu, click **Build** → **Set Active Configuration** and select **Debug**.
 - 4.
- You can now execute the build by clicking **Build** → **Build** from the main menu. Once successful, click **Build** → **Execute** to see the "Hello World - Debug" output in the Build tool window.

Handling Complex Build Commands

A small console application like this one doesn't have any extensive dependencies, and most of the default options for the C# compiler are fine. However, more complex projects will require many more options to be passed on the command line. For these cases, it can be useful to create an options file for

all of the command line switches. To do this:

1. From the main menu, click **File** → **New** to create a new text file and add it to the project. For this example, we are creating the command line options for the Debug build in a file called `Debug . opts`.
2. Insert the following into the new file. This should match your first few options on the Debug build command line:

```
/warn:3 /target:exe /define:DEBUG;TRACE /debug+
```

3. From the main menu, click **Project** → **Project Properties**.
4. Click to display the **Tools** tab.
5. In the **Settings for** box, select the **Debug** configuration.
6. In the **Tool name** box, select the **Build** tool.
7. In the **Command line** box, type the following, where **SeHello.exe** is your own executable name:

```
% (DOTNETDIR)\csc.exe @Debug . opts /out:%bdSeHello.exe %{* . cs}
```

8. Close the Project Properties dialog.
9. From the main menu, click **Build** → **Build** to make sure the options file was correctly read.

Setting Up the Console Debugger

Optionally, to set up the console debugger, complete the following steps. This demonstrates how to hook up an external tool, and how it can be used interactively inside the Build tool window.

1. From the main menu, click **Project** → **Project Properties**.
2. Click to display the **Run/Debug** tab.
3. Select the option to **Use External Debugger**.
4. In the **Debugger:** box, type the name of the debugger you wish to use.

```
% (WINDKDIR)\Bin\mdbg . exe
```

5. In the **Other options:** box, type additional options to be passed to the debugger on the command line, for example, if **SeHello.exe** is the name of your own executable:

```
SeHello . exe
```

6. Click to display the **Tools** tab.
7. In the **Settings for** box, select the **Debug** configuration.

8. Make sure **Run from dir** is set to **%bd**.
 9. For command line debugging tools, make sure the **Capture output** and **Output to build window** options are selected.
- 1 Click **OK** to save your settings and close the Project Properties dialog.
 - 0.
 - 1 From the main menu, click **Debug** → **Start** to start the debugger.
 - 1.
 - 1 From the main menu, click **Build** → **Show Build**. The Build tool window now has focus with a blinking cursor just after the first **mdbg>** prompt.
 - 2.
 - 1 In the Build window, type the following commands in order, pressing **Enter** after each one:
 - 3.
 - **print args** - To show the value of the **args** variable.
 - **next** - To do a step over.
 - **go** - To continue execution.
 - **quit** - To quit the debugger. The debugger must be stopped in order for SlickEdit to regain control of the Build window.

The entire debug session output should look similar to the following:

```
C:\Dev\Lab\TestProjects\SeHello\Debug
```

```
> C:\WinSDK\Bin\mdbg.exe SeHello.exe
```

```
MDbg (Managed debugger) v2.0.50727.312 (rtmLHS.050727-3100) started.  
Copyright (C) Microsoft Corporation. All rights reserved.  
For information about commands type "help";  
to exit program type "quit".  
run SeHello.exe  
STOP: Breakpoint Hit  
16: static void Main(string[] args) {  
[p#:0, t#:0] mdbg> print args  
args=array [0]  
[p#:0, t#:0] mdbg> next  
19: Console.WriteLine("Hello World - Debug");  
[p#:0, t#:0] mdbg> go  
Hello World - Debug  
STOP: Process Exited  
mdbg> quit  
C:\Dev\Lab\TestProjects\SeHello\Debug  
>
```

Hello World Tutorial (Java) (Pro only)

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, click **Project** → **New**.
2. On the **Project** tab, click to expand **Java**, then 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**.

This creates a workspace and project by the name **HelloWorld** at **C:\HelloWorld**.

Create the File

1. From the main menu, click **Project** → **New**.
2. Select the **File** tab.
3. Select **Java** from the list and enter a **Filename**, **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.

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, as shown in the following 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, click **Run** → **Execute**. The words **Hello World** are displayed in the window.

Vim Tutorial

SlickEdit® provides an emulation mode for the Vim text editor. If you want to learn Vim, you can use the **vimtutor** command. Most Vim installations come with this command, which displays a special "learn-by-doing" practice file in the editor that you can actually edit as you learn how to use the Vim commands. This file has been customized for SlickEdit users.

To use the command, open the SlickEdit command line, then type the command and press **Enter**. The practice file is displayed in the editor. Each time you use the **vimtutor** command, SlickEdit creates a fresh copy of this file.

Tip

- You will be prompted to switch to the Vim emulation when you invoke the command if the editor is set to a different emulation. See [Changing Emulations](#) for more information.
- When in the Vim emulation, you can open the SlickEdit command line with **Ctrl+A**, or in any emulation, by clicking in the message area with the mouse. See [Activating the Command Line](#) for more information.

Creating and Distributing Custom Toolbars (Pro only)

Use the following steps to write a macro which will load a custom toolbar that you can easily distribute to other users:

1. Create a new form by clicking **Macro → New Form**.
2. Change the form's name by editing the **name** property. For this example, we will name the form "mytoolbar1". You will need to remember the name of the form.
3. Save the new form by pressing **Ctrl+S**, or, right-click on the form and select **Save Form**.
4. Close the form and its properties.
5. Create a new toolbar. To do this, click **Tools → Options → Appearance → Toolbars**, and click **New**. Enter a new toolbar name in the **Tool Bar Name** field. This is the caption that will be used on the title bar of your toolbar when it is floating (i.e. not docked). For this example, we will use "My Toolbar 1". Next, expand the dialog by clicking the **Advanced** button. In the **Form Name** box, enter the form name that you just created in Step 1, then click **OK**. The new toolbar is displayed as floating.
6. Add buttons to the new toolbar by dragging and dropping them onto the new toolbar from the **Toolbar Customization** options page (**Tools → Options → Appearance → Toolbar Customization**).
7. Activate the SlickEdit® command line (see [Activating the Command Line](#)) and run the **save_config** command.

8. Open the file `vusrobs.e` (this file resides in your SlickEdit config directory, which can be found by going to **Help → About SlickEdit**). An example of this file is located in the `docs/samples` installation subdirectory.
9. Copy all of the code which is related to your new toolbar as well as the `defmain()` function at the bottom of the file, and paste it into a new file with a `.e` extension. An example of this file is `newToolbar.e`, located in the `docs/samples` installation subdirectory.

1 Add the following include statements to the very top of the file:

0.

```
#include "slick.sh"
#include "toolbar.sh"
```

Be sure that `slick.sh` is included first.

- 1 At the end of the `defmain()` method, add the following line directly below the call to 1. `_config_modify_flags` (as seen in `newToolbar.e`):

```
_tbAddForm("mytoolbar1", TBFLAG_NEW_TOOLBAR, false, 0, true);
// Where mytoolbar1 is the name of your new form and _not_ the
name of your toolbar.
```

This will actually add the toolbar to SlickEdit and update the SlickEdit state file.

- 1 Save the file.
- 2.
- 1 This macro (remember that it ends in `.e`) will not compile like standard macros because of the 3. `defmain()` call. Therefore, activate the SlickEdit command line and enter the full path to the newly created macro file (in this case, `newToolbar.e`). This will load the macro and the new toolbar.

In order to share this toolbar, distribute the new macro file that you created in Step 10, and have each user run it on the SlickEdit command line as described in Step 14. The new toolbar can be displayed by clicking **View → Toolbars → My Toolbar 1**.

Your final macro should look very similar to `newToolbar.e`. Of course you can add more than one toolbar to the macro file using the same steps.

Encoding

To provide better support for editing Unicode and non-Unicode files, file data can be loaded natively as either Utf-8 (Unicode mode) or SBCS/DBCS in the active code page. This means that no conversion is necessary to open a file in one of these encodings. For example, the default text encoding for Ubuntu and Mac is Utf-8. When you open a huge Utf-8 file on Ubuntu or Mac, the file will open in less than 1 second. Unicode mode supports most of the features supported by SBCS/DBCS, but there are a few limitations.

For more information, see [Unicode Limitations](#)

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 using the platform specific default text encoding (Windows: SBCS/DBCS, Mac: Utf-8 Unix: depends on LANG).

There are many encodings available, including:

- **Automatic** - Encoding is chosen based on file settings, file type (extension), and global settings.
- **Text** - Load default text encoding file as Unicode (default text encoding is Utf-8 on some platforms).
- **Text, SBCS/DBCS mode** - Load active code page file in SBCS/DBCS editing mode. Only active code page characters are supported. Has the advantage of providing the best performance when a fixed font is chosen.
- **Binary** - Load file in Utf-8 encoding. Disables all options which automatically translate data in the file (expand tabs, change line ends, etc.).
- **Binary, SBCS/DBCS mode** - Load active code page file in SBCS/DBCS editing mode. Disables all options which automatically translate data in the file (expand tabs, change line ends, etc.).
- **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 or Utf-8 depending on the OS default text encoding.
- **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 or Utf-8 depending on the OS default text encoding. This option is NOT fool-proof and may give incorrect results.
- **Auto Unicode Utf-8** - 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 Utf-8.

- **Auto Unicode2 Utf-8** - 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 Utf-8. 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 or Utf-8 depending on the OS default text encoding. 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.
- **Auto HTML** - This encoding specifies that the file encoding be determined based on the first HTML charset specified. Otherwise, the file is loaded as SBCS/DBCS or Utf-8 depending on the OS default text encoding.
- **Auto HTML5** - This encoding specifies that the file encoding be determined based on the first HTML charset specified. Otherwise, the file is loaded as Utf-8.
- **EBCDIC, SBCS/DBCS mode** - Load file in SBCS/DBCS editing mode. Only active code page characters are supported. Has the advantage of providing the best performance when a fixed font is chosen.

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 UCN 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 **Automatic**, then the global setting defined at **Tools → Options → Languages → File Extension Manager** 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 **Edit → Other → 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 **Edit → Other → Copy Unicode As → Java/C# (UTF-16 \uHHHH)** menu item.
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:

- Proportional font tab character expansion is not yet fully supported. Since Unicode support requires the use of proportional fonts, tab expansion is not fully supported.
- The Unicode line end character **0x2048** is not supported.
- Casing features (uppercase, lowercase, ignore case) do not support surrogates. On Windows, the WIN32 calls are relied upon for casing support, and Windows casing features do not support surrogates. See [Surrogate Support](#).
- Version control supports files containing Unicode data but does not support file names that contain characters not in the active code page.
- Truncation line length and bounds columns are 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.

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 or UTF-8, loading these files is significantly faster. This is very important to our customers who expect superior performance from SlickEdit®.
- Unicode editing mode requires using proportional fonts. Not all features can work exactly like 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.

Environment Variables

Below is a list of environment variables that can be used within SlickEdit®. Configuration environment variables are set in the operating system or in `user.cfg.xml` file. For more information see [Setting Environment Variables in user.cfg.xml](#)

You can also use the `set` command from the SlickEdit command line 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 `SLICKEDITCONFIG` environment variable in `user.cfg.xml`. `SLICKEDITCONFIG` determines where the editor looks for `user.cfg.xml`. When the editor starts up, it sets the value of environment variables specified in `user.cfg.xml`. For more information, see [Setting Environment Variables in user.cfg.xml](#).

Environment Variable	Description
<code>VSLICKRESTORE</code>	Directory to store Auto Restore files.
<code>SLICKEDITCONFIG</code>	<p>Directory where user's local configuration files are stored. Used in multi-user environments. Defaults to:</p> <ul style="list-style-type: none"> • (Windows) .../My Documents/My SlickEdit Config/[version]/ • (Mac) \$HOME/Library/Application Support/SlickEdit/[version]/ • (Linux, UNIX) \$HOME/.slickedit/[version]/ <p>Note that <code>VSLICKCONFIG</code> and <code>VSLICKCLASSICCONFIG</code> are no longer supported. <code>VSLICKCONFIG</code> has been replaced with a new variable, <code>SLICKEDITCONFIG</code>.</p>
<code>VSLICK</code>	Specifies additional command line arguments to editor as if you were typing them in when invoking the editor. See also Invocation Options .
<code>VSLICKPATH</code>	One or more directories separated with a semicolon

Environment Variables

Environment Variable	Description
	(;) (or a colon [:] on UNIX) where batch macros or executable files are searched.
VSLICKMACROS	One or more directories separated with a semicolon (;) (or a colon [:] on UNIX) that contain macro files (* .e). VSLICKPATH must also contain the directories listed here.
VSLICKBIN	One or more directories separated with a semicolon (;) (or a colon [:] on UNIX) that contain binary files. VSLICKPATH must also contain the directories listed here.
VSLICKBITMAPS	One or more directories separated with a semicolon (;) (or a colon [:] on UNIX) that contain bitmap files (*.bmp). VSLICKPATH must also contain the directories listed here.
VSLICKMISC	One or more directories separated with a semicolon (;) (or a colon [:] on UNIX) that contain miscellaneous files including *.api, *.idx, vslick.sta, main.dct, *.pif, *.ini, and *.lst. VSLICKPATH must also contain the directories listed here.
VSLICKTAGS	Specifies global tag files. One or more file names separated with a semicolon (;) (or a colon [:] on UNIX) that contain tags. Do not put this environment variable in user.cfg.xml.
VSLICKBACKUP	Directory to place backup files. Affects +D (default) and -D backup configurations only.
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 dos command and shell function. The complete path to the xterm program must be specified. You may not specify the -e option in the command string. For example, setting VSLICKXTERM to: /usr/X11/bin/xterm -geometry 80x40

Setting Environment Variables in user.cfg.xml

Environment Variable	Description
	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.
VSLICKXNOPLUSNEWMSG	Suppresses a message when starting a second instance of SlickEdit.

Setting Environment Variables in user.cfg.xml

Along with whatever facilities are provided by your operating system to set environment variables, you can set configuration environment variables in the file `user.cfg.xml`. 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/[version]/`
- Mac: `$HOME/Library/Application Support/SlickEdit/[version]/`
- Linux and UNIX: `$HOME/.slickedit/[version]/`

Below is text from a sample `user.cfg.xml` file with an environment profile.

```
<options>
    <misc.environment n="misc.environment" version="1" >
        <!-- Use semicolons for separating directories on Windows -->
        <p n="VSLICKPATH"
v="%VSLICKBIN%;%VSLICKMACROS%;%VSLICKBITMAPS%;%SLICKEDITCONFIGVERSION%;%SLICKEDITCONFIG%" configs="win"/>
            <!-- Use colons for separating directoris on Mac and Unix platforms. -->
            <p n="VSLICKPATH"
v="%VSLICKBIN%:%VSLICKMACROS%:%VSLICKBITMAPS%:%SLICKEDITCONFIGVERSION%:%SLICKEDITCONFIG%" configs="unix mac"/>
                <p n="VSLICKMACROS" v="%VSLICKMACROS%;%SLICKEDITCONFIG%" configs="win"/>
                    <p n="VSLICKMACROS" v="%VSLICKMACROS%:%SLICKEDITCONFIG%"
```

```
configs="unix mac" />
    <p n="VSLICKINCLUDE" v="%SLICKEDITCONFIG%" />
    <p n="MYPROJECTVERSION" v="c:\myprog4.2" />
  </misc.environment>
</options>
```

When the editor starts, the following environment variables are created by the editor:

- **VSDRIVE** - Drive letter followed by a colon (:) where editor executable resides.
- **VSDIR** - Directory of editor executable with a 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 using the **set** command on the SlickEdit command line. 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). 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 under **Help → 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 Slick-C 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, click **Macro → Set Macro Variable** (or use the **gui_set_var** command) 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. For more information, see [Set Variable Dialog](#).
- 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
def_alias_case	Controls whether alias identifier matching is case-sensitive. Set to i to make alias matching case-insensitive (default). Set to e to turn on case-sensitivity.

Table of Configuration Variables

Configuration Variable	Description
def_auto_linecomment	Change to 0 to turn off automatic line comment insertion.
def_binary_ext	This variable is used for the Brief emulation, or in other emulations if def_list_binary_files is set to false . The space-delimited extensions listed by this variable are filtered out by the edit command's completion. The default is .ex .obj .exe .lib .
def_buflist	<p>Change this variable to find the initial file in Buffer List. The default is 3. This macro variable determines how the list_buffers 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"> • SORT_BUFLIST_FLAG - 1 • SEPARATE_PATH_FLAG - 2 <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>
def_ctags_flags	This variable is a safeguard against parsing past the end of a proc when the braces mismatch. To have SlickEdit® recognize the second dd , set the value of this variable to 10 .
def_debug_logging	(Pro only) If you change this value to 1 , then run the integrated debugger and let it time out, a vs.log file will be created in your config directory under the logs subdirectory.
def_deselect_copy	Set to 1 in Brief emulation to deselect after a copy.

Table of Configuration Variables

Configuration Variable	Description
<code>def_do_block_mode_key</code>	Set this variable's value equal to 0 to stop SlickEdit from inserting characters on every line of a block selection.
<code>def_error_re2</code>	Edit this variable to change from the SlickEdit regular expression used for compile/build errors.
<code>def_fast_auto_READONLY</code>	When set to 1 , this option speeds up the Auto read only feature by only checking the attribute on disk (not opening every file). See Load File Options for more information.
<code>def_filelist_show_dotfiles</code>	Controls the global Show files beginning with a dot option (Tools → Options → Appearance → General). 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. (Dot files are files with names beginning with a dot character.)
<code>def_from_cursor</code>	Default is 0 . If non-zero, the commands upcase_word , lowercase_word , and cap_word will start case change from the cursor position instead of the beginning of the current word.
<code>def_linenewrap</code>	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.
<code>def_linux1_shell</code>	To use an alternate shell, set this variable to the shell that you want to run (for example, /bin/bash -i). This will cause the editor to use your process shell.
<code>def_max_filehist</code>	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.
<code>def_max_mffind_output</code>	This variable is set for performance reasons. You

Table of Configuration Variables

Configuration Variable	Description
	can increase the amount of information displayed in the Output tool window during a multi-file search by changing this to your desired setting.
def_max_workspace_hist	Increases the length of the Workspace history list in the Project menu. Enter the number of files you want to see in the history.
def_modal_paste	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.
def_plusminus_blocks	When the value is set to 1 , the plusminus command will try to find code blocks to expand or collapse if the cursor is on a line that does not have a Plus or Minus bitmap on it. The default is 1 .
def_preplace	Default is 1 . If the value is set to 0 , the save command will NOT prompt you if you are inadvertently overwriting a file. For example, if you invoke the command savexyz , and an <i>xyz</i> file already exists, and <i>xyz</i> is not the name of the current buffer, you are prompted by default whether you wish to overwrite the file.
def_rwprompt	Default is 1 . Change this to 0 to suppress the pop-up that asks: Do you want to update the read-only attribute of the file on disk?
def_save_macro	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.
def_shift_updown_line_select	Set this value to 1 for Shift+Up or Shift+Down to select the current line.
def_show_makefile_target_menu	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).

Configuration Directories and Files

Configuration Variable	Description
def_switchbuf_cd	Set this variable equal to 1 to change the current working directory to the file that currently has focus in the editor. This variable is on by default in the GNU Emacs emulation, and off in all other emulations.
def_top_bottom_push_bookmark	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 (.process buffer). The default value is 0 .
def_undo_with_cursor	Set this value to 1 to enable the undo of each cursor movement.
def_update_context_max_file_size	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).
def_vc_advanced_options	Set to this variable to 0 to remove advanced options that decrease performance when using ClearCase version control.
def_vtg_tornado	(Pro only) Set this variable value to 0 to prevent Context Tagging of Tornado files.
def_xml_no_schema_list	To prevent SlickEdit from accessing the Internet to validate and get color coding information from DTDs, 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: .xml .xsl .xsd . 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, [Table of User Configuration Files](#).

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.

Configuration changes are saved when you exit the editor. Note that 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.

Configuration Directory Location

By default, the user configuration directory is in the following location, depending on the operating system you are using:

- Windows: .../My Documents/My SlickEdit Config/Editor_Version/
- Mac: \$HOME/Library/Application Support/SlickEdit/Editor_Version/
- Linux, UNIX: \$HOME/.slickedit/editor_version/

SlickEdit® creates a versioned config subdirectory for each version of SlickEdit you run.

Tip

You can view the path to the config directory by clicking **Help → About SlickEdit**. When SlickEdit displays the config directory, it includes the versioned subdirectory. For example: C:\Documents and Settings\SlickEditUser\My Documents\My SlickEdit Config\20.0.3\.

If you want to use a different directory for your config files, set the **SLICKEDITCONFIG** environment variable (see [Environment Variables](#)), or specify the location by using the **-sc** invocation option on the command line (see [Invocation Options](#)). In both cases, SlickEdit will create a versioned subdirectory below the specified directory.

Resetting the Configuration Directory

For troubleshooting purposes, it is sometimes helpful to reset the configuration directory to the defaults. SlickEdit Product Support may also ask you to use a default configuration to help debug problems. To reset to the default configuration:

1. Exit any instances of SlickEdit® that are running.

2. Make a backup of the user config directory and store the backup in a safe location.
3. Delete the contents of the versioned config directory (i.e. 20.0.3) but NOT the directory itself.
4. Start SlickEdit.

After completing these steps, SlickEdit opens with the default configuration settings. Note that you will be prompted for all of the initial setup information just as if you were running SlickEdit for the first time (see [Running SlickEdit](#)).

Note

If, instead of deleting the contents of the directory, you delete the directory itself, SlickEdit will attempt to migrate your settings from a previous version.

Table of User Configuration Files

The table below provides a list of the user configuration files.

User Config File	Description
compilers.xml	An XML file that contains compiler include/jar file configurations. Often the settings found there have been auto generated. The compiler profiles are user customizable. It is only safe to delete this file if you've never created or modified compiler profiles.
diffmap.ini	A text file which stores root directory mappings. It's used to automatically set other text box when typing in the first text box of the DIFFzilla® dialog. This data is a lot like auto-restore data but since there's a standalone vsdiff program, it made more sense for this to be in a separate file. This file can be deleted without causing any problems or losing important settings.
diffsessions.xml	A text file which contains diff session history and named diff sessions.
perfile.xml	An XML text file that contains buffer history information which is used when you open an addition file. The most valuable information this stores is your previous edit location so that it can be restored when you reopen a file. It also contains the following buffer settings: encoding, Soft Wrap, Language mode, and XML Wrap scheme. This file can be deleted without causing any problems or

User Configuration Directory

User Config File	Description
	losing important settings.
personal.sca	An XML file that contains global annotation options and settings. The annotation feature allows you to make various notes about certain lines of files without modifying the file itself. Great for code review notes.
project.vpe (UNIX: uproject.vpe)	A text file that contains user-defined language-specific projects.
user.cfg.xml	(very import) An XML file containing almost all non-Auto Restore like configuration settings. Includes settings for fonts, color profiles, emulation, key bindings, environment variables, language settings, beautifier profiles, color coding profiles, printing profiles, FTP profiles, and more.
(Windows only) usercpp.h	A text file that contains global defines (default preprocessing) for Context Tagging® of C++ and C code. (Windows only)
(Unix and macOS only) unxcpp.h	A text file that contains global defines (default preprocessing) for Context Tagging® of C++ and C code. (Unix and macOS only)
usersystemverilog.svh	A text file that contains global defines (default preprocessing) for Context Tagging® of SystemVerilog code.
userverilog.v	A text file that contains global defines (default preprocessing) for Context Tagging® of Verilog code.
usrprjtemplates.vpt	A text file that contains user-defined project packages.
vrestore.slk	A text file that contains auto-restore information, such as buffer information and command line history. This file can be deleted without causing any problems or losing important settings.
vslick.sta	A binary file that contains dialog boxes, menus, macro pcode, key bindings, and all other

System Configuration Files

User Config File	Description
	configuration data not stored in one of the other configuration files. Both user and system configuration information is stored here. It is safe to delete this file but only if you are not running any instances of SlickEdit. The changes to this file will be re-applied when you restart SlickEdit.
vusrobs.e (UNIX: vunxobj.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, vusr10e.e.
*.vpwhist	A text file that contains workspace auto-restore information including windows, files, break points, tool windows, toolbars, and more. This file can be deleted without causing any problems or losing important settings.

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
com_slickedit.base.zip	A plug-in that contains almost all default system settings. Most of the settings are stored in .cfg.xml files. A .cfg.xml file typically contains one profile with any number of properties.
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.

System Config File	Description
syscpp.h (UNIX: usyscpp.h)	A text file that contains system-defined default preprocessing for Context Tagging® of C++ and C code.
systemverilog.svh	A text file that contains system-defined default preprocessing for Context Tagging® of SystemVerilog
verilog.v	A text file that contains system-defined default preprocessing for Context Tagging® of Verilog
vslick.sta	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.

.cfg.xml File Format

.cfg.xml files contain profiles with properties. Profile names are case insensitive. Property names are case sensitive. There's rarely a need for case insensitive property names and you can always add a "display_name" attribute to a property for a pure case insensitive implementation when needed.

Normalized and Unnormalized Profiles

When profiles are loaded, they are normalized if they aren't already normalized.

A normalized profile with a normalized property looks like this:

```
<options>
    <profile n='misc.options' version="1">
        <p n="buffer_kcache_size" v="2005"/>      <-- example of
normalized standard property -->
    </profile>
</options>
```

All Slick-C profile/property APIs are designed only to handle the above normalized profile format (_plugin_get_profile, _plugin_set_profile, etc.). Before user.cfg.xml is saved, the Slick-C function _xmlcfg_apply_profile_style is called to unnormalize the profile.

The unnormalized version of the above profile looks like this:

```
<options>
    <misc.options n="misc.options" version="1">
        <buffer_kcache_size v="2005"/>      <-- example of unnormalized
standard property -->
    </misc.options>
</options>
```

It's much easier to define an XML Schema for unnormalized profiles (used for XML validation, color coding, and auto completion). Otherwise, profiles would always be stored in the normalized form.

Standard and XML Properties

For simplicity, all property names are case sensitive. There's rarely a need for case insensitive property names and you can always add a "display_name" attribute to a property for a pure case insensitive implementation when needed.

There are two types of properties: Standard and XML.

Normalized standard properties look like this:

```
<p n="name_of_property" v="value-of-property" [apply="0" | "1"]  
[configs="space-delimited-configs"] />
```

Unnormalized standard properties look like this:

```
<name_of_property v="value-of-property" [apply="0" | "1"]  
[configs="space-delimited-configs"] />
```

Only property names that are valid XML element names can be stored in the unnormalized standard property form. Usually the normalized form is used if the property names may not be valid XML element names.

Valid predefined configs for *space-delimited-configs* are win, mac, unix (all UNIX platforms except Mac), linux, intelsolaris, sparsolaris, aix, and hpx.

The standard property form is very restricting but it is often all that is needed. If you need to add additional attributes or need child elements, you must use the less restrictive XML property form.

Examples

<pre><!-- Normalized --> <p n="n1" v="v1"/> <p n="n2" v="v1" apply="1" configs="win mac"/> apply="1" configs="win mac"/></pre>	<pre>Unnormalized --> <n1 v="v1"/> <n2 v="v1"</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------

XML properties look like this:

Normalized XML properties look like this:

```
<p n="name_of_property"  
[any-custom-attribute-value-pairs-but-no-v-attribute] [apply="0" | "1"]  
[configs="space-delimited-configs"]/>
```

or this:

```
<p n="name_of_property"  
[any-custom-attribute-value-pairs-but-no-v-attribute] [apply="0" | "1"]  
[configs="space-delimited-configs"]>  
[any-xml-here]  
</p>
```

Unnormalized XML properties look like this:

```
<name_of_property  
[any-custom-attribute-value-pairs-but-no-v-attribute] [apply="0"|"1"]  
[configs="space-delimited-configs"]/>
```

or this:

```
<name_of_property  
[any-custom-attribute-value-pairs-but-no-v-attribute] [apply="0"|"1"]  
[configs="space-delimited-configs"]>  
    [any-xml-here]  
</name_of_property>
```

Only property names that are valid XML element names can be stored in the unnormalized XML property form. Usually the normalized form is used if the property names may not be valid XML element names.

Examples

<pre><!-- Normalized <p n="n1" a="avalue" b="bvalue"/> b="bvalue"/> <p n="n2" apply="1" configs="win mac">avalue</p> configs="win mac">avalue</n2></pre>	<pre>Unnormalized --> <n1 a="avalue" <n2 apply="1" avalue</n2></pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------

The Standard property form is used for the very common name/value pair scenario. The XML property form is used for everything else. You may have both forms of properties in a single profile.

File Search Order

Search Order for Executable Files

The search order for executable files, batch macro programs, and miscellaneous files is:

1. Current directory.
2. Configuration directory.
3. Paths specified in **VSLICKPATH** environment variable.
4. Paths specified in **PATH** environment variable.

Color Coding Profiles

For more basic information about using Color Coding, see [Colors, Color Coding, and Symbol Colors](#).

To modify the color coding profiles, use one of the following methods:

- Use the language-specific **Color Coding** options screen (see [Language-Specific Color Coding Options](#)). This is much easier to learn and helps avoid mistakes.
- Hand edit an XML color coding profile in `user.cfg.xml`. The `user.cfg.xml` file defines language-specific color coding as well as many other settings.

Creating a New Color Coding Profile

Perform the steps below to create a new color coding profile for a language.

1. From the main menu, click **Tools** → **Options** → **Languages**, expand your language category and language, then select **Color Coding**.
2. Create the new color coding profile by clicking the **New** button.
3. Enter the new color coding **Profile name** and click the **OK** button.
4. Use the Color Coding dialog to set the various options (add keywords, define comments/strings/numbers, etc.).
5. Click **OK** to commit the changes or **Apply** to try out your changes.

User created or modified profiles are stored in `user.cfg.xml`.

Color Coding Profile Format

Take a look at some of the existing color coding profiles shipped with SlickEdit to help you figure out how to add certain color coding constructs.

The system color coding profiles shipped with SlickEdit are stored in `.cfg.xml` files in the `com_slickedit.base.zip` plug-in along with many other profiles. You can open files out of `com_slickedit.base.zip` without unzipping it first. Type "c:\Program Files\SlickEdit Pro ???.?plugins\com_slickedit.base.zip\com_slickedit.base\color_profiles\" in the Open tool window to see a list of color coding profiles. It's a good idea to look at some of these profiles to better understand of how the XML properties in color coding profiles work. Features such as Smart Open (Open tool window and "e" command support it), multi-file search & replace, recursive diffs, and the file manager all support reading files inside a zip file.

Note

The standard operating system open dialog does not support opening files inside a zip file. It's easiest to use the Open tool window to browse and open files in a zip file.

Part of the "cpp" color coding profile is shown below:

```
<options>
    <colorcoding_profiles n="colorcoding_profiles.cpp" version="2">
        <case_sensitive v="1"/>
        <idchars v="a-zA-Z_\$ 0-9"/>
        <styles v="idparenfunction doxygen xmldoc javadoc color_inactive_cpp
embeddedasm"/>
        ...
        <p n="comment, // ">
            <attrs line_continuation_char="\\" color_to_eol="comment" />
        </p>
        <p n="k, case" />
        ...
    </colorcoding_profiles>
</options>
```

Note

If you use the Slick-C _plugin_get_profile() function to fetch an XML profile, the profile will be returned in normalized format and not the XML format shown above which is the unnormalized format stored on disk. See [Normalized and Unnormalized Profiles](#).

A Word About the Color Coding Profile XML Format

You will notice that the "n" attribute of the "p" element contains comma delimited data instead of being split into multiple attributes. This is done because the configuration engine uses the property name (not a configurable combination) as the key when diffing profiles. This format was chosen for best diff results. Only changes to properties are stored in user.cfg.xml. This standardized configuration engine allows the Dev Team to avoid reinventing the wheel for system/user configuration data which only stores user changes to modified/added properties. When you update to a newer version of SlickEdit, you automatically get new property settings that you haven't modified.

Examples:

```
<p n="comment, // ">
    <attrs line_continuation_char="\\" color_to_eol="comment" />
</p>
<p n="k, case" />
```

Property Syntax for Matching Language Elements

The XML property syntax for matching color coding language elements is:

```
<p n="color,start-delimiter">
  [ <attrs [flags="flags"]
    [end="end-delimiter"]
    [nest_start="nest-start" nest_end="nest-end"]
    [start_col="start-col" [end_col="end-col"] ]
    [end_start_col="end-start-col" [end_end_col="end-end-col"] ]
    [start_color="color"] [end_color="color"]
    [color_to_eol="color"] [end_color_to_eol="color"]
    [case_sensitive="case-sensitive"]
    [escape_char="escape-char"] [doubles_char="doubles-char"]
    [order="order"]
    [embedded_lexer="embedded-lexer"]
  >

  [ <iattrs
    [type="color"]
    [start="start-delimiter"]
    [flags="flags"]
    [end="end-delimiter"]
    [nest_start="nest-start" nest_end="nest-end"]
    [start_col="start-col" [end_col="end-col"] ]
    [end_start_col="end-start-col"
      [end_end_col="end-end-col"] ]
    [start_color="color"] [end_color="color"]
    [color_to_eol="color"] [end_color_to_eol="color"]
    [case_sensitive="case-sensitive"]
    [escape_char="escape-char"]
    [doubles_char="doubles-char"]
    [order="order"]
    [embedded_lexer="embedded-lexer"]
  >
    <iattrs>
  ]
  <attrs>
]
</p>
```

color - Predefined color. See [Color Coding Colors](#).

start-delimiter - Plain text string or regular expression to match depending on if flags specify **regex** or **perlre**. See [Section Replacing with Regular Expressions](#) for information on tagged expressions and

special characters in the replace string.

flags - (Optional) Predefined space delimited flags See [Color Coding Flags](#).

end-delimiter - (Optional) Plain text string or regular expression to match after start-delimiter is found depending on if flags specify **end_regex** or **end_perlre**. See [Section Replacing with Regular Expressions](#) for information on tagged expressions and special characters in the replace string.

nest-start - (Optional) Plain text string or regular expression to match after start-delimiter is found depending on if flags specify **end_regex** or **end_perlre**.

start-col and *end-col* - (Optional) Specifies the columns in which the **start-delimiter** is considered a match. Specify a begin and end column to set a range of columns. Leave the end column blank, to specify that the start delimiter is recognized anywhere after the start column.

end-start-col and *end-end-col* - (Optional) Specifies the columns in which the **end-delimiter** is considered a match. Specify a begin and end column to set a range of columns. Leave the end column blank, to specify that the end delimiter is recognized anywhere after the start column.

start-color - (Optional) Overrides the default start delimiter color.

end-color - (Optional) Overrides the default end delimiter color.

color-to-eol - (Optional) When non-blank, specifies that color coding continues to end of line (if **multiline** flag not specified), end-delimiter (if specified), or end of file (no end-delimiter and **multiline** flag specified) after the start delimiter is matched with this predefined color. See [Color Coding Colors](#).

case-sensitive - (Optional) Overrides the default case sensitive defined by the *case_sensitive* element (see [Other Color Coding Profile Properties](#)).

escape-char - (Optional) Has effect only when **end-delimiter** is non-blank or **color-to-eol** is non-blank. Allows you to define the escape character where the next character is skipped so the **end-delimiter** can be correctly found. Many string type constructs support escaping with a character like backslash (some languages uses a different escape character)

line-continuation-char - (Optional) Has effect only for single line constructs. When this character is found at the end of the line, coloring will continue to the next line.

order - (Optional) A signed integer which determines the order of evaluation of items. In more complicated scenarios where there are multiple **start-delimiter** patterns matching the same text, this is used to choose which match gets processed. Lower values are matched first and take precedence. Sometimes a regular expression which is intended to match a longer pattern also exactly matches a shorter duplicate pattern.

embedded-lexer - (Optional) When non-blank, indicates this item is an embedded language even if it doesn't match an existing color coding profile. When the **start-delimiter** is a regular expression, replacements for tagged expressions and escapes just like a typical search and replace will be performed. See [Section Replacing with Regular Expressions](#) for information on tagged expressions and special characters in the replace string.

Color Coding Colors

Color is one of the following color names:

- **k** -- Keyword
- **comment** -- Comment
- **doc_comment** -- Document comment
- **linenum** -- Line number
- **string** -- String
- **pp** -- Preprocessing
- **pu** -- Punctuation
- **lib** -- Library
- **op** -- Operator
- **user** -- User defined color
- **modified_line** -- Modified line color
- **inserted_line** -- Inserted line color
- **deleted_line** -- Deleted line color
- **number** -- Number color
- **float** -- Floating point number color
- **hex_int** -- Hexadecimal integer color
- **function** -- Function color
- **attribute** -- Attribute color
- **unknown_xml_element** or **unknown_tag** -- Unknown XML or HTML tag color
- **xhtml_element_in_xsl** -- XHTML element in XSL color
- **tag** -- Tag color
- **xml_character_reference** -- XML character reference color
- **unknown_attribute** -- Unknown attribute color
- **doc_comment** -- Documentation comment color
- **doc_keyword** -- Documentation comment keyword color
- **doc_punctuation** -- Documentation comment punctuation color

- **doc_attribute** -- Documentation comment attribute color
- **doc_attr_value** -- Documentation comment attribute value color
- **identifier** -- Identifier color
- **identifier2** -- Identifier 2 color
- **inactive_code** -- Inactive code color
- **inactive_keyword** -- Inactive code keyword color
- **inactive_comment** -- Inactive code comment color
- **markdown_header** -- Markdown header color
- **markdown_code** -- Markdown code color
- **markdownblockquote** -- Markdown blockquote color
- **markdown_link** -- Markdown link color
- **markdown_link2** -- Markdown link 2 color
- **markdown_bullet** -- Markdown bullet color
- **markdown_emphasis** -- Markdown emphasis color
- **markdown_emphasis2** -- Markdown emphasis 2 color
- **css_element** -- CSS element color
- **css_class** -- CSS class color
- **css_property** -- CSS property color
- **css_selector** -- CSS selector color
- **other** -- Other color

Color Coding Flags

flags is a space delimiter list zero or more of the following:

Flag name	Description
first_non_blank	Specifies that the match must occur as the first non-blank character. Space or tab characters are considered blanks. This option can not be used in combination with the check_first option.
check_first	Specifies that the match be checked before

Flag name	Description
	scanning for other matches. This is most useful for column oriented languages where a character in a particular column means the entire line is a line comment (COBOL). This option can not be used in combination with the first_non_blank option.
regex	Specifies that start-delimiter is a SlickEdit regex.
perlre	Specifies that start-delimiter is a Perl regex.
end_regex	Specifies that end-delimiter is a SlickEdit regex.
end_perlre	Specifies that end-delimiter is a Perl regex.
multiline	Specifies that color coding continues after start-delimiter either to the end-delimiter or end of file.
embedded_lexer_prefix_match	When specified, the prefix of embedded_lexer is matched against other color coding profiles (ex "cppEOF" would match "cpp").
apply_multiline_at_eol	Has effect only when multiline flag specified. This option is typically used for here-document constructs where the start should be colored as Other color, then the text until the end of line is colored as if this construct was never hit, and then subsequent lines continue with this construct.
end_embedded_at_bol_if_possible	When the start-delimiter for an embedded language construct starts at the end of a line and the end-delimiter is the first non-blank in a line, choose this option. That way only lines in between the start and end delimiters are colored in embedded language color.
dont_color_as_embedded_if_possible	Continue to use the current background color which may already be embedded.
color_as_embedded_if_found	If embedded-lexer is found, switch to the background to embedded. Otherwise, the current background color which may already be embedded is used.
embedded_end_is_token	This is useful for interpolated strings where finding

Flag name	Description
	the end-delimiter requires tokenizing the text so that tokens like strings which could contain the end-delimiter are skipped. Note that defining an interpolated string requires the outer string start/end delimiters to be defined and then an inner start/end delimiters (often \\$\{ and \} when start delimiter is a SlickEdit regular expression) to be defined where this option is checked. Use iattrs element to add the sub item for interpolation and set the start/end delimiters. You may need to set the nest-start and nest-end to { and }. Scala interpolated strings require the interpolation to support nested braces where nest-start and nest-end are { and } respectively.

Adding Color Coding for HTML/XML Tags, Attributes, and Values

The HTML/XML tag support is designed specifically for these languages and bulletin board tag languages which are very similar.

First you need to define the multi-line construct used for all tags.

```
<!-- comment color won't be used if tags are defined -->
<p n="comment,<">
    <attrs end=">" flags="multiline"/>
</p>
```

Add tags like this:

```
<p n=",<,p," v="id class style title lang dir align onclick ondblclick ..." />
<p n=",<, /p," v=" " />
```

The property name must start with a comma. The next item indicates which multi-line construct the tag should be added to. This is typically '<' but can also be '|'. The word that follows is the tag name.

The "v" attribute above defines valid attributes for the tag.

Define valid attribute values like this.

```
<p n=",<,,align" v="center right left justify char top middle bottom
baseline" />
```

or like this:

```
<p n=",<,p,align" v="center right left justify char top middle bottom  
baseline"/>
```

The property name must start with a comma. The next item indicates which multi-line construct the tag should be added to. This is typically '<' but can also be '|'. The word that follows is optionally the tag name which is only needed if the attribute values depend on a specific tag.

Other Color Coding Profile Properties

The table below lists other properties which are not used for added keywords, comments, tags, attributes, or attribute values

XML Property syntax	Description
<code><case_sensitive v="[0 1]" /></code>	Defines the case sensitivity for the language. This is typically near the top of the profile but order of properties does not matter.
<code><idchars v="start_id_chars after_id_chars" /></code>	Defines the characters that are the start of a valid identifier and additional valid characters that may follow. start_id_chars and after_id_chars must be a valid SlickEdit syntax regular expression when enclosed in [] (i.e [start_id_chars] must be a valid SlickEdit syntax regex). You may use a dash (-) character to specify a range, for example, A-Z specifies uppercase letters. To specify a dash or backslash (\) character as a valid word character, place a backslash before the character. Many character classes are supported. See SlickEdit® Regular Expressions .
<code><styles v="styles" /></code>	Defines zero or more space delimited styles. See Color Coding Style Values below for a list of available styles.
<code><mn_flags v="mn-flags" /></code>	Defines zero or more space delimited number flags. See Color Coding Number Flags for a list of available flags.
<code><mn_int_suffixes v="suffix-list" /></code>	Space delimited list of supported integer suffixes. By default, all suffixes are case insensitive. Prefix the suffix with "\c" to specify a case sensitive suffix.

XML Property syntax	Description
	If your prefix starts with a backslash, uses two backslashes (ex "\\").
<code><mn_float_suffixes v="suffix-list"/></code>	Space delimited list of supported floating point suffixes. By default, all suffixes are case insensitive. Prefix the suffix with "\c" to specify a case sensitive suffix. If your prefix starts with a backslash, uses two backslashes (ex "\\").
<code><mn_hex_suffixes v="suffix-list"/></code>	Space delimited list of supported hexadecimal integer suffixes. By default, all suffixes are case insensitive. Prefix the suffix with "\c" to specify a case sensitive suffix. If your prefix starts with a backslash, uses two backslashes (ex "\\").
<code><mn_digit_separator_char v="digit-separator-char"/></code>	Specifies a single character which is allowed between digits. For example, C++ supports a single quote character (ex 123'000'000). Perl and many other languages support an underscore (123_000_000).
<code><inherit v="cc_profile"/></code>	Inherits token settings from the specified color coding profile.

Color Coding Style Values

The table below describes the *styles* that can be used:

Style Name	Description
bbc	Enables additional support for BBC language.
cics	Enables special support for CICS language.
cobol	Enables special support for Cobol
color_inactive_cpp	Enabled inactive colors for C++ #if'zeroed code.
css	Enables special support for CSS language.
doxygen	Adds support for color coding Doxygen keywords in doc_comment's.

Color Coding Profile Format

Style Name	Description
eof	Adds a table entry to color everything after 0x1a (EOF char) as a comment.
heredocument	Adds support for generic HERE documents similar to Perl syntax but primarily backward compatible with previous versions of SlickEdit. Note that the Perl color coding definition no longer uses this option. Instead, more precise Perl specific table items are defined.
html	Enables additional support for HTML language.
idparenfunction	An identifier followed by an open parenthesis indicates a function (like C++ and Java).
idstartnum	Indicates that identifiers may start with a number. Special identifier support for COBOL.
javadoc	Adds support for color coding JavaDoc keywords in doc_comment's.
jcl	Special support for JCL
linenum	Line numbers may be found as the first non-blank symbol of a line like BASIC.
markdown	Enables special support for Markdown language syntax.
model204	Enables special support for Model 204 language.
os390asm	Enables special support for OS390 assembler
packageimport	Language has Java syntax package and import statement where non-quoted file name follows package and import keyword.
perl	Enables additional support for Perl language.
ppkeywordsanywhere	Specifies that preprocessing keywords may appear anywhere in a line and not just after leading whitespace.
puppet	Enables additional support for puppet language.

Style Name	Description
ruby	Enables additional support for Ruby language.
Scala	Enables additional support for Scala language.
Scala	Enables additional support for XML language.
xmldoc	Adds support for color coding XML Doc keywords in doc_comment's.
xml_literals	Adds support for embedded XML literals.

Color Coding Number Flags

The table below describes the *mn_flags* that can be used:

Flag Name	Description
allow_hex_digits	This option is useful for coloring hexadecimal numbers which start with a digit (no prefix characters) and may contain hexadecimal digits. Typically there is a suffix character like 'H' added to mn_hex_suffixes .
digit_float	Indicates floating point starting with a digit (ex. 1.2) is colored in float point number color.
digit_int	Indicates integers starting with a digit (ex. 123) is colored in number color.
dote_float	Indicates floating point numbers allow the exponent to immediately follow the decimal point (ex 1.e4).
dotp_float	Indicates hexadecimal floating point numbers allow the exponent to immediately follow the decimal point (ex 1.p4).
dot_float	Indicates floating point numbers may start with a decimal point (ex .123).
d_exponent	Indicates floating point numbers also allow the exponent to be specified with a "D" (ex 1.4d5).

Flag Name	Description
f_exponent	Indicates floating point numbers also allow the exponent to be specified with an "F" (ex 1.4f5).
no_exponent	Indicates that floating point numbers do not have an exponent.
verilog_base_squote_hex	Indicates Verilog syntax base single quote numbers like 16'hFFFF and 16'd1234 are color coded in number color.
zerob_binary	Indicates binary numbers such as 0b1010 is color coded in number color.
zerod_decimal	Indicates whether decimal numbers such as 0d89 is color coded in number color.
zeroo_octal	Indicates octal numbers such as 0o777 is color coded in number color.
zerox_hex	Indicates hexadecimal numbers such as 0x123ABC is color coded in number color.
zerox_p_float	Indicates floating point hexadecimal numbers are colored in floating point number color. Many languages have adopted this standard syntax for hexadecimal floating point syntax.

Editing a Key Binding Profile

If you are creating a new emulation or if you change many key bindings, you might want to edit your key binding profile instead of using the [Key Binding Options](#) screen. Using the dialog is less error prone than editing a key binding profile. User modified key binding profiles are stored in `user.cfg.xml`. A sample profile with a few key binding changes is shown below:

```
<eventtab_profiles n="eventtab_profiles.emulation-CUA" version="1">
    <!-- Change or add Ctrl+F12 and bind it to the insert-docbook-id
command -->
    <p n="'C-F12'" v="insert-docbook-id"/>
    <!-- Change or add Command+/ and bind it to the cut-word -->
    <p n="'M-/'" v="cut-word"/>
    <!-- Remove the key binding for Alt+F -->
    <d n="'A-F'" />
    <!-- Bind the range of keys Ctrl+0..Ctrl+9 to the alt-bookmark command
-->
    <!-- The "d" element above does not support a key range -->
    <p n="'C-0'-'C-9'" v="alt-bookmark"/>
</eventtab_profiles>
```

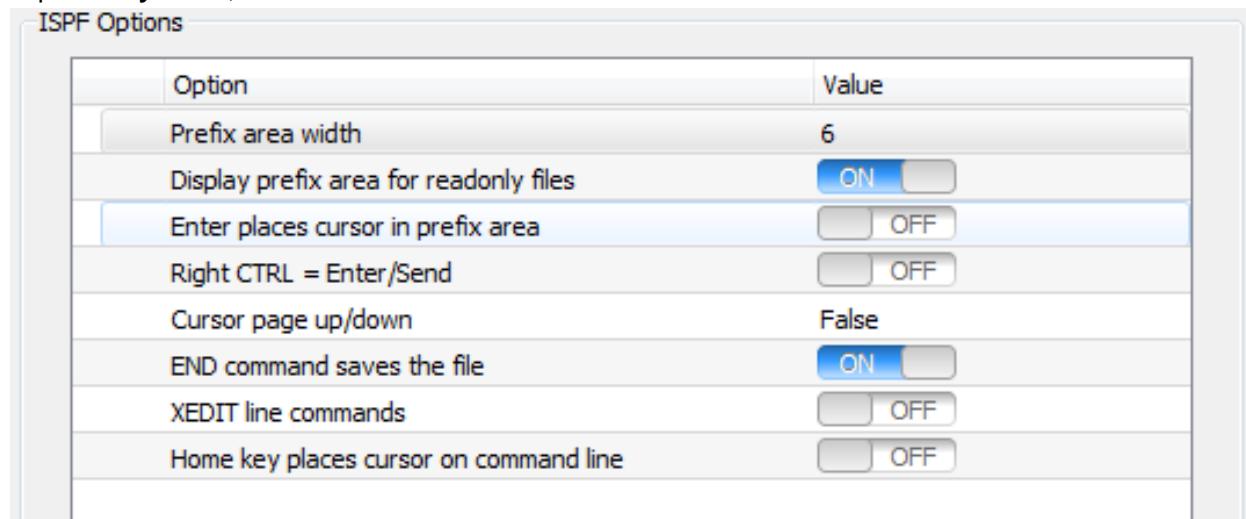
See [Event Names](#) for a list of valid key names. Enter key names in upper case (except in the rare case where case matters like "a"). Property names (the key names) are case sensitive. If you enter a key in the wrong case, it may bind correctly but the configuration engine will not be able to correctly generate user differences.

Using the ISPF and XEDIT Emulations

This section describes the features of the ISPF editor emulation and outlines some XEDIT line commands.

ISPF Options

The ISPF Emulation options screen is used to tune various ISPF emulation behaviors. To access these options, SlickEdit must be in ISPF emulation mode. Then, from the main menu, click **Tools** → **Options**, expand **Keyboard**, and select **ISPF Emulation**.



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**.
- **Display prefix area for readonly files** - The prefix area is used to enter commands. By default, the prefix area is not displayed for read-only files since most of the commands cannot be used. When set to **True** the following line commands are allowed in read-only mode:

Command	Description
ISPF Line Labels	Define a label.
ISPF Line Command BNDS	Insert a column boundary ruler line.
ISPF Line Command COLS	Insert a column ruler line.
ISPF Line Command First	Expose one or more lines at the beginning of a

	block of excluded lines.
ISPF Line Command Last	Expose one or more lines at the beginning of a block of excluded lines.
ISPF Line Command Show	Expose one or more lines having the leftmost indentation level in a block of excluded lines.
ISPF Line Command TABS	Displays the tab definition line.
ISPF Line Command Exclude	Specifies one or more lines to be hidden (excluded).
ISPF Line Command Select	Select a block of lines.

- **Enter places cursor in prefix area** - When this check box is selected, the **Enter** key places the cursor in the prefix area of the next line. When this check box is cleared, the **Enter** key places the cursor in column 1 of the next line.
- **Right CTRL = Enter/Send** - When this check box is selected, 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 this check box is cleared, 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 this check box is selected, 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 this check box is cleared, page up/down always scrolls one page.
- **END command saves the file** - When this check box is selected, 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 this check box is selected, the prefix area will support XEDIT-style line commands.
- **Home places cursor on command line** - When this check box is selected, 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.

More ISPF-related options are available on the language-specific **General** and **Editing** options screens (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **General** and **Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **Editing**). These options include **Auto CAPS** mode and editing of boundaries and the truncation column. The **Bounds** setting is unique to ISPF. It controls column bounds for specific ISPF commands that operate on column ranges. See [Bounds \(Language-Specific\)](#) for more information.

ISPF Primary Commands

The following table of 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 press the **Esc** key, click on the message line, 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

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.

Command	Description
ispf_autosave	Turn on or off prompting to save changes.
ispf_bounds	Set or reset the left and right edit boundaries.
ispf_bnds	Set or reset the left and right edit boundaries
ispf Browse	Browse a data set or member.
ispf_cancel	Closes the current file or PDS member without saving changes.
ispf_caps	Turn on or off automatic capitalization mode.
ispf_change	Replace one string with another within the current buffer.
ispf_chg	Replace one string with another within the current buffer.
ispf_compare	Compare the file you are editing with another file.

ispf_copy	Insert the contents of a file. This command requires a full path, and will not work with only a PDS member name specified.
ispf_create	Create a new file or PDS member containing the contents of the buffer.
ispf_cut	Cut lines out of the current buffer and place them in the clipboard.
ispf_delete	Delete lines in the given line range, or the entire buffer.
ispf_edit	This command is identical to the built-in edit command.
ispf_end	Close the current file.
ispf_exclude	Hide (exclude) lines that match the given search string.
ispf_find	Find occurrences of the given search string in the current buffer.
ispf_flip	Reverse the exclude status of lines.
ispf_hex	Toggle display of the document in Hexadecimal mode.
ispf_hilite	Specify the use of color-coding in the editor.
ispf_locate	Find lines with a specific line prefix.
ispf_move	Move the contents of a file or PDS member into the buffer.
ispf_nonumber	Turn off numbering mode.
ispf_number	Controls line numbering mode. Unlike ISPF, this command does affect how lines are inserted.
ispf_paste	Copy lines from the clipboard to the buffer.
ispf_preserve	Controls saving of trailing blanks.
ispf_rchange	Repeat the change requested by the most recent

	change command.
ispf_renumber	Immediately update the line numbers in a file.
ispf_replace	Save the contents of the current buffer to an existing file.
ispf_reset	Reset the contents of the line prefix area.
ispf_return	Close the current file.
ispf_rfind	Repeat the last find operation requested.
ispf_save	This command is identical to the built-in save command.
ispf_sort	Sort lines of data in a specified order.
ispf_submit	Submit the contents of the current buffer for batch processing.
ispf_swap	Switch to the next buffer.
ispf_tabs	Define logical tab positions.
ispf_unnumber	Blank out the line numbers in a file.
ispf_undo	This command is identical to the undo command.

ISPF Line Commands

The table below shows ISPF edit line commands that 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, or move the cursor left or backspace 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.

ISPF Line Commands

Command	Description
ISPF Line Labels	Define a label.
ISPF Line Command Shift	Shift data left or right.
ISPF Line Command A	Identify a line after which lines are to be inserted.
ISPF Line Command B	Identify a line before which lines are to be inserted.
ISPF Line Command BNDS	Insert a column boundary ruler line.
ISPF Line Command Copy S	Specify lines to be copied to another location.
ISPF Line Command COL	Insert a column ruler line.
ISPF Line Command Delete	Delete one or more lines.
ISPF Line Command First	Expose one or more lines at the beginning of a block of excluded lines.
ISPF Line Command I	Insert one or more blank data entry lines.
ISPF Line Command Lowercase	Convert all uppercase letter alphabetic characters in one or more lines to lowercase.
ISPF Line Command Last	Expose one or more lines at the beginning of a block of excluded lines.
ISPF Line Command Move	Specify lines to be moved to another location.
ISPF Line Command MASK	Display the contents of the mask used with the insert (I) and text entry (TE) line commands.
ISPF Line Command Make Data	Convert one or more no-save lines to data so that they may be saved when the buffer is saved.
ISPF Line Command Overlay	Identify one or more lines over which the copy or move block is to be overlaid.
ISPF Line Command Repeat	Specify lines to be repeated immediately following this line or block.
ISPF Line Command Show	Expose one or more lines having the left-most indentation level in a block of excluded lines.

ISPF Line Command TABS	Display the tab definition line.
ISPF Line Command TE	Insert one or more blank lines to allow power typing for text entry.
ISPF Line Command TF	Reflow paragraphs according to the current column boundary settings.
ISPF Line Command TJ	Join this line with the next line.
ISPF Line Command TS	Divide a line so that data can be added.
ISPF Line Command Uppercase	Convert all lowercase letter alphabetic characters in one or more lines to uppercase.
ISPF Line Command Exclude	Specify one or more lines to be hidden (excluded).
ISPF Line Command Select	Select a block of lines.

ISPF Line Command Documentation

ISPF Line Labels .label

Usage

.label, where *label* does not start with a **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:

- **.zfirst** - The first line in the buffer (abbreviated **.zf**).
- **.zlast** - The last line in the buffer (abbreviated **.zl**).
- **.zcsr** - The current line the cursor is on (abbreviated **.zc**).

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**
- **(([n]** - Shift the block of lines *n* columns left, default **2**
- **) [n]** - Shift the current line *n* columns right, default **2**
- **)) [n]** - Shift the block of lines *n* columns right, default **2**
- **< [n]** - Data shift the current line *n* columns left, default **2**
- **<< [n]** - Data shift the block of lines *n* columns left, default **2**
- **> [n]** - Data shift the current line *n* columns right, default **2**
- **>> [n]** - Data shift the block of lines *n* columns right, default **2**

Remarks

This set of commands is used for shifting data left or right. The versions using parenthesis shift text literally, while the other versions attempt 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, lowcase, 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 cannot 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 press **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, uppercase

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 (**Tools** → **Options** → **ISPF Options**).

XEDIT	ISPF	Description
/	R	Repeat the marked line.
F	A	Paste text following line.

XEDIT	ISPF	Description
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 table below shows ISPF primary commands that are not supported in the ISPF emulation mode. The unsupported commands fall into two 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.

Unsupported Command	Description
autolist	Control the automatic printing of data to the ISPF list data set.
builtin	Process a built-in command, even if overloaded by a macro.
define	Define a name as an alias or macro.
imacro	Save the name of an initial macro in the edit profile.
level	Set the modification level number in PDF library statistics.

model	Copy a model into the buffer or defines a model class.
notes	Control whether the MODEL command display notes or not.
nulls	Control null spaces.
pack	Control whether data is to be stored compressed or not.
profile	Display edit profile.
recovery	Specify edit recovery options.
rmacro	Save a recovery macro in the edit profile.
setundo	Control the UNDO mode.
stats	Generate library statistics.
version	Set the version number in the PDF library statistics.
view	Save as browse command but prompts on save.

The following commands are supported in ISPF emulation mode.

Supported Command	Description
ispf_bottom	Move cursor to the end of the buffer.
ispf_down	Move cursor to next page of text.
ispf_enter	Handle the Enter key or Right Control key in ISPF emulation.
ispf_home	Place the focus on the command line in ISPF emulation.
ispf_retrieve	Does command line retrieval, getting the next command line from the list.
ispf_retrieve_back	Identical to the ispf_retrieve back command.

Menu Editing

ispf_top	Move cursor up to the top of the buffer.
ispf_up	Move cursor up to the previous page of text.
ispf_do_lc	Immediately process all commands found in the line prefix area.

Menu Editing

For information about accessing menus in SlickEdit and associated options, see [Accessing Menus](#).

Creating and Editing Menus

Menus in SlickEdit are easily customized using a Menu Editor that allows you to add, delete, or change menu entries. Modifications made through this UI will be preserved when you upgrade to a newer version of SlickEdit.

Warning

SlickEdit menus are controlled by Slick-C macro files. You can customize menus by editing the corresponding Slick-C files. However any such modifications will be lost when you upgrade to a newer version of SlickEdit. Only modifications made through the **Menu Editor** will be preserved.

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.

To access the **Menu Editor** dialog, click **Macro** → **Menus** from the main menu (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.

Creating a New Menu Resource

Use the Menu Editor to create a new menu resource. From the main menu, click **Macro** → **Menus** (or use the **open_menu** command), then click **New** on the Open Menu dialog. The Menu Editor is displayed. See [Menu Editor Dialog](#) for more information.

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 click **Tools** → **Options** → **Appearance** → **Advanced**, then 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 click **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. This dialog box can be accessed by clicking the **Alias** button on the Menu Editor. 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 attributes to enable or disable predefines which you can specify for any command. When these predefined auto-enabling attributes are not enough, 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 clicking **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](#).

Emulation Tables

CUA Keys

CUA Cursor Movement

Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
Down arrow	Cursor down
Ctrl+Home	Top of buffer
Ctrl+End	Bottom of buffer
Home	Begin line
End	End line
PgUp	Page up
PgDn	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Indent to next tab stop
Shift+Tab	Back indent text to previous tab stop
Ctrl+J	Go to line
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor

CUA Keys

Ctrl+Shift+Alt+Up	Add new cursor above current cursor
-------------------	-------------------------------------

CUA Inserting Text

Ins	Insert/overwrite toggle
Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+Q	Quote next character typed

CUA Deleting Text

Del	Delete char under cursor
Backspace	Delete char before cursor
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Ctrl+Shift+K	Cut word

CUA Searching

Ctrl+F	Find
Ctrl+R	Replace
Ctrl+G	Find next occurrence
Ctrl+Shift+G	Find previous occurrence
Ctrl+I	Incremental search

CUA Keys

Ctrl+Shift+I	Reverse incremental search
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

CUA Selection

Ctrl+A	Select all
Ctrl+B	Select block/column
Ctrl+L	Select line
F8	Select character/stream
Ctrl+U	Deselect
Ctrl+X	Cut selection
Backspace, Del	Delete selection
Tab	Indent selection
Shift+Tab	Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Alt+=	Execute commands in selection
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor

CUA Keys

Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

CUA Clipboard

Ctrl+C, Ctrl+Ins	Copy selection to clipboard
Ctrl+Shift+C	Append selection to clipboard
Ctrl+K	Copy word to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Shift+Ins	Paste
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Ctrl+Shift+K	Cut word
Ctrl+X, Shift+Del	Cut selection
Ctrl+Shift+X	Append cut selection

CUA Command Line and Text Box Editing

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Ctrl+Shift+C	Append selection to clipboard

CUA Keys

Ctrl+Shift+X	Append cut selection
Ctrl+Shift+V	List clipboards and optionally paste one
Esc	Cursor to command line toggle
Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+U	Upcase word
Ctrl+Shift+L	Lowcase word
Ctrl+Shift+K	Cut word
Ctrl+E	Cut to end of line
Ctrl+Backspace	Cut line
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character
Ctrl+K	Copy word to clipboard
Ctrl+X	Retrieve previous command
Ctrl+C	Retrieve next command
Ctrl+V	Start or extend char/stream selection
Ctrl+Shift+C	Start char/stream selection
Ctrl+Shift+X	Append cut selection
Ctrl+Shift+V	List clipboards and optionally paste one
Esc	Cursor to command line toggle

CUA Files and Buffers

F2, Ctrl+S	Save current buffer
Ctrl+N	Next buffer
Ctrl+P	Previous buffer
F7, Ctrl+O	Edit a file or find buffer
Ctrl+Shift+B	List buffers
F6	File compare

CUA Windowing

Ctrl+H	Split window horizontally
Ctrl+Tab, Ctrl+F6	Next window
Ctrl+Shift+Tab, Ctrl+Shift+F6	Previous window
Ctrl+Shift+Z	Zoom window toggle
Ctrl+F4	Close window
Alt+F2	Move window edge
Alt+F3	Create window edge
Ctrl+F7	Move
Ctrl+F8	Size
Ctrl+F9	Minimize
Ctrl+F10	Maximize

CUA Compiling and Programming Support

Alt+Dot	(Pro only) List symbols

CUA Keys

Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only)
Ctrl+Space	(Pro only) Complete symbol
Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+/ Ctrl+Space	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
Ctrl+G	Find next reference
Ctrl+Shift+G	Find previous reference
Ctrl+M	(Pro only) Build project
Ctrl+F5	(Pro only) Execute project
F4, Ctrl+Shift+Down	Next error
Shift+F4, Ctrl+Shift+Up	Previous error
Ctrl+Shift+S	Set next error
Ctrl+Shift+E	List errors
Shift+F10	(Pro only) Compile current buffer
Alt+1	Cursor to error/include file
F12	Make and load current macro buffer
Ctrl+Shift+M	(Pro only) Start concurrent process
Ctrl+Shift+P	Expand extension specific alias

CUA Keys

Ctrl+Shift+O	Expand global alias
Alt+F7	Project properties

CUA Debugging (Pro only)

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+PadStar	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

CUA Macros

Ctrl+F11	Start/end macro recording
Ctrl+F12	Terminate recording & run last recorded macro

CUA Keys

Ctrl+Shift+F12 <key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
F12	Make and load current macro buffer
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event() .
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and run dialog box/macro.

CUA Miscellaneous

F1	Help for mode or context
Ctrl+F1	Help on word at cursor
Alt+F4	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Esc	Cancel or command line toggle
Ctrl+Z, Alt+Backspace	Undo
Shift+F9	Undo with cursor motion grouping
Ctrl+Y	Redo
Ctrl+Shift+H	Hex display toggle
Ctrl+Shift+U	Upcase word
Ctrl+Shift+L	Lowcase word
Ctrl+Shift+J	Go to bookmark

BBEdit Keys

Ctrl+0..Ctrl+9	Set bookmark 0..9
Ctrl+Shift+N	Activate Bookmarks tool window
Ctrl+]	Match parenthesis
Ctrl+\	Expand or collapse selective display
Ctrl+Shift+O	Expand alias at cursor
Ctrl+D	Change directory
Alt+F5	Restore MDI window
Alt+F10	Maximize MDI window

BBEdit Keys

BBEdit Cursor Movement

Shift+Tab	Back indent text to previous tab stop
Command+Left arrow	Begin line
Command+Down arrow	Bottom of buffer
Down arrow	Cursor down
Ctrl+PgDn	Bottom of window
Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
End	End line
Command+J	Go to line
Tab	Indent to next tab stop
Alt+Right	Next word

BBEdit Keys

PgDn	Page down
PgUp	Page up
Alt+Left	Previous word
Command+Down arrow	Top of buffer
Ctrl+PgUp	Top of window
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

BBEdit Inserting Text

Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character typed

BBEdit Deleting Text

Ctrl+Backspace	Cut line
Ctrl+Shift+K	Cut word
Backspace	Delete char before cursor
Del	Delete char under cursor

BBEdit Selection

Ctrl+Shift+Right-Click	Copy selection to cursor
Command+X	Cut selection
Backspace, Del	Delete selection
Command+Shift+A	Deselect
Command+=	Execute commands in selection
Shift+Click	Extend selection
Tab	Indent selection
Ctrl+Right-Click	Move selection to cursor
Command+A	Select all
F8	Select character/stream
Command+L	Select line
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words
Command+Shift+L	Select paragraph
Double-Click	Select word
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Right-Click & Drag	Start block/column selection
Click & Drag	Start char/stream selection C

BBEdit Keys

Shift+<Cursor keys> selection	Start or extend char/stream
Shift+Tab	Unindent selection

BBEdit Searching

Command+F	Find
Alt+Command+F	Find in files
Command+G	Find in next occurrence
Command+Shift+G	Find previous occurrence
Ctrl+R	Replace

BBEdit Command Line and Text Box Editing

Ctrl+Shift+X	Append cut selection
Ctrl+Shift+C	Append selection to clipboard
Space	Complete argument
F3, Command+C	Copy selection to clipboard
Ctrl+K	Copy word to clipboard
Esc	Cursor to command line toggle
Ctrl+Backspace	Cut line
Command+X	Cut selection
Ctrl+Shift+K	Cut word
Shift+Click	Extend selection
Ins	Insert/overwrite toggle
?	List arguments

BBEdit Keys

Ctrl+Shift+V	List clipboards, optionally paste one
Ctrl+Shift+L	Lowcase word
Alt+Right	Next word
Command+V	Paste
Command+Shift+V	Paste previous clipboard
Alt+Left	Previous word
Ctrl+Q	Quote next character
Down arrow	Retrieve next command
Up arrow	Retrieve previous command
Triple-Click	Select line
Double-Click	Select word
Command+E	Set search string
Click & Drag	Start char/stream selection
Shift+<Cursor keys> selection	Start or extend char/stream selection
Ctrl+Shift+U	Upcase word

BBEdit Files and Buffers

F7, Command+O	Edit a file or find buffer
F6	File compare
Ctrl+Shift+B	List buffers
Alt+Command+Right arrow	Next buffer
Command+N	New file
Alt+Command+Shift+N	New file from clipboard

BBEdit Keys

Command+Shift+N	New file from selection
Ctrl+Tab	Open associated file
Alt+Command+Left arrow	Previous buffer
F4	Save and quit current buffer
F2, Command+S	Save current buffer

BBEdit Clipboard

Ctrl+Shift+X	Append cut selection
Ctrl+Shift+C	Append selection to clipboard
Command+C	Copy selection to clipboard
Ctrl+K	Copy word to clipboard
Ctrl+Backspace	Cut line
Command+X, Shift+Del	Cut selection
Ctrl+Shift+K	Cut word
Ctrl+Shift+V	List clipboards, optionally paste one
Command+V	Paste

BBEdit Windowing

Command+W, Ctrl+F4	Close window
Alt+Command+W	Close All Windows
Command+F10	Maximize
Command+M	Minimize
Command+F2	Move window edge

BBEdit Keys

Ctrl+F6	Next window
Ctrl+Shift+F6	Previous window
Command+F8	Size
Command+Shift+®	Split window horizontally
Command+/-	Zoom window toggle

BBEdit Compiling and Programming Support

Ctrl+M	(Pro only) Build project
Shift+F10	(Pro only) Compile current buffer
Ctrl+Dot, F5, Alt+Esc	(Pro only) Complete symbol
Command+D	Cursor to error/include file
Ctrl+F5	(Pro only) Execute project
Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias
Command+G	Find next reference
Command+Shift+G	Find previous reference
Ctrl+Shift+E	List errors
Command+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
F12	Make and load current macro buffer
Alt+Command+Down arrow	Next error
Alt+Comma	(Pro only) Parameter Info

BBEdit Keys

Ctrl+Alt+Comma	(Pro only) Symbol Information Help
Command+Comma	Pop a pushed bookmark
Alt+Command+Up arrow	Previous error
Command+Dot	Push a bookmark, go to definition of symbol at cursor
Shift+Command+ /	Push a bookmark, go to first reference to symbol at cursor
Ctrl+Shift+S	Set next error
Ctrl+Shift+M	(Pro only) Start concurrent process

BBEdit Debugging (Pro only)

Alt+Command+B	Activate breakpoints window
Command+7, Ctrl+Alt+C	Activate call stack
Ctrl+Command+H	Activate threads window
Command+4	Activate variables window
Command+3, Ctrl+Alt+W	Activate watch window
Ctrl+Shift+F9	Clear all breakpoints
Ctrl+Shift+F5	Restart debugging
Ctrl+F10	Run to cursor
Command+PadStar	Show next statement
F5	Start/continue debugging
Command+Shift+I	Step into
Command+Shift+T	Step out
Command+Shift+O	Step over
Shift+F5	Stop debugging

BBEdit Keys

Alt+Command+ /	Toggle breakpoint enable
Command+ /	Toggle breakpoint

BBEdit Macros

Ctrl+Shift+Space	Edit current dialog box if running a dialog box. Close dialog box that won't close. Load and run dialog box/macro if editing dialog box or macro.
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Terminate infinite loops.
F12	Make and load current macro buffer
Ctrl+F11	Start/end macro recording
Ctrl+F12	Terminate recording, run last recorded macro

BBEdit Miscellaneous

Esc	Cancel or command line toggle
Ctrl+Shift+Space	Complete more
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+O	Expand alias at cursor
Command+D	Go to bookmark
F1	Help for mode or context
Ctrl+Shift+H	Hex display toggle
Ctrl+Shift+L	Lowcase word

Brief Keys

Ctrl+B	Match parenthesis
Command+F10	Maximize MDI window
Command+F7	Move MDI window
Command+Shift+Z	Redo
Command+F5	Restore MDI window
Command+Q	Safe exit
Command+F1	Help on word at cursor
Ctrl+0..Ctrl+9	Set bookmark 0..9
Alt+F8	Size MDI window
Command+Z, Alt+Backspace	Undo
Shift+F9	Undo with cursor motion grouping
Ctrl+Shift+U	Upcase word

Brief Keys

Brief Cursor Movement

Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
Down arrow	Cursor down
Ctrl+PgUp, Home(3x)	Top of buffer
Ctrl+PgDn, End(3x)	Bottom of buffer
Home	Begin line

Brief Keys

End	End line
Alt+G	Go to line
Alt+J	Go to bookmark
PgUp	Page up
PgDn	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Home, Home(2x)	Top of window
Ctrl+End, End(2x)	Bottom of window
Tab	Insert tab or next tab stop
Shift+Tab	Previous tab stop
Shift+Home	Left side of window
Shift+End	Right side of window
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

Brief Inserting Text

Alt+I	Insert/overwrite toggle
Enter	Maybe split insert line
Ctrl+Enter	No split insert line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)

Brief Keys

Shift+Space	Insert a space (no syntax expansion)
Alt+Q	Quote next character typed
Ctrl+A	Insert buffer name

Brief Deleting Text

Del	Delete char or selection
Backspace	Delete char before cursor
PadMinus	Cut line or selection
Alt+K	Delete to end line
Ctrl+K	Delete word
Alt+D	Delete line
Ctrl+Backspace	Delete previous word

Brief Searching

Alt+S, F5	Search forward
Shift+F5	Search again
Alt+F5, Ctrl+Shift+F5	Search backward
Ctrl+F5	Case sensitivity toggle
Ctrl+F6	Regular expression toggle
Alt+T, F6	Translate forward
Shift+F6	Translate again
Alt+F6, Ctrl+Shift+F6	Translate backward
Ctrl+S	Forward incremental search

Brief Keys

<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.
-----------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Brief Clipboard

Ctrl+E	Copy word to clipboard
Ctrl+L	Paste recent clipboard
Ins, Ctrl+Y, Shift+Ins	Paste
PadPlus, Ctrl+Ins	Copy selection to clipboard
PadMinus, Shift+Del	Cut line or selection

Brief Command Line and Text Box Editing

The following keys are different in all Text Boxes except the command line if the CUA Text Box check box is enabled (**Tools > Options > Redefine Common Keys**):

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Alt+A..Alt+Z	Taken over by dialog manager for selecting controls

Brief Command Line Keys

F10, Esc	Cursor to command line toggle
Space	Complete argument
?	List arguments
Alt+I	Insert/overwrite toggle
Alt+Q	Quote next character

Brief Keys

Ctrl+E	Copy word to clipboard
Ins, Ctrl+Y	Paste
Ctrl+L	Paste recent clipboard
Ctrl+A	Insert buffer name
Ctrl+Space	Expand alias at cursor. Use alias command to define aliases.
Ctrl+Right	Next word
Ctrl+Left	Previous word
Ctrl+F1	Upcase word
Ctrl+F2	Lowcase word
Ctrl+F7	Capitalize word
Alt+K	Delete to end line
Ctrl+K	Delete word
Alt+D	Delete line
Ctrl+Backspace	Delete prev word
Up arrow	Retrieve previous command
Down arrow	Retrieve next command
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

Brief Files and Buffers

Brief Keys

Alt+W	Save current buffer
Ctrl+Minus	Quit current buffer
Alt+E	Edit a file or find buffer
Alt+X	Safe exit w/write all option
Alt+O	Change buffer name
Ctrl+F10	File compare
Alt+B	List buffers
Alt+N	Next buffer
Alt+Minus	Previous buffer
Ctrl+X	Save all buffers and exit
Alt+R	Read file

Brief Windowing

F1	Change window
F2	Move window edge
F3	Create window edge
F4	Delete window edge
Ctrl+Z	Zoom window toggle
Ctrl+J	Split window horizontally
Ctrl+W, Ctrl+Tab	Next window
Ctrl+Shift+Tab	Previous window
Shift+Left	Switch to left window
Shift+Right	Switch to right window

Shift+Up	Switch to window above
Shift+Down	Switch to window below

Brief Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only) Symbol Information Help
Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+ /	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
Ctrl+Space	(Pro only) Complete symbol
Alt+F9, Ctrl+Shift+F9	(Pro only) Build project
Alt+F10, Ctrl+Shift+F10	(Pro only) Compile current buffer
Ctrl+N	Next error
F9	Make and load macro
Ctrl+P	List errors
Ctrl+O	Stop concurrent process
Ctrl+I	(Pro only) Start concurrent process
Ctrl+G	List buffer procedures

Brief Keys

Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias

Brief Debugging (Pro only)

Ctrl+F9	Toggle breakpoint enable
F10	Step over
F11	Step into
Ctrl+F10	Run to cursor
Alt+PadStar	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

Brief Selection

Alt+C	Select block/column
Alt+L	Select line
Alt+M	Inclusive char selection
Alt+A	Non-inclusive char selection
Del	Delete selection
PadMinus	Cut selection
Tab	Indent selection

Brief Keys

Shift+Tab	Unindent selection
Shift+F8	Shift selection right
Alt+Y	Go to beginning of selection
Ctrl+F3	Upcase selection
Ctrl+F4	Lowcase selection
Alt+=	Execute commands in selection
Alt+F	Fill selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

Brief Macros

F7	Start/end macro recording
Shift+F7	Pause recording toggle
F9	Make and load macro module
F8	Run last recorded macro

Brief Keys

Alt+F7, Ctrl+Shift+F7	List recorded macros
Alt+F8, Ctrl+Shift+F8	Save last recorded macro
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and run dialog box/macro.

Brief Miscellaneous

Alt	Menu bar
Alt+H	Help for mode or context
Ctrl+F	Configuration menu
Alt+X	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Ctrl+X	Write all buffers and exit
Esc	Cancel
Alt+H	Help for mode or context
Ctrl+F	Configuration menu
Ctrl+Shift+H	Hex display toggle
Ctrl+T	Line to top

CodeWarrior Keys

Ctrl+B	Line to bottom
Alt+U, PadStar	Undo
Ctrl+F9	Undo with cursor motion grouping
Ctrl+U	Redo
Shift+F1	Scroll up
Ctrl+D, Shift+F2	Scroll down
Shift+F3	Scroll left
Shift+F4	Scroll right
Ctrl+F1	Upcase word
Ctrl+F2	Lowcase word
Ctrl+F7	Capitalize word
Ctrl+R	Repeat next key stroke
Alt+Z	Shell to DOS
Alt+V	Version
Alt+P	Print selection
Ctrl+C	Center line
Alt+1, Alt+2, Alt+0	Set bookmark
Alt+Shift+J	Activate Bookmarks tool window
Ctrl+Q	Fundamental mode for next key press
Ctrl+Shift+O	Expand alias at cursor
Alt+/-	Alias change directory

CodeWarrior Keys

CodeWarrior Cursor Movement

Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
Down arrow	Cursor down
Ctrl+Home	Top of buffer
Ctrl+End	Bottom of buffer
Home	Begin line
End	End line
PgUp	Page up
PgDn	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Indent to next tab stop
Shift+Tab	Back indent text to previous tab stop
Ctrl+G	Go to line
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

CodeWarrior Inserting Text

--	--

CodeWarrior Keys

Ins	Insert/overwrite toggle
Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+Q	Quote next character typed

CodeWarrior Deleting Text

Del	Delete char under cursor
Backspace	Delete char before cursor
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Ctrl+Shift+K	Cut word

CodeWarrior Searching

Alt+F4, Ctrl+F	Find
Ctrl+R, Ctrl+=	Replace
F3	Find next occurrence
Shift+F3	Find previous occurrence
Ctrl+I	Incremental search
Ctrl+Shift+I	Reverse incremental search
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background

CodeWarrior Keys

	multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.
--	----------------------------------------------------------------------------------------------------------------

CodeWarrior Selection

Ctrl+A	Select all
Ctrl+L	Select line
F8	Select character/stream
Ctrl+U	Deselect
Ctrl+X	Cut selection
Backspace, Del	Delete selection
Tab	Indent selection
Shift+Tab	Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Alt+=	Execute commands in selection
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection

CodeWarrior Keys

Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

CodeWarrior Clipboard

Ctrl+C, Ctrl+Ins	Copy selection to clipboard
Ctrl+Shift+C	Append selection to clipboard
Ctrl+K	Copy word to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Shift+Ins	Paste
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Ctrl+Shift+K	Cut word
Ctrl+X, Shift+Del	Cut selection
Ctrl+Shift+X	Append cut selection

CodeWarrior Command Line and Text Box Editing

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Ctrl+Shift+C	Append selection to clipboard
Ctrl+Shift+X	Append cut selection
Ctrl+Shift+V	List clipboards and optionally paste one
Esc	Cursor to command line toggle

CodeWarrior Keys

Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+U	Upcase word
Ctrl+Shift+L	Lowcase word
Ctrl+Shift+K	Cut word
Ctrl+E	Cut to end of line
Ctrl+Backspace	Cut line
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character
Ctrl+K	Copy word to clipboard
Up arrow	Retrieve previous command
Down arrow	Retrieve next command
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

CodeWarrior Files and Buffers

F2, Ctrl+S	Save current buffer
Ctrl+N	Next buffer

CodeWarrior Keys

Ctrl+P	Previous buffer
F4	Save and quit current buffer
F7, Ctrl+O	Edit a file or find buffer
Ctrl+Shift+B	List buffers
F6	File compare

CodeWarrior Windowing

Ctrl+H	Split window horizontally
Ctrl+Tab, Ctrl+F6	Next window
Ctrl+Shift+Tab, Ctrl+Shift+F6	Previous window
Ctrl+Shift+Z	Zoom window toggle
Ctrl+F4	Close window
Alt+F2	Move window edge
Ctrl+F7	Move
Ctrl+F8	Size
Ctrl+F9	Minimize
Ctrl+F10	Maximize

CodeWarrior Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Alt+Comma	(Pro only) Parameter Info

CodeWarrior Keys

Ctrl+Alt+Comma	(Pro only) Symbol Information Help
Ctrl+Space	(Pro only) Complete symbol
Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+/-	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
F3	Find next reference
Shift+F3	Find previous reference
Ctrl+M	(Pro only) Build project
Ctrl+F5	(Pro only) Execute project
Ctrl+Shift+Down	Next error
Ctrl+Shift+Up	Previous error
Ctrl+Shift+S	Set next error
Ctrl+Shift+E	List errors
Ctrl+F7	(Pro only) Compile current buffer
Alt+1, Ctrl+D	Cursor to error/include file
F12	Make and load current macro buffer
Ctrl+Shift+M	(Pro only) Start concurrent process
Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias

CodeWarrior Debugging (Pro only)

CodeWarrior Keys

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+PadStar	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

CodeWarrior Macros

Ctrl+F11	Start/end macro recording
Ctrl+F12	Terminate recording & run last recorded macro
F12	Make and load current macro buffer
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite

CodeWarrior Keys

	loops.
Ctrl+Shift+Space	If editing dialog box or macro, load and run dialog box/macro.

CodeWarrior Miscellaneous

F1	Help for mode or context
F10	Menu bar
Ctrl+F1	Help on word at cursor
Alt+F4	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Esc	Cancel or command line toggle
Ctrl+Z, Alt+Backspace	Undo
Shift+F9	Undo with cursor motion grouping
Ctrl+Y	Redo
Ctrl+Shift+H	Hex display toggle
Ctrl+Shift+U	Upcase word
Ctrl+Shift+L	Lowcase word
Ctrl+Shift+J	Go to bookmark
Ctrl+0..Ctrl+9	Set bookmark 0..9
Ctrl+Shift+T	Activate Bookmarks tool window
Ctrl+B	Match parenthesis
Ctrl+Shift+O	Expand alias at cursor

Alt+F5	Restore MDI window
Alt+F10	Maximize MDI window

CodeWright Keys

CodeWright Cursor Movement

Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
Down arrow	Cursor down
Ctrl+Home	Top of buffer
Ctrl+End	Bottom of buffer
Home	Begin line
End	End line
PgUp	Page up
PgDn	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Page left
Ctrl+PgDn	Page right
Tab	Indent to next tab stop
Shift+Tab	Back indent text to previous tab stop
Ctrl+J	Go to line
Ctrl+	Add multiple cursors

CodeWright Keys

Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

CodeWright Inserting Text

Ins	Insert/overwrite toggle
Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+Q	Quote next character typed

CodeWright Deleting Text

Del	Delete char under cursor
Backspace	Delete char before cursor
Ctrl+Backspace	Delete previous word
Ctrl+Del	Delete to end of line
Shift+Backspace	Delete word

CodeWright Searching

Ctrl+F	Find
Ctrl+H	Replace
Ctrl+Shift+S, Ctrl+G	Find next occurrence

Ctrl+Shift+Q	Quick search
Ctrl+I	Incremental search
Ctrl+Shift+R	Repeat last replace
Ctrl+Shift+I	Reverse incremental search
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

CodeWright Selection

Ctrl+A	Select all
Ctrl+B	Select block/column
Ctrl+L	Select line
Ctrl+I	Inclusive character
Ctrl+M	Non-inclusive character selection
Ctrl+W	Save selection
Ctrl+X	Cut selection
Backspace, Del	Delete selection
Tab	Indent selection
Shift+Tab	Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Alt+=	Execute commands in selection
Shift+Cursor keys	Start or extend char/stream selection
Click & Drag	Start char/stream selection

CodeWright Keys

Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

CodeWright Clipboard

Ctrl+C, Ctrl+Ins	Copy selection to clipboard
Ctrl+Shift+C	Append selection to clipboard
Ctrl+K	Copy word to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Shift+Ins	Paste
Ctrl+X, Shift+Del	Cut selection

CodeWright Command Line and Text Box Editing

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Ctrl+Shift+C	Append selection to clipboard

CodeWright Keys

Ctrl+Shift+V	List clipboards and optionally paste one
F9	Toggle command line
Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+U	Upcase word
Ctrl+Shift+D	Lowcase word
Ctrl+Del	Delete to end of line
Shift+Backspace	Delete word
Ctrl+Backspace	Delete previous word
Ctrl+D	Delete line
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character
Ctrl+K	Copy word to clipboard
Up arrow	Retrieve previous command
Down arrow	Retrieve next command
Shift+Cursor keys	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

CodeWright Files and Buffers

F2, Ctrl+S	Save current buffer
Ctrl+Shift+N	Next buffer
Ctrl+Shift+P	Previous buffer
Ctrl+O	Edit a file or find buffer
F6	File compare
Ctrl+N	Open new file or create new project
Ctrl+Shift+F	Insert file at cursor

CodeWright Windowing

Ctrl+Tab, Ctrl+F6	Next window
Ctrl+Shift+Tab, Ctrl+Shift+F6	Previous window
Ctrl+Shift+Z, Alt+F2	Zoom window toggle
Ctrl+F4	Close window
Alt+F3	Create window edge
Ctrl+F7	Move
Ctrl+F8	Size

CodeWright Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Alt+Comma, Alt+F1	(Pro only) Parameter Info
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.

CodeWright Keys

Ctrl+Space	(Pro only) Complete symbol
Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+/-	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
Ctrl+G, Ctrl+Shift+S	Find next reference
Ctrl+F9, Ctrl+M	(Pro only) Build project
Ctrl+F10, Shift+F10	(Pro only) Compile current buffer
Ctrl+F5	(Pro only) Execute project
Ctrl+Shift+Down	Next error
Ctrl+Shift+Up	Previous error
Ctrl+Shift+S	Set next error
Ctrl+Shift+E	List errors
Alt+1	Cursor to error/include file
F12	Make and load current macro buffer
Ctrl+Shift+M	(Pro only) Start concurrent process

CodeWright Debugging (Pro only)

F5	Start/continue debugging
Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
Ctrl+Shift+F9	Clear all breakpoints

CodeWright Keys

F10	Step over
F11	Step into
Alt+PadStar	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

CodeWright Macros

F7, Ctrl+F11	Start/end macro recording
F8, Ctrl+F12	Terminate recording & run last recorded macro
F12	Make and load current macro buffer
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and run dialog box/macro.

CodeWright Miscellaneous

F1	Help for mode or context
Ctrl+F1	Help on word at cursor

CodeWright Keys

Alt+Shift+Left	Back (like web browser)
Alt+Shift+Right	Forward (like web browser)
Alt+F4	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Esc	Cancel
Ctrl+T	Line to top
F9	Line toggle
Ctrl+Shift+G	Show functions headings
Ctrl+U, Ctrl+Z	Undo
Shift+F9	Undo with cursor motion grouping
Ctrl+Y, Alt+Ins	Redo
Ctrl+Shift+H	Hex display toggle
Ctrl+Shift+U	Upcase word
Ctrl+Shift+D	Lowcase word
Ctrl+Shift+J	Set bookmark
Ctrl+0..Ctrl+9	Set bookmark 0..9
Ctrl+Shift+B	Activate Bookmarks tool window
Ctrl+]	Find matching parenthesis
Ctrl+\	Expand or collapse selective display
Ctrl+Shift+K	Show matching parenthesis
Alt+F5	Restore MDI window

Eclipse Keys

Alt+F10	Maximize MDI window
---------	---------------------

Eclipse Keys

Eclipse Cursor Movement

Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
Down arrow	Cursor down
Ctrl+Home	Top of buffer
Ctrl+End	Bottom of buffer
Home	Begin line
End	End line
PgUp	Page up
PgDn	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Indent to next tab stop
Shift+Tab	Back indent text to previous tab stop
Ctrl+L	Go to line
Ctrl+	Add multiple cursors

Eclipse Keys

Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

Eclipse Inserting Text

Ins	Insert/overwrite toggle
Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)

Eclipse Deleting Text

Del	Delete char under cursor
Backspace	Delete char before cursor
Ctrl+D	Delete line
Ctrl+Del	Delete word
Ctrl+Backspace	Delete previous word
Ctrl+Shift+Del	Delete to end of line

Eclipse Searching

Ctrl+F	Find
Ctrl+R	Replace
Ctrl+K	Find next occurrence
Ctrl+Shift+K	Find previous occurrence

Eclipse Keys

Ctrl+J	Incremental search
Ctrl+Shift+J	Reverse incremental search
Ctrl+Shift+J	Reverse incremental search
Ctrl+Shift+U	Find all occurrences of word at cursor excluding comments and strings
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

Eclipse Selection

Ctrl+A	Select all
Ctrl+U	Deselect
Ctrl+X	Cut selection
Backspace, Del	Delete selection
Tab	Indent selection
Shift+Tab	Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Alt+=	Execute commands in selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor

Eclipse Keys

Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

Eclipse Clipboard

Ctrl+C, Ctrl+Ins	Copy selection to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Shift+Ins	Paste
Ctrl+X, Shift+Del	Cut selection

Eclipse Command Line and Text Box Editing

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V, Shift+Ins	Paste
Ctrl+Shift+V	List clipboards and optionally paste one
Esc	Cursor to command line toggle
Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+X	Upcase word or selection

Eclipse Keys

Ctrl+Shift+Y	Lowcase word or selection
Ctrl+D	Delete line
Ctrl+Del	Delete word
Ctrl+Shift+Del	Delete to end of line
Ctrl+Backspace	Delete previous word
Ins	Insert/overwrite toggle
Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V, Shift+Ins	Paste
Esc	Cursor to command line toggle

Eclipse Files and Buffers

Ctrl+S	Save current buffer
Ctrl+PgDn	Next buffer
Ctrl+PgUp	Previous buffer
Ctrl+N	New project
Ctrl+Shift+E, Ctrl+F6	List buffers
Ctrl+Shift+F4	Close all

Eclipse Windowing

Ctrl+Shift+Tab	Previous window
Ctrl+Shift+Z	Zoom window toggle
Ctrl+W, Ctrl+F4	Close window

Eclipse Keys

Alt+F2	Move window edge
Alt+F3	Create window edge
Ctrl+F7	Next view

Eclipse Compiling and Programming Support

Alt+Dot, F2	(Pro only) List symbols
Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only) Symbol Information Help
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Ctrl+Space	(Pro only) Complete symbol
Ctrl+Dot, F3	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+Shift+G	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
Ctrl+K	Find next reference
Ctrl+Shift+K	Find previous reference
Ctrl+B	Complete symbolBuild all
Ctrl+F5, Ctrl+F11	(Pro only) Execute project
Ctrl+Shift+D	Open javadoc editor
Ctrl+Shift+F	Beautify selection
Ctrl+Shift+M	Add import

Eclipse Keys

Ctrl+Shift+O	Organize imports
Shift+F10	(Pro only) Compile current buffer
Ctrl+H	Activate find symbol tool window
Ctrl+I	Reindent current line
Ctrl+=	Display diff dialog
Alt+1	Cursor to error/include file
F12	Make and load current macro buffer
Ctrl+Shift+I	Expand extension specific alias
Ctrl+O	Activate Outline tool window
F4	Activate Class tool window
Ctrl+`	Edit associated file. Edit .h file if editing .cpp file or .cpp file if editing .h file.

Eclipse Debugging (Pro only)

F5	Debug step into
F6	Debug step over
F7	Debug step out
F8	Debug continue out
F10	Step over
F11	Start/continue debugging
Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
F9	Toggle break point
Ctrl+F9	Toggle breakpoint enable

Eclipse Keys

Ctrl+Shift+F9	Clear all breakpoints
Ctrl+R	Run to cursor
Alt+PadStar	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

Eclipse Macros

Alt+F11	Start/end macro recording
Alt+F12	Terminate recording & run last recorded macro
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
F12	Make and load current macro buffer
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event() .
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and run dialog box/macro.

Eclipse Miscellaneous

F1	Help for mode or context
Ctrl+F1	Help for word at cursor

GNU Emacs Keys

Alt+F4	Safe exit
Ctrl+Shift+Q	Toggle modified line display
Ctrl+Shift+R	Open file
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Esc	Cancel or command line toggle
Ctrl+Z, Alt+Backspace	Undo
Shift+F9	Undo with cursor motion grouping
Ctrl+Y	Redo
Ctrl+Shift+X	Uppercase word or selection
Ctrl+Shift+Y	Lowercase word or selection
Ctrl+Shift+N	Activate bookmarks tool window
Ctrl+0..Ctrl+9	Set bookmark 0..9
Ctrl+]	Match parenthesis
Ctrl+\	Expand or collapse selective display
Alt+F5	Restore MDI window
Alt+F10	Maximize MDI window

GNU Emacs Keys

GNU Emacs Cursor Movement

Alt+G	Go to line
Ctrl+A	Begin line

Ctrl+E	End of line
Left arrow, Ctrl+B	Cursor left
Right arrow, Ctrl+F	Cursor right
Ctrl+N, Down arrow	Next line
Ctrl+P, Up arrow	Previous line
Ctrl+V, PgDn	Page down
Alt+V, PgUp	Page up
Home, Alt+<	Top of buffer
End, Alt+>	Bottom of buffer
Alt+Left, Ctrl+Left, Alt+B	Previous word
Alt+Right, Ctrl+Right, Alt+F	Next word
Ctrl+Alt+B	Previous level
Ctrl+Alt+F	Next level
Alt+)	Match parenthesis
Alt+[, Alt+Up	Previous paragraph
Alt+], Alt+Down	Next paragraph
Alt+M	First non blank
Alt+A, Ctrl+Up	Previous sentence
Alt+E, Ctrl+Down	Next sentence
Ctrl+X, 'G'	Go to line
Alt+Home	Top of file in next window
Alt+End	End of file in next window
Alt+PgDn	Page down next window

GNU Emacs Keys

Alt+PgUp	Page up next window
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

GNU Emacs Inserting Text

Ins	Insert/overwrite toggle
Tab, Ctrl+I	Indent to previous line or insert tab character
Enter, Ctrl+M, Ctrl+J	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+O	Split line
Ctrl+Q	Insert literal

GNU Emacs Deleting Text

Alt+\	Delete horizontal space
Ctrl+X Ctrl+O	Delete blank lines
Alt+Del	Cut previous word
Ctrl+Alt+K	Cut level
Ctrl+Backspace	Cut previous word
Ctrl+K	Cut to end of line

Alt+D	Cut word
Alt+K	Cut sentence
Ctrl+Alt+H	Cut previous word
Del, Ctrl+D	Delete character under cursor
Backspace, Ctrl+H	Delete character before cursor

GNU Emacs Searching

Ctrl+S	Incremental search
Ctrl+R	Reverse incremental search
Ctrl+Alt+S	Regular expression search
Ctrl+Alt+R	Reverse regular expression search
Alt+%	Search and replace with prompting
Alt+&	Search and replace without prompting
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

GNU Emacs Selection

Ctrl+Shift+2	Select character/stream
Alt+@	Select next word
Alt+H	Select paragraph
Ctrl+X Ctrl+X	Exchange point and start selection
Ctrl+X 'W'	Write selection to file
Ctrl+Alt+\	Indent selection

GNU Emacs Keys

Ctrl+X Ctrl+Alt+I	Tabify selection
Ctrl+X Alt+I	Untabify selection
Shift+F7	Shift block selection left
Shift+F8	Shift block selection right
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

GNU Emacs Clipboard

Ctrl+W, Shift+Del	Cut selection
Ctrl+Y, Shift+Ins	Paste
Alt+Y	Select clipboard to insert
Alt+W, Ctrl+Ins	Copy selection to clipboard
Ctrl+X Ctrl+A	Copy word
Ctrl+K	Cut to end of line

Alt+D	Cut word
Alt+K	Cut sentence
Alt+Del	Delete previous word
Ctrl+Alt+K	Cut level
Ctrl+Alt+H	Cut previous word
Ctrl+Alt+W	Append next clipboard

GNU Emacs Files and Buffers

Ctrl+X Ctrl+S	Save current buffer
Ctrl+X 'K'	Quit buffer
Ctrl+X 'B'	Select buffer
Ctrl+X Ctrl+F	Edit a file or find buffer
Ctrl+X Ctrl+W	Save and rename buffer
Ctrl+F7	Write buffer to file
Ctrl+F2, Ctrl+X 'C'	Compare windows
Ctrl+X Ctrl+B, Ctrl+X Alt+B	List buffers
Ctrl+X Ctrl+V	Replace buffer with a file on disk
Ctrl+X 'I'	Insert file
Ctrl+X 'W'	Write selection to file
Ctrl+X 'D'	Directory edit mode

GNU Emacs Windowing

Ctrl+X '2'	Split window horizontally

Ctrl+X '1'	One window
Ctrl+X 'O'	Next window
Ctrl+X 'P'	Prev window
Ctrl+PgDn	Shrink window
Ctrl+PgUp	Expand window

GNU Emacs Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only) Symbol Information Help
Ctrl+Space	(Pro only) Complete symbol
Ctrl+X 'M'	(Pro only) Build project
Ctrl+F5	(Pro only) Execute project
Ctrl+X Ctrl+N, Alt+F10	Next error
Alt+F6	(Pro only) Compile current buffer
Ctrl+C C	Stop concurrent process
Ctrl+X Ctrl+M	(Pro only) Start concurrent process
Ctrl+X Comma, Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+X Ctrl+H, Ctrl+Comma	Pop a pushed bookmark

Ctrl+X Dot	Go to definition
Ctrl+X Alt+Dot	Make tag file
Ctrl+X Alt+Comma	Select tag file
F3	Make and load current macro buffer
Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias

GNU Emacs Debugging (Pro only)

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+PadStar	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

GNU Emacs Macros

Ctrl+X 'E'	Run last recorded macro
F3	Make and load current buffer
Ctrl+X Alt+N	Save last recorded macro
Ctrl+X '('	Start/end macro recording
Ctrl+X ')'	End macro recording
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that will not close. If editing dialog box or macro, load and run the dialog box/macro.

GNU Emacs Command Line and Text Box Editing

The following keys are different in all Text Boxes except the command line if the CUA Text Box check box is enabled (**Tools > Options > Redefine Common Keys**):

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Alt+A..Alt+Z	Taken over by dialog manager for selecting controls

GNU Emacs Command Line Keys

F2, Alt+X	Cursor to command line toggle
Ctrl+D	Delete character

Backspace	Delete previous character
Tab, Ctrl+I	Insert tab character
Ctrl+J, Enter Ctrl+M	ENTER argument
Space	Complete argument
?	List arguments
Ins	Insert toggle
Alt+\	Delete horizontal space
Ctrl+Q	Quote next character
Ctrl+Space	Expand alias at cursor. Use alias command to define aliases.
Ctrl+K	Cut to end of line
Ctrl+Alt+H	Cut previous word
Alt+D	Cut word
Ctrl+W, Shift+Del	Cut selection
Ctrl+Y, Shift+Ins	Paste
Alt+Y	Paste recent clipboard
Ctrl+Alt+W	Append next clipboard
Alt+C	Capitalize word
Alt+L	Lowcase word
Alt+U	Upcase word
Ctrl+T	Transpose characters
Alt+T	Transpose words
Ctrl+N, Down	Retrieve previous argument
Ctrl+P, Up	Retrieve next argument

ESC, Ctrl+[Complete and enter argument
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

GNU Emacs Miscellaneous

F1, Alt+?, Ctrl+-	Help for mode or context
Ctrl+H	GNU Emacs Help
Alt+P	Configuration menu
F10	Main menu for mode
Ctrl+X Ctrl+C	Safe exit
Ctrl+Shift+Comma	Select from cursor to beginning of buffer
Ctrl+Shift+Dot	Select from cursor to end of buffer
Ctrl+Shift+Space	Complete more
Ctrl+G	Cancel
Ctrl+/, F9, Ctrl+X 'U'	Undo
Shift+F9	Redo
Ctrl+F9	Undo w/cursor grouping
Ctrl+Shift+H	Hex display toggle
Ctrl+T	Transpose characters
Alt+T	Transpose words

GNU Emacs Keys

Ctrl+X Ctrl+T	Transpose lines
Alt+U	Upcase word
Alt+L	Lowcase selection
Alt+C	Capitalize word
Ctrl+X 'F'	Set margins
Alt+Q	Reflow paragraph
Alt+S, Pad5	Center line within margins
Ctrl+Shift+F2	Activate Bookmarks tool window
Alt+)	Find matching start paren
Ctrl+L	Center line within window
Ctrl+Shift+O	Expand alias at cursor
Ctrl+X /	Alias change directory
F8, Ctrl+F8	Set macro variable
F4	Bind to key
F6	What is key
F7	Change directory
Ctrl+X Ctrl+E	Shell
Ctrl+X Ctrl+Z	Minimize editor
Ctrl+X '='	Display information on cursor position
Ctrl+X 'L'	Count lines
Alt+~	Modify toggle
Alt+Z	Zap to char
Ctrl+Z	Iconize MDI window

GNU Emacs Argument/Repeating a Key

Ctrl+U	Select number of times to invoke a command. Sets argument-count.
Ctrl+X Ctrl+O	Change number of blank lines at or before cursor to argument-count
Ctrl+X Ctrl+I, Ctrl+X Tab	Indent or unindent selected lines by argument-count characters
Alt+0..Alt+9	Select argument-count 0..9
Ctrl+0..Ctrl+9	Select argument-count 0..9

Epsilon Keys

Epsilon Cursor Movement

Ctrl+A, Alt+Left	Begin line
Ctrl+E, Alt+Right	End of line
Left arrow, Ctrl+B	Cursor left
Right arrow, Ctrl+F	Cursor right
Ctrl+N, Down arrow	Next line
Ctrl+P, Up arrow	Previous line
Ctrl+V, PgDn	Page down
Alt+V, PgUp	Page up
Ctrl+Home, Alt+<	Top of buffer
Ctrl+End, Alt+>	Bottom of buffer
Ctrl+Left, Alt+B	Previous word
Ctrl+Right, Alt+F	Next word
Home	Beginning of window

Epsilon Keys

End	End of window
Ctrl+Alt+B	Previous level
Ctrl+Alt+F	Next level
Alt+)	Match parenthesis
Alt+[, Alt+Up	Previous paragraph
Alt+], Alt+Down	Next paragraph
Alt+M	First non blank
Alt+A, Ctrl+Up	Previous sentence
Alt+E, Ctrl+Down	Next sentence
Ctrl+X, 'G'	Go to line
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

Epsilon Inserting Text

Ins	Insert/overwrite toggle
Tab, Ctrl+I	Indent to previous line or insert tab character
Enter, Ctrl+M, Ctrl+J	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+O	Split line

Epsilon Keys

Ctrl+Q	Insert literal
--------	----------------

Epsilon Deleting Text

Alt+\	Delete horizontal space
Ctrl+X Ctrl+O	Delete blank lines
Alt+Del	Cut previous level
Ctrl+Alt+K	Cut level
Ctrl+Backspace	Cut line
Ctrl+K	Cut to end of line
Alt+D	Cut word
Alt+K	Cut sentence
Ctrl+Alt+H	Cut previous word
Del, Ctrl+D	Delete character under cursor
Backspace, Ctrl+H	Delete character before cursor

Epsilon Searching

Ctrl+S	Incremental search
Ctrl+R	Reverse incremental search
Ctrl+Alt+S	Regular expression search
Ctrl+Alt+R	Reverse regular expression search
Alt+%	Search and replace with prompting
Alt+&	Search and replace without prompting
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background

	multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.
--	----------------------------------------------------------------------------------------------------------------

Epsilon Selection

Ctrl+2	Select character/stream
Alt+@	Select block/column
Alt+H	Select paragraph
Ctrl+X Ctrl+X	Exchange point and start selection
Ctrl+X 'W'	Write selection to file
Ctrl+Alt+\	Indent selection
Ctrl+X Ctrl+Alt+I	Tabify selection
Ctrl+X Alt+I	Untabify selection
Shift+F7	Shift block selection left
Shift+F8	Shift block selection right
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/

Epsilon Keys

	column selection.
Ctrl+Double-Click	Add selection of words

Epsilon Clipboard

Ctrl+W, Shift+Del	Cut selection
Ctrl+Y, Shift+Ins	Paste
Alt+Y	Select clipboard to insert
Alt+W, Ctrl+Ins	Copy selection to clipboard
Ctrl+X Ctrl+A	Copy word
Ctrl+K	Cut to end of line
Alt+D	Cut word
Alt+K	Cut sentence
Alt+Del	Cut previous level
Ctrl+Alt+K	Cut level
Ctrl+Alt+H	Cut previous word
Ctrl+Alt+W	Append next clipboard

Epsilon Files and Buffers

Ctrl+X Ctrl+S	Save current buffer
Ctrl+X 'K'	Quit buffer
Ctrl+X 'B'	Select buffer
Ctrl+X Ctrl+F	Edit a file or find buffer
Ctrl+X Ctrl+W	Save and rename buffer

Epsilon Keys

Ctrl+F7	Write buffer to file
Ctrl+F2, Ctrl+X 'C'	Compare windows
Ctrl+X Ctrl+B, Ctrl+X Alt+B	List buffers
Ctrl+X Ctrl+V	Replace buffer with a file on disk
Ctrl+X 'I'	Insert file
Ctrl+X 'W'	Write selection to file
Ctrl+X 'D'	Directory edit mode

Epsilon Windowing

Ctrl+X '2'	Split window horizontally
Ctrl+X '1'	One window
Ctrl+X 'O', Alt+PgDn	Next window
Ctrl+X 'P', Alt+PgUp	Prev window
Ctrl+X Ctrl+D	Delete window
Home	Cursor to beginning of window
End	Cursor to end of window
Ctrl+PgDn	Shrink window
Ctrl+PgUp	Expand window

Epsilon Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.

Epsilon Keys

Alt+Comma	(Pro only) Info
Ctrl+Space	(Pro only) Complete symbol
Ctrl+X 'M'	(Pro only) Build project
Ctrl+F5	(Pro only) Execute project
Ctrl+X Ctrl+N, Alt+F10	Next error
Alt+F6	(Pro only) Compile current buffer
Ctrl+C	Stop concurrent process
Ctrl+X Ctrl+M	(Pro only) Start concurrent process
Ctrl+X Comma, Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+X Ctrl+H, Ctrl+Comma	Pop a pushed bookmark
Ctrl+X Dot	Go to definition
Ctrl+X Alt+Comma	Context Tagging® - Tag Files dialog box
F3	Make and load current macro buffer
Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias

Epsilon Debugging (Pro only)

F5	Start/continue debugging
Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
F9	Toggle breakpoint

Epsilon Keys

Ctrl+F9	Toggle breakpoint enable
Ctrl+Shift+F9	Clear all breakpoints
F10	Step over
F11	Step into
Ctrl+F10	Run to cursor
Alt+PadStar	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

Epsilon Macros

Ctrl+X 'E'	Run last recorded macro
F3	Make and load current buffer
Ctrl+X Alt+N	Save last recorded macro
Ctrl+X '('	Start/end macro recording
Ctrl+X ')'	End macro recording
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and

	run dialog box/macro.
--	-----------------------

Epsilon Command Line and Text Box Editing

The following keys are different in all Text Boxes except the command line if the CUA Text Box check box is enabled (**Tools > Options > Redefine Common Keys**):

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Alt+A..Alt+Z	Taken over by dialog manager for selecting controls

Epsilon Command Line Keys

F2, Alt+X	Cursor to command line toggle
Ctrl+D	Delete character
Ctrl+H, Backspace	Delete previous character
Tab, Ctrl+I	Insert tab character
Ctrl+J, Enter, Ctrl+M	ENTER argument
Space	Complete argument
?	List arguments
Ins	Insert toggle
Alt+\	Delete horizontal space
Ctrl+Q	Quote next character
Ctrl+Space	Expand alias at cursor. Use alias command to define aliases.
Ctrl+K	Cut to end of line
Ctrl+Alt+H	Cut previous word

Epsilon Keys

Alt+D	Cut word
Ctrl+W, Shift+Del	Cut selection
Ctrl+Y, Shift+Ins	Paste
Alt+Y	Paste recent clipboard
Ctrl+Alt+W	Append next clipboard
Alt+C	Capitalize word
Alt+L	Lowcase word
Alt+U	Upcase word
Ctrl+T	Transpose characters
Alt+T	Transpose words
Ctrl+N, Down	Retrieve previous argument (cmdline only)
Ctrl+P, Up	Retrieve next argument (cmdline only)
F2, Alt+X	Complete and enter argument (cmdline only)
Ctrl+D	Start or extend char/stream selection
Ctrl+H, Backspace	Start char/stream selection
Tab, Ctrl+I	Extend selection
Ctrl+J, Enter, Ctrl+M	Select word
Space	Select line

Epsilon Miscellaneous

F1, Alt+?, Ctrl+-	Help for mode or context
Alt+P	Configuration menu
Ctrl+X Ctrl+C	Safe exit

Epsilon Keys

Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Ctrl+G	Cancel
F9, Ctrl+X 'U'	Undo
Shift+F9, Ctrl+X 'R'	Redo
Ctrl+F9	Undo w/cursor grouping
Ctrl+Shift+H	Hex display toggle
Ctrl+T	Transpose characters
Alt+T	Transpose words
Ctrl+X Ctrl+T	Transpose lines
Alt+U	Upcase word
Alt+L	Lowcase word
Alt+C	Capitalize word
Ctrl+X 'F'	Set margins
Alt+Q	Reflow paragraph
Alt+S	Center line within margins
Alt+)	Find matching start paren
Ctrl+L	Center line within window
Ctrl+Shift+O	Expand alias at cursor
Ctrl+X /	Alias change directory
F8, Ctrl+F8	Set macro variable
F4	Bind to key

ISPF Keys

F6	What is
F7	Change directory
Ctrl+X Ctrl+E	Shell
Ctrl+X Ctrl+Z	Resume
Ctrl+X '='	Display information on cursor position
Ctrl+X 'L'	Count lines
Alt+~	Modify toggle
Alt+Z	Scroll down
Ctrl+Z	Scroll up

Epsilon Argument/Repeating a Key

Ctrl+U	Select number of times to invoke a command. Sets argument-count.
Ctrl+K	Cuts argument-count complete lines
Ctrl+X Ctrl+O	Change number of blank lines at or before cursor to argument-count
Ctrl+X Ctrl+I, Ctrl+X Tab	Indent or unindent selected lines by argument-count characters
Alt+0..Alt+9	Select argument-count 0..9

ISPF Keys

ISPF Cursor Movement

Left arrow	Cursor left, if in first column, cursor moves into prefix area
Right arrow	Cursor right, if at end of prefix area, moves to first column of line
Up arrow	Cursor up
Down arrow	Cursor down

ISPF Keys

Ctrl+Home	Top of buffer
Ctrl+End	Bottom of buffer
Home	Begin line
End	End line
PgUp	Page up
PgDn	Page down
F7	Page up
F8	Page down
F10	Page left
F11	Page right
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Indent or move to next tab stop
Shift+Tab	Move to previous tab stop, or if in first column, move to prefix area
Ctrl+J	Go to line
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

ISPF Line Prefix Commands

.label	Define line prefix label
--------	--------------------------

ISPF Keys

bnbs	Insert left and right boundary ruler line
tabs	Insert tabs ruler line
cols	Insert column ruler line

ISPF Inserting Text

Ins	Insert/overwrite toggle
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+Q	Quote next character typed

ISPF Inserting Text - Line Prefix Commands

i [n]	Insert n lines after this line
te [n]	Insert n lines for word-wrap text entry
tj	Join line
ts	Split line
mask	Insert new line mask
d [n]	Delete n lines
dd	Delete a block of lines

ISPF Deleting Text

Del	Delete char under cursor
Backspace	Delete char before cursor

ISPF Keys

Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Ctrl+Shift+K	Cut word

ISPF Deleting Text - Line Prefix Commands

d [n]	Delete n lines
dd	Delete a block of lines

ISPF Searching

Ctrl+F	Find
Ctrl+R	Replace
Ctrl+G	Find next occurrence
Ctrl+Shift+G	Find previous occurrence
F5	Find next occurrence
F6	Repeat last change
Ctrl+I	Incremental search
Ctrl+Shift+I	Reverse incremental search
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

ISPF Selection

Ctrl+A	Select all
Ctrl+B	Select block/column

ISPF Keys

Ctrl+L	Select line
Ctrl+U	Deselect
Ctrl+X, Shift+F1	Cut selection
Backspace, Del, Shift+F4	Delete selection
Tab	Indent selection
Shift+Tab	Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Alt+=	Execute commands in selection
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

ISPF Selection - Line Prefix Commands

([n], (([n]	Shift block n columns left
---------------	----------------------------

ISPF Keys

) [n],)) [n]	Shift block n columns right
< [n], << [n]	Shift data n columns right
> [n], >> [n]	Shift data n columns right
c [n]	Copy n lines
cc	Copy a block of lines
m [n]	Move n lines
mm	Move a block of lines
z [n]	Select n lines
zz	Select a block of lines

ISPF Clipboard

Ctrl+C, Ctrl+Ins, Shift+F2	Copy selection to clipboard
Ctrl+Shift+C	Append selection to clipboard
Ctrl+K	Copy word to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Shift+Ins, Shift+F3	Paste
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Ctrl+Shift+K	Cut word
Ctrl+X, Shift+Del, Shift+F1	Cut selection
Ctrl+Shift+X	Append cut selection

ISPF Clipboard - Line Prefix Commands

--	--

a [n]	Insert block after, repeat n times
b [n]	Insert block before, repeat n times
o [n]	Overlay n lines
oo	Overlay a block of lines
r [n]	Repeat a line n times
rr [n]	Repeat a block n times
z [n]	Select n lines
zz	Select a block of lines

ISPF Command Line and Text Box Editing

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Ctrl+Shift+C	Append selection to clipboard
Ctrl+Shift+X	Append cut selection
Ctrl+Shift+V	List clipboards and optionally paste one
Esc	Cursor to command line toggle
Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+U	Upcase word
Ctrl+Shift+L	Lowcase word

ISPF Keys

Ctrl+Shift+K	Cut word
Ctrl+E	Cut to end of line
Ctrl+Backspace	Cut line
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character
Ctrl+K	Copy word to clipboard
Up arrow	Retrieve previous command
Down arrow	Retrieve next command
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line
F12	Retrieve previous command from command line
Shift+F12	Retrieve next command from command line

ISPF Files and Buffers

F2, Ctrl+O	Edit a file or find buffer
F3, F4	Save (if autosave is on) and quit current buffer
F3, Ctrl+S	Save current buffer
Ctrl+N	Next buffer
Ctrl+P	Previous buffer
F9	Next buffer or window

ISPF Keys

Shift+F9	Previous buffer or window
Ctrl+Shift+B	List buffers

ISPF Windowing

Ctrl+H	Split window horizontally
Ctrl+Tab, Ctrl+F6	Next window
Ctrl+Shift+Tab, Ctrl+Shift+F6	Previous window
Ctrl+Shift+Z	Zoom window toggle
Ctrl+F4	Close window
Alt+F2	Move window edge
Alt+F3	Create window edge
Ctrl+F7	Move
Ctrl+F8	Size
Ctrl+F9	Minimize
Ctrl+F10	Maximize

ISPF Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only)
Ctrl+Space	(Pro only) Complete symbol

ISPF Keys

Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+/-	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
Ctrl+G	Find next reference
Ctrl+Shift+G	Find previous reference
Ctrl+M	(Pro only) Build project
Ctrl+F5	(Pro only) Execute project
Ctrl+Shift+Down	Next error
Ctrl+Shift+Up	Previous error
Ctrl+Shift+S	Set next error
Ctrl+Shift+E	List errors
Shift+F10	(Pro only) Compile current buffer
Alt+1	Cursor to error/include file
Ctrl+Shift+M	(Pro only) Start concurrent process
Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias

ISPF Debugging (Pro only)

Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
F9	Toggle breakpoint

ISPF Keys

Ctrl+F9	Toggle breakpoint enable
Ctrl+Shift+F9	Clear all breakpoints
F10	Step over
F11	Step into
Ctrl+F10	Run to cursor
Alt+PadStar	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

ISPF Macros

Ctrl+F11	Start/end macro recording
Ctrl+F12	Terminate recording & run last recorded macro
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that will not close. If editing a dialog box or macro, load and run the dialog box/macro.

ISPF Selective Display

--	--

ISPF Keys

x [n]	Exclude n lines
xx	Exclude a block of lines
f [n]	Expose first n lines of excluded block
l [n]	Expose last n lines of excluded block
s [n]	Expose n lines at first indentation level

ISPF Miscellaneous

F1	Help for mode or context
Ctrl+F1	Help on word at cursor
Alt+F4	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Esc	Cancel or command line toggle
Ctrl+Z, Alt+Backspace	Undo
Ctrl+Y	Redo
Ctrl+Shift+H	Hex display toggle
Ctrl+Shift+U	Upcase word
Ctrl+Shift+L	Lowcase word
Ctrl+Shift+J	Go to bookmark
Ctrl+0..Ctrl+9	Set bookmark 0..9
Ctrl+Shift+N	Activate Bookmarks tool window
Ctrl+]	Match parenthesis

macOS Keys

Ctrl+\	Expand or collapse selective display
Ctrl+Shift+O	Expand alias at cursor
Ctrl+D	Change directory
Alt+F5	Restore MDI window
Alt+F10	Maximize MDI window
Enter	Execute line prefix commands or, if there are none, go to the next line.
Ctrl+Enter	Execute line prefix commands
Right-Ctrl	Alternate key for executing line prefix commands

ISPF Miscellaneous - Line Prefix Commands

lc [n]	Lowcase n lines
lclc, lcc	Lowcase block of lines
md [n]	Make n data lines
mdmd, mdd	Make data lines
uc [n]	Upcase n lines
uclc, ucc	Upcase block of lines
tf	Reflow paragraph

macOS Keys

macOS Cursor Movement

Shift + Tab	Back indent text to previous tab stop
Command+Left arrow, Ctrl+A	Begin line

macOS Keys

Command+Down arrow, Ctrl+End	Bottom of buffer
Ctrl+PgDn	Bottom of window
Down arrow	Cursor down
Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
End, Ctrl+E	End line
Ctrl+J	Go to line1
Tab	Indent to next tab stop
Option+Right, Ctrl+Right	Next word
PgDn	Page down
PgUp	Page up
Option+Left, Ctrl+Left	Previous word
Ctrl+Home, Command+Up arrow	Top of buffer
Ctrl+PgUp	Top of window
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

macOS Inserting Text

Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line

macOS Keys

Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character types

macOS Deleting Text

Ctrl+Backspace	Cut line
Ctrl+K	Cut to end of line
Ctrl+Shift+K	Cut word
Backspace	Delete char before cursor
Del	Delete char under cursor

macOS Selection

Command+X	Cut selection
Backspace, Del	Delete selection
Ctrl+U	Deselect
Shift+Click	Extend selection
Tab	Indent selection
Ctrl+Right-Click	Move selection to cursor
Command+A	Select all
F8	Select character/stream
Ctrl+L, Command+L	Select line
Triple-Click	Select line

macOS Keys

Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words
Ctrl+W, Double-Click	Select word
Command+]	Indent selection
Command+[Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Right-Click & Drag	Start block/column selection
Click & Drag	Start char/stream selection
Shift+<Cursor keys>	Start or extend char/stream selection

macOS Searching

Command+F	Find
Shift+Command+F	Find in files
Command+E	Use selection for find
Command+G	Find next occurrence
Command+Shift+G	Find previous occurrence
Ctrl+R	Replace
Ctrl+Shift+R	Replace in files

macOS Command Line and Text Box Editing

Ctrl+Shift+X	Append cut selection
--------------	----------------------

macOS Keys

Ctrl+Shift+C	Append selection to clipboard
Option+Command+Comma	Complete argument
Command+C, Ctrl+C	Copy selection to clipboard
Esc	Cursor to command line toggle
Ctrl+Backspace	Cut line
Command+X	Cut selection
Ctrl+Shift+K	Cut word
Shift+Click	Extend selection
Ins	Insert/overwrite toggle
Ctrl+Shift+V	List clipboards, optionally paste one
Ctrl+Shift+L	Lowcase selection
Option+Right	Next word
Command+V, Ctrl+V	Paste
Option+Left, Ctrl+Left	Previous word
Ctrl+Q	Quote next character
Down arrow	Retrieve next command
Up arrow	Retrieve previous command
Triple-Click	Select line
Double-Click	Select word
Command+E	Use selection for search
Click & Drag	Start char/stream selection
Ctrl+Shift+U	Upcase word

macOS Files and Buffers

Command+O, Ctrl+O	Edit a file or find buffer
Ctrl+=	File compare
Ctrl+Shift+B	List buffers
Ctrl+N	Next Buffer
Command+N	New File
Ctrl+Backtick	Open associated file
Ctrl+P	Previous buffer
Command+S, Ctrl+S	Save current buffer

macOS Clipboard

Ctrl+Shift+X	Append cut selection
Ctrl+Shift+C	Append selection to clipboard
Command+C, Ctrl+C	Copy selection to clipboard
Ctrl+Backspace	Cut line
Ctrl+K	Cut to end of line
Command+X, Ctrl+X	Cut selection
Ctrl+Shift+Command+Right	Select next expression
Ctrl+Shift+V	List clipboards, optionally paste one
Command+V	Paste

macOS Windowing

Option+Tab	Next window

macOS Keys

Option+Shift+Tab	Previous window
Command+M	Minimize window
Command+Backtick	Cycle through application windows

macOS Macros

Ctrl+Shift+Space	Edit current dialog box if running a dialog box. Close dialog box that won't close. Load and run dialog box/macro if editing dialog box or macro.
Ctrl+Option+Command+S	Halt Slick-C® macro prompting for a key with get_event()
Ctrl+Option+Command+T	Halt Slick-C macro that is executing. Terminate infinite loops.
F4, F12	Make and load current macro buffer
Shift+F4	Start/end macro recording
Ctrl+F12	Terminate recording, run last recorded macro

macOS Miscellaneous

Esc	Cancel or command line toggle
Ctrl+Shift+Space	Complete more
Ctrl+Shift+Period	Complete next word/variable
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+O	Expand alias at cursor
Ctrl+Shift+J	Toggle bookmark
Ctrl+Shift+N	Activate Bookmarks tool window
F1	Help for mode or context
Ctrl+Shift+H	Hex display toggle

SlickEdit® Keys

Ctrl+Shift+L	Lowcase selection
Ctrl+Shift+U	Upcase word
Ctrl+]	Match parenthesis
Ctrl+\	Expand or collapse selective display
Option+F10, Command+F10	Maximize MDI window
Command+F7	Move MDI window
Command+Shift+Z	Redo
Option+F5	Restore MDI window
Command+Q	Safe exit
Ctrl+0..Ctrl+9	Set bookmark 0..9
Command+Z, Option+Backspace	Undo

SlickEdit® Keys

SlickEdit® Cursor Movement

Left arrow, Ctrl+J	Cursor left
Right arrow, Ctrl+L	Cursor right
Up arrow, Ctrl+I	Cursor up
Down arrow, Ctrl+K	Cursor down
Ctrl+Home, Ctrl+X Ctrl+U	Top of buffer
Ctrl+End, Ctrl+X Ctrl+J	Bottom of buffer
Home, Ctrl+U	Begin line
End, Ctrl+O	End line
PgUp, Ctrl+P	Page up

PgDn, Ctrl+N	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Next tab stop
Shift+Tab	Previous tab stop
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

SlickEdit® Inserting Text

Ins, Ctrl+X Ctrl+O	Insert/overwrite toggle
Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Alt+N	Insert buffer name
Shift+Space	Insert a space (no syntax expansion)
Ctrl+X Tab	Move text tab
Ctrl+Q	Quote next character typed
Alt+S	Split line at cursor

SlickEdit® Deleting Text

Del, Ctrl+D	Delete char under cursor
Backspace	Delete char before cursor
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Alt+W	Cut word
Alt+J	Join line to cursor

SlickEdit® Searching

Ctrl+F	Find next occurrence
Ctrl+Shift+F	Find previous occurrence
Ctrl+S	Incremental search
Ctrl+X Ctrl+Z	Resume search and replace (Supports command line replace command only)
Ctrl+X Ctrl+R	Reverse incremental search
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

SlickEdit® Selection

Alt+B	Select block/column
Alt+L	Select line
Alt+Z	Select character/stream
Alt+U	Deselect

SlickEdit® Keys

Alt+C	Copy selection to cursor
Alt+K	Cut selection
Alt+M	Move selection to cursor
Alt+F	Fill selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Ctrl+F3	Uppercase selection
Ctrl+F4	Lowercase selection
Alt+A	Move/overlay block
Alt+O	Overlay block selection
Alt+E	Go to end of selection
Alt+Y	Go to beginning of selection
Ctrl+X Ctrl+P	Reflow selection
Alt+=	Execute commands in selection
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line

SlickEdit® Clipboard

Alt+V, Ctrl+Ins	Copy selection to clipboard
Ctrl+X Ctrl+W	Copy word to clipboard
Ctrl+X Ctrl+Y	List clipboards and optionally paste one
Ctrl+Y, Shift+Ins	Paste
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Alt+W	Cut word
Alt+K, Shift+Del	Cut selection

SlickEdit® Command Line and Text Box Editing

The following keys are different in all Text Boxes except the command line if the CUA Text Box check box is enabled. (**Tools > Options > Redefine Common Keys**):

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Alt+A..Alt+Z	Taken over by dialog manager for selecting controls

SlickEdit® Command Line Keys

Esc, Ctrl+A	Cursor to command line toggle
Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word

SlickEdit® Keys

Ctrl+F1	Upcase word
Ctrl+F2	Lowcase word
Alt+W	Cut word
Ctrl+E	Cut to end of line
Ctrl+Backspace	Cut line
Alt+V	Copy selection to clipboard
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character
Ctrl+X Ctrl+W	Copy word to clipboard
Ctrl+Y	Paste
Ctrl+X Ctrl+Y	List clipboards and optionally paste one
Alt+N	Insert buffer name
Ctrl+Space	Expand alias at cursor. Use alias command to define aliases.
Up arrow, Ctrl+I	Retrieve previous command
Down arrow, Ctrl+K	Retrieve next command
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

SlickEdit® Files and Buffers

F2, Ctrl+X Ctrl+S	Save current buffer

SlickEdit® Keys

F3, Ctrl+X 'K'	Quit current buffer
F8, Ctrl+B	Next buffer
Ctrl+F8, Ctrl+V	Previous buffer
F4	Save and quit current buffer
F7	Edit a file or find buffer
Ctrl+X Ctrl+B	List buffers
Ctrl+X 'B'	Find buffer
F6	File compare

SlickEdit® Windowing

Ctrl+X '2'	Split window horizontally
Ctrl+W, Ctrl+Tab	Next window
Ctrl+Shift+Tab	Previous window
Ctrl+Z	Zoom window toggle
Ctrl+X '1'	One window

SlickEdit® Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only) Symbol Information Help
Ctrl+H, Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor

SlickEdit® Keys

Ctrl+Alt+H, Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+/ Ctrl+Shift+P	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+X Ctrl+H, Ctrl+Comma	Pop a pushed bookmark
Ctrl+F	Find next reference
Ctrl+Shift+F	Find previous reference
Ctrl+Space	(Pro only) Complete symbol
Ctrl+X 'M'	(Pro only) Build project
Ctrl+F5	(Pro only) Execute project
Alt+F10, Ctrl+X Ctrl+N	Next error
Ctrl+X 'N'	Set next error
Ctrl+F6	(Pro only) Compile current buffer
Alt+1	Cursor to error/include file
Ctrl+X Ctrl+L	Make and load current macro buffer
Ctrl+C	Stop concurrent process
Ctrl+X Ctrl+M	(Pro only) Start concurrent process
Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias

SlickEdit® Debugging (Pro only)

Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
Ctrl+Shift+F9	Clear all breakpoints

SlickEdit® Keys

F10	Step over
F11	Step into
Ctrl+F10	Run to cursor
Alt+PadStar	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

SlickEdit® Macros

Ctrl+R	Start/end macro recording
Ctrl+T	Terminate recording & run last recorded macro
Ctrl+Shift+T,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+X Ctrl+L	Make and load current buffer
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and run dialog box/macro.
Ctrl+X 'E'	Run last recorded macro
Ctrl+X '('	Start recording macro
Ctrl+X ')'	End recording macro

SlickEdit® Miscellaneous

F1	Help for mode or context
F5	Configuration menu
Alt+X, Ctrl+X Ctrl+C	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Esc, Ctrl+G	Cancel
F9	Undo
Ctrl+F9	Undo with cursor motion grouping
Shift+F9, Ctrl+X R	Redo
Ctrl+Shift+H	Hex display toggle
Ctrl+F1	Upcase word
Ctrl+F2	Lowcase word
Shift+F5	Center line
Shift+F1	Scroll up
Shift+F2	Scroll down
Shift+F3	Scroll left
Shift+F4	Scroll right
Alt+S	Split line
Alt+J	Join line
Alt+P	Reflow paragraph
Ctrl+Shift+N	Activate Bookmarks tool window

Alt+T	Match parenthesis
Alt+R	Fundamental mode for next key press
Ctrl+X Ctrl+E	OS Shell
Ctrl+X Ctrl+D	Alias change directory

Vim Keys

For a complete list of supported EX command, see [Vim EX commands](#).

Differences Between SlickEdit Vim and gvim

SlickEdit Vim emulation does not support all Vim EX commands. Some are probably not worth adding. Others haven't been done yet.

Note

Please post changes/enhancements you would like to see for SlickEdit's Vim emulation on the forum.

Some more significant differences are listed below:

\%V, \%#, \%'m, \%l, \%c, \%V, \%[], \ze, \z1..\z9, \z*(\)	Not yet supported in Vim regex syntax.
	Not supported in EX command line. Vim uses this to add additional commands.

Vim Cursor Movement

Left arrow, Ctrl+J	Cursor left
Right arrow, Ctrl+L	Cursor right
Up arrow, Ctrl+I	Cursor up
Down arrow, Ctrl+K	Cursor down
Ctrl+Home, Ctrl+X Ctrl+U	Top of buffer

Vim Keys

Ctrl+End, Ctrl+X Ctrl+J	Bottom of buffer
Home	Begin line
End, Ctrl+O	End line
PgUp, Ctrl+B	Page up
PgDn, Ctrl+F	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Indent to next tab stop
Shift+Tab	Indent to previous tab stop
Ctrl+	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

Vim Cursor Movement - Normal Mode Only

gj	Down screen line (different when lines wrap)
gk	Up screen line (different when lines wrap)
-	Begin previous line
^	Begin text
0	Begin line
\$	End line
G, gg	Go to line

Vim Keys

	Go to column
w	Next word
W	Next non-white space word
b	Previous word
B	Previous non-white space word
ge	Backward to end of word
gE	Backward to end of non-white space word
e	End of word
E	End of non-white space word
(Previous sentence
)	Next sentence
{	Previous paragraph
}	Next paragraph
[[Previous section
]]	Next section
%	Find matching paren
N %	Move N % down a buffer
H	Move to upper-left corner of window
M	Move to middle of window
L	Move to lower-left corner of window
'	Jump to bookmarked line
~	Jump to bookmarked column
gm	Move to middle of window on the current line

Vim Cursor Movement - Visual Mode Only

aw	Select a word
iw	Select inner word
aW	Select WORD
iW	Select inner WORD
as	Select a sentence
is	Select inner sentence
ap	Select a paragraph
ip	Select inner paragraph
ab	Select a block
ib	Select inner block
aB	Select a Block
iB	Select inner Block

Vim Inserting Text

Ins	Insert/overwrite toggle
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)

Vim Inserting Text - Normal Mode Only

i	Insert text
I	Insert text at beginning of line
a	Append text

Vim Keys

A	Append text after end of line
o	Insert text below current line
O	Insert text above current line

Vim Deleting Text

Del	Delete character under cursor
Ctrl+E	Cut to end line
Ctrl+Backspace	Cut line

Vim Deleting Text - Normal Mode Only

x	Delete character under cursor
(visual) x	Delete selection
X	Delete character before cursor
(visual) d	Delete selection
d	Delete text
D	Delete to end of line

Vim Searching

Command+F	Find dialog
Ctrl+R (insert mode only)	Replace dialog
Command+G	Find next occurrence
Ctrl+Shift+G, Command+Shift+G	Find previous occurrence
Ctrl+I	Incremental search
Ctrl+Shift+I	Reverse incremental search

<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.
-----------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Vim Searching - Normal Mode Only

/	Search forward (accommodates multipliers)
?	Search backward (accommodates multipliers)
n	Forward repeat last search
N	Backward repeat last search
f	Forward character search
F	Backward character search
t	Move cursor up to character
T	Move cursor backward after character
Semicolon (;)	Repeat character search
Comma (,)	Reverse repeat character search
m	Set bookmark

Vim Selection

Ctrl+L	Select line
Shift+F7	Shift block selection left
Shift+F8	Shift block selection right
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection

Vim Keys

Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

Vim Selection - Normal Mode Only

v	Character visual mode
V	Line visual mode
Ctrl+V	Block visual mode
o	Move cursor to beginning (or end) of selection

Vim Clipboard

Ctrl+C, Command+C	Copy selection to clipboard
Ctrl+K	Copy word to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Command+V, Shift+Ins	Paste
Ctrl+Backspace	Cut line
Ctrl+E	Cut to end of line
Ctrl+X (Insert mode only),	Cut selection

Command+X, Shift+Del	
----------------------	--

Vim Clipboard - Normal Mode Only

p	Paste text after cursor
]p	Paste text after cursor (adjust indent)
gp	Paste text after cursor (leave cursor after new text)
(visual) p	Paste clipboard contents over selection
P	Paste text before cursor
[p	Paste text before cursor (adjust indent)
gP	Paste text before cursor (leave cursor after new text)
y	Copy text to clipboard
(visual) y	Copy selection to clipboard
Y	Copy line to clipboard

Vim Command Line and Text Box Editing

The following keys are different in all Text Boxes except the command line if the CUA Text Box check box is enabled (**Tools > Options > Redefine Common Keys**):

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Alt+A..Alt+Z	Taken over by dialog manager for selecting controls

Vim Command Line and Text Box Editing - Normal Mode Only

Ctrl+Q	Cursor to command line toggle

Vim Keys

Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+U	Upcase selection
Ctrl+Shift+L	Lowcase selection
Ctrl+E	Cut to end of line
Ctrl+Backspace	Cut line
Ins	Insert/overwrite toggle
Ctrl+V	Paste
Ctrl+K	Copy word to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+Space	Expand alias at cursor. Use the alias command to define aliases or the Ex command : abbr .
Up arrow, Ctrl+I	Retrieve previous command
Down arrow, Ctrl+K	Retrieve next command
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

Vim Files and Buffers

F2, Ctrl+S, Command+S	Save current buffer
-----------------------	---------------------

F3	Quit current buffer
F8	Next buffer
Ctrl+F8	Previous buffer
F4	Save and quit current buffer
F7, Ctrl+O, Command+O	Open a file or find buffer
Ctrl+Shift+B	List buffers
F6	File compare

Vim Files and Buffers - Normal Mode Only

:w	Save current buffer
:q	When in "One file per window mode" close window and buffer. When in "Multiple files share window" mode, close tile without closing buffer or close buffer if only one tile left.
:clo[se], :bdelete	When in "One file per window" mode close window and buffer. When in "Multiple files share window" mode, close buffer.
:bn[ext]	Next buffer. When in "One file per window" mode, switches to next window. When in "Multiple files share window" mode, switches to next buffer within the current window.
:bp[revious]	Previous buffer. When in "One file per window" mode, switches to previous window. When in "Multiple files share window" mode, switches to previous buffer within the current window.
:buffers	List buffers.
:e[dit] filename	Open file specified
:wq	Save and quit current buffer
:q!	Quit current buffer without saving
:wa[ll]	Write all buffers
:qa[ll]	Quit all buffers

<code>:wqa[ll]</code>	Write and quit all buffers
<code>:qa[ll]!</code>	Quit all buffers without saving
<code>:xa[ll]!</code>	Write all changed buffers and exit

Vim Windowing

Ctrl+Tab	Next window
Ctrl+Shift+Tab	Previous window
Ctrl+Shift+Z	Zoom window toggle

Vim Windowing - Normal Mode Only

<code>:b[uffer] [buffer-id]</code>	Edits the file corresponding to the buffer id specified.
<code>:sb[uffer] [buffer-id]</code>	Split window horizontally. If [buffer-id] is specified, the file corresponding to [buffer-id] is opened.
<code>:sb[uffer] [file]</code>	Split window horizontally. If [file] is specified, [file] is opened.
<code>:sp[lit] [file]</code>	Split window horizontally. If [file] is specified, [file] is opened.
<code>:vs[plit] [file]</code>	Split window vertically. If [file] is specified, [file] is opened.
Ctrl+w]	Split window and jump to symbol under cursor
Ctrl+w f	Split window and edit file name under the cursor
Ctrl+w n	Split window with empty new window
Ctrl+w o	Make current window the only visible window
Ctrl+w j	Move cursor to window below
Ctrl+w k	Move cursor to window above
Ctrl+w Ctrl+w	Move cursor to window below (wrap)
Ctrl+w W	Move cursor to window above (wrap)

Ctrl+w t	Move cursor to top window
Ctrl+w b	Move cursor to bottom window

Vim Compiling and Programming Support

Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only)
Ctrl+Space	(Pro only) Complete symbol
Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Ctrl+ /	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
Ctrl+M	(Pro only) Build project
Ctrl+Shift+Down	Next error
Ctrl+Shift+Up	Previous error
Ctrl+Shift+S	Set next error
Ctrl+F5	(Pro only) Execute project
Shift+10	(Pro only) Compile current buffer
Alt+1	Cursor to error
F12	Make load current macro buffer

Vim Keys

Ctrl+Shift+M	(Pro only) Start concurrent process
Ctrl+Shift+P	Expand extension specific alias
Ctrl+Shift+O	Expand global alias

Vim Debugging (Pro only)

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+PadStar	Show next statement
Alt+F9	Activate breakpoints window
Ctrl+Alt+W	Activate watch window
Ctrl+Alt+V	Activate variables window
Ctrl+Alt+C	Activate call stack
Ctrl+Alt+H	Activate threads window

Vim Macros

Ctrl+F11	Start/end macro recording

Vim Keys

F12	Make and load current macro buffer
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If editing dialog box or macro, load and run dialog box/macro.

Vim Macros - Normal Mode Only

Dot	Repeat last insert or delete
q{a-zA-Z0-9}	Recording keyboard macro to clipboard(register) id that follows. Press 'q' again to end recording.
@{a-zA-Z0-9}	Playback keyboard macro for clipboard(register) id that follows.

Vim Miscellaneous

Alt+F4, Command+Q, Command+F4	Safe exit
F1	Help for mode or context
Command+,	Configuration menu
Ctrl+Shift+H	Hex display toggle
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+Space	Complete more
Esc	Cancel
Ctrl+Z (insert mode only), Command+Z, Alt+Backspace	Undo

Vim Keys

Shift+F9	Undo with cursor motion grouping
Ctrl+Y (insert mode only), Command+Shift+Z	Redo
Ctrl+A (command or visual mode only)	Increment number
Ctrl+X (command or visual mode only)	Decrement number
Ctrl+Shift+U	Upcase selection
Ctrl+Shift+L	Lowcase selection
Ctrl+]	Match parenthesis
Ctrl+\	Expand or collapse selective display
Ctrl+Shift+O	Expand alias at cursor
Alt+/, Command+/-	Alias change directory

Vim Miscellaneous - Normal Mode Only

u	Full undo
(visual) u	Change selected text to lowercase
U	Undo with cursor motion grouping
(visual) U	Change selected text to uppercase
~	Toggle the case of current character
(visual) ~	Toggle the case of the selected text
c	Change text
C	Change to end of line
(visual) c, C	Change text in selection
r	Overstrike character(s)

(visual) r	Replace text in selection
R	Overstrike text
s	Substitute character(s)
S	Substitute line
<	Shift text left
>	Shift text right
J	Join line (insert spaces)
(visual) J	Join the selected lines (insert spaces)
(visual) gJ	Join the selected lines (no spaces)
:	EX command mode
Q	EX editor mode
&	Repeat last SUBSTITUTE
Esc, Ctrl+[Normal mode

Vim EX command line

Many Vim EX commands take the following form:

:[range]ex-cmd

range can either be a single range specifier (see table below) or a two comma (or semicolon) delimited range specifies (as in 1,\$).

Vim EX range specifiers

{number}	Absolute line number.
.	Current line.
\$	Last line of file.
%	Same as 1,\$ which specifies the entire file.

't	Line number corresponding to book mark specified.
'<	Line number corresponding to first line of selection.
'>	Line number corresponding to last line of selection.
/pattern[!]	Next line which matches <i>pattern</i> . Pattern defaults to Vim regular expression syntax (see Vim Regular Expressions).
?pattern[?]	Previous line which matches <i>pattern</i> . Pattern defaults to Vim regular expression syntax (see Vim Regular Expressions).

Vim EX commands

:!external-program	Run <i>external-program</i> specified.
:[range]!external-program	Filter line(s) specified with <i>external-program</i> specified.
:[range]<[NumLines]	Unindents line or <i>NumLines</i> specified by syntax indent width or "shiftwidth" setting. You may specify multiple less thans to unindent by a multiple of the syntax indent. For example << unindents by syntax indent*2.
:[range]>[NumLines]	Indents line or <i>NumLines</i> specified by syntax indent width or "shiftwidth" setting. You may specify multiple greater thans to indent by a multiple of the syntax indent. For example >> indents by syntax indent*2.
:bd[elete]	When in "One file per window" mode close window and buffer. When in "Multiple files share window" mode, close buffer.
:bn[ext]	Next buffer. When in "One file per window" mode, switches to next window. When in "Multiple files share window" mode, switches to next buffer within the current window.
:bp[revious]	Previous buffer. When in "One file per window" mode, switches to previous window. When in "Multiple files share window" mode, switches to previous buffer within the current window.
:bufdo cmd	Execute <i>cmd</i> for each buffer.
:b[uffer] [buffer-id]	Edits the file corresponding to the buffer id specified.
:buffers	List buffers.

:cd [path]	Change directory to <i>path</i> if given or display current directory.
:clo[se]	When in "One file per window" mode close window and buffer. When in "Multiple files share window" mode, close buffer.
:<i>[range]</i>co[py] destLine	Copy line(s) after <i>destLine</i>
:<i>[range]</i>d[elete]	Delete line(s) specified
:e[dit] filename	Open file specified
:f[ile]	Displays file info which includes filename, current line number, and number of lines in the file
:<i>[range]</i>g[lobal] /<i>pattern</i>! [<i>cmd</i>]	Mark lines with occurrences of pattern in the line range specified. Apply <i>cmd</i> to all marked lines. <i>partern</i> defaults to Vim regular expression syntax (see Vim Regular Expressions). <i>cmd</i> defaults to ":print" if not specified.
:<i>[range]</i>g[lobal]! /<i>pattern</i>! [<i>cmd</i>]	Same as :global except lines without occurrences of <i>pattern</i> are marked.
:h[elp] [<i>vim-help-item</i>]	Provides help on the <i>vim-help-item</i> specified. Currently supports EX commands and range specifiers. For example, :help :s will display help on the :substitute EX command. Help with no parameters displays start of Vim emulation keys section which provides a link to a list of the supported EX commands.
:<i>[range]</i>j[oin] [<i>NumLines</i>]	Join <i>NumLines</i> specified to current line. If a range is specified, <i>NumLines</i> is ignored and lines in range are joined to first line of range.
:k{a-z}	Set bookmark with the specified name. Note that in normal mode " goes to the next bookmark which is different than gvim.
:<i>[range]</i>l[ist] [<i>NumLines</i>]	Displays the line(s) specified with a \$ at the end of each line. Same as :print but appends \$ to the end of each line.
:<i>[range]</i>m[ove] destLine	Moves line(s) after <i>destLine</i>
:n[ext]	Switches to next buffer.
:noh[lsearch]	Clears highlighted search strings.
:<i>[range]</i>nu[mber] [<i>NumLines</i>]	Displays the line(s) specified with a line number at the beginning of each line. Same as :print but line number is displayed at the

Vim Keys

	beginning of each line.						
:p[rint] [NumLines]	Displays the line(s) specified.						
:[range]pu[t] [x]	Paste clipboard or text specified after the last line or range specified. <i>x</i> can either be a clipboard(register) name (a-z0-9+), ="text", ="\\text", or =mathematical_expression (ex. =0x4a+0x20).						
:q[uit]	When in "One file per window mode" close window and buffer. When in "Multiple files share window" mode, close tile without closing buffer or close buffer if only one tile left.						
:q[uit]!	Same as :quit but doesn't prompt to save changes.						
:qa[ll]	Close all buffers						
:[range]r[ead] file	Insert <i>file</i> specified after the last line of range specified.						
:[range]r[ead] !external-program	Insert output from <i>external-program</i> specified after the last line of range specified.						
:red[o]	Undoes an undo operation.						
:reg[isters]	Display all named and unnamed clipboards (registers).						
:rew[ind][!]	Revert the current buffer to the contents on disk. When ! is specified, buffer is reverted without prompting whether to discard changes.						
:sb[uffer] [buffer-id]	Split window horizontally. If [buffer-id] is specified, the file corresponding to [buffer-id] is opened.						
:sb[uffer] [file]	Split window horizontally. If [file] is specified, <i>file</i> is opened.						
	Replace occurrences of <i>pattern</i> with occurrences of <i>string</i> in the line <i>range</i> specified. By default, <i>pattern</i> is interpreted as a Vim regular expression (see Vim Regular Expressions). <i>options</i> is a string of one or more options.						
	<table border="1"> <thead> <tr> <th>Options</th><th>Description</th></tr> </thead> <tbody> <tr> <td>c</td><td>Confirm each substitution. 'y' to substitute match. 'l' to substitute match and then quit, 'n' to skip match, Esc to quit substituting, 'a' to substitute remaining matches, and 'q' to quit substituting.</td></tr> <tr> <td>g</td><td>Replace all occurrences in the line. With this</td></tr> </tbody> </table>	Options	Description	c	Confirm each substitution. 'y' to substitute match. 'l' to substitute match and then quit, 'n' to skip match, Esc to quit substituting, 'a' to substitute remaining matches, and 'q' to quit substituting.	g	Replace all occurrences in the line. With this
Options	Description						
c	Confirm each substitution. 'y' to substitute match. 'l' to substitute match and then quit, 'n' to skip match, Esc to quit substituting, 'a' to substitute remaining matches, and 'q' to quit substituting.						
g	Replace all occurrences in the line. With this						

/[options]	Options	Description
		option, only the first occurrence in each line is matched.
	i	Case insensitive matching.
	I	Case sensitive matching.
	e	Case sensitive matching.
	<	If found, place cursor at beginning of word.
	>	If found, place cursor at end of word.
	r	Interpret search <i>pattern</i> as a SlickEdit® regular expression.
	I	Interpret <i>pattern</i> as a Perl regular expression.
	~	Interpret <i>pattern</i> as a Vim regular expression.
	n	Interpret <i>pattern</i> as literal text (plain text search) and not any regular syntax.
	u	Interpret <i>pattern</i> as a Perl regular expression. Unix syntax regular expressions are no longer supported.
	p	Print current line after substitution.
	w	Limit search to words. Used to search for variables.
	v	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.
:se[t]	Display list of supported option settings.	

Visual C++ Keys

:se[t] option=value	Set <i>option</i> to <i>value</i> .
:sh[ell]	Runs the default shell. The shell command executed is specified in the 'shell' option.
:sp[lit] [file], :sbuffer [file]	Split window horizontally. If [<i>file</i>] is specified, <i>file</i> is opened.
:[range]t	Synonym for copy.
:ta[g] [tag]	If <i>tag</i> is specified, navigates to tag specified and pushes a bookmark at the previous location. Otherwise, bookmark is popped and cursor is placed on the previous location.
:u[ndo] [tag]	Undo last change.
:[range]v[global] /pattern/ [cmd]	Same as :global!.
:ve[rSION]	Displays product help about information.
:vs[plit] [file]	Split window vertically. If [<i>file</i>] is specified, <i>file</i> is opened.
:[range]:w[rite][!] [file]	Save lines specified to current buffer. If ! is specified, prompting to replace existing file is suppressed.
:[range]w[rite][!] [file]	Save specified lines to file specified. If ! is specified, prompting to replace existing file is suppressed.
:[range]w[rite][!] >>file	Append specified lines to file specified. If ! is specified, prompting to replace existing file is suppressed.
:wa[ll]	Write all buffers
:wq	Save the current buffer and quit.
:wqa[ll]	Save and quit all buffers
:xa[ll]!	Save all changed buffers and exit
:x	Same as :wq.
:[range]y[ank] [x]	Copy specified line(s) to the clipboard(register) specified.
:[range]z [NumLines]	Displays <i>NumLines</i> that follow the last line of the <i>range</i> specified.

Visual C++ Keys

Visual C++ Cursor Movement

Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
Down arrow	Cursor down
Ctrl+Home	Top of buffer
Ctrl+End	Bottom of buffer
Home	Begin line
End	End line
PgUp	Page up
PgDn	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Indent to next tab stop
Shift+Tab	Back indent text to previous tab stop
Ctrl+G	Go to line, offset, bookmark, error, definition, declaration, or reference
Ctrl+J	Previous preprocessing condition
Ctrl+K	Next preprocessing condition
Ctrl+	Add multiple cursors

Visual C++ Keys

Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

Visual C++ Inserting Text

Ins	Insert/overwrite toggle
Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+Q	Quote next character typed

Visual C++ Deleting Text

Del	Delete char under cursor
Backspace	Delete char before cursor
Ctrl+L	Cut line
Alt+Shift+L	Cut sentence

Visual C++ Searching

Ctrl+F	Find
Ctrl+H	Replace
F3	Find next occurrence
Shift+F3	Find previous occurrence
Ctrl+I	Incremental search

Visual C++ Keys

Ctrl+Shift+I	Reverse incremental search
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

Visual C++ Selection

Ctrl+A	Select all
Ctrl+B	Select block/column
Ctrl+F8	Select line
F8	Select character/stream
Ctrl+Shift+J	Select previous preprocessing condition
Ctrl+Shift+K	Select next preprocessing condition
Alt+U	Deselect
Ctrl+X	Cut selection
Backspace, Del	Delete selection
Tab	Indent selection
Shift+Tab	Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Alt+=	Execute commands in selection
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection

Visual C++ Keys

Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

Visual C++ Clipboard

Ctrl+C, Ctrl+Ins	Copy selection to clipboard
Ctrl+Shift+C	Append selection to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Shift+Ins	Paste
Ctrl+L	Cut line
Ctrl+X, Shift+Del	Cut selection
Ctrl+Shift+X	Append cut selection
Alt+Shift+L	Cut sentence

Visual C++ Command Line and Text Box Editing

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Ctrl+Shift+C	Append selection to clipboard

Visual C++ Keys

Ctrl+Shift+X	Append cut selection
Ctrl+Shift+V	List clipboards and optionally paste one
Esc	Cursor to command line toggle
Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+U	Upcase word
Ctrl+U	Lowcase word
Ctrl+L	Cut line
Ctrl+Shift+L	Delete line
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character
Alt+N	Insert buffer name
Up arrow	Retrieve previous command
Down arrow	Retrieve next command
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

Visual C++ Files and Buffers

Visual C++ Keys

F2, Ctrl+S	Save current buffer
Ctrl+O	Edit a file or find buffer
Ctrl+Shift+B	List buffers
F6	File compare

Visual C++ Windowing

Ctrl+Tab, Ctrl+F6	Next window
Ctrl+Shift+Tab	Previous window
Ctrl+F4	Close window

Visual C++ Compiling and Programming Support

Ctrl+Alt+T, Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Ctrl+Shift+Space, Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only)
Ctrl+Space	(Pro only) Complete symbol
F12, Ctrl+Dot	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Shift+F12, Ctrl+ /	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
F3	Find next reference

Visual C++ Keys

Shift+F3	Find previous reference
F4, Ctrl+Shift+Down	Next error
Shift+F4, Ctrl+Shift+Up	Previous error
F7	(Pro only) Build project
Shift+F10	(Pro only) Compile current buffer
Ctrl+Shift+G	Cursor to error/include file
Ctrl+F12	Make and load current macro buffer
Ctrl+Shift+M	(Pro only) Start concurrent process

Visual C++ Debugging (Pro only)

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+PadStar	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

Visual C++ Keys

Alt+7, Ctrl+Alt+C	Activate call stack
Ctrl+Alt+H	Activate threads window

Visual C++ Macros

Ctrl+Shift+R	Start/end macro recording
Ctrl+Shift+P	Terminate recording & run last recorded macro
Ctrl+F12	Make and load current buffer
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and run dialog box/macro.

Visual C++ Miscellaneous

F1	Help for mode or context
Ctrl+F1	Help on word at cursor
Alt+F4	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable
Esc	Cancel or command line toggle
Ctrl+Z, Alt+Backspace	Undo
Shift+F9	Undo with cursor motion grouping

Visual Studio Default Keys

Ctrl+Y, Ctrl+Shift+Z	Redo
Ctrl+Shift+H	Hex display toggle
Ctrl+0..Ctrl+9	Set bookmark 0..9
Alt+Shift+F2	Activate Bookmarks tool window
Ctrl+E, Ctrl+]	Match parenthesis
Ctrl+Shift+O	Expand alias at cursor
Ctrl+D	Activates search history
Alt+F5	Restore MDI window
Alt+F10	Maximize MDI window
Ctrl+Up	Scroll up
Ctrl+Down	Scroll down
Ctrl+Shift+T	Transpose words
Alt+Shift+T	Transpose lines

Visual Studio Default Keys

Visual Studio Cursor Movement

Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
Down arrow	Cursor down
Ctrl+Home	Top of buffer
Ctrl+End	Bottom of buffer
Home	Begin line

Visual Studio Default Keys

End	End line
PgUp	Page up
PgDn	Page down
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+PgUp	Top of window
Ctrl+PgDn	Bottom of window
Tab	Indent to next tab stop
Shift+Tab	Back indent text to previous tab stop
Ctrl+G	Go to line, offset, bookmark, error, definition, declaration, or reference
Ctrl+J	Previous preprocessing condition
Ctrl+Shift+\	Add multiple cursors
Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

Visual Studio Inserting Text

Ins	Insert/overwrite toggle
Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ctrl+Q	Quote next character typed

Visual Studio Deleting Text

Del	Delete char under cursor
Backspace	Delete char before cursor
Ctrl+L	Cut line

Visual Studio Searching

Ctrl+F	Find
Ctrl+H	Replace
F3	Find next occurrence
Shift+F3	Find previous occurrence
Ctrl+I	Incremental search
Ctrl+Shift+I	Reverse incremental search
Ctrl+Shift+F	Find in Files
Ctrl+Shift+H	Replace in Files
<Any Key>	Press any key to be prompted whether to cancel a foreground search/replace, or multi-file search/replace. To cancel a background multi-file search/replace, use the "Stop" button on the Find in Files tab of the Find and Replace tool window.

Visual Studio Selection

Ctrl+A	Select all
Ctrl+F8	Select line
F8	Select character/stream
Ctrl+Shift+J	Select previous preprocessing condition

Visual Studio Default Keys

Ctrl+Shift+K	Select next preprocessing condition
Alt+U	Deselect
Ctrl+X	Cut selection
Backspace, Del	Delete selection
Tab	Indent selection
Shift+Tab	Unindent selection
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Alt+=	Execute commands in selection
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Right-Click & Drag	Start block/column selection
Shift+Click	Extend selection
Ctrl+Right-Click	Move selection to cursor
Ctrl+Shift+Right-Click	Copy selection to cursor
Double-Click	Select word
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words

Visual Studio Clipboard

Ctrl+C, Ctrl+Ins	Copy selection to clipboard
------------------	-----------------------------

Visual Studio Default Keys

Ctrl+Shift+C	Append selection to clipboard
Ctrl+Shift+V	List clipboards and optionally paste one
Ctrl+V, Shift+Ins	Paste
Ctrl+L	Cut line
Ctrl+X, Shift+Del	Cut selection
Ctrl+Shift+X	Append cut selection
Alt+Shift+L	Cut sentence

Visual Studio Command Line and Text Box Editing

Ctrl+X	Cut selection
Ctrl+C	Copy selection to clipboard
Ctrl+V	Paste
Ctrl+Shift+C	Append selection to clipboard
Ctrl+Shift+X	Append cut selection
Ctrl+Shift+V	List clipboards and optionally paste one
Esc	Cursor to command line toggle
Space	Complete argument
?	List arguments
Ctrl+Left	Previous word
Ctrl+Right	Next word
Ctrl+Shift+U	Upcase word
Ctrl+U	Lowcase word
Ctrl+L	Cut line

Visual Studio Default Keys

Ctrl+Shift+L	Delete line
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character
Alt+N	Insert buffer name
Up arrow	Retrieve previous command
Down arrow	Retrieve next command
Shift+<Cursor keys>	Start or extend char/stream selection
Click & Drag	Start char/stream selection
Shift+Click	Extend selection
Double-Click	Select word
Triple-Click	Select line

Visual Studio Files and Buffers

F2, Ctrl+S	Save current buffer
Ctrl+O	Edit a file or find buffer
Ctrl+N	Open new file or create new project
Ctrl+Shift+B	List buffers
Ctrl+Shift+PadMinus	Next document. Next window if in one file per window mode (default). Otherwise, next buffer.
Ctrl+PadMinus	Previous document. Previous window if in one file per window mode (default). Otherwise, previous buffer.
F6	File compare

Visual Studio Windowing

--	--

Visual Studio Default Keys

Ctrl+Tab, Ctrl+F6	Next window
Ctrl+Shift+Tab	Previous window
Ctrl+F4	Close window
Alt+Shift+Enter	Fullscreen mode toggle

Visual Studio Compiling and Programming Support

Ctrl+Alt+T, Alt+Dot	(Pro only) List symbols
Ctrl+PgUp/Ctrl+PgDn	When listing symbols. Next/previous definition
Shift+PgUp/Shift+PgDn	When listing symbols. Page up/down argument list section.
Ctrl+Shift+Space, Alt+Comma	(Pro only) Parameter Info
Ctrl+Alt+Comma	(Pro only)
Ctrl+Space	(Pro only) Complete symbol
F12, Ctrl+Dot, Ctrl+F12	(Pro only) Push a bookmark and go to the definition of the symbol at cursor
Ctrl+Alt+Dot	(Pro only) Push a bookmark and go to the declaration of the symbol at cursor
Shift+F12, Ctrl+ /	(Pro only) Push a bookmark and go to the first reference to the symbol at cursor
Ctrl+Comma	Pop a pushed bookmark
F3	Find next reference
Shift+F3	Find previous reference
Ctrl+Alt+J	Activate Symbols tool window
F4, Ctrl+Shift+Down	Next error
Shift+F4, Ctrl+Shift+Up	Previous error
Ctrl+Shift+B	(Pro only) Build project

Visual Studio Default Keys

Shift+F10, Ctrl+F7	(Pro only) Compile current buffer
Ctrl+Shift+G, Alt+1	Cursor to error/include file
Ctrl+Alt+A, Ctrl+Shift+M	(Pro only) Activate Build tool window
Ctrl+Alt+L	Activate Projects tool window
Alt+Shift+A	Add current file to project
Ctrl+Shift+N	Create new workspace
Ctrl+Shift+O	Open workspace

Visual Studio Debugging (Pro only)

F5	Start/continue debugging
Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
F9, Ctrl+B	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+PadStar	Show next statement
Ctrl+Alt+B, Alt+F9	Activate breakpoints window
Alt+3	Activate watch window 1
Ctrl+Alt+W [1..4]	Activate watch window N
Ctrl+Alt+G	Activate registers tool window

Visual Studio Default Keys

Ctrl+Alt+M [1..4]	Activate memory window N
Alt+4, Ctrl+Alt+V A	Activate Autos variables window
Ctrl+Alt+V L	Activate Locals variables window
Ctrl+Alt+V T	Activate Members variables window
Alt+7, Ctrl+Alt+C	Activate call stack
Ctrl+Alt+H	Activate threads window

Visual Studio Macros

Ctrl+Shift+R	Start/end macro recording
Ctrl+Shift+P	Terminate recording & run last recorded macro
Ctrl+Shift+F12 <key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro that is prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Halt Slick-C macro that is executing. Use this to terminate infinite loops.
Ctrl+Shift+Space	If running a dialog box, edits current dialog box. Use this to close a dialog box that won't close. If editing dialog box or macro, load and run dialog box/macro.

Visual Studio Miscellaneous

F1	Help for mode or context
Ctrl+F1	Help on word at cursor
Alt+F4	Safe exit
Ctrl+Shift+Comma	Complete previous word/variable
Ctrl+Shift+Dot	Complete next word/variable

Visual Studio Default Keys

Esc	Cancel or command line toggle
Ctrl+Z, Alt+Backspace, Shift+Backspace	Undo
Ctrl+Y, Ctrl+Shift+Z, Alt+Shift+Backspace	Redo
Ctrl+0..Ctrl+9	Set bookmark 0..9
Ctrl+], Ctrl+Shift+]	Match parenthesis
Ctrl+D	Activates search history
Alt+F5	Restore MDI window
Alt+F10	Maximize MDI window
Ctrl+Up	Scroll up
Ctrl+Down	Scroll down
Ctrl+T	Transpose characters
Ctrl+Shift+T	Transpose words
Alt+Shift+T	Transpose lines
Ctrl+K Ctrl+N	Go to next bookmark
Ctrl+K Ctrl+P	Go to previous bookmark
Ctrl+K Ctrl+L	Clear all bookmarks
Ctrl+K Ctrl+K	Toggle bookmark
Ctrl+K Ctrl+W	Activate Bookmarks tool window
Ctrl+M Ctrl+P	Show all
Ctrl+M Ctrl+U	Show selection
Ctrl+M Ctrl+H	Hide selection
Ctrl+R Ctrl+W	Toggle viewing of whitespace

Xcode Keys

Ctrl+Alt+O	Activate Output tool window
Ctrl+Alt+R	Go to URL

Xcode Keys

Xcode Cursor Movement

Shift + Tab	Back indent text to previous tab stop
Command+Left arrow	Begin line
Command+Down arrow	Bottom of buffer
Ctrl+PgDn	Bottom of window
Down arrow	Cursor down
Left arrow	Cursor left
Right arrow	Cursor right
Up arrow	Cursor up
End	End line
Command+L	Go to line1
Tab	Indent to next tab stop
Alt+Right	Next word
PgDn	Page down
PgUp	Page up
Alt+Left	Previous word
Command+Down arrow	Top of buffer
Ctrl+PgUp	Top of window
Ctrl+	Add multiple cursors

Xcode Keys

Ctrl+Shift+Alt+Down	Add new cursor below current cursor
Ctrl+Shift+Alt+Up	Add new cursor above current cursor

Xcode Inserting Text

Enter	Insert a line
Ctrl+Enter	Open a new line below current line
Ctrl+Shift+Enter	Open a new line above current line
Shift+Enter	Insert a line (no syntax expansion)
Shift+Space	Insert a space (no syntax expansion)
Ins	Insert/overwrite toggle
Ctrl+Q	Quote next character types

Xcode Deleting Text

Ctrl+Backspace	Cut line
Ctrl+Shift+K	Cut word
Backspace	Delete char before cursor
Del	Delete char under cursor

Xcode Selection

Ctrl+Shift+Right+Click	Copy selection to cursor
Command+X	Cut selection
Backspace, Del	Delete selection
Ctrl+U	Deselect

Xcode Keys

Command+=	Execute commands in selection
Shift+Click	Extend selection
Tab	Indent selection
Ctrl+Right-Click	Move selection to cursor
Command+A	Select all
F8	Select character/stream
Ctrl+L	Select line
Triple-Click	Select line
Ctrl+Click	Add cursor or selection
Shift+Right-Click	Add multiple partial line characters selections based on a block/column selection.
Ctrl+Double-Click	Add selection of words
Double-Click	Select word
Shift+F7	Shift selection left
Shift+F8	Shift selection right
Right-Click & Drag	Start block/column selection
Click & Drag	Start char/stream selection
Shift+<Cursor keys>	Start or extend char/stream selection
Shift+Tab	Unindent selection

Xcode Searching

Command+F	Find
Alt+Command+F	Find in files
Command+G	Find next occurrence

Xcode Keys

Command+Shift+G	Find previous occurrence
Ctrl+R	Replace

Xcode Command Line and Text Box Editing

Ctrl+Shift+X	Append cut selection
Ctrl+Shift+C	Append selection to clipboard
Space	Complete argument
F3, Command+C	Copy selection to clipboard
Ctrl+K	Copy word to clipboard
Esc	Cursor to command line toggle
Ctrl+Backspace	Cut line
Command+X	Cut selection
Ctrl+Shift+K	Cut word
Shift+Click	Extend selection
Ins	Insert/overwrite toggle
?	List arguments
Ctrl+Shift+V	List clipboards, optionally paste one
Ctrl+Shift+L	Lowcase word
Alt+Right	Next word
Command+V	Paste
Alt+Left	Previous word
Ctrl+Q	Quote next character
Down arrow	Retrieve next command

Xcode Keys

Up arrow	Retrieve previous command
Triple-Click	Select line
Double-Click	Select word
Command+E	Set search string
Click & Drag	Start char/stream selection
Ctrl+Shift+U	Upcase word

Xcode Files and Buffers

F7, Command+O	Edit a file or find buffer
F6	File compare
Ctrl+Shift+B	List buffers
Alt+Command+Right arrow	Next Buffer
Command+N	New File
Alt+Command+Up arrow	Open associated file
Alt+Command+Left arrow	Previous buffer
F4	Save and quit current buffer
F2, Command+S	Save current buffer

Xcode Clipboard

Ctrl+Shift+X	Append cut selection
Ctrl+Shift+C	Append selection to clipboard
Command+C	Copy selection to clipboard
Ctrl+K	Copy word to clipboard

Xcode Keys

Ctrl+Backspace	Cut line
Command+X, Shift+Del	Cut selection
Ctrl+Shift+K	Select next condition
Ctrl+Shift+V	List clipboards, optionally paste one
Command+V	Paste

Xcode Macros

Ctrl+Shift+Space	Edit current dialog box if running a dialog box. Close dialog box that won't close. Load and run dialog box/macro if editing dialog box or macro.
Ctrl+Shift+F12,<key>	Stops macro recording and binds macro to <key> (which can be 0-9, A-Z, or F1-F12).
Ctrl+Break	Halt Slick-C® macro prompting for a key with get_event()
Ctrl+Alt+Shift+F2	Halt Slick-C macro that is executing. Terminate infinite loops.
F12	Make and load current macro buffer
Ctrl+F11	Start/end macro recording
Ctrl+F12	Terminate recording, run last recorded macro

Xcode Miscellaneous

Esc	Cancel or command line toggle
Ctrl+Shift+Space	Complete more
Ctrl+Shift+Dot	Complete next word/variable
Ctrl+Shift+	Complete previous word/variable
Ctrl+Shift+O	Expand alias at cursor
Command+D	Go to bookmark

Xcode Keys

Ctrl+Shift+D	Activate Bookmarks tool window
F1	Help for mode or context
Ctrl+Shift+H	Hex display toggle
Ctrl+Shift+L	Lowcase word
Ctrl-]	Match parenthesis
Command+F10	Maximize MDI window
Command+F7	Move MDI window
Command+Shift+Z	Redo
Command+F5	Restore MDI window
Command+Q	Safe exit
Command+F1	Help on word at cursor
Ctrl+0..Ctrl+9	Set bookmark 0..9
Alt+F8	Size MDI window
Command+Z, Alt+Backspace	Undo
Shift+F9	Undo with cursor motion grouping
Ctrl+Shift+U	Upcase word
Command+F7	Move MDI window

Slick-C® Macro Programming Guide

Note

Many of the Slick-C macro programming features described here are only available in the Pro edition and not the Standard or Community editions. The Standard and Community editions have very limited macro programming capabilities. These editions support limited macro recording and some configuration batch macros (like vusrdefs.e (UNIX: vunxdefs.e)) but not much else. If you want to do significant macro programming such as writing dialog boxes or sophisticated macros you can share with others, you need the Pro edition. Dialog boxes written in Slick-C will run on all platforms SlickEdit supports.

This guide contains the following topics:

- [Introduction](#)
- [Differences Between Slick-C® and C++](#)
- [Four Ways to Use Slick-C®](#)
- [Language Constructs](#)
- [Types](#)

-
- [Mathematical Operators](#)
 - [Declarations](#)
 - [Statements](#)
 - [Functions](#)
 - [Preprocessing](#)
 - [Defining Controls](#)
 - [Defining Events and Event Tables](#)
 - [Event-Driven Dialog Boxes](#)
 - [Module Initializations](#)
 - [Compiling and Loading Macros](#)
 - [Debugging Macros](#)
 - [Error Handling and the rc Variable](#)
 - [Dialog Editor](#)
 - [Creating Dialog Boxes](#)
 - [Clipboard Inheritance®](#)
 - [Objects and Instances](#)
 - [Using Functions as Methods](#)
 - [Built-in Controls](#)
 - [Menus](#)
 - [Common Macro Dialog Boxes](#)
 - [String Functions](#)
 - [Search Functions](#)
 - [Selection Functions](#)
 - [Writing Selection Filters](#)
 - [Unicode and SBCS or DBCS Macro Programming](#)
 - [Shelling Programs from a Slick-C® Macro](#)
 - [Interfacing With Other Languages \(DLL\)](#)
 - [Command Line Interface](#)
-

- [Hooking Startup and Exit](#)
- [State File Caching](#)
- [Windows Data Structure](#)
- [Tutorials](#)
- [Events](#)

Introduction

Slick-C® is a macro programming language that blends object-oriented features from C++, Java, and Python. Much of the code in the SlickEdit® editor is written in Slick-C, which covers many of the actions normally performed in a code editor including navigation and buffer modification. The Slick-C source is provided when SlickEdit is installed. You can use Slick-C to modify the look and feel of the editor, write macros to perform custom operations, add new language support, and essentially extend the editor's functionality until it is completely customized according to your preferences.

Working with the Slick-C® Source Code

After SlickEdit® is installed, the Slick-C macro files are located in the `macros` subdirectory of your installation directory.

Slick-C macros are stored in files ending in the `.e` extension. The Slick-C macro translator compiles these files to byte code which is saved in a corresponding file with the `.ex` extension.

Slick-C follows a C-style linking model with the distinction that macros can be loaded and reloaded dynamically. Compiled macros and dialog box templates are stored in the state file `vslick.sta`, which is located in your configuration directory.

Slick-C is preprocessed like C. Slick-C header files use the `.sh` extension. All Slick-C source files `#include slick.sh`.

Slick-C® Naming Conventions

The table below outlines Slick-C naming conventions.

Type	Example Name	Details
Namespaces	<code>se.example</code>	Lowercase, with an underscore or dot to separate multiple words.
Classes	<code>ExampleName</code>	Mixed case, first letter must be capitalized, all caps only acceptable for acronyms like "FTP".
Interfaces	<code>IExampleName</code>	Like class names, but with "I" prefix.
Enums	<code>ExampleName</code>	Like class names (idea of "E" prefix rejected).
Enum Flags	<code>ExampleFlags</code>	Like enums, but ends with "Flags"

**Differences Between Slick-C®
and C++**

Type	Example Name	Details
		(idea of "F" prefix rejected).
Member Funcs	exampleName	Mixed case with the first letter lowercase.
Member Vars	m_exampleName	Mixed case, first letter lowercase, and an "m_" prefix.
Properties	m_exampleName	Same as member variables (should not distinguish from var).
Class Vars	s_exampleName	Mixed case, first letter lowercase, and an "s_" prefix.
Namespace Vars	g_exampleName	Like member vars except with a "g_" prefix.
Namespace Funcs	example_name	Lowercase with words separated with underscores.
Global Vars	<anything>	No rules.
Global funcs	<anything>	No rules.
Typedefs	<anything>	No rules.

Differences Between Slick-C® and C++

Structures

- Space for structure member variables is allocated when you access the member.
- Structure data is not continuous. This is obvious for string, array, and hash table member variables that contain variable size data. However, even other types are sometimes stored elsewhere.
- There is not a **sizeof** function that tells you the size of a structure in bytes.

Arrays

- Space for array elements is allocated when you index into the array.
- You cannot use pointer variables to traverse array elements.
- You cannot limit the number of elements that the array may contain.
- Specifying an array variable without the [] operator does not return a pointer to the first element. Instead it refers to the entire array. This allows you to copy one array to another or define a function that returns a copy of an array.
- There is not a **sizeof** function that tells you the size of the array in bytes. There is a **_length** method that tells you the number of elements in the array.

Example:

```
struct PHONERECORD {  
    _str name;  
    _str PhoneNumber;  
};  
  
void defmain()  
{  
    PHONERECORD list[]; // No size limit is allowed here.  
  
    // Allocate space for 0 index and name member.  
    list[0].name=Joe;  
    // Allocate space for PhoneNumber member.  
    list[0].PhoneNumber="555-1234";  
  
    PHONERECORD list2[];  
    list2=list; // Copy the entire array into list2.  
    t=list2; // Now copy the entire array into a container variable.
```

}

Hash Tables

Slick-C® provides a `:[]` hash table operator that is similar to the array operator `[]` except that hash tables are indexed with a string type or by class objects. See [Hash Tables](#) for more information.

Assignment Statement

Assignment statements in Slick-C® are not as shallow as C++. Array, hash table, and structure types are recursively traversed. Pointers are not traversed.

Example:

```
struct {
    int a[];
} s1,s2;
s1.a[0]=1;
s2=s1; // Copy structure and all elements of array.
```

Comparison Operator

The `==` and `!=` operators support comparing container types, arrays, hash tables, and structures. Complex types are traversed recursively, like the assignment statement. Strings within an array, hash table, or struct must match exactly (spaces matter).

Preprocessing

Preprocessing expressions can use string and floating point expressions.

switch Statement

The `switch` statement supports string expressions and integer expressions.

enum

Slick-C® also supports `enum_flags` where bit flags are automatically generated. Specifying a type for `enum` (`enum myenum:long`) is not yet supported. Scoped enums (enum class or enum struct) are not yet supported.

Example:

```
enum COLOR {
    RED,
    YELLOW=4,
    GREEN,
}
enum_flags OptionFlags {
    FLAG1=0x4,
    FLAG2, // 0x8
    FLAG3, // 0x10
    FLAGS_ALL=FLAG1|FLAG2|FLAG3};
```

Const

The use of the **const** keyword is very different than C++. The **const** keyword is used to define global string or numeric constants. Currently **const** doesn't support a type (const int VALUE=4;) or making any variable read-only. **const** is intended as a better alternative to using a #define to define a constant.

```
const GL_MODE="GL";
const GL_VALUE= 1.4;
```

Labeled Loops

The **break** and **continue** statements accept an optional label parameter so that you can break a specific loop (like Java).

Example:

```
outerlabel:
for (;;) {
    for (;;) {
        if () break outerlabel;
        if () continue outerlabel;
    }
}
```

Variable Argument Functions

An **arg** function allows you to define functions that accept a variable number of arguments. The **arg**

function can be used on the left side of an assignment statement.

Example:

```
void defmain()
{
    p(Param1,2,x);
}
void p()
{
    messageNwait("Called with arg() arguments");
    for (i:=1;i<=arg();++i) {
        messageNwait("arg(\"i\")="arg(i));
    }
    // All undeclared variable parameters are passed by reference so when
    // a variable is passed, we can change the contents of the callers
variable.
    arg(3)=... // New value for x;
}
```

Built-in Graphics Primitives

You can define dialog box resources and menu resources. There are primitives for defining event handlers for dialog boxes and declaring control types. This allows the Slick-C® linker to detect a reference to a control that does not exist on a dialog box before you execute the code.

Clipboard Inheritance®

Clipboard Inheritance provides inheritance specifically for dialog boxes. This feature enables the copying of parts of existing dialog boxes to the clipboard and pasting them elsewhere, and the original code still runs. New code can be attached to the new controls without affecting the original controls, and to affect both instances of the controls (inheritance). Creating inheritance for parts of dialog boxes is very natural because the Slick-C® language has been designed for this feature. See [Clipboard Inheritance®](#) for more information.

End of Statement Semicolon

Slick-C® assumes that the end of line is a semicolon except under a few conditions. Expressions may extend across line boundaries if the line ends in a binary operator or if the line ends with a backslash, and expressions in parentheses may extend across line boundaries.

Type Checking

Capabilities not Supported by Slick-C®

Type checking in Slick-C® is identical to C++ except for the following:

- The **typeless** type is compatible with ALL other types.
- String constants are automatically converted to numeric types where necessary.
- Integer types are automatically converted to string types.
- Functions do not require prototypes. However, when a prototype is given, strict type checking is enforced like you would expect. A **#pragma** option to require prototypes will eventually be added.
- Classes and Interfaces are defined using a Java-style syntax.

Capabilities not Supported by Slick-C®

- Only one syntax is currently supported for making a call with a pointer to a function variable. The **pfn(p1,p2,)** syntax is not supported. This limitation is necessary for container variables because the compiler does not know the type of the variable.
- **char** and **short** types are not available.
- Template classes are not supported. Container variables are sometimes a more powerful mechanism for accomplishing much of what is done with template classes. However, container variables lack the speed and additional type checking of template classes.
- Function overloading is not supported.
- Slick-C only supports the less ambiguous C-style type casting.
- Because Slick-C does not allow low level manipulation of memory, you cannot do things like type cast an **int *** to a **long ***.
- There are no character constants defined using single quote characters. Slick-C currently allows the use of single quotes to define strings. Single quoted strings are much more readable for file names or regular expressions that require the use of backslashes.
- **goto** is not supported. (Slick-C supports labeled loops.)

Four Ways to Use Slick-C®

There are four ways to extend the SlickEdit® code editor using Slick-C:

- [Recording Slick-C® Macros](#)
- [Key Bindable Command](#)
- [Event-Driven Dialog Boxes](#)
- [Batch Macros](#)

Recording Slick-C® Macros

When using macro recording, Slick-C source code is created for a key bindable command. To create a recorded macro, complete the following steps:

1. From the main menu, select **Macro** → **Record Macro**.
2. Perform the actions that you want the macro to repeat.
3. When finished, select **Macro** → **Stop Recording**.

The macro is saved as Slick-C source code and you can edit the recorded macro through the user interface. Recorded macros are saved in the `vusrmacs.e` file in the user configuration directory.

Key Bindable Command

A key bindable command is the most common way to extend the editor. Command macros can be bound to keys or invoked from a menu. To create a Slick-C® command named **hello**, complete the following steps:

1. Place the macro code below into a new file named `test.e`:

```
_command void hello()
{
    message("Hello World");
}
```

2. With the file still open, press **F12** or use the **load** command to compile and load the macro. Or, from the main menu, click **Macro** → **Load Module**, then browse and select the macro to load.

Now you can type **hello** in the command line and the message **Hello World** is displayed.

The **hello** command can be bound to a key. To bind the hello command to **Alt +5**, complete the following steps:

1. From the main menu, click **Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**.
2. In the **Search by command** combo box, type **hello**.
3. Click **Add**.
4. Press **Alt+5**.
5. Click **Bind**.
6. Click **OK** on the Options dialog.
7. Now press **Alt+5**. The message **Hello World** is displayed.

Event-Driven Dialog Boxes

Slick-C® includes a dialog editor that allows you to create event-driven forms using a predefined set of controls.

This section describes:

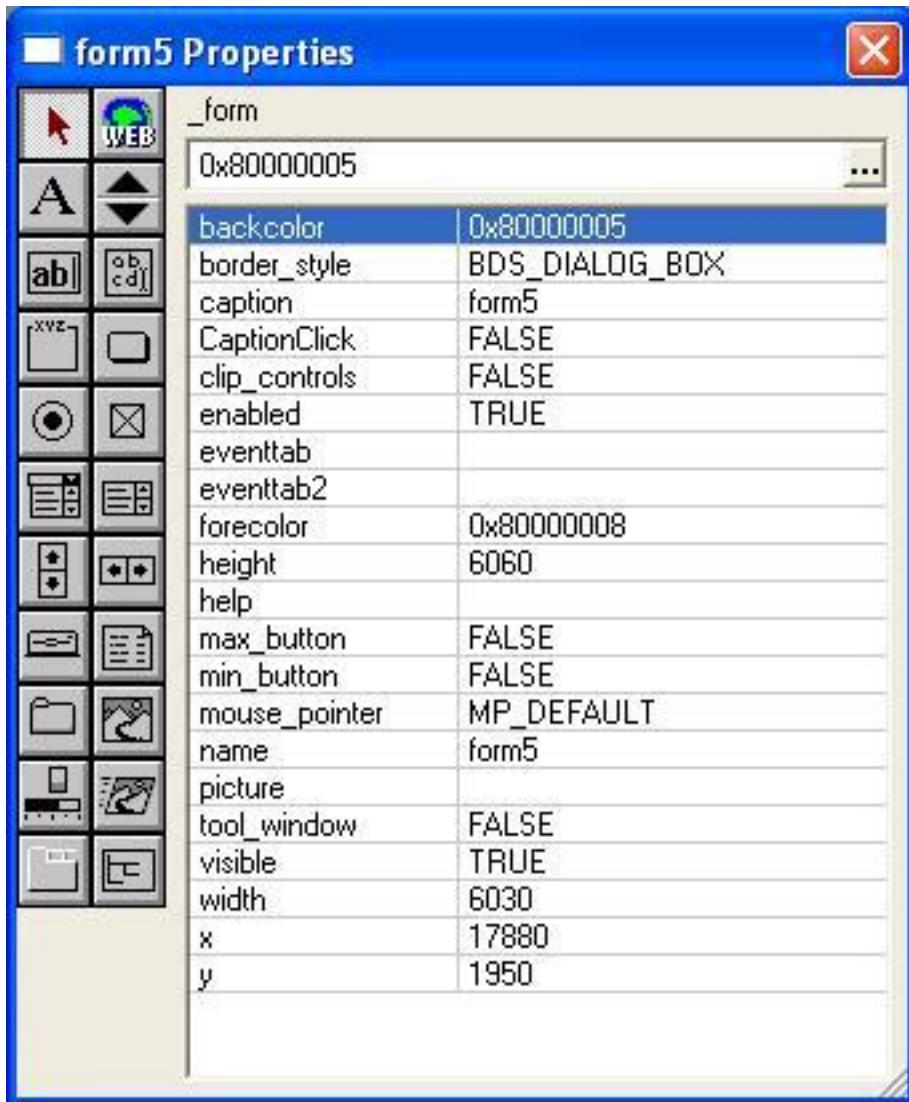
- [Creating a Simple Event-Driven Dialog Box](#)
- [Loading Code and Displaying Dialog Boxes](#)
- [Binding Commands to Keys for Dialog Box Display](#)

For more information, see also [Creating Dialog Boxes](#).

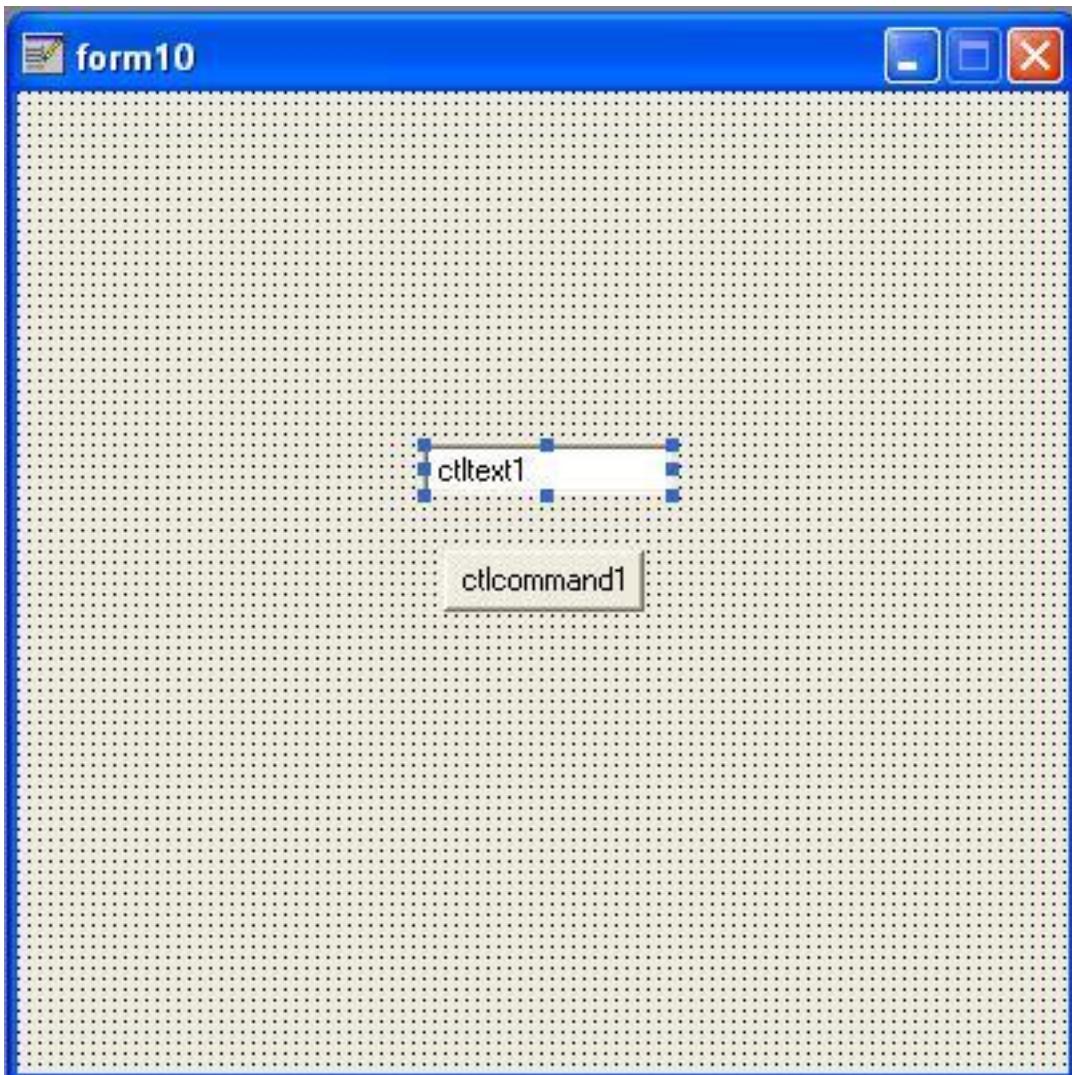
Creating a Simple Event-Driven Dialog Box

To create a simple event-driven dialog box, complete the following steps:

1. From the main menu, select **Macro** → **New Form**.
2. In the dialog editor Properties dialog box, double-click **Insert Button Control**.



3. Double-click **Insert Text Cox Control** in the dialog editor Properties dialog box.
4. Move the command button or the text box so that they do not overlap. Click on the object with the left mouse button, hold it, and drag to move the object.



5. Double-click on the command button that appears on the form (not the bitmap in the dialog editor Properties dialog box). The Select An Event dialog box appears with **lbutton_up** displayed in the combo box.
6. Press **Enter** to select the event.
7. The Open dialog box is displayed for a new file that is to contain the source code for this dialog box. Type `form1.e` and press **Enter**. A file is displayed named `form1.e` with the following lines of code:

```
#include "slick.sh"

defeventtab form1;
void ctlcommand1.lbutton_up()
{
}
```

8. If the previous lines of code are not displayed, then a `form1.e` file might already exist. If so, modify the existing `form1.e` file to contain the previous lines of code.
9. Modify the code to add the following statement: `ctltext1.p_text="Hello World";`

Example:

```
#include "slick.sh"

defeventtab form1;
void ctlcommand1.lbutton_up()
{
    // Set the p_text property of the text box control
    ctltext1.p_text="Hello World";
}
```

- 1 From the main menu, select **Macro → Load Module**.
- 0.

Loading Code and Displaying Dialog Boxes

To load the dialog box, and then display it, complete the following steps:

1. Right-click on the form and select **Load and Run Form**.
2. Click **ctlcommand1**. **Hello World** is displayed in the text box.
3. To close the Form1 dialog box, press the close button on the title bar of the window.
4. Type **show form1** to display this dialog box from the command line.
5. To display the dialog box modally, type **show -modal form1** on the command line.

The dialog source is saved in the `vuserdefs.e` file in the user configuration directory, `My SlickEdit Config`. Press **Ctrl+Shift+Space** while any dialog box is running to edit it (including the Properties dialog box).

Binding Commands to Keys for Dialog Box Display

To bind a command to a key that displays a dialog box, use the following example to write the necessary command:

```
#include "slick.sh"
_command void run_form1()
{
    show( "-modal form1" );
}
```

See [Key Bindable Command](#) for more information about binding a command to a key.

Batch Macros

Slick-C® allows you to write batch macros. Batch macros are macros that can be run, much like shell scripts, from within the editor. They do not need to be loaded, and they do not remain resident in the editor after they have been run.

Use a batch macro when working with Slick-C® primitives that you want to share among multiple users. Batch macros cannot be bound to a key; however, you can execute a batch macro from the command line or a menu item.

1. Open an empty buffer and type the following code:

```
#include "slick.sh"
void defmain()
{
    message("Hello World");
}
```

2. Save the file as `hellow.e`, then press **Esc** to open the SlickEdit® command line.

Note

To be able to run a batch macro without specifying the full path, save the file in a directory listed in the **VSLICKPATH** environment variable. Otherwise, you will need to include an absolute or relative (to the current directory) path to run your batch macro. For more information, see [Environment Variables](#).

3. Type **xcom hellow**, and press **Enter**.

Note

You can type **hellow** instead of **xcom hellow** to run the batch macro as long as internal command doesn't have the same name. On non-Windows platforms, it's best to always prefix external batch macros with "xcom" because a leading "/" will be interpreted as a find command and not a path separator.

4. The status line displays the message **Hello World** is displayed.

Batch programs must be saved before they are executed so that the macro can compile. Also, batch programs are automatically compiled if there is no corresponding `.ex` file, or if the date of the source file is newer than the date of the `.ex` file.

Language Constructs

The Slick-C® language is rooted in the C language. Slick-C contains some constructs from REXX and a dialog system usually found only in languages such as Microsoft® Visual Basic®. Slick-C also blends in object-oriented features from C++, Java, and Python.

Topics in this section:

- [Identifiers](#)
- [Comments](#)
- [String Literals](#)
- [Numeric Literals](#)
- [Defining Constants](#)
- [Namespaces](#)

Identifiers

A variable or identifier may contain any of the characters "A-Za-z\$_0-9" and must start with one of the characters "A-Za-z_\$".

See [Reserved Words and Keywords](#) for a list of reserved words and keywords.

All identifiers starting with **p_** are reserved to be used as Slick-C property names.

Comments

Slick-C® supports both of the C++ comment styles.

- Use **//** to declare that the rest of the line is a comment
- Use **/*** to open a block comment and ***/** to close a block comment.
- Block comments can be nested.

Example:

```
i=1; //this is a comment
/* this is a /* nested */ comment */
```

String Literals

Strings can be surrounded with single or double quotes. Double-quoted strings are identical to C++ string literals.

A backslash followed by a character has special meaning, as outlined in the table below.

Characters	Meaning
\a	Bell character (7)
\b	Backspace character (8)
\f	Form feed character (12)
\n	New line character (10)
\r	Carriage return (13)
\t	Tab character (9)
\v	Vertical tab character (11)
\?	Question mark character
'	Single quote character
"	Double quote character
\	Backslash character
\x hh	Hexadecimal character code
\ooo	Octal character code

If single quotes are used, two single quotes consecutively represent one single quote character. If double quotes are used, a backslash followed by a double quote represents one double quote character. The operator `==` used in the example below compares two strings for exact equality. The Slick-C® language does have an operator `==`. However, this operator strips leading and trailing spaces and tabs from both operands.

Examples:

```
"abc" == 'abc'  
"Can't find file" == 'Can''t find file'  
"\t" == _chr(9)  
\t == _chr(9)
```

```
" spaces " == "spaces"
```

A backslash (not inside quotation marks) followed by a character or a number has the special meaning, as shown in the table below.

Characters	Meaning
\a	Bell character (7)
\b	Backspace character (8)
\f	Form feed character(12)
\n	New line (10)
\r	Carriage return (13)
\t	Tab character (9)
\v	Vertical tab character (11)
\x dd	Hexadecimal character code dd
\ ddd	Decimal character code ddd

Caution

Using the above feature is not recommended. Use a quoted string.

Numeric Literals

The Slick-C® language supports integer constants in both decimal and hexadecimal formats. Hexadecimal numbers are defined using **0x[hexdigits]** just like they are in C.

The Slick-C language supports floating point numbers. The mantissa is limited to 32 digits and the exponent is limited to nine digits. When precision is lost, the result is rounded. Overflow and underflow are detected. Floating point numbers have the following syntax:

[+ | -] digits [.][digits][E[+ | -] digits]

or

[+ | -] [.][digits][E[+ | -] digits]

There may be blank spaces before and after the leading sign.

Example:

```
4.04  
4e2  
4e2  
4E-2  
4E-2
```

Defining Constants

There are multiple ways to define constants.

Defining Constants Using #define

Slick-C® supports the **#define** preprocessor directive. The **#define** directive is for defining constants or in-line functions. Use the following syntax to define the constant or in-line function:

```
#define name[ (param1,param2,) ] value
```

Use a backslash at the end of a line to indicate that the *value* text continues to the next line. Any occurrence of *name* is replaced with the text *value* before the source is compiled.

Caution

When *value* represents an expression, place parentheses around it to make sure that there is not a problem with operator precedence.

Example:

```
#define MAXLINES      15  
#define MAXLINESP1    (MAXLINES+1)  
#define max(a,b)      (((a) >= (b)) ? (a) : (b))  
#define min(a,b)      (((a) <= (b)) ? (a) : (b))  
  
defmain() {  
    x=MAXLINES;  
    y=MAXLINESP1;  
    a=max(x,y);  
}
```

Defining Constants Using `const`

A **const** declaration is used to define a constant. A constant can be scoped within a class, namespace, or globally. The advantage to using **const** instead of **#define** is that **const** constants are pure constants and can be introspected using **find_index()** and **name_info()**, but a **#define** is just a lexical substitution. For consistency, we recommend that constants use all uppercase identifiers, optionally using underscores to separate words.

Examples:

```
const MAXLINES = 15;
const MAXLINESP1 = (MAXLINES+1);
const SEARCHKEY = "<Search>";
static const MY_LIMIT = INFINITY;
```

A few notes about constants:

- The **const** declaration works with Slick-C classes and namespaces.
- Constants support type inference so that the compiler can tell ints from booleans from strings.
- Constant names and values are stored in the state file in order to allow introspection.
- Static constant names are local to the current module, and are not stored in the state file (and do not allow introspection).
- Context Tagging® recognizes the **const** declaration correctly.

Defining Constants Using Enumerators

Slick-C® also allows you to define constants using enumerators. Enumerated types share the advantages of **const** declarations. They are discussed in more detail in the section [Enumerated Types](#).

Namespaces

Slick-C® supports namespaces which allow you to partition functions and classes into independent areas in order to organize your code better, and to better isolate your code from name conflicts. Slick-C namespaces use `".` instead of `::`. Slick-C does not allow un-named namespace declarations. Slick-C supports two types of namespace declarations, as shown in the following code:

```
// Module-wide (like Java).
namespace slickedit.tagging;

// Scoped namespace declaration (like C++ and C#).
namespace slickedit.search {
    . . .
```

}

Namespace imports use the C++-style **using** syntax:

```
// Pull all symbols from slickedit.tagging into current scope.  
using namespace slickedit.tagging;  
  
// Pull one symbol from slickedit.search into scope.  
using slickedit.search.Regex;  
  
// Qualified access to a symbol in the namespace.  
slickedit.diff.Diff( f1, f2 );
```

Slick-C includes the **default;** namespace, which will return you to the "default" global namespace.

Types

Slick-C® types are similar to the types in C. The following types are available in Slick-C:

- [Strings](#) - Slick-C has a built-in string type `_str`.
- [Enumerated Types](#) - Slick-C supports C-style enumerated types. In addition, Slick-C supports bit-flag style enumerated types (`enum_flags`).
- [Arrays](#) - Array types are declared like C arrays, but cannot have a size limit. Array elements are always dynamically allocated.
- [Hash Tables](#) - Slick-C provides a `:[]` hash table operator which is similar to the array operator `[]`, except that hash tables are indexed with a string type.
- [Structs](#) and [Unions](#) - Slick-C supports C-style structs and unions. Static structure members are not supported.
- [Classes](#) and [Interfaces](#) - Slick-C supports Java-style classes and interfaces. Static class member variables are not supported.
- [Pointers](#) - Slick-C provides pointer and reference types in the same manner as in the C language.
- [Typeless](#) - Typeless variables are declared using the `typeless` type. A typeless variable can be assigned to or from any type, including structs, arrays, and hash tables.
- **Numeric types** - The numeric types are `int`, `long`, and `double`. All numeric types are signed. Slick-C does not support `char`, `short`, or `float` types.
- **Boolean type** - The built-in Slick-C Boolean type is `bool`.
- **Void type** - `void` is only permitted as the return type of a function.
- **Typedefs** - Slick-C supports C-style `typedef` type declaration statements.

Strings

String variables are declared using the `_str` type. You can get the length of the string using the `length` built-in.

Slick-C® has additional string operators so that the compiler always knows whether to perform a string or numeric operation. The `+` operator always means add two numbers, and the concatenation operator `:+` always means concatenate two strings. The `:=+` operator can be used to append to the end of a string.

See also [Implicit Conversion to Strings](#).

Enumerated Types

Slick-C® enumerated types are very much like C enumerated types, with the exception of also having a form of enumerated type for bit flags. By default, Slick-C enumerated types have very relaxed type checking with respect to assignment, arithmetic, and bit operations. However, when strict error checking is enabled (see [strictenums](#)), the type checking is more like C++11 with respect to assignment, arithmetic, and bit operators.

```
enum BasicOptions {
    OPTION1=1,
    OPTION2,
    OPTION3,
};
```

Slick-C® enumerated types support operators `++` and `--`, making it easy to iterate over a set of enumerators using a for loop.

In addition, Slick-C enumerated types introduce enumerated type flags, a convenient way to create a set of bit flags.

```
enum_flags OptionFlags {
    FLAG1=0x4,
    FLAG2, // 0x8
    FLAG3, // 0x10
    FLAGS_ALL=FLAG1 | FLAG2 | FLAG3
};
```

Slick-C® enumerated type flags support all bitwise arithmetic operations, as well as boolean `!` and non-zero test.

Arrays

Array types are declared like C arrays, but cannot have a size limit. Array elements are always dynamically allocated.

Use array variables to keep a list of items. To define an array variable, use the following syntax:

```
[static] TypeName variable1[][{e1 ,e2 , ...}] , variable2[][{e1 ,e2 , ...}] ...;
```

The first element of an array starts at 0. Use more than one set of brackets (`[]`) for multi-dimensional arrays. Do not define the maximum number of elements in the array, because array elements are allocated when you access them. The maximum number of elements that can be placed in an array is approximately 2 billion. Use the `_length()` method to determine the number of elements in an array. The syntax for using this method is `variable._length()`.

To empty an array, use the following statements:

```
array._makeempty();           // Empty the array.  
array=null;                  // Empty the array. Same as above.
```

You can delete and insert items into an array using the **_deleteI()** and **_insertI()** built-in methods, respectively.

A Slick-C® class instance can be indexed using array syntax provided that the class implements the **sc.lang.IIndexable** interface. This is similar to overloading operator [] in C++.

Differences from C++

- Space for array elements is allocated when you index into the array.
- You cannot use pointer variables to traverse array elements.
- You cannot limit the number of elements that the array may contain.
- Specifying an array variable WITHOUT the [] operator does not return a pointer to the first element. Instead, it refers to the entire array. This allows you to copy one array to another, or define a function which returns a copy of an array.
- There is no **sizeof** function which tells you the size of the array in bytes. There is a **_length** method which tells you the number of elements in the array.
- You can append an element to the end of an array using the built-in **:+=** operator.
- Array initializers are not supported for local variables.

Example:

```
int gai[]={1, 7, 12};  
int gaai[][]={{1},{1,2},{1,2,3}}; // Two dimensional array.  
_str    gastring1[]{"Value1", "Value2"};  
typeless  gat[]{"String", 1, 2.4};  
  
void defmain()  
{  
    t=gai;           // Copy all the array elements  
                    // into a local container variable.  
    t[t._length()]=45; // Add another array element.  
    t :+= 46;         // Add another array element (shorthand).  
    for (i:=0;i<t._length();++i) {  
        messageNwait("t["i"]="t[i]");  
    }  
}
```

Hash Tables

Hash tables are declared similar to array types and indexed with a string :[] operator. Use the following syntax to define a hash table variable:

```
[static] TypeName variable1:[] [= {s1=>e1, s2=>e2, ...}],  
variable2:[] [= {s1=>e1, s2=>e2, ...}] ...;
```

You can delete an item from a hash table using `_delete()`. Hash table initializers are not supported for local variables.

Hash tables support indexing by class objects. The class must implement the `getHashKey()` member of the **IHashable** interface. For example:

```
#include "slick.sh"  
#import "stdprocs.e"  
#import "sc/lang/IHashable.e"  
  
class FileName : sc.lang.IHashable {  
    private _str m_file;  
    _str getHashKey() {  
        return m_file;  
    }  
    _str getExtension() {  
        return get_extension(m_file);  
    }  
    _str getPath() {  
        return _strip_filename(m_file,'N');  
    }  
    _str getFileName() {  
        return _strip_filename(m_file,'P');  
    }  
    void makeAbsolute(_str toDir=null) {  
        m_file = absolute(m_file, toDir);  
    }  
    _str getRelative(_str toDir) {  
        return relative(m_file,toDir);  
    }  
    FileName(_str fname=null) {  
        m_file = fname;  
    }  
    _str get() {  
        return m_file;  
    }  
    void set(_str fname) {
```

```
        m_file = fname;
    }
};

void defmain() {
    bool ht[];
    FileName a("C:\\temp\\test.txt");
    FileName b("C:\\Program Files\\");
    FileName c("F:\\Public\\xkcd108.jpg");
    ht:[a] = true;
    ht:[b] = false;
    ht:[c] = true;
    FileName i;
    foreach ( i => auto v in ht ) {
        _assert(i instanceof FileName);
        say("i="i.getHashKey() " v="v);
    }
}
```

A Slick-C® class instance can be indexed using hash table syntax provided that the class implements the **_hash_el(_str key)** function of the **sc.lang.IHashIndexable** interface. This is somewhat similar to overloading operator **[]** in C++. See [Overloading Array Index Operators](#) for an example of using **IHashIndexable**.

Structs

Structures (structs) are typically used to logically group data. For example, a record in a database might have a name, address, and phone number. This can be logically grouped into a ContactInfo structure which is more convenient to use than accessing the fields individually. Structures can also have the added effect of reducing the number of global variables.

Slick-C® supports C-style structs. Slick-C structs cannot have member functions.

For consistency, we recommend that structs use initial caps (camel case) identifiers. Use the following syntax for defining a struct:

```
[static] struct StructName {
    member-variable-decl1;
    member-variable-decl2;
} ([variable1[={e1,e2, ...}], variable2[={e1,e2, ...}], ...);
```

The **struct** declaration provides the option of defining your own type called *StructName* and to declare one or more variables. The syntax of *member-variable-decls* is identical to declaring other variables, except that static structure members are not supported. Use the following syntax for accessing a member

of a struct variable:

```
variable.member_name
```

Example:

```
struct PHONERECORD {      // Define a type called PHONERECORD.
    _str Name;
    _str PhoneNumber;
} gPhoneRecord;           // Declare a variable of that type.

PHONERECORD gPR={        // Declare a variable of type PHONERECORD.
    "Steve", "555-1346"
};

PHONERECORD gRecordArray[]; // See arrays below.
struct PHONERECORD2 {     // Define a type called PHONERECORD2.
    _str Name;
    _str PhoneNumber;
    _str FaxNumber;
};

void defmain()
{
    messageNwait("Name=gPR.Name" PhoneNumber=gPR.PhoneNumber);
    typeless t = gPR; // Copy phone record data into a local container
variable.          // Container variables can access structure
elements          // as an array.
    messageNwait("Name=t[0]" PhoneNumber=t[1]);
}
```

Slick-C structs support designated initializers:

```
struct PhoneRecord {
    _str name;
    _str phoneNumber;
};

PhoneRecord shouldHaveKnown = {
    .phoneNumber = "867-5309",
    .name="Jenny"
};
```

Differences from C++

- There is no **sizeof** operator like in C++. Since the Slick-C® interpreter stores all types as container variables, the **sizeof** operator has no meaning.
- Space for structure elements is allocated when you access the element.
- Structure data is not contiguous. The Slick-C interpreter stores all types as container variables, including the members of a struct.

Unions

Slick-C® supports C-style unions. Unions are typically used in place of a struct in the case where you have mutually exclusive member variables. In this case, a union requires less memory than a struct. Memory is only allocated for one member variable at a time. The syntax for defining a union is shown in the following example:

```
[static] union [UnionName] {  
    member-variable-decl1;  
    member-variable-decl2;  
} [variable1[={e1}] , variable2[={e1}], ...];
```

The **union** declaration provides the option to define your own type named *UnionName* and to declare one or more variables. The syntax of *member-variable-decls* is identical to declaring other variables, except that static union members are not supported. The syntax for accessing a member union variable is *variable.member_name*.

Example:

```
union {  
    int i;  
    _str s;  
    double d;  
} gu={1};      // Type checking here is with first member variable.  
  
#define KIND_INT 1  
#define KIND_STRING 2  
#define KIND_DOUBLE 3  
void defmain()  
{  
    struct {  
        int kind;  
        // Here we are nesting a union inside a struct.  
        // This union only requires space for one of these members at a  
time.  
        union {
```

```
    int i;
    _str s;
    double d;
} u;
} x;

x.kind=KIND_INT;
x.u.i=1;
...
switch (x.kind) {
case KIND_INT:
    messageNwait("x.u.i=x.u.i");
    break;
case KIND_STRING:
    messageNwait("x.u.s=x.u.s");
    break;
case KIND_DOUBLE:
    messageNwait("x.u.d=x.u.d");
    break;
}
}
```

Anonymous Unions

An anonymous union is a union member variable that is not named. This saves you from having to type the union member variable name.

Example:

```
void defmain()
{
    struct {
        int kind;
        union {
            int i;
            _str s;
            double d;
        }; // No name for this union member variable.
    } x;
    x.kind=KIND_INT;
    x.i=1;
}
```

Interfaces

Interfaces use Java-like syntax. They do not allow constructors, destructors, or member variables; only prototypes. Interfaces can inherit from other interfaces. All the prototypes in an interface are implicitly public.

Example:

```
interface ICommunicationDevice {  
    void talk();  
    void hangup();  
};
```

Classes

Classes use a Java-like syntax. For example:

```
class Phone : ICommunicationDevice {  
  
    protected typeless m_dialer = null;  
    private typeless m_line = null;  
    private static typeless s_operator = null;  
  
    Phone(_str number="") {  
    }  
    ~Phone() { }  
    void talk() {  
    }  
    void hangup() {  
    }  
    static void getOperator() {  
    }  
};
```

For consistency, Slick-C® class names should be in camel case. Member variables within classes should start with "m_". Static member variables should start with "s_". Finally, methods should be lowercase. If a method name contains multiple words, the trailing words should be camel case.

A few notes about Slick-C classes:

- A class can extend or inherit from only one other class.
- A class can implement multiple interfaces.
- Use the **instanceof** operator to test if a class instance derives from a specific class or interface.
- Member variables can have constant initializer expressions.

- All member variables must be initialized, either using initializers or in the class constructor.
- All member variables must be declared before the constructor.
- There are no **extends** or **implements** keywords.
- Classes are not allowed to derive from **struct** types.
- The default access level is public. There is a **public** keyword, but it essentially does nothing.
- Class members support **protected** and **private**.
- There is no concept of a package scope like there is in Java.
- Member functions are virtual by default, except for **static** member functions.
- **static** member variables may have initializers.
- **extern** member function prototypes are implemented in a DLL.
- A class is allowed one and only one constructor.
- If a class constructor takes arguments, they must have defaults.
- No explicit calls to **new** or **delete** (no **new** or **delete** keywords).
- No function overloading.
- No operator overloading.
- No friend relationships.
- No templates or generics.
- No final and no **const**.
- No C#-style properties or delegates.
- No default root "object" class.
- No static constructors.

The life-span of a Slick-C class instance is identical to that of a similar Slick-C struct. There are no **new** or **delete** operators.

```
// Construct an instance of a class, like C++.  
C1 a;  
C1 b;  
  
// Assign a class instance to another (deep copy).  
a = b;
```

```
// An array of class instances. Constructor not called here.  
C1 array[ ];  
// Constructor called with no args followed by deep copy.  
array[1] = a;
```

See the following topics in this section for more information:

- [Introspection](#)
- [Implicit Conversion to Strings](#)
- [Overloading Comparison and Assignment Operators](#)
- [Overloading Array Index Operators](#)

Introspection

Slick-C® supports introspection of **struct** and **class** instances through the built-in functions shown below. In each of the functions, "index" can be either an integer index or a string containing the field or method name.

- **v._callmethod(index)** - Call a class method.
- **v._construct()** - Construct an instance of a class.
- **v._fieldindex(name)** - Find the position of a class field.
- **v._fieldname(i)** - Get the name of a class field.
- **v._findmethod(name)** - Find a class method.
- **v._getfield(index)** - Get a reference to a class field.
- **v._instanceof(name)** - Return true if variable is instance of or derives from the given class.
- **v._length** - Return the number of fields in a class.
- **v._setfield(index,value)** - Modify a class field.
- **v._typename()** - Return the name of variables type.

The C++ API for Slick-C includes the following functions:

- **vsHvarTypename(hvar)**
- **vsHvarFieldIndex(hvar,name)**
- **vsHvarFieldName(hvar,i)**
- **vsHvarGetField(hvar,index)**
- **vsHvarGetFieldByName(hvar,name)**

- `vsHvarSetField(hvar,index,value)`
- `vsHvarSetFieldByName(hvar,name,value)`
- `vsHvarFindMethod(hvar,name)`
- `vsHvarCallMethod(hvar,index,args)`
- `vsHvarCallMethodByName(hvar,name,args)`
- `vsHvarInstanceOf(hvar,name)`
- `vsHvarConstruct(name,args)`

Implicit Conversion to Strings

Slick-C provides the interface **IToString** for implicit string conversion (see [Strings](#)). If a class implements **sc.lang.IToString**, then an instance of that class can be implicitly converted to a string, without explicitly calling the `toString()` method.

Overloading Comparison and Assignment Operators

By default, Slick-C® class instances are compared using a deep member-wise equality test. To override the default comparison methods for a class, Slick-C provides the interfaces **sc.lang.IEquals** and **sc.lang.IComparable**. If a class implements **sc.lang.IEquals**, an instance of that class can be compared to another instance using operator `==` or operator `!=` as defined by the `equals()` method. If a class implements **sc.lang.IComparable**, then instances of the class can be compared using the standard comparison operators, as defined by the `compare()` method. If a class implements **IComparable**, it does not have to implement **IEquals** to support equality and inequality tests.

Overloading Array Index Operators

Slick-C® supports the overloading of the `[]` and `:[]` operators. For more information, see [Hash Tables](#).

Below is an example of **IIndexable**:

```
#import "sc/lang/IIndexable.e"

class PerfectSquares : sc.lang.IIndexable {
    typeless _array_el(int i) {
        return i*i;
    }
}

void defmain()
{
    PerfectSquares ps;
    say("defmain: 3^2="ps[3]);
    say("defmain: 16^2="ps[16]);
```

```
}
```

Below is an example of **IHashIndexable**:

```
class PhoneBook : sc.lang.IHashIndexable {
    _str m_numbers: [];
    void loadNumbers() {
        m_numbers: ["Brittany"] = "555-3825";
        m_numbers: ["Vanessa"] = "555-1024";
    }
    typeless _hash_el(_str name) {
        return m_numbers:[name];
    }
}
void defmain()
{
    PhoneBook pb;
    pb.loadNumbers();
    say("defmain: Brittany's number is " pb:["Brittany"]);
    say("defmain: Vanessa's number is " pb:["Vanessa"]);
}
```

Overloading Assignment/Copy Semantics

By default, Slick-C® class instances are copied using a deep, member-wise copy. To override this behavior, a class can implement the **sc.lang.IAssignTo** interface and implement a custom **copy()** method.

Overloading Iteration Semantics

A Slick-C® class can be customized to work seamlessly in a **foreach** loop by implementing the **sc.lang.IIterable** interface. The **sc.lang.Range** class, which is included in the Slick-C class library, is an excellent example of how to implement and use **Iterable**.

SlickEdit® Class Libraries

SlickEdit ships with a small but growing core of Slick-C® classes and interfaces to build upon. There are two top-level namespaces: **sc** (Slick-C) and **se** (SlickEdit). The **sc** namespace encompasses general purpose classes that support programming in Slick-C and are application-independent. It can be compared to **java.lang** and **java.util** in Java, the **System** namespace in C#, or the **std** namespace in C++ with respect to its purpose (not feature-by-feature). The **se** namespace includes the foundations and implementations of select features of the SlickEdit editor. Not all SlickEdit features use Slick-C classes.

Differences from C++ and Java

- Slick-C® uses per-member access specifiers like Java rather than the grouping syntax employed by C++.

- Slick-C supports destructors, just like C++ (Java does not have destructors).
- Slick-C has no **new** or **delete**.
- Slick-C does not support overloaded methods or **const** methods.
- Like C++, Slick-C class instances are passed by value, unless you specifically pass them by pointer or reference.
- Like Java, **this** is a reference to the current class instance, not a pointer as it is in C++.

Additionally:

- No function overloading.
- No operator overloading.
- No friend relationships.
- No templates or generics.
- No final and no **const**.
- No C#-style properties or delegates.
- No default root "object" class.
- No static constructors.

Pointers

Pointers to Variables

Pointer variables are declared using the following syntax:

```
[static] TypeName *variable1[=&v1] , *variable2[=&v2] ...;
```

The unary **&** operator is used to return the address of a variable. The unary ***** operator is used to dereference a pointer. Use the operator **->** (for example, **p->m-variable**) to access members of a pointer to a structure.

Caution

When a module is reloaded, static variable addresses change. Make sure you reinitialize global pointer variables which point to static (module scope) variables.

Pointers to Functions

Function pointer variables are useful for callback functions. The syntax for function pointers is:

```
[static] TypeName (*variable1)([ArgDecl1, ArgDecl2,...]) {=function_name};
```

Where *ArgDecl* has the almost the same syntax as a variable declarations, except **static** is not supported and the ampersand (**&**) operator is used to specify call by reference parameters. Call by reference array and hash table parameters require parentheses around the ampersand (**&**) and *id*.

The syntax for calling a pointer to function variable is:

```
(*pfn)([e1, e2,...])
```

If accessing an invalid function pointer, the Slick-C® macro stops.

Caution

When a module is reloaded, static function addresses change. Make sure you reinitialize global function pointer variables which point to static (module scope) functions.

Typeless

A typeless variable can be assigned to or from any type, including structs, arrays, and hash tables.

Typeless container variables can be declared using the **typeless** type. A typeless container can be passed to a function using the **var** type. The container variable can store the contents of any typed variable. This is easy for the interpreter since all typed variables are stored as container variables. At run time, the interpreter must check the current type of the container variable (and sometimes convert it) to perform an operation.

The compiler performs (double) floating point arithmetic on container variables. Currently, there is only a very small difference in speed between arithmetic operations on integer type variables and container variables, because the Slick-C® language has been optimized for string and container operations.

Note that there is no **sizeof** operator. Since the Slick-C interpreter currently stores all types as container variables, the **sizeof** operator has no meaning.

Example:

```
typeless t;
t=1;    // Store an integer.
        // Convert the contents of the variable t to a
        // floating point number (double type) and add 1.
        // NOTE: The interpreter is smart and will only perform
        // integer arithmetic here.
t=t+1;
// Since + always means addition, the compiler converts
```

```
// string constants to the smallest possible numeric type.  
t=t+"1";  
  
// Declare string variable.  
_str s;  
s=1; // Compiler will convert int to string.  
t=1.2;  
// Must cast string type to int or compiler will complain.  
t=(int)t+(int)s; // Result is 2, not 2.2, because of the cast of t to  
int.  
  
// Destroy the integer and make an array.  
// Also make the 0 element an integer.  
t[0]=1;  
t[1]=2; // Add another element.  
  
t2=t; // Copy the array and all its elements.  
struct {  
    int x;  
    int y;  
} st;  
st.x=1;  
st.y=2;  
t=st;  
// Print out the elements of the structure.  
for (i:=0;i<t._length();++i) {  
    messageNwait("t["i"]=""t[i]);  
}
```

Mathematical Operators

Slick-C® uses the operator precedence of C. The table below contains the unary operators that an expression can use.

Operator	Description
<code>! e1</code>	Logical NOT. Result is true if <code>e1</code> evaluates to false or 0 . Otherwise the result is true .
<code>~ e1</code>	Bitwise complement.
<code>- e1</code>	Negation.
<code>+ e1</code>	No change.
<code>++ v1</code>	Increments the variable <code>v1</code> and returns the result.
<code>v1 ++</code>	Returns the value of <code>v1</code> and then increments the variable <code>v1</code> .
<code>-- v1</code>	Decrements the variable <code>v1</code> and returns the result.
<code>v1 --</code>	Returns the value of <code>v1</code> and then decrements the variable <code>v1</code> .

The binary and ternary operators for the Slick-C language are listed in the table below. In addition to the operators listed in the previous table, string concatenation is implied. If a binary operator does not exist between two unary expressions, concatenation is automatically performed.

All numeric operators, except bitwise operators, support floating point numbers. Bitwise operators support 32-bit integers and 64-bit long integers for all platforms. Bitwise operators also support enumerated flag types (**enum_flags**).

Operator	Description
<code>=</code>	Assign right operand to left operand.
<code>:=</code>	Declare new variable with type matching right operand and assign it the value of the right operand.
<code>+=</code>	Add left operand to right operand and assign to left operand.

Mathematical Operators

Operator	Description
-=	Subtract right operand from left operand and assign to left operand.
/=	Divide left operand by right operand and assign to left operand.
*=	Multiply left operand with right operand and assign to left operand.
=	Bitwise OR left operand with right operand and assign to left operand.
^=	Bitwise XOR left operand with right operand and assign to left operand.
&=	Bitwise AND left operand with right operand and assign to left operand.
e1 ? e2 : e3	If expression e1 is TRUE (not the string 0), expression e2 is returned. Otherwise, expression e3 is returned.
&&	Logical AND. If left hand expression is false, right-hand expression is not evaluated.
	Logical OR. If left hand expression is true, right-hand expression is not evaluated.
	Bitwise OR.
^	Bitwise XOR.
&	Bitwise AND.
==	Equal. Performs a numeric or string comparison depending on the operands. This function is NOT identical to the C strcmp function (see := operator below). If both operands are numbers, a numeric comparison is performed. Otherwise a string comparison is performed. In any case, leading and trailing spaces and tabs are stripped before the comparison is performed.
>	Greater than. Performs a numeric or string

Mathematical Operators

Operator	Description
	comparison depending on the operands. See == operator.
>=	Greater than or equal. Performs a numeric or string comparison depending on the operands. See == operator.
<	Less than. Performs a numeric or string comparison depending on the operands. See == operator.
<=	Less than or equal. Performs a numeric or string comparison depending on the operands. See == operator.
!=	Not equal. Performs a numeric or string comparison depending on the operands. See == operator.
:==	Exactly equal. Always performs string comparison. This is equivalent to the C expression: (strcmp(a,b)==0)
:!=	Not exactly equal. Always performs string comparison.
:<=	Exactly less than or equal. Always performs string comparison.
:<	Exactly less than. Always performs string comparison.
:>=	Exactly greater than or equal. Always performs string comparison.
:>	Exactly greater than. Always performs string comparison.
instanceof	Can be used to test if a class instance derives from a specific class or interface. It can be used in two ways: x instanceof MYCLASS , or x instanceof "MYCLASS" . "MYCLASS" does not need to be a constant string, and x may be a typeless container variable. Slick-C's instanceof is slightly more powerful than Java's, since the right operand can be a string value rather than just a class name.

Mathematical Operators

Operator	Description
	Otherwise, it is essentially the same concept.
<code>:+</code>	Concatenation.
<code>:=</code>	If the left operand is a string type, append right operand (string) to the end of the left operand and assign to the left operand. This is similar to the built-in strappend() function, but more convenient to use. If the left operand is an array type, append the right operand to the end of the array. The statement <code>a := b;</code> is equivalent <code>a[a._length()] = b;</code>
<code><<</code>	Bitwise shift left.
<code>>></code>	Bitwise shift right.
<code>+</code>	Addition.
<code>-</code>	Subtraction.
<code>/</code>	Division with possible floating point result.
<code>intdiv</code>	Division with integer result.
<code>*</code>	Multiplication.
<code>%</code>	Modulo (integer remainder).

Two sets of comparison operators exist. The operators `<`, `>`, `=`, `!=`, `<=`, and `>=` perform a numeric comparison if both string expressions are valid numbers. The operators `:<`, `:>`, `==`, `:!=`, `:<=`, and `:>=` always perform a string comparison.

Select the appropriate comparison operator for performing a string or numeric comparison. Expressions may extend across line boundaries if the line ends in a binary operator or if the line ends with a backslash.

The table below shows examples of math operators in Slick-C.

Example	Operator
<code>(1.0==1)</code>	<code>== true</code>

Declarations

Example	Operator
(1e2==100)	== true
(1e2:==100)	== false
(" abc "==="abc")	== true
(" abc ":=:"abc")	== false
(" abc "!="abc")	== true
(" 1 "=="1)	== true
(" 1 ":=:1)	== false
("abc":<"def")	== true
1 2	:==>"12"
1 (2)	:==>"12"
pow(4,2)	==16
5%2	==1
5/2	==2
5/2.0	==2.5
5 intdiv 2.0	==2
5&2	==0
5 2	==7
(10<7)	== false
(10:<7)	== true

Declarations

Variables and functions are declared in Slick-C® the same way they are defined in C.

This section contains the following topics:

- [Scoping and Declaring Variables](#)
- [Simple Variables](#)
- [Implicit Local Variables](#)
- [Declaring Local Variables With :=](#)
- [Declaring Variables With auto](#)

Scoping and Declaring Variables

The Slick-C® language supports global, namespace, static (module), and local scope variables. Global variables can be accessed by any module. The scope of static and local variables are limited to the module in which they are defined. Variables are declared the same way that they are defined in C++. See [Types](#) for a list of types available in Slick-C.

Namespace level variables are visible within the current namespace but can be accessed from another namespace if they are qualified with the namespace name or imported with the **using** directive.

Simple Variables

The syntax for defining a simple variable is:

```
[static] TypeName variable1[=expression1] , variable2[=expression2] ...;
```

The comma is used to declare more than one variable of the same type. Local variables do not have to be defined. Using a variable not already defined as global or constant declares the variable to be a local typeless variable. However, you should declare variables within the scope of a function to ensure that the variable will be local even if the name is declared elsewhere as a global or constant.

Example:

```
// Declare a global integer.  
int gi=1;  
// Declare a module scope integer.  
static int si=2+4;  
// Declare some global string variables.  
_str gstring1="Value1", gstring2="Value2";
```

```
// Declare a global large floating point variable.  
double gd=1.4;  
// Declare a global typeless variable.  
typeless gt="xyz";  
void defmain( )  
{  
    _str s="ess";  
    // Declare a local string variable and initialize it to "ess".  
    t=gi;  
    // Copy gi into local container variable t.  
    message( "t="t"s="s );  
}
```

Details About Variable Initializations

The following are some details about variable initializations:

- Global and static numeric variables, which include **bool**, **int**, **long**, **double**, and enumerated types, are initialized to **0** when there is no specified value provided. Local variables of any type are not initialized.
- Global and static variables declared as **typeless** or **_str** are initialized with "" (a zero length string) when there is no initialization value provided.
- Global, static, and local variables declared as array, hash tables, and structure types are initialized as **null** when there is no initialization value provided.
- Global, static, and local variables of class type are initialized by running their constructor with default arguments. Global, static, and local variables of interface type are initialized to **null**.
- Local numeric, string, enumerated, and typeless variables require initialization.

Example:

```
bool      globalbool=true;  
int       globalint;  
double    globaldouble;  
void defmain()  
{  
    // Will print message "globalbool=1 globalint=0 globaldouble=0".  
    message( "boolean="globalbool" globalint="globalint"  
    globaldouble="globaldouble");  
}
```

Type Casting

Slick-C® enforces string type checking on everything except typeless variables. However, there are times when you need to convert an expression from its actual type to another. Type casting helps communicate that to the compiler. Note that some type conversions can change the value of an expression. The syntax

for type casting is as follows:

```
(TypeName) expression
```

Some casts are not permitted in Slick-C. For example, you cannot cast a struct type to another struct type. Also, Slick-C does not support the C++ function style cast mechanism, and does not permit pointer types to be cast.

Example:

```
void defmain()
{
    int i;
    double d;
    d=1.7;
    i=(int)d; // i gets the value 1, NOT 1.7
    typeless t;
    t=1.2;
    i=t;      // Here i gets 1.2 BUT
    bool b;
    b= i!=0; // Can't use cast here.
    i=(int)b; // Need cast here.
}
```

Implicit Local Variables

Local variables do not have to be declared. Using a variable not already declared as global or constant declares the variable to be a local typeless variable. However, you should declare variables within the scope of a function to ensure that the variable will be local even if the name is declared elsewhere as a global or constant. Turning on any of the compiler pragmas **autodeclvars**, **strict**, or **pedantic** will flag implicit local variables as errors.

Example:

```
_str cheese1 = "provolone";
_str cheese2 = "cheddar";
temp = cheese2; // Same as typeless temp = cheese2;
cheese2 = cheese1;
cheese1 = temp;
```

Declaring Local Variables With :=

Slick-C® supports type inference using the **:=** operator, which both declares, and initializes a local

variable with inferred type. This syntax provides you with the syntactic convenience of implicit local variables without sacrificing strong type checking.

In the following statement, *id* is declared as a local variable with the same type as *expr*:

```
id := expr;
```

Examples:

```
b := false;           // bool b = false;
i := 0;              // int i = 0;
j := i;              // int j = i;
s := "test";          // _str s = "test";
p := &s;              // _str *p = &s;
c := _process_comment(line); // COMMENT_TYPE c=_process_comment(line);
p := &obj;            // Object *p = &obj;
fp := func;           // int (*fp)() = func;
x := y := 0;          // int x=0; int y=0;
for (a:=1; a<10; ++a); // count to 10
```

Declaring Variables With auto

Slick-C® supports type inference using the **auto** keyword. The syntax for auto variable declarations is:

```
[static] auto variable1[=expression1] , variable2[=expression2] ...;
```

Like the **:=** operator, **auto** variable declarations use type inference to assign a type to the variable being declared and initializes the variable with the specified expression. Auto declarations are allowed in both local and global scopes, whereas **:=** can only be used inside functions for local declarations.

Examples:

```
auto b=false;           // bool b=false;
auto x=0, y=1;          // int x=0; int y=0;
auto i=x+1, s="test"; // int i=x+1; _str s="test";
```

You can also use **auto** to introduce a new local variable when calling a function that takes an "out" argument by reference, or with the **parse** statement. You can think of this identical to using implicitly declared variables, except that you prefix the variable with the **auto** keyword to introduce it. The type of the variable will be inferred from the point of use. In a parse statement, it will become a string type. In a function call, it will acquire the type of the formal argument from the function prototype. The advantage of using **auto** for output-only pass by reference variables is that, when coding a function call, you do not have to backtrack to declare the variable, you can just introduce it at its point of use and keep coding.

Examples:

```
struct Position {
    double x,y,z;
    // ...
};

struct SpaceTimeContinuum {
    _str timeVal;
    // ...
};

void warp(SpaceTimeContinuum &stc)
{
    // ...
}

void travelFast(Position destinations[])
{
    warp(auto stc);
    parse stc.timeVal with auto realPart '+' auto imaginaryPart;
    // ...
    foreach (auto p in destinations) {
        // ...
    }
}
```

Statements

Slick-C® statements are constructed in the same manner as the statements in the C language.

Topics in this section:

- [Assignment Operator](#)
- [if Statement](#)
- [Block Statement](#)
- [Loops](#)
- [parse Statement](#)
- [switch Statement](#)

Assignment Operator

The simple assignment statement has the syntax *variable*=*expression*. For example:

```
i=1;  
i=i+1;
```

Assignment statements can be cascaded (**x=y=z**). Assignment statements within **if** and **while** conditions are not allowed. The compiler flags assignments within **if** and **while** statements as an error. See [Declaring Local Variables With :=](#) for more information.

if Statement

The syntax for an **if** statement is the following:

```
if (expression) statement [else statement]
```

statement can be a C-style statement block which contains multiple statements. For more information, see [Block Statement](#).

Caution

CAUTION The value **0** for all types is false. All other values are true. Like C++, Slick-C® uses the value **0** for null pointers. For the string type, only a one-byte length string where the first character is an ASCII 0 is false. A 0 length string ("") is true when used in a boolean expression. Slick-C

also considers an empty (`=null`) pointer variable or class instance as false.

Example:

```
if (x<y) a==1;
if (x=="a") {
    y=1;
} else if (x=="b") {
    y=2;
} else if (x=="c") {
    y=3;
} else if (x=="d") {
    y=4;
}
```

Block Statement

A statement block is typically used to allow multiple statements within an `if` or loop construct. However, it can also be used to declare a new local scope. A statement block has the following syntax, where `statement` may declare local variables:

```
{
    statement1;
    statement2;
    ...
}
```

Example:

```
int i=0;
if (i<1) {
    int x=1;
    {
        int x;
        // Can do the assignment here.
        x=3;
    }
    // The variable x will be 1 here and not 3.
}
```

Loops

Slick-C® supports C-style [do](#), [for](#), and [while](#) loops. In addition, Slick-C also supports Java/C#-style [foreach](#) loops and the Ada-style [loop](#) statement. You can use [break](#) and [continue](#) with all styles of loops.

do

The **do** loop executes *statement* first and then evaluates *condition_exp*. If *expression* is true (not the value 0), the *statement* continues to be executed until *expression* becomes false (0) or a break statement is reached.

Example:

```
[label:] do statement while ( condition_exp );
```

for

The C-style **for** loop is free-form. The expressions before the first semicolon of the **for** loop are executed before entering the loop. The *condition_exp* expression is checked before entering the **for** loop also. If *condition_exp* is true (not the value 0), the *statement* is executed. The *statement* continues to be executed until *condition_exp* becomes false (0) or a break statement is reached. When the bottom of the **for** loop is reached, but before *condition_exp* is checked again, the expressions after the second semicolon are executed.

The syntax of **for** is:

```
[label:] for (b4e1 ,b4e2 ... .b4e3]; [condition_exp] ;  
{cont_e1,cont_e2 ... ,cont_e3}) statement
```

Examples:

```
// The following loops are equivalent.  
loop1:  
    for ( i=1;i<10;++i ) {  
        messageNwait("i="i);  
    }  
  
loop2:  
    i=1;  
    for ( ;i<10; ) {  
        messageNwait("i="i);  
        ++i;  
    }  
  
loop3:
```

```
for ( i=1;i<10;++i ) messageNwait("i="i);

loop4:
    i=1;
    while ( i <10 ) {
        messageNwait("i="i++);
    }

loop5:
    i=1;
    do {
        messageNwait("i="i);
    } while ( i<10 );
```

foreach

The **foreach** statement works with arrays, hash tables, strings (same as Bourne shell), structs (iterates over the fields of the struct), and classes (if instance of **sc.lang.IIterable**, otherwise like structs). The syntax of **foreach** is:

```
foreach ( [ k => ] v in a ) {
    statements;
}
```

Example:

```
void printStats(int (&statistics)[])
{
    foreach (auto name => auto count in statistics) {
        say("testForeach: "name"="count);
    }
    int i,j=0;
    foreach (i in range(10, 20, 2)) {
        say("printStats: sequence[ "j++" ]="i);
    }
}
```

There is an optional key which is useful for hash tables. The value can be omitted (**key=> . in ht**):

```
foreach( key => value in ht ) {
    statements;
}
```

Both **value** and **index** can be auto-declared using the **auto** keyword. If **value** is auto-declared, its type

will be inferred from the type of the collection. The implementation uses `_nextel()`.

loop

The generic **loop** statement is similar to that found in the Ada and D languages. The following statements are equivalent:

```
for(;;) { ... }
loop { ... }
```

Example:

```
status := search( ":" , "@" );
loop {
    if ( status ) break;
    get_line(line);
    messageNwait("found match line="line);
    status = repeat_search();
}
```

Another example:

```
defmain()
{
    i:=0;
    j:=0;
    loop {
        say("test, i="i);
        i++;
        if (i>1000) {
            break;
        }
        inner: loop {
            say("defmain: j="j);
            if (j++ > 750) break inner;
        }
        say("defmain: H1");
        if (i < 500) {
            continue;
        }
        say("defmain: H2");
    }
}
```

while

The **while** loop evaluates *condition_exp* first and then executes *statement* if *condition_exp* is true (not the value **0**). The *statement* will continue to be executed until *condition_exp* becomes false (**0**) or a break statement is reached.

Example:

```
[label:] while (condition_exp) statement
```

break

Loops are exited with the **break** primitive. The **break** primitive supports an optional **label** argument (like JavaTM). If specified, the label must match the *label* of one of the loops that you are currently using.

continue

The **continue** primitive can be used to skip to the top of a loop. Using **continue** on a **for** loop causes the expressions after the second semicolon to be executed before *condition_exp* is checked. When **continue** is used on a **do** statement, the *condition_exp* is not checked and execution resumes at the top of the loop.

Loops can also be exited with the **continue** primitive. The **continue** primitive supports an optional **label** argument (like Java). If specified, the label must match the *label* of one of the loops that you are currently using.

Example:

```
outerloop:  
for ( i=1;i<3;++i ) {  
    for ( j=1;;++j ) {  
        if ( j==2 ) continue outerloop; // Exit inner loop.  
        if ( j==3 ) break outerloop; // Exit both loops.  
        messageNwait("i="i);  
    }  
}
```

parse Statement

The syntax for **parse** is **parse string with template**. This statement parses *string* as specified by *template*.

The table below shows what *template* may contain.

Item	Description
<i>variable_name</i>	Output variable.
.	Null output variable.
<i>nnn</i>	Number specifying new parse column.
+ <i>nnn</i>	Amount to increment parse column relative to start of last string found or last column setting.
- <i>nnn</i>	Amount to decrement parse column relative to start of last string found or last column setting.
'text'[, <i>search_options</i>]	<p>String constant to search for. If found, parse column becomes first character after <i>text</i>. Otherwise parse column becomes first character after length of string being parsed. <i>search_options</i> is an optional expression that may evaluate to a string of one or more of the option letters U, R, B, I, and Y:</p> <ul style="list-style-type: none"> • R Specifies SlickEdit regular expressions. • L Specifies Perl regular expressions. • ~ Specifies Vim regular expressions. • U Specifies Perl regular expressions. Unix syntax regular expressions are no longer supported. • B Specifies Perl regular expressions. Brief syntax regular expressions are no longer supported. • & specifies wildcard regular expressions. • I Specifies a case insensitive search. • Y Specifies a binary which search allows positions in the middle of a DBCS character (only affects Japanese operating systems). <p>See the topic "regular expressions" in the SlickEdit® Help system for more information.</p>
<i>(expression)</i> [, <i>search_options</i>]	String <i>expression</i> to search for. If found, parse column becomes first character after text. Otherwise parse column becomes first character after length of string being parsed. See above for a description of the search options.

Item	Description
------	-------------

The rules for parse column are:

- The parse column is initialized to column 1.
- If a column or column increment specifies a column greater than the length of the string being parsed, the parse column is set to the length of the string being parsed plus one.
- If a column decrement specifies a column less than the length of the string being parsed, the parse column is set to column 1.

The rules for setting output variables are:

- Output variables are set in groups. An output variable group is defined to be consecutive variables with no search or column specifiers between them.
- Before variables of an output variable group can be set, the end parse column within the source string must be found. In the case the end parse column is set by a search, the end parse column for this output variable group becomes the first character to the left of the text found. In the case the end parse column is set by a column or column increment the end parse column becomes the first character to the left of the column. The start parse column is the current parse column as specified by the template.
- A word parse of the text between the start and end columns is performed to set the variables in an output variable group if the group contains more than one variable. Otherwise the one output variable is set to the text between the start and end columns of the source string. Each variable set by a word parse will have no leading or trailing tabs/spaces except for the last output variable which is set to the rest of the sub-string.
- If the start column is greater than the end column the variables in the output group are set to null.

Wildcard regular expressions are supported for **parse**. You can also use the **auto** keyword to auto-declare the output string variables in a **parse** statement. For example, the following statement declares "firstword" and "secondword" as strings:

```
parse s with auto firstword auto secondword;
```

Examples of **parse**:

```
// Results are a=='1', b=='2', c=='3'.
parse '1 2 3' with a b c;

// Results are a=='1', b=='2', c=='3'. Note that tab and space
characters are stripped.
parse '1 '\t' 2 '\t' 3' with a b c;
```

```
// Results are a=='1', b=='3'.
parse '1 2 3' with a . b;

// Results are a=='1', b=='2', c=='3', d=='4', e=='5'.
parse 'xxx1 2 3yyy 4 5' with 'xxx' a b c 'yyy' d e;

// Results are a=='1 2 3', b==' 4 5'.
parse 'xxx1 2 3yyy 4 5' with 'xxx' a 'yyy' b;

// Results are a=='xxx1 2 3', b=='yyy 4 5'.
parse 'xxx1 2 3yyy 4 5' with 'xxx' +0 a 'yyy' +0 b;

// Results are delim=='/', s1=='x', s2=='y', options=="".
parse 'c/x/y' with 2 delim +1 s1 (delim) s2 (delim) options;
```

switch Statement

Slick-C® supports the C **switch** statement. The Slick-C **switch** supports integers and string types. The **switch** statement uses the following syntax:

```
switch (expression) {
[ case expression:
statements
]
[ case expression:
statements
]
...
[ default:
statements
]
}
```

The **switch** expression is evaluated and compared against all the case expressions. After a match is found, ALL statements below the case are executed, including those statements found in the next case and the default, until a break statement is reached.

Example:

```
outerloop:
for ( i=1;;++i ) {
switch ( i ) {
case 1:
case 2:
    messageNwait("i=1 or i=2");
```

```
        break;
        // Done with these cases.
    case 3:
        break outerloop;
    }
}
```

Functions

A function can be called from the macro language. Slick-C® has five kinds of functions: procedures, commands, class methods, library functions, and built-ins. These are described in the following sections:

- [Defining a Procedure](#)
- [Defining a Command](#)
- [Class Methods](#)
- [Function Prototypes](#)
- [Differences Between Commands, Built-ins, and Defs](#)

Defining a Procedure

Procedures and functions are the basic building blocks for most modern, imperative languages. Slick-C® procedures cannot be bound to keys. A procedure name must be a valid Slick-C identifier (same as C identifier). Use the following syntax to define a procedure:

```
[static] [TypeName] id(TypeName1 [&] id1, TypeName2 [&] id2, ...)  
{  
    statement1;  
    statement2;  
    ...  
}
```

TypeName specifies the return type of the function. For more information, see [Types](#). If the return type is not specified, the function will return **typeless**. When the **void** type is used, a value cannot be specified to the return statement. The return statement is used to specify the result of the function call and exit the function.

The optional **static** keyword is used to limit the scope of a procedure to the module in which it is defined. By default, procedures are global and can be accessed by any module. Procedures are called by specifying the name followed by comma delimited arguments, if any, in parentheses.

```
[ result =] id( expr1, expr2, ... );
```

In the above example, **expr1** matches the type of **id1** and **expr2** matches the type of **id2**, etc.

Example:

```
int increment(int x)
{
    return x+1;
}
bool proc(int &p1, _str p2, _str (&list)[], int (*pfn)(int))
{
    return(true)
}
void defmain()
{
    p1 := 0;
    p2 = "Hello world";
    if ( proc(p1, p2, auto list, increment) ) {
        // ...
    }
}
```

Note

The **list** and **p1** parameters are call by reference parameters. Like C++, **list** parameter requires parentheses around the **&** reference operator and the name, because the **[]** operator would otherwise be processed first. The **pfn** parameter is a pointer to a function.

Argument Declarations

The syntax for an argument declaration is the same as for declaring a variable, except that the **static** keyword cannot be used. An ampersand (**&**) before the *id* declares a call by reference parameter. Call by reference array and hash table parameters require parentheses around the **&** and *id*.

The last argument in the declaration list may be an ellipsis to indicate that the function accepts more arguments of any type. Use the **arg** function to access these optional arguments.

TypeName specifies the return type of the function. For more information, see [Types](#). If the return type is not specified, the function will return **typeless**. When the **void** type is used, a value cannot be specified to the return statement. The return statement is used to specify the result of the function call and exit the function.

The optional **static** keyword is used to limit the scope of a procedure to the module in which it is defined. By default, procedures are global and can be accessed by any module. Procedures are called by specifying the name followed by comma delimited arguments, if any, in parentheses.

Example:

```
bool proc(int &p1,_str p2,_str (&list)[],int (*&pfn)(int))
{
```

```
        return(true)
}
```

Note

The **list**, **p1**, and **pfn** parameters are call by reference parameters. Like C++, the **list** parameter requires parentheses around the & reference operator and the name, because the [] operator would otherwise be processed first. This avoids deviating much from C++ syntax. The command **pfn** is a reference to a pointer to a function.

Procedures can have up to 15 arguments defined. The procedure can be called with more arguments than defined by the procedure declaration. These extra arguments and the arguments defined in the procedure declaration can be retrieved by the **arg** function. Calling the **arg** function with no parameters returns the number of parameters with which the function was called. The minimum number of arguments with which the procedure may be called is defined by the procedure heading. A parameter of type **var** specifies a typeless variable passed by reference.

Default Arguments

Defining arguments with default values instead of using the **arg** function makes your code more understandable. The assignment operator has special meaning in an argument declaration. It defines a default value for an argument. The default value is used if the caller does not specify the parameter. Default arguments must always be specified in the function definition. Unlike C++, default arguments in prototypes do not have an effect on the compiled code.

Example:

```
static int proc2()
{
    return("before");
}
int proc(_str p1=proc2():+"after",int p2=2)
{
    return(p1:+p2);
}
void defmain()
{
    proc();           // Use defaults ("beforeafter" ,2).
    proc("param1");  // Use the second default value.
    proc("param1",3); // Specify both values.
    proc(,3);        // This is not allowed.
}
```

Named Arguments

When calling a function, the formal parameter name can optionally be specified for each actual parameter. Using named arguments can make code more directly understandable. Named arguments can also be used to skip ahead over default arguments and only specify the most pertinent arguments in a function call. When using a named argument, the argument name must match the formal parameter name in the function prototype. Named arguments must be specified in the same order the parameters are specified in the function prototype. ... : can be used to specify the start of a function's variable argument list as a named argument.

Example:

```
int proc(_str p1,int p2=2,_str p3="after")
{
    return(p1:+p2:+p3);
}
void defmain()
{
    proc("before");           // Uses default args:      proc("before"
,2, "after")
    proc("before", p2:32);    // Use default for p3:
proc("before", 32, "after")
    proc(p1:[ ,p3:] );      // Use default for p2:      proc("[ ", 2,
" ]")
    proc(p1:( ,32,p3:) );   // All args specified:     proc("( ", 32,
" )")
    proc(p2:300);           // Not allowed (first parameter has no
default)
    proc(p3:/ ,p1:/ );      // Not allowed (parameters out of
order)
    proc("before",5,p4:null); // Not allowed (no parameter named
'p4')

    // illustrates how named arguments can make function call
self-documenting
    ext := _get_extension(p_buf_name, returnDot:true);
}
```

Defining a Command

The **_command** primitive is used to define a new command with argument completion. A command can be invoked by typing its name on the SlickEdit® command line, selecting it from a menu item definition, pressing a key, calling it in a Slick-C® function, or typing its name followed by arguments in parentheses in a Slick-C expression. Command procedures always have global scope and can be bound to a key with the Key Bindings option screen (**Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**).

The syntax for defining a command is:

```
_command [TypeName | void] name1[,name2 [,name3... ]]( [ArgDecl1,  
ArgDecl2, ...] )  
    [name_info(const_exp)]  
    {  
        statements  
    }
```

TypeName specifies the return type of the command (see [Types](#)). If *TypeName* or **void** is not specified, the return type is **typeless**. When the **void** type is used, a value cannot be specified to the return statement. The return statement is used to specify the result of the function call and exit the function.

The syntax for *ArgDecl*s is the same as for declaring a variable, except that the **static** keyword may not be used. In addition, an **&** before the *id* declares a call by references parameter. Call by reference array and hash table parameters require parentheses around the **&** and the *id*. However, all typed or named arguments must have a default value.

The last argument in the declaration list can be an ellipsis to indicate that the function accepts more arguments of any type. Use the **arg** function to access these optional arguments.

The name of a command may be a valid Slick-C identifier, or a string constant of a length of one, such as **"/"**. SlickEdit uses the slash to define a search command.

Example:

```
// Allow command in read only mode.  
// Use ellipsis because this accesses arguments.  
_command int goto_line(...)  
    name_info(' ','VSARG2_READ_ONLY|VSARG2_REQUIRES_EDITORCTL)  
{  
    param=arg(1);  
    if (param==" " || ! isinteger(param)) {  
        message('Please specify line number');  
        return(1);  
    }  
    p_line=param;  
    return(0);  
}  
_command void mycommand(_str filename=" ") name_info(FILE_ARG)  
{  
    if (filename==" ") {  
        _message_box("No filename specified");  
    }  
    message("filename="filename);  
}
```

Commands receive unnamed command line arguments by calling the **arg** function. When a command is

invoked from the command line, the expression **arg(1)** contains the rest of the command line after the name with leading spaces removed. For example, invoking the edit command **e file1file2** calls the **e** command with **file1 file2** in **arg(1)**. The **parse** built-in is an excellent function for parsing a command line string (see the Help system for more information on parsing). When another macro calls a command, more than one argument string can be passed. Calling the **arg** function with no parameters returns the number of parameters with which the command or procedure was called.

name_info Attributes

The optional **name_info** expression is used to specify command argument completion rules and restricts when the command may be executed.

const_exp is a single constant expression. A comma (,) character in the string indicates the end of an argument.

The first argument in *const_exp* indicates the type of word arguments the command accepts and is used for argument completion purposes. For a list of already defined argument types, look in the **slick.sh** file for constants that end in **_ARG**. *const_exp* may contain one or more of the **_ARG** constants. Separate each **_ARG** constant with a space. An asterisk (*) character may be appended to the end of a completion constant to indicate that one or more of the arguments may be entered. The second argument (after the quoted comma) specifies when the command should be or disabled. One or more of the flags in the table below can be specified and ORed together with the bitwise OR (|) operator.

Flag	Description
VSARG2_CMDLINE	Command supports the command line. VSARG2_CMDLINE allows a fundamental mode key binding to be inherited by the command line.
VSARG2_MARK	ON_SELECT event should pass control on to this command and not deselect text first. Ignored if command does not require an editor control.
VSARG2_QUOTE	Indicates that this command must be quoted when called during macro recording. Needed only if command name is an invalid identifier or keyword.
VSARG2_LASTKEY	Command requires last_event value to be set when called during macro recording.
VSARG2_MACRO	This is a recorded macro command. Used for completion.
VSARG2_TEXT_BOX	Command supports any text box control. VSARG2_TEXT_BOX allows a fundamental mode key binding to be inherited by a text box.
VSARG2_NOEXIT_SCROLL	Do not exit scroll caused by using scroll bars.

Flag	Description
	Ignored if command does not require an editor control.
VSARG2_EDITORCTL	Command allowed in editor control. VSARG2_EDITORCTL allows a fundamental mode. Key binding to be inherited by a non-MDI editor control.
VSARG2_NOUNDOS	Do not automatically call <code>_undo('s')</code> . Require macro to call <code>_undo('s')</code> to start a new level of undo.
VSARG2_READ_ONLY	Command allowed when editor control is in strict read only mode. Ignored if command does not require an editor control
VSARG2_ICON	Command allowed when editor control window is iconized. Ignored if command does not require an editor control.
VSARG2_REQUIRES_EDITORCTL	Command requires an editor control.
VSARG2_REQUIRES_MDI_EDITORCTL	Command requires MDI editor control.
VSARG2_REQUIRES_AB_SELECTION	Command requires selection in active buffer.
VSARG2_REQUIRES_BLOCK_SELECTION	Command requires block/column selection in any buffer.
VSARG2_REQUIRES_CLIPBOARD	Command requires editorctl clipboard.
VSARG2_REQUIRES_FILEMAN_MODE	Command requires active buffer to be in fileman mode.
VSARG2_REQUIRES_TAGGING	Command requires <code><ext>_proc_search/find-tag</code> support.
VSARG2_REQUIRES_SELECTION	Command requires a selection in any buffer.
VSARG2_REQUIRES_MDI	Command requires MDI interface maybe because it opens a new file or uses <code>_mdi</code> object. Commands with this attribute are removed from pop-up menus in which the MDI interface is not available (editor control OEMs).

Example:

```
#include "slick.sh"
// This command supports completion where the first argument
// is a filename and the second argument is an environment variable.
_command test1(...) name_info(FILE_ARG" "ENV_ARG)
{
    parse arg(1) with file_name env_name;
    message("file_name="file_name" env_name="env_name);
}
// This command is enabled only when the target is an editor control
// which has a selection.
_command void gui_enumerate()
    name_info(','VSARG2_REQUIRES_EDITORCTL|VSARG2_REQUIRES_AB_SELECTION)
{
    ...
}
// This command supports completion on multiple filenames.
_command int e,edit(...)
name_info(FILE_ARG'*,'VSARG2_CMDLINE|VSARG2_REQUIRES_MDI)
{
    ...
}
```

The **edit** command allows any number of file name arguments to be given. When the user is presented with a selection list of file names, many files may be selected with the spacebar key. If an asterisk (*) is appended to the end of a completion constant, that command must support a space-delimited list of strings. Double quotes are placed around arguments with embedded spaces.

The value of *const_exp* may be retrieved by the built-in function **name_info**.

OnUpdate Functions

A Slick-C® command can have a corresponding **_OnUpdate_commandname** function. This function is used to provide more precise control over the enabling and disabling of a command than the **name_info** command can provide.

Example:

```
int _OnUpdate_linehex(CMDUI &cmdui,int target_wid,_str command)
{
    if ( !target_wid || !target_wid._isEditorCtl() ) {
        return(MF_GRAYED);
    }
    if (p_UTF8) {
        return(MF_UNCHECKED|MF_GRAYED);
    }
}
```

```
    if (p_hex_mode==2) {
        return(MF_CHECKED|MF_ENABLED);
    }
    return(MF_UNCHECKED|MF_ENABLED);
}
```

Class Methods

Slick-C® classes can contain methods which implement the class behaviors. Slick-C supports static class methods. These methods may be called without having an instance of the class available. Like Java, all other Slick-C class methods are virtual. Unlike Java and C++, Slick-C class methods do not support overloading. A class method may have up to 14 arguments. Like Java and C++, the first argument is hidden and contains the class instance (this) for virtual methods.

Example:

```
namespace outer;

interface IShape {
    double area();
    void draw();
};

class Rectangle : IShape {
    int m_w=0;
    int m_h=0;
    double area() {
        return m_w*m_h;
    }
    void draw() {
        // Draw box.
    }
};
class Circle : IShape {
    int m_r=0;
    double area() {
        return m_r*m_r*3.1459;
    }
    void draw() {
        // Draw round thing.
    }
};
class Factory {
    static IShape makeShape(int x, int y, _str type, ...)
    {
        switch ( type ) {
```

```
        case "Rectangle":  
            // ...  
        case "Circle":  
            // ...  
        }  
        return null;  
    }  
};  
namespace default;  
void draw_car()  
{  
    body := outer.Factory.makeShape( 0, 10, "Rectangle", 40, 10);  
    cab := outer.Factory.makeShape(10, 10, "Rectangle", 20, 10);  
    axl1 := outer.Factory.makeShape( 5, 5, "Circle", 5);  
    axl2 := outer.Factory.makeShape(30, 5, "Circle", 5);  
  
    outer.IShape car[];  
    car[car._length()] = body;  
    car[car._length()] = cab;  
    car[car._length()] = axl1;  
    car[car._length()] = axl2;  
    double area = 0.0;  
    foreach ( auto s in car ) {  
        area += s.area();  
    }  
    foreach ( s in car ) {  
        s.draw();  
    }  
}
```

Function Prototypes

Function prototypes provide the compiler with type information about a function without providing any code. Slick-C® reduces the need for prototypes by performing some argument checks at link time. When the linker finds an uninitialized variable error, it recommends that you add a function prototype to your source so the compiler can find your error. You might need a function prototype if you want to use the function address in an expression. Prototypes are not allowed for event functions.

The syntax for defining a function prototype is identical to defining a function except that a semicolon (;) is placed after the closing parentheses of the parameter list. Unlike C++, default arguments in prototypes have no effect on the compiled code. No code or **name_info** is given.

The need for function prototypes is also mitigated in Slick-C because of the **#import** directive which allows the compiler to import declarations from another Slick-C module. This is more convenient than C++, where you need to put declarations in a header file to support calling functions across modules. It is also more convenient that Java, because **#import** gets declarations directly from the source code, so the imported module does not need to be compiled to be imported. This simplifies compiling modules with

circular dependencies.

Example:

```
int proc(_str s,_str list[]);           // Function prototype.  
int (*pfn)(_str s,_str list[])=proc;   // Pointer to function.  
_command void command1(...);            // Function prototype.  
_command void command1(...) {          // Must have ... here to match  
prototype.                           // Use arg function here to get or  
set                                // arguments.  
}
```

Library Functions

A library function is a function that was implemented in a dynamically loaded library and was not written in the Slick-C® language. A library function must follow Slick-C calling conventions and be registered with the interpreter. Prototypes for library functions should use the **extern** keyword to indicate that they are implemented outside of Slick-C code.

Built-in Functions

A built-in function is a function that was implemented in the interpreter and was not written in the Slick-C® language.

Finding Functions

There are over 1200 documented functions and 200 properties. There are two ways to find the function that you seek. First, you can use the menu item **Help → Macro Functions by Category**, which displays smaller lists of these functions by category. Second, you can view source code for existing commands. If you do not know the name of the command but you do know the key that invokes the command, use the **what_is** command or **Help → What Is Key** to find the name of the command that is executed. Then, use the **find_proc** command or **Macro → Find Slick-C Proc** to display the macro source code.

Differences Between Commands, Built-ins, and Defs

- A command definition looks like a procedure that starts with the **_command** primitive, and has an optional **name_info** construct after the arguments. Built-ins are not defined.
- Commands always have global or namespace scope. Built-ins always have global scope. Procedures can have static (module), global scope, or namespace scope.
- Commands can be bound to keys. Built-ins and procedures cannot.
- Commands can be invoked from the command line or the **execute** function. Built-ins and procedures cannot.

Differences Between Commands, Built-ins, and Defs

- A command may be given the same name as a built-in. However, this limits how the command may be called within a macro (use the **execute** function). None of the commands have the same name as a built-in so you can call any command just like any other function.
- Only commands may be given non-alphanumeric single character names such as +, =, !, @, #, \$, etc. However, this limits how the command can be called within a macro (place the command in quotes or use the **execute** function).

There are several differences between defining a procedure and defining a command with the **_command** primitive:

- The scope of a procedure can be limited to a module.
- Command functions are invoked by typing the name on the SlickEdit® command line, from a menu item definition, by using the **execute** function, or by typing the command name followed by arguments in parentheses in a Slick-C® expression. Procedures can only be called by the latter method and cannot be bound to keys.
- A procedure name must be a valid Slick-C identifier (same as C identifier). The name of a command can be a string constant containing a single character such as "/" (SlickEdit uses the slash to define a search command).

defmain: Writing Slick-C® Batch Files

A batch macro contains a special function named **defmain**. Slick-C batch files have the extension .e. Batch macros can be invoked by typing the name (extension not required) followed by arguments on the SlickEdit® command line, quoting the name in a macro, or by using the **execute** function. If the batch macro needs to be recompiled, the Slick-C translator is invoked before the batch macro is executed. Do not use the **load** command to load a batch program, because **defmain** is not invoked and an error will result. If you load a batch program that you do not want, use the **unload** command to unload it. When a batch program is executed, the **defmain** procedure is called after the procedure **definit** is called. For more information, see [Module Initializations](#).

The syntax of the **defmain** function is:

```
[TypeName | void] defmain()
{
    statement
    statement
    ...
}
```

TypeName specifies the return type of the function. If *TypeName* or **void** is not specified, the return type is typeless. When the **void** type is used, a value cannot be specified to the return statement. The return value of **defmain** is placed in the predefined **rc** global variable.

Note

The **execute** function only supports returning an **int** type. Check the global **rc** variable for other types.

The **arg** function is used to retrieve the command line arguments passed to the **defmain** procedure. All of the command line arguments will be in **arg(1)**. Use the **parse** statement to easily parse multiple space delimited arguments.

The following example displays the arguments given to the macro on the SlickEdit message line. If you define a procedure in a batch program, use the **static** keyword to conserve memory. SlickEdit stores the names of global procedures and variables in a names table.

```
int defmain()
{
    messageNwait("Arguments given: " arg(1));
    parse arg(1) with word1 word2 .;
    messageNwait("word1=" word1" word2=" word2);
    return(0);
}
```

Extending the editor with a batch macro has the advantage of conserving memory and reducing the size of the state file. Also, batch macros can be easily shared between multiple users. The editor keeps the batch macro loaded only while it is executing. External batch macro names and arguments are not supported by completion. To provide completion, you must define a command with the **_command** primitive and have it call the external batch program. If you name the command the same name as the batch program (without the extension), use the **xcom** command to bypass internal command searching. There are two ways to invoke a Slick-C batch macro:

- Type the name of the module followed by arguments on the SlickEdit command line.
- Type **vs -p program** at the shell prompt, where *program* is the name of the batch program and **vs** is the name of the editor. Alternatively, you may use the **-r** option to have SlickEdit remain resident after the batch program completes.

For the above methods, SlickEdit invokes the translator to compile the source code file if the source code file exists and its date is later than the date of the **.ex** file.

Preprocessing

Preprocessing in Slick-C® is identical to C/C++. Preprocessing allows you to conditionally compile source code or define textual replacements.

This chapter includes the following topics:

- [#if](#)
- [#pragma](#)
- [#region and #endregion](#)
- [Including Header Files](#)
- [Importing Slick-C Modules](#)

#if

The syntax of the Slick-C® language conditional **if** block is any of the following examples:

```
#if expression
[statements]
{#elif expression
[statements]}
[#else
statements]
#endif
```

There may be nothing more than space or tab characters preceding a **#**. Text on the same line following **#else** or **#endif** is not permitted. The expression specified MUST be valid. To display an error message and end the compile, use the **#error** directive: **#error expression**.

Usually, preprocessing is used to write macros that operate on multiple operating systems or environments. The table below shows the constants that are automatically defined by the Slick-C translator.

Note

The following platform-specific constants are deprecated. Using them will result in a Slick-C compilation error if you are using **#pragma option(pedantic,on)** or **#pragma option(deprecation,on)**. Use the **machine()** built-in function, or the **_isUnix()**, **_isWindows()** or **_isMac()** built-in functions instead.

Constant	Description
<code>_PCDOS_</code>	Deprecated. Use the built-in <code>_isWindows()</code> function instead. Non-zero if the current operating system is Windows. Use <code>machine()</code> built-in function to determine at run time which of these operating systems you are running.
<code>_UNIX_</code>	Deprecated. Use the built-in <code>_isUnix()</code> function instead. Non-zero if current operating system is UNIX compatible.
<code>_NT_</code>	Deprecated. Use the built-in <code>_isWindows()</code> function instead. Non-zero if the current operating system is Microsoft Windows NT® compatible.
<code>_MACOSX_</code>	Deprecated. Use the built-in <code>_isMac()</code> function instead. Non-zero if the current operating system is macOS.
<code>VERSION</code>	Version number of SlickEdit®.
<code>COLUMN</code>	Number of characters from start of current line.
<code>FILE</code>	Current file name.
<code>LINE</code>	Current line number.
<code>PATH</code>	Current file name and path.
<code>DATE</code>	Current date.
<code>TIME</code>	Current time.

Use the Slick-C translator `-d` option to define a constant for use by preprocessing. To test if a constant has been defined, use the `defined()` function.

Example:

```
#if !defined(my_constant)
    #define my_constant "default value"
#endif
#if __PCDOS__
    name= "c:\util\myprog"
#elif __UNIX__
    name= "/usr/bin/myprog"
#else
    #error "Don't know what to do for this OS"
#endif
```

The above example could be written as follows (without using deprecated preprocessor constants).

```
const my_constant = "default value";

if (_isWindows()) {
    name= "c:\util\myprog"
} else if (_isUnix()) {
    name= "/usr/bin/myprog"
} else {
    _message_box("Don't know what to do for this OS");
}
```

#pragma

The **#pragma** preprocessor directive is used to change various options during the course of a compile. The syntax is:

```
#pragma option(OptionName [, ( on | off ) ] )
```

Slick-C® options (*OptionName*) are shown in the table below.

Note

- For each option, if the second argument is not given, the value is restored to the command line invocation value.
- All **#pragma** options may be specified by command line compiler options. Run `vst.exe` (UNIX: `vst`) with no arguments to view compiler options. You can use the **VST** environment variable to specify compiler options for all of your macros.

OptionName	Default Value	Description
autodecl	On	Enables autodeclvars and autodeclctls . See those options for more information.
autodeclctls	On	When enabled, the compiler attempts to automatically declare control variables. This option is automatically enabled when autodecl , pedantic , strict , or strict2 is enabled.
autodeclvars	On	When enabled, the compiler attempts to automatically declare typeless variables when an assignment is made. This option is automatically enabled when autodecl is enabled. This option is automatically disabled when pedantic , strict , or strict2 is enabled.
deprecation	Off	Allows you to configure properties and built-in functions as deprecated. When enabled, the Slick-C compiler catches when a deprecated item is used and flags it as an error. A function is considered as deprecated if it has a Javadoc function comment containing the @deprecated tag. Deprecation is automatically enabled when pedantic is enabled. Note that using the deprecation pragma may result in your macro not loading when you upgrade to a new release of SlickEdit if the code calls a function that becomes deprecated.
pedantic	Off	Enabling this option automatically enables all existing and future strict syntax and type-checking options. Unlike other Slick-C pragmas, the meaning of

OptionName	Default Value	Description
		pedantic could be augmented in future releases of SlickEdit. This means that if you use the pedantic pragma in your own macros, they may not load when you upgrade to a new release of SlickEdit if, for example, a function it is using becomes deprecated or stricter type checking reveals a problem.
redeclvars	Off	This is used to generate code for variables without having the type information. When enabled, any variable can be redeclared as a typeless variable.
strict	Off	This option is used to turn on a high level of type checking and syntax enforcement in the Slick-C translator. It automatically enables autodeclctls , autodeclvars , strictnumbers , strictparens , strictsemicolons , and strictstrings . We recommend using #pragma option(strict,on) in user-written macros because it gives the best combination of high level of error checking and forward compatibility.
strict2	Off	Second generation of strict Slick-C compilation checks. Automatically enables all options that strict enables, plus strictarglists , strictincludes , and twopass .
strictarglists	Off	When disabled, a function can have implicitly typeless arguments. When enabled, formal parameter lists for functions and prototypes must have types. The example illustrates an error case:

OptionName	Default Value	Description
		<pre>void first_char(_str s) { return substr(s,1,1); }</pre> <p>This option is automatically enabled when pedantic or strict2 is enabled.</p>
strictboolean	Off	<p>When enabled, Slick-C variables with boolean types cannot be assigned to integers without using a cast. This means, for example, that the following would be flagged as an error:</p> <pre>bool b = 0;</pre> <p>This option is automatically enabled within classes and namespaces.</p>
strictpointers	Off	<p>When enabled, Slick-C variables with pointer types can only be assigned to compatible pointer types or null. This means, for example, that the following would be flagged as an error:</p> <pre>int *p = 0;</pre> <p>The correct code would be:</p> <pre>int *p = null;</pre> <p>This option is automatically enabled within classes and namespaces.</p>
strictenums	Off	<p>When enabled, Slick-C variables with enumerated types cannot be assigned to integers without using a cast. This means, for example, that the following would be flagged as an error:</p>

OptionName	Default Value	Description
		<pre>enum MyOptions {O1, O2, O3}; MyOptions e = 0;</pre> <p>The correct code would be:</p> <pre>enum MyOptions {O_DEF=0, O1, O2, O3}; MyOptions e = O_DEF;</pre>
strictellipsis	Off	When enabled, the ellipsis must be given as the last argument to a function or prototype for type checking to succeed when calling function with extra arguments. This option is automatically enabled when pedantic or strict2 is enabled.
strictincludes	Off	When enabled, verifies that all #import and #include statements precede any real code in the current module. This is required for twopass compilation. This option is automatically enabled when pedantic or strict2 is enabled.
strictnames	Off	When enabled, enforces naming conventions for symbols declared in Slick-C classes, namespaces, and enumerated types. See Slick-C® Naming Conventions for more information. This option is automatically enabled when pedantic is enabled.
strictnumbers	Off	When enabled, Slick-C numeric constants are treated strictly as integer or double precision floating point types, rather than typeless variables. This makes it

OptionName	Default Value	Description
		possible to use precise type inference with integer types, like <code>i := 0;</code> . This option is automatically enabled when pedantic is enabled. It is also automatically enabled within classes and namespaces.
strictparens	Off	Use this pragma for more readable code. When enabled, parentheses must be given on all built-in functions. This option is automatically enabled when pedantic , strict , or strict2 is enabled.
strictprotos	Off	When enabled, all function calls require the function to be previously declared or imported. When disabled, when the Slick-C compiler encounters a function call to a previously undefined function, it assumes that the function is a global function. The function call is resolved at link time, and an error will show up at run time if the function does not exist or is not provided enough parameters. This option is automatically enabled within classes and namespaces.
strictreturn	On	When enabled, and an explicit return type is given to a function, the compiler will flag an error if a return statement is potentially missing.
strictsemicolons	Off	Use this pragma so that smart editing features work, and to prevent compilation errors. When enabled, semicolons must terminate all statements. This option is automatically enabled when pedantic , strict , or strict2

OptionName	Default Value	Description
		is enabled.
strictstrings	Off	When enabled, Slick-C string constants are treated strictly as string types, rather than typeless variables. This means, for example, that you can no longer assign "0" to an integer variable. This option is automatically enabled when pedantic , strict , or strict2 is enabled.
twopass	Off	When enabled, the Slick-C compiler does a two-pass compilation. This allows the compiler to verify all function call signatures even for functions that are declared later in the file. This option is automatically enabled when pedantic or strict2 is enabled.
metadata	Undefined	The metadata #pragma is specified in Slick-C header files to indicate that a header file (.sh) is strongly associated with a particular compilation unit. Normally, when a header file is included by a Slick-C module, the metadata for struct, class, enum, and const declarations is compiled into that module. However, the metadata #pragma is given, it indicates that the metadata should only be compiled in for the specified module. This is done to reduce code size.

#region and #endregion

The **#region** directive lets you specify a block of code that you can expand or collapse when using Selective Display. The **#endregion** directive marks the end of a **#region** block. A **#region** block must be terminated with **#endregion**. The syntax of these directives is:

```
#region name
#endregion name
```

The *name* parameter (optional) is used to indicate the name of the region. This name is displayed in the editor window when the region is collapsed.

Example:

```
#region Region_1
void Test() {}
void Test2() {}
void Test3() {}
#endregion Region_1

void defmain()
{
}
```

Including Header Files

The syntax of the **include** statement is:

```
#include string_constant
```

This statement includes the file specified by *string_constant* for compiling. If *string_constant* does not specify a path, the Slick-C® translator will look in the same directory of the main source file. Otherwise, the path specified by *string_constant* is searched. If the file is not found, the Slick-C translator looks for the include file in the directories specified by the **VSLICKINCLUDE** and **VSLICKPATH** environment variables (see "environment variables" in **Help → Index**). Include files may be nested.

Unlike C++, Slick-C header files do not require guards. Our preprocessor automatically guards against recursive header file inclusion, and will never include the same header file twice for a single module either.

Importing Slick-C Modules

#import is a preprocessing directive but it is more than a **#include** in that it does the following:

- Imports all public declarations from a Slick-C® module.
- Uses an implicit header guard to prevent recursive or multiple importation of the same module.

#imports are not recursive. If you #import a module (abc.e) that #imports another module (def.e), you will not get the declarations from def.e. This is an important consideration for compilation performance and to minimize inter-module dependencies.

#require is a preprocessing directive, like **#import**, but it is recursive. If you want a module to always pull in another required module when it is #imported, use the **#require** directive. For example, a class (Abc) that derives from another class (Def) should #require the parent class module (Def.e). That way when another module #imports Abc.e, it will also have the declaration for the parent class Abc. As a general rule, a module needs to use **#require** when its classes or function signatures use types that are declared in another module. Use **#import** when your code (within function bodies) needs to call functions or use classes and global variables from other modules.

When processing a **#import**, the following rules are in effect:

- All function definitions are treated as prototypes.
- Global variable definitions are treated as declarations.
- Static globals are ignored.
- Forms, menus, event tables, and event handlers are ignored.
- **#includes** continue to be treated as part of the **#import**.

Examples:

```
#import "stdcmds.e"
#import "slickedit/stringutil.e"
#import "slickedit/search.sh"
```

Defining Controls

Usually, you do not need to communicate with the compiler about a control to which you refer; however there are a couple of cases in which you must declare a control. This can happen when the compiler cannot safely assume that you are referring to a control, or when the compiler cannot find the location of the dialog box of the control that you are trying to access. The compiler needs to tell the linker which dialog box is supposed to contain your control. The syntax for declaring a control variable is:

```
[_nocheck] ObjectName ControlName;
```

Or you can use:

```
[_nocheck] _control ControlName;
```

ObjectName can be one of the following:

- **_check_box**
- **_combo_box**
- **_command_button**
- **_gauge**
- **_hscroll**
- **_image**
- **_label**
- **_list_box**
- **_picture_box**
- **_radio_button**
- **_text_box**
- **_vscroll**

The **_nocheck** keyword tells the compiler not to check if the control exists on the current dialog box.

The **[_nocheck] ObjectNameControlName;** declaration is only permitted outside the scope of a function.
The **[_nocheck] _control ControlName;** declaration already supports local procedure scope.

Example:

Defining Events and Event Tables

```
// Create a form with a command button named ctlcancel, and gauge named
ctlgauge1.
// Set the cancel and default properties of the command button to true.
//
#include "slick.sh"
static bool gcancel;
_command void test()
{
    // Need to tell compiler ctlgauge1 is a control because
    // the form1_wid.ctlgauge1 is too ambiguous.
    _control ctlgauge1;

    // Show the form modeless so there is no modal wait.
    form1_wid=show("form1");
    // Disable all forms except form1_wid.

    disabled_wid_list=_enable_non_modal_forms(0,form1_wid);
    gcancel=0;
    for (i:=1;i<=100;++i) {
        // Read mouse, key, and all other events until none are left or
        // until the variable gcancel becomes true.
        process_events(gcancel);
        if (gcancel) {
            break;
        }
        // Do work here. Replace the delay below with the operation you
want to do.
        delay(10);

        form1_wid.ctlgauge1.p_value=i;
    }
    // Enable all forms that were disabled.
    _enable_non_modal_forms(1,0,disabled_wid_list);
    form1_wid._delete_window();
}
defeventtab form1;
ctlcancel.lbutton_up()
{
    gcancel=1;
}
```

Defining Events and Event Tables

Event tables are used for describing event or key bindings by source code, creating event-driven dialog boxes, and describing inheritance.

def Primitive

The **def** primitive is used to bind a key sequence or event to a command or procedure and is not typically used when creating event-driven dialog boxes. The **defeventtab** primitive selects the active event table that the **def** primitive sets the bindings to. If there is no **defeventtab** declaration before the first **def** primitive, the **default_keys** event table is used. The **default_keys** event table defines the event handlers for Fundamental mode. The source code representing the bindings is translated and then the event tables are loaded either by the **load** command or by executing the module as a batch program. For more information on batch programs, see [defmain: Writing Slick-C® Batch Files](#). Even though executing the module as a batch program unloads the module when the **defmain** function terminates, the event table changes remain present. The following syntax is used for defining a key:

```
def {prefix_key} event [- event] [, event [- event]] ... = [command];
```

command can be either a command (defined with **_command**) or global procedure. If *command* is not specified, the existing event is unbound. The words *prefix_key* and *event* may be any valid event name. Some event names do not need to be enclosed in quotes.

Example:

```
def "A-x"=safe_exit;
// Note that "A-a" is different than "A-A" which
// requires the Alt and Shift keys to be pressed.
def "A-?"=help;
def "C-X" "b"=list_buffers;
def \0 - \255= nothing;
```

The **defeventtab** primitive is used to define a new event table. The syntax for defining a an event table is:

```
defeventtab name;
```

name may contain a period (.) character. The period is used to separate the form name from the control name. The **def** primitive changes the binding of events of the last event table defined. If no event table is defined, the **default_keys** event table is used. The **default** keyword may be used in place of *name* to end the scope of the current event table and reset back to the default event table.

Example:

```
defeventtab c_keys;
```

```
def " " =c_space;  
def "ENTER" =c_enter;
```

Event tables are global in scope. When an event table is loaded by the **load** command or by executing the module as a batch program, the new bindings replace the event bindings of the existing event table. If the event table specified by **defeventtab** does not exist, a new one is created.

Event-Driven Dialog Boxes

Event tables are for creating event-driven dialog boxes and inheritance. The event table definition code is automatically inserted by the dialog editor. To begin working with event tables, see [Creating Dialog Boxes](#). To attach an event table to a form (dialog box outer window) or form control, define an event table with the same name (**p_name** property) as the form. Dot the form name with the control name if you want to specify inheritance for an event table that is attached to a control.

Example:

```
defeventtab form_name[.control_name] [_inherit [etab_name]];
```

Using the **_inherit** primitive, you can link one event table to another. This makes it possible to perform Clipboard Inheritance® (see [Clipboard Inheritance®](#)). If no name follows the **_inherit** keyword, the inheritance is unlinked. To add event handlers using the **def** primitive or by defining an event handler function, use the following syntax:

```
[ReturnType] ctl_name.event [- event] [, event [- event]] ... ( [ArgDecl1,  
ArgDecl2,...])  
{  
    statements  
}
```

If *ctl_name* is the same name as the last event table form name (name before dot), the event handler is attached to an event table named *form_name*. Otherwise, the event handler is attached to an event table named *form_name.ctl_name*.

The word *event* in the previous code can be any valid event name. Some event names do not need to be enclosed in quotes. It is a best practice to always enclose the event names in quotes.

The syntax for *ArgDecls* is the same as is the syntax for declaring a variable except that the **static** keyword may not be used. An ampersand (&) before the *id* declares a call by references parameter. Call by reference array and hash table parameters require parentheses around the ampersand and *id*.

The following is an example of a form with a text box and **OK** button:

```
#include "slick.sh"  
// Define an event table for the dialog box window.  
defeventtab form1;  
  
// Since this is the first event handler defined for this control  
// and the name of this control does not match the last defined event, the  
// table, the Slick-C translator automatically defines the event table  
// form1.ctlcommand1 and defines the lbutton_up event handler within  
// this new event table.
```

```
void ctlcommand1.lbutton_up( )
{
    // Set the p_text property of the text box control.
    ctltext1.p_text="Hello World";
}
```

When the above code is loaded with the **load** command (**Macro → Load Module**), the editor attaches the **form1.ctlcommand1** event table to a control named **ctlcommand1** on **form1**. A **form1** event table is not created because an event handler for this event table was defined. When you save the configuration, event tables that are not used are deleted.

Module Initializations

The Slick-C® language provides two module initialization functions called **definit** and **defload**. If the two are present, the procedures **definit** and **defload** are called when a module is loaded. The **definit** module is called before **defload**. When the module is saved by the **write_state** command, the **definit** procedure is invoked each time the editor is invoked. This gives your module an opportunity to perform initializations such as creating a temporary file, or allocating a selection, or bookmark. The following syntax is used for defining the special functions **definit** and **defload**:

```
definit()
{
    statements
}
defload()
{
    statements
}
```

The return value of these functions is always **void**. You cannot specify an argument to the return statement. To enhance the performance of SlickEdit®, use the **defload** primitive instead of the **definit** primitive. The **definit** primitive forces a module to be loaded when the editor is invoked. When **definit** is called, the expression **arg(1)** indicates whether the module was loaded with the **load** command or when the editor initialized. When a module is loaded, **arg(1)** returns **L**. Otherwise **arg(1)** returns a null string ("").

Example:

```
int gmarkid= -1;
definit()
{
    // If editor was invoked and is using the local state file
    // which already contains this macro file.
    if (arg(1)!="L") {
        // Must manually initialize this variable.
        gmarkid=-1;      // indicate no mark is allocated.
    } else {
        // If this macro file has been loaded for the first time,
        // the variable gets its initial value from the
        // variable declaration "int gmarkid= -1;".
        // If this macro file has been reloaded,
        // retain the variables state.
    }
}
```

There are two subtle points to this example when assuming that the **gmarkid** variable is used to contain

an allocated mark id (also called selection handle). First, the variable **gmarkid** is scoped as global and not static. This is because the mark needs to remain allocated when this module is reloaded. When the module is reloaded, an unload of the module occurs first and the **_free_selection** built-in is not called to free a mark already allocated (there is no **defunload** primitive). Modules with static variables (module scope) lose their value when reloaded. Second, the value of **arg(1)** is used to make sure that the variable **gmarkid** is initialized only when the editor is invoked and not when the module is loaded. Use this as a template for creating a temporary buffer in the hidden window.

Example:

```
#include "slick.sh"
void definit()
{
    get_view_id(view_id);
    activate_view(HIDDEN_VIEW_ID);
    status=find_view(".bookmark");
    if (status) {
        /* Create a buffer and view in hidden window. */
        status=load_files("+c +t");
        if (status) {
            // The nls function may be used for national language support
            // in the future.
            _message_box(nls('Could not create bookmark buffer'))
            return;
        }
        p_buf_name=".bookmark";_delete_line();
        p_buf_flags= THROW_AWAY_CHANGES|HIDE_BUFFER|KEEP_ON_QUIT;
    }
    // Note: ELSE case cannot empty bookmark buffer unless mark ids
    // are freed. Might as well leave them.
    get_view_id(bookmark_view_id);
    activate_view(view_id);
}
```

Compiling and Loading Macros

The commands **st** and **load** are used to compile Slick-C® modules from within the editor. The **st** command translates the module specified into binary code. When a module is not specified, the current buffer is translated. The **load** command (**F12** or **Macro → Load Module**) translates the module specified if necessary, and loads the resulting byte code. When a module is not specified, the current buffer is saved, translated, and loaded. If a module is loaded that has already been loaded, it is replaced. Both the commands invoke the external program **vstw.exe** (UNIX: **vstw**) to translate the source module into byte code. DO NOT use the **load** command on batch programs. After doing so, you are no longer able to execute the batch program until you use the **unload** command (**Macro → Unload Module**).

A module that is loaded with the **load** command can be unloaded using the **unload** command (**Macro → Unload Module**). However, the symbol table or the names table still contains the names of globally scoped variables, procedures, and commands until you save the configuration. The configuration is automatically saved when you exit the editor. You can invoke the **save_config** command from the command line to save the configuration at any time.

Debugging Macros

The Slick-C® translator **vstw.exe** (UNIX: **vstw**) enables debug messages to be inserted into the code and compiled. Use the **messageNwait** function to display a message and wait until a key is pressed. The **_message_box** function can be used to display a dialog box with a message and wait until you press **Enter** to proceed. Useful defs tab .e extension aliases are listed in the table below.

Alias Name	Value
m	messageNwait(%\n: %\c);
mb	_message_box(%\n: %\c);

The following sections will help you debug and work on Slick-C macros:

- [Finding Procedures](#)
- [Finding Run-Time Errors](#)
- [Performance Profiling](#)
- [Slick-C® Debugger](#)

Finding Procedures

The **find_proc** command (**Macro → Go to Slick-C Definition**) finds Slick-C® source code or Help for a Slick-C symbol name that you specify. Use this function if you are browsing a macro and you want to find out more about a function. You can find the procedure at the cursor by pressing **Ctrl+Dot**. The syntax of the **find_proc** command is:

```
find_proc proc_name
```

Tip

Instead of **find_proc**, use the command **fp**, which is a shortcut. It functions exactly the same as **find_proc**.

The table below shows some examples of using **find_proc** on the command line.

Command	Description
find_proc find_proc	Finds the source code for find_proc .

Command	Description
find_proc cursor_up	Finds the source code for cursor_up .
find_proc substr	Displays Help on substr built-in.

Finding Run-Time Errors

When a Slick-C® error occurs, a dialog box with the title "Slick-C Error" is displayed. Usually the Slick-C Stack tool window is displayed listing the call stack at the time of the error. Double-click in this tool window to view source for a call stack entry. The **find_error** command (**Macro → Find Slick-C Error**) finds the last Slick-C interpreter run-time error. The module with the error is loaded and the cursor is placed on the line causing the error.

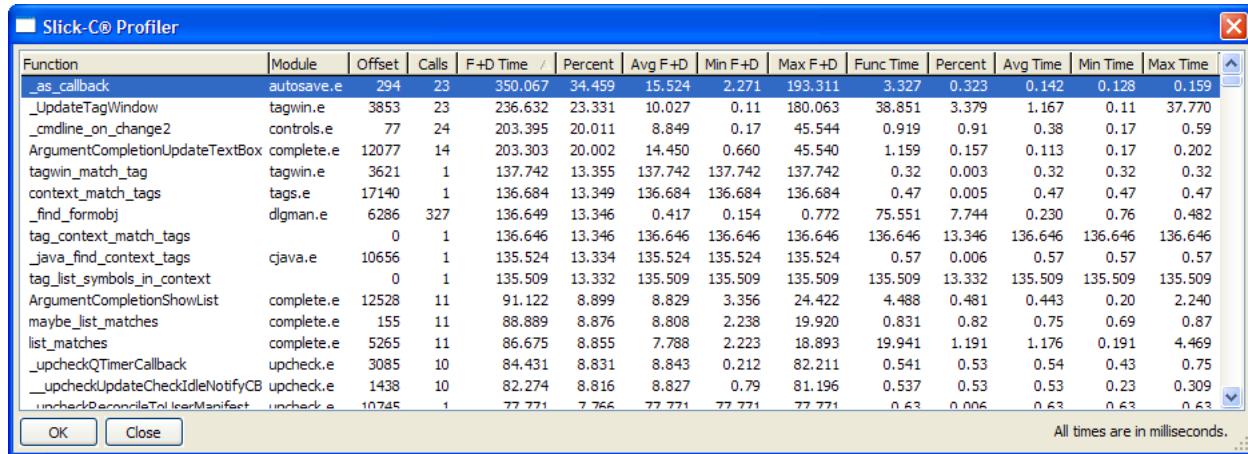
Performance Profiling

The Slick-C® interpreter supports performance profiling. This is useful to identify bottlenecks or other inefficiencies in Slick-C code. The profiler does not affect performance when it is inactive, and there is only a minimal effect on performance when it is collecting data.

To use this feature, invoke the **profile** command on the SlickEdit® command line with the following options:

- **profile on** - Starts profiling data collection (also resets counters).
- **profile off** - Stops profiling data collection.
- **profile view** - Displays profiling data (also stops collection).
- **profile command args** - Executes the specified Slick-C command with the specified arguments, then displays the profiling data. For example, to profile a CVS update, type **profile cvs-gui-mfupdate**.
- **profile save** - Saves the profiling data for loading/viewing at a later time.
- **profile load** - Loads previously saved profiling data for viewing.

Prior to displaying the profiling data, the applicable Slick-C source files are scanned in order to resolve the names of static functions. Then the Slick-C Profiler dialog is displayed showing the data in multi-column, non-modal tree format. Each line represents one function, which is either a Slick-C function or an exported DLL function, depending on what was called when the profiling data was collected. All times are displayed in milliseconds.



The screenshot shows a Windows application window titled "Slick-C® Profiler". Inside, there is a table with 15 columns. The columns are: Function, Module, Offset, Calls, F+D Time / Percent, Avg F+D, Min F+D, Max F+D, Func Time, Percent, Avg Time, Min Time, and Max Time. The data in the table includes various function names like "as_callback", "UpdateTagWindow", "cmdline_on_change2", etc., along with their respective module names, offsets, call counts, and performance metrics in milliseconds. At the bottom of the window, there are "OK" and "Close" buttons, and a note stating "All times are in milliseconds."

Function	Module	Offset	Calls	F+D Time / Percent	Avg F+D	Min F+D	Max F+D	Func Time	Percent	Avg Time	Min Time	Max Time
_as_callback	autosave.e	294	23	350.067 / 34.459	15.524	2.271	193.311	3.327	0.323	0.142	0.128	0.159
_UpdateTagWindow	tagwin.e	3853	23	236.632 / 23.331	10.027	0.11	180.063	38.851	3.379	1.167	0.11	37.770
_cmdline_on_change2	controls.e	77	24	203.395 / 20.011	8.849	0.17	45.544	0.919	0.91	0.38	0.17	0.59
ArgumentCompletionUpdateTextBox	complete.e	12077	14	203.303 / 20.002	14.450	0.660	45.540	1.159	0.157	0.113	0.17	0.202
tagwin_match_tag	tagwin.e	3621	1	137.742 / 13.355	137.742	137.742	137.742	0.32	0.003	0.32	0.32	0.32
context_match_tags	tags.e	17140	1	136.684 / 13.349	136.684	136.684	136.684	0.47	0.005	0.47	0.47	0.47
_find_formobj	dlgman.e	6286	327	136.649 / 13.346	0.417	0.154	0.772	75.551	7.744	0.230	0.76	0.482
tag_context_match_tags		0	1	136.646 / 13.346	136.646	136.646	136.646	136.646	13.346	136.646	136.646	136.646
_java_find_context_tags	cjava.e	10656	1	135.524 / 13.334	135.524	135.524	135.524	0.57	0.006	0.57	0.57	0.57
tag_list_symbols_in_context		0	1	135.509 / 13.332	135.509	135.509	135.509	135.509	13.332	135.509	135.509	135.509
ArgumentCompletionShowList	complete.e	12528	11	91.122 / 8.899	8.829	3.356	24.422	4.488	0.481	0.443	0.20	2.240
maybe_list_matches	complete.e	155	11	88.889 / 8.876	8.808	2.238	19.920	0.831	0.82	0.75	0.69	0.87
list_matches	complete.e	5265	11	86.675 / 8.855	7.788	2.223	18.893	19.941	1.191	1.176	0.191	4.469
_upcheckQTimerCallback	upcheck.e	3085	10	84.431 / 8.831	8.843	0.212	82.211	0.541	0.53	0.54	0.43	0.75
_upcheckUpdateCheckIdleNotifyCB	upcheck.e	1438	10	82.274 / 8.816	8.827	0.79	81.196	0.537	0.53	0.53	0.23	0.309
inheritedVariablesToClearManifest	incheck.e	10745	1	77.771 / 7.765	77.771	77.771	77.771	0.63	0.006	0.63	0.63	0.63

The profiling data can be sorted by clicking any sortable column. Double-click on any function to open the associated file in SlickEdit, with the cursor at the function location.

The Slick-C Profiler displays the following columns:

- **Function** - Name of the function called.
- **Module** - Name of the module from which the function comes.
- **Offset** - The P-code offset of the function within the module.
- **Calls** - Number of calls to the function.
- **F+D Time** - Total time spent in the function and its descendants.
- **Percent** - Percentage of the total time spent in the function and its descendants.
- **Avg F+D** - Average time spent in the function and its descendants.
- **Min F+D** - Minimum time spent in the function and its descendants.
- **Max F+D** - Maximum time spent in the function and its descendants.
- **Func Time** - Total time spent in the function only.
- **Percent** - Percentage of the total time spent in the function.
- **Avg Time** - Average time spent in the function.
- **Min Time** - Minimum time spent in the function.
- **Max Time** - Maximum time spent in the function.

Slick-C® Debugger

The Slick-C Debugger helps you trace Slick-C code. The debugger has no effect on performance when it is inactive, and only a minimal effect on performance when it is running.

To activate the Slick-C Debugger, from the main menu, click **Macro** → **Start Slick-C® Debugger**, or use the **slickc_debug_start** command on the SlickEdit® command line.

When you start the debugger, a separate instance of SlickEdit launches in debug mode (the "debugger instance") and attaches to the original instance of SlickEdit (the "debuggee"). In the debugger instance, you can set breakpoints, step through code, inspect globals and properties, and more.

Use the Debug menu items or key bindings to perform debug operations. See "debugging" in the Help system (**Help** → **Index**) for more information about how to use the debugger in SlickEdit and other options that are available.

You can also use the **slickc_debug** command on the SlickEdit command line to perform various actions:

- **Step into commands** - Use **slickc_debug** *command*, where *command* is the SlickEdit command you want to step into. The debugger terminates when the command completes. For example, use **slickc_debug list_tags** to launch the debugger and step into the **list_tags** command, which scans the current buffer for tags and displays them in a selection list.
- **Debug batch macros** - Use **slickc_debug** *PathToBatchMacro* to activate the debugger for the specified batch macro. For example, use **slickc_debug C:TEMP\bm164.e** to open the batch macro file *bm164.e* in the editor and start the debug session.
- **Enable remote attachment** - Use **slickc_debug on** to enable debugging so that someone else can attach to your instance of SlickEdit remotely. Use the **slickc_debug off** to disable debugging.

To stop the debugging session, from the debug instance main menu, click **Debug** → **Stop Debugging**. This detaches the debugger instance and closes it.

The debugger instance connects to the debuggee using a lightly extended version of JDWP (Java Debug Wire Protocol), although there is no JVM (Java Virtual Machine) involved. By default, it attaches to port 8003.

In order to run in a safe, clean environment, the Slick-C Debugger creates and uses its own configuration directory, named **SCDebug**, located in the user config. Additionally, an empty workspace is created and stored in the debug config which is used thereafter each time the debugger is run. This workspace, **SCDebug.vpw**, is used to store breakpoints that you set in the debug instance. It also stores the list of open files and watch expressions.

The Loaded Classes tool window is a useful tool for examining the state of the debuggee with respect to Slick-C. It shows all the loaded modules and loaded classes, all global variables, all MISC_TYPE variables, and loaded event tables. Many of these items are found under the imaginary "sc.lang.*" namespaces. The Loaded Classes tool window is not active by default in debugging mode. To display it, from the main menu, click **Debug** → **Windows** → **Loaded Classes**. See "Loaded Classes tool window" in the Help system (**Help** → **Index**) for more information.

Error Handling and the rc Variable

The **rc** variable is a predefined global variable that is accessible from all loaded modules. The following functions require that you use the **rc** variable for error handling: **buf_match** and **get_env**.

By convention, functions that use integer error codes return negative error codes that correspond to the error codes in `rc.sh`. For these functions, **0** means success and positive codes means the error code is not in `rc.sh`.

Some functions display an error message on the message line. Use the **clear_message** function to clear the message.

Example:

```
// Cause a message.  
_deselect();  
_copy_to_cursor();  
// Clear the message.  
clear_message();
```

Dialog Editor

The dialog editor is used to create dialog boxes: It provides controls to build the text boxes, combo boxes, radio buttons, image controls, menu items, and forms for a dialog box.

Microsoft Visual Basic and Slick-C®

Creating event-driven dialog boxes (see [Event-Driven Dialog Boxes](#)) in Slick-C is similar to Microsoft Visual Basic except that the language has C++-style syntax. The following list contains some of the differences between Slick-C and Microsoft Visual Basic:

- When an event is sent to a control or dialog box, the object receiving the event MUST be the active object (not necessarily the same as the system focus). This is a major difference between Slick-C and Microsoft Visual Basic. If a button control receives an event and executes a statement such as this: **p_caption=New button caption**, the caption on the button is changed and NOT the caption for the dialog box.
- Built-in properties all start with the prefix **p_** to avoid these keywords from conflicting with their own identifiers.
- A more general method of object instance referencing is used.
- Almost all properties that can be accessed at design time can also be accessed at run time. For example, the **p_name** property for a control or dialog box may be set after the dialog box is displayed.
- Event tables are used to group event handlers for controls. Event tables in Slick-C are used in a similar fashion to classes in C++.
- Slick-C has sophisticated and powerful Dialog Box Inheritance Order. For more information, see [Dialog Box Inheritance Order](#).
- Parent, child, next, and previous (**p_parent**, **p_child**, **p_next**, **p_prev**) creation order relationships are all maintained when dialog boxes are created.
- Event tables can be linked together. One event table can inherit the event handlers of another event table. The event table links can be changed at run time.
- The dialog editor allows event tables to be transferred through the clipboard. Controls from the same or different dialog boxes may reference the same event tables. There is no need for control arrays. For more information, see [Clipboard Inheritance®](#).
- Functions can be used as methods that operate on an instance of an object.

Creating Dialog Boxes

This chapter contains the following topics:

- [Dialog Editor Summary](#)
- [Adding and Deleting Controls](#)
- [Setting Properties](#)
- [Aligning Controls](#)
- [Sizing Controls](#)
- [Moving Controls](#)
- [Miscellaneous Assignments When the Form is Active](#)
- [Miscellaneous Menu Items](#)
- [Creating a Form](#)
- [Saving a Form](#)
- [Adding Event Handlers](#)
- [Inherited Code Found Dialog Box](#)
- [Loading and Running the Form](#)
- [Modal and Modeless Dialog Boxes](#)
- [Dialog Box Parent Window](#)

Dialog Editor Summary

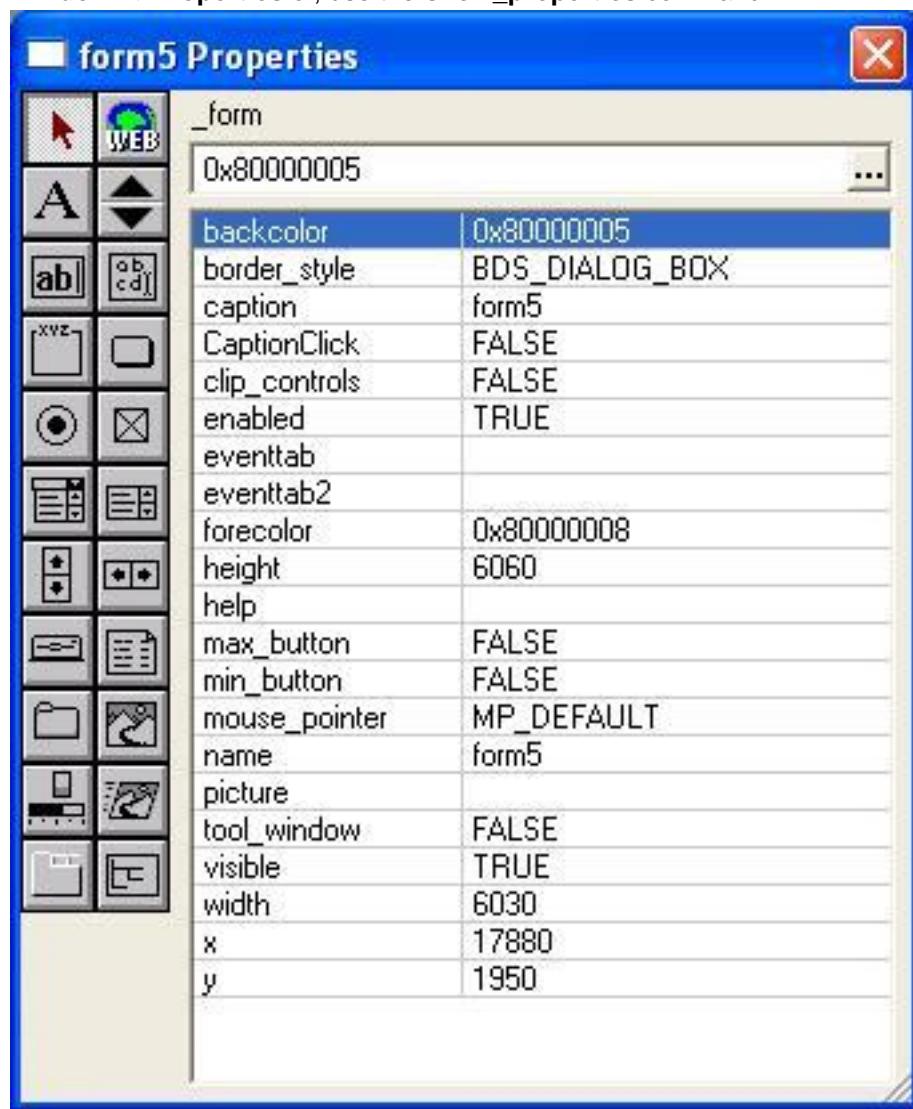
To edit a dialog box that is being run, press **Ctrl+Shift+Space** or right-click on the top of a form and select **Edit**. If you press **Ctrl+Shift+Space** while the Properties dialog box is active, you edit the Properties dialog box. Double-click the system menu to close the edited Properties form. Some UNIX window managers do not close windows when you double-click on the system menu.

Adding and Deleting Controls

The bitmaps on the left of the Properties dialog box are used to create controls. Hover over a bitmap to display the function of a bitmap. There are two methods for creating a control. The first method is to double-click the left mouse button on the bitmap of the control that you want to create. This places a new control in the middle of the selected form.

The **Picture Box** and **Frame** controls enable you to place controls inside of them. To do so, select

Window → Properties or, use the **show_properties** command.



To use the other method for creating a control, complete the following steps:

1. Single-click on the **Text Box** bitmap.
2. Move your mouse so that it appears on top of the form that you are editing. If you cannot see the form that you are editing, display it by selecting **Window → Selected Form**.
3. To create the text box control, click the left mouse button, and, while holding it down, move the mouse pointer to the right to create a dotted rectangle. When you release the mouse, the text box control is displayed within the rectangle.

To delete a control, select the control(s) to remove, then press **Backspace** or **Delete**.

Setting Properties

To set properties, complete the following steps:

1. Select the control. Left-click the mouse button on the property in **Properties** list box.
2. Type the new value in **Properties** combo box. Press **Enter** when the **Properties** list box is active to set the property.
3. Select the control. Double-click the left mouse button on the property in the **Properties** list box to go to the next value of the property. For **color** and **picture** properties, a dialog box is displayed.

Aligning Controls

Select the control with which you want to align the other controls. Select the other controls with **Shift+LButton**. Double-click the left mouse button on one of the properties **x** or **y** to align the controls in the **x** or **y** direction. Press **Enter** on the value in the **Properties** combo box.

Sizing Controls

To size controls, use one of the following methods:

- To size a single control, select the control and click and drag one of the selection handles with the left mouse button.
- To size multiple controls, select the controls and set the **width** or **height** property.
- To size multiple controls, select the controls and press **Shift+Left**, **Shift+Right**, **Shift+Up**, or **Shift+Down** to move the lower right corner of the selected controls by one pixel.

Moving Controls

To move controls, use one of the following methods:

- Select the control(s), then click and drag with the left mouse button.
- Select the control(s), then set the **x** or **y** property.
- Select the control(s), then press the **Left**, **Right**, **Up**, or **Down** arrow key to move the selected controls by one pixel.

Miscellaneous Assignments When the Form is Active

The table below shows a list of miscellaneous button and key assignments that can be used when the form is active.

Assignment	Action
Right mouse click	Displays menu with various dialog editor commands.
Ctrl+Shift+Space	Loads form and Slick-C® code. Runs dialog box. If you accidentally press Ctrl+Shift+Space when in the Properties dialog box, you will be editing the Properties dialog box. Double-click on the system menu to close the edited Properties form. Some UNIX window managers do not close windows when you double-click on the system menu.
Ctrl+S	Loads form and saves into state file. Under UNIX, this may just list source for the form that can be executed.
Ctrl+L	Loads form.
Ctrl+C	Copies selected controls.
Ctrl+V	Pastes controls from the clipboard.
Ctrl+X	Cuts selected controls.
Ctrl+A	Selects all controls with same parent as the already selected control(s).
Tab	Deselects all controls and selects next control in tab order (p_tab_index).
Shift+Tab	Deselects all controls and selects previous control in tab order.
Left mouse click	Double-click (on control) displays Select an Event Function dialog box for adding or modifying event handlers.

Miscellaneous Menu Items

The table below shows the miscellaneous menu items.

Menu Item	Description
System Box of form, Show Properties	Display Properties dialog box.
Window → Properties	Display Properties dialog box.
Window → Selected Form	Display selected form (form being edited).
Macro → New Form	Creates a new dialog box with a default name.
Macro → Open Form	Open existing dialog box or create new dialog box.
Macro → Grid	Sets the distances between the dots on edited form.

Creating a Form

A form is the outer window of a dialog box. The objects within the dialog box are called controls. The form also refers to the entire dialog box. A new form can be created by using one of the following methods:

- Use the **New Form** menu item (**Macro → New Form**).

OR

- Use the **Open Form** menu item (**Macro → Open Form**) and specify the name of a new form.

Saving a Form

Click on the form being edited and press **Ctrl+S**.

Inserting a Form

You can insert a form's definition into a file by using the **Insert Form or Menu Source...** menu item (**Macro → Insert Form or Menu Source...**). Select the dialog you want to insert, and the selected item's source will be inserted into the current file. This command inserts the dialog source only, which defines the object's properties. It will not insert any event tables that have been defined for the object. You do not need to insert the source into a file to use the dialog, as it is automatically saved in your configuration. This command is useful when you want to share your dialog with another user.

Adding Event Handlers

Set the form name and the control names (**name** property in **Properties** list box) before adding code to

Inherited Code Found Dialog Box

the dialog box because these names are referenced in the code. Prefix your control names using the letters **ctl** so that they are easily recognizable. To add an event handler, complete the following steps:

1. Double-click on the control in the dialog box for which you want to add code (not the bitmap in the Properties dialog box). The Select An Event dialog box is displayed.
2. Select an event and click **OK**. If this is the first event handler for this dialog box, you will be prompted with an Open dialog box for a new file to contain the source code for this dialog box.
3. Type a unique file name. Usually this file name is derived from the name of the dialog box you are creating, such as `form1.e`.

After performing the above steps, the dialog editor inserts an event function definition into your source file and places your cursor in the function.

Inherited Code Found Dialog Box

This dialog box is displayed when there is no code for the event you have chosen and the control is using an inherited event table. You will see this dialog box if you copy a control with existing code, paste elsewhere and then double-click on the new instance of the control.

The following options are available on the dialog:

- **Inherit code** - When this option is selected, a statement which links a new event table to an inherited event table (event table not belonging to the control and possibly copied through the clipboard). This affects user level 1 inheritance code (`p_eventtab`) only.
- **Go to inherited code** - When this option is selected, no code is inserted. The cursor is placed on the existing inherited event handling code.
- **Don't inherit code** - Select this option when you do not want to inherit the existing user level 1 inheritance code (`p_eventtab`). Sometimes when you copy a control with existing code to the clipboard, you will not want to inherit the existing event handlers.

Loading and Running the Form

To run the current dialog box that you are editing, click **Macro → Load and Run Form** or use the **run_selected** command. This loads the code, loads the dialog box, and runs the dialog box. To close the dialog box, double-click on the system menu (some UNIX window managers do not close windows when you double-click on the system menu) or press **Ctrl+Shift+Space** (in the running version of your dialog box and not the edited copy). Press **Ctrl+Shift+Space** when any dialog box is running to edit it (this includes the Properties dialog box).

Display the dialog box from the command line by typing **show <FormName>**. To display the dialog box modally enter **show -modal <FormName>** on the command line. For more information about this command, see [Displaying Dialog Boxes](#). Dialog box templates and compiled macros are stored in the state file `vslick.sta`.

The example code below shows how to write a command that displays a dialog box. This is used when binding a command to a key that displays a dialog box.

```
#include "slick.sh"
_command void run_form1()
{
    // The -modal option displays other windows
    // while the dialog box is displayed.
    show( "-modal form1");
}
```

Adding a Cancel Button

To add a **Cancel** button, complete the following steps:

1. Double-click **Insert Button Control**.
2. Set the **caption** property to **Cancel**.
3. Set the **cancel** property to **TRUE** by double-clicking the left mouse button on the **cancel** property in the **Properties** list box.
4. Set the **name** property of a control (never the form) to "" if you are not going to reference the control by name.
5. Clicking **Cancel** when your dialog box is running will close the dialog box even though you have not written any code. If you do add code to your **Cancel** button, you must close the dialog box by typing the following in the command line: **p_active_form._delete_window();**

Adding an OK Button and Closing a Dialog Box

To add an **OK** button and close a dialog box, complete the following steps:

1. Create a command button control by double-clicking **Insert Button Control** in the Properties dialog box.
2. Set the **caption** property to **OK**, set the **default** property to **TRUE**, and set the **name** property to **ctlok**.
3. Double-click on the command button control in the dialog box for which you want to add code (not the bitmap in the Properties dialog box). The Select An Event dialog box is displayed.
4. Choose the **Ibutton_up** event and click **OK**.
5. If this is the first event handler for this dialog box, the Open dialog box for a new file to contain the source code for this dialog box is displayed. Type a unique file name. Usually this file name is derived from the name of the dialog box that you are creating, such as **form1.e**.

After completing the previous steps, the dialog editor inserts an event function definition into your source file and places your cursor in the function. Add the code as shown in the following example:

```
#include "slick.sh"
defeventtab form1
// Code for OK button.
void ctlok.lbutton_up()
{
    // Close the dialog box and return a value. The _delete_window
    // function allows modal dialog boxes to return a value. For
    // more information, see "Displaying Dialog Boxes" below. Each
object
    // in the dialog box will receive an on_destroy event.
    // NOTE: If "" is a valid return value. Return 1 here and store
    // your results in the global _param1 variable.
    p_active_form._delete_window("return value");
    // Statements after closing a dialog box are executed.
}
```

Before closing a dialog box, review the following information:

- First, if a modal dialog box returns a value, the value "" (zero length string) MUST be returned to indicate that the dialog box has been canceled. This convention is used so that when running a dialog box, you can press **Ctrl+Shift+Space** to safely cancel and edit the dialog box.
- Use the global container variables **_param1.._param10** to return multiple strings. Alternately, you can make an array or structure and place it in **_param1**. If you do place your string results in the global variables **_param1.._param10**, make sure your dialog box returns 1 (or any value other than "") to indicate that the dialog box was not canceled.

Displaying Dialog Boxes

The **show** command is called in function-style syntax from within a macro. It can also be invoked from the command line or a menu item.

The command line call syntax is:

```
show cmdline
```

cmdline is a string in the format:

```
[option] form_name
```

The function call syntax is:

```
show (cmdline [,arg1 [,arg2 ... [argN]]])
```

option can be one of the options in the table below.

Option	Description
-mdi	Keep the form on top of the MDI window.
-app	Keep the form on top of the SlickEdit® application window. This allows the MDI window to be displayed on top of the form.
-xy	Restore the previous x,y position of the dialog box. If the old position cannot be found, the dialog box is centered. When the dialog box is closed, the x,y position is automatically saved (the dialog manager calls _save_form_xy).
-hidden	Do not make the form visible. Run the form modally. All other forms are disabled. Control returns to the caller when the form window is deleted with _delete_window .
-nocenter	Do not center the form.
-new	Normally, when a form is already displayed, the existing form is given focus. This option allows for multiple instances of a form to be displayed.
-reinit	(UNIX only) This option causes the _delete_window function to make the form invisible instead of deleting the form. The destroy events are dispatched even though no windows are actually destroyed. Next time show is called for the same dialog box, the invisible dialog box is made visible, some properties are reinitialized, and the create events are sent. Be careful when using this option. Not all dialog boxes can use this option without minor modifications. The form_parent() function does not work because the next time the form is used, the parent is not changed to the new parent specified.
-hideondel	(UNIX only) This option is the same as the -reinit option except no properties are reinitialized when the invisible dialog box is shown again.

form_name specifies a form or menu resource. If it is an integer, it must be a valid index into the names

table of a form or menu. Otherwise, it should be the name of an existing form or menu that can be found in the names table.

on_create and on_load Events

The array of args (*arg1...argN*) is passed to **on_create**. When a dialog box and all its objects are created, each object receives an **on_create** event. The **on_create** event receives the *arg1, arg2,...,argN* arguments given to the **show** function. After the **on_create** events are sent, the form receives an **on_load** event. You CANNOT set the final focus in an **on_create** event. Use the **_set_focus** function during the **on_load** event to set the initial focus to a control other than the control with lowest tab index (**p_tab_index**) that is enabled and visible.

Return Value of show

If the **-modal** option is given, the return value given to **_delete_window** is returned. "" is returned if the dialog box is edited or destroyed during an **on_create** event. Use the global variables **_param1..._param10** to return more than one string value. Alternately, you can make an array or structure and place it in **_param1** for non-string return types.

If the **-modal** option is not given, the form window *id* is returned if successful. Otherwise, a negative error code is returned.

Example:

```
// This example requires that you create a form called form1 with a
// command button and load this file.
#include "slick.sh"
_command void mytest()
{
    result := show("-modal form1");
    if (result=="") {
        return(COMMAND_CANCELLED_RC);
    }
    message( "_param1=_param1" "_param2=_param2");
}

defeventtab form1;
void ctlcommand1.on_create()
{
    // Global variable _param1.._param10 are defined in "slick.sh"
to
    // allow for multiple strings to be returned in separate
variables.
    // Alternatively, if the return strings do not contain spaces,
you
    // could concatenate them together with a space and use the
parse
    // built-in to easily separate them.
    _param1="string1";
```

```
_param2="string2";
// Close the dialog box and indicate that the dialog box was not
canceled.
// Each object in the dialog box will receive an on_destroy
event.
p_active_form._delete_window(1);
}
```

Example:

```
// This example requires that you create a form called
// form1 with a command button and load this file.
#include "slick.sh"
_command void mytest()
{
    show("-modal form1", "param1 to on_create", "param2 to
on_create");
}

defeventtab form1;
void ctlcommand1.on_create(_str arg1="", _str arg2="")
{
    _str tmpArg1, tmpArg2;
    tmpArg1=arg(1);
    tmpArg2=arg(2);
    _message_box("arg1=" arg1" arg2=" arg2);
    _message_box("tmpArg1=" tmpArg1" tmpArg2=" tmpArg2);
}
```

Example:

```
#include "slick.sh"
int defmain()
{
    index=find_index("form1",oi2type(OI_FORM));
    if (!index) {
        messageNwait("form1 not found");
        return(1);
    }
    // Can specify name table index instead of name. When show is
    // called
    // without the "-modal" option, the positive window id (instance
    // handle)
    // of the form created is returned.
    form_wid=show("-hidden -nocenter "index);
    if (form_wid<0) {
```

```
        return(1);
    }
    // Place the form at the top left corner of the display.
    form_wid.p_x=form_wid.p_y=0;
    // Make the form visible.
    form_wid.p_visible=true;
    return(0);
}
```

Modal and Modeless Dialog Boxes

If you do not want the MDI window or any other form to get focus when your dialog box is displayed, specify the **-modal** option to the **show** command (see [Displaying Dialog Boxes](#)). When the **-modal** option is given, other forms, including the MDI window, are disabled (**p_enabled=false**) until the form is closed. In addition, the **_delete_window** function can be used to return a value (see the previous example).

Modeless example:

```
#include "slick.sh"
int defmain()
{
    // When show is called without the "-modal" option, the positive
    // window id (instance handle) of the form created is returned.
    form_wid=show("-hidden -nocenter form1");
    if (form_wid<0) {
        return(1);
    }
    // Place the form at the top left corner of the display.
    form_wid.p_x=form_wid.p_y=0;
    // Make the form visible.
    form_wid.p_visible=true;
    return(0);
}
```

If you need to display a status dialog box during processing, you might require a modeless dialog box so control is returned to you. However, it is a best practice to disable all other dialog boxes including the MDI window during processing.

Advanced modeless example:

```
#include "slick.sh"
static typeless gcancel;
_command void test()
{
    // Show the form modeless so there is no modal wait.
```

```
form1_wid=show( "form1" );
// Disable all forms by the one with p_window_id==form_wid. A space-
// delimited string of disabled form window ids is returned.
disabled_wid_list=_enable_non_modal_forms(0,form_wid);
gcancel=0;
for (;;) {
    // Read mouse, key, and all other events until none are left
    // or until the variable gcancel becomes true.
    process_events(gcancel);
    if (gcancel) break;
    // Do your processing here.
}
// Enable the forms that were disabled.
enable_non_modal_forms(1,0,disabled_wid_list);
form1_wid._delete_window();
}
defeventtab form1;
void ctlcancel.lbutton_up()
{
    gcancel=1;
}
```

Dialog Box Parent Window

The parent window of a dialog box form has two uses. First, the dialog box remains on top of the parent window. Use the **show** command and specify the **-app** option if you want to allow a modeless dialog box be displayed behind the MDI window. The **-mdi** option of the **show** command can be used to make sure a dialog box stays on top of the MDI window.

Command line examples:

```
show -app _calc_form
show -mdi -new _calc_form
```

Second, the parent window is used by some dialog boxes (such as the Print and Spelling dialog boxes) to determine on which buffer to operate. This permits the dialog boxes to support the editor control. To do this, they call the **_form_parent** function during an **on_create** event to get the window *id* of the window which contains the buffer to be operated on. These dialog boxes only support certain parent windows. For example, the Print dialog box will not run correctly if the **-app** option of the **show** command is used.

Remembering a Dialog Box's Previous Position

The **show** command centers the dialog box to the current form or MDI window. Usually this is fine, but sometimes it is helpful for a dialog box to reappear in the same position that it was in when the user closed the dialog box. To do this, specify the **-xy** option to the **show** command. This adds the **IS_SAVE_XY** flag to the **p_init_style** property. When the dialog box is closed, the **x** and **y** position of the

dialog box is stored and later saved in the auto restore file (`vrestore.slk` by default) when you exit the editor. The form is centered if the old **x,y** position information cannot be found.

Clipboard Inheritance®

This section contains the following topics:

- [Clipboard Inheritance® Overview](#)
- [Clipboard Inheritance® Example](#)
- [Dialog Box Inheritance Order](#)

Clipboard Inheritance® Overview

Clipboard Inheritance enables the transferring of objects from one place to another using the clipboard to create new instances that inherit the code of the original objects. Code for the new instances can be added that affects only the new instances, and code of the original instances can be modified, affecting both instances.

For example, you may want to create a group of controls that are needed by the SlickEdit® File Open dialog to allow the user to specify the various supported file formats. SlickEdit supports the following file formats:

- **DOS** - Each line is separated with a carriage return, followed by linefeed.
- **Macintosh®** - Each line is separated with a carriage return only.
- **UNIX** - Each line is separated with a linefeed only.
- **Record width** - A user-specified number of bytes placed in each line.
- **Separator character** - A single user-specified line separator character.

The following partial dialog box can be used to handle the file formats of SlickEdit.



Example:

```
// The names of the controls do not need to be declared.  
//
```

Clipboard Inheritance® Overview

```
// The names of the radio button controls are
// ctlopendos, ctlopenmac, ctlopenunix, ctlopenauto.
//
// The first text box is named ctlopenlinesep and
// the text box below it is named ctopenwidth.

defeventtab form1;

// Define the lbutton_up event for the DOS radio button. This function
will
// get called when any of the radio buttons get turned on. The event
// table automatically created here is called form1.ctlopendos.
ctlopendos.lbutton_up()
{
    // Set the text displayed in both text boxes to nothing so the users
    // knows that the radio button format has been chosen.
    ctlopenlinesep.p_text="";
    ctopenwidth.p_text="";
}

static void zap_radio_buttons()
{
    ctlopendos.p_value=0;
    ctlopenmac.p_value=0;
    ctlopenunix.p_value=0;
    ctlopenauto.p_value=0;

}

// Define the on_change event for the first text box. For a text box,
the
// on_change event gets called when the user modifies the text in the
text box.
// The event table automatically created here is form1.ctlopenlinesep.
ctlopenlinesep.on_change()
{
    if (p_text!="") {
        ctopenwidth.p_text=""; // Clear out the other text boxes text.
        zap_radio_buttons(); // Turn off all the radio buttons.
    }
}
// Define the on_change event for the second text box. The event table
// automatically created here is form1.ctopenwidth.
ctopenwidth.on_change()
{
    if (p_text!="") {
        ctlopenlinesep.p_text=""; // Clear out the other text boxes text.
        zap_radio_buttons(); // Turn off all the radio buttons.
```

```
    }  
}
```

Only the first radio button **ctlopendos** has an event handler defined. The other radio buttons use the **form1.ctlopendos** event table. This can be accomplished in the dialog editor using Clipboard Inheritance or, if the radio buttons are already created, you can set the **p_eventtab** property of the other radio buttons to **form1.ctlopendos**.

To use Clipboard Inheritance:

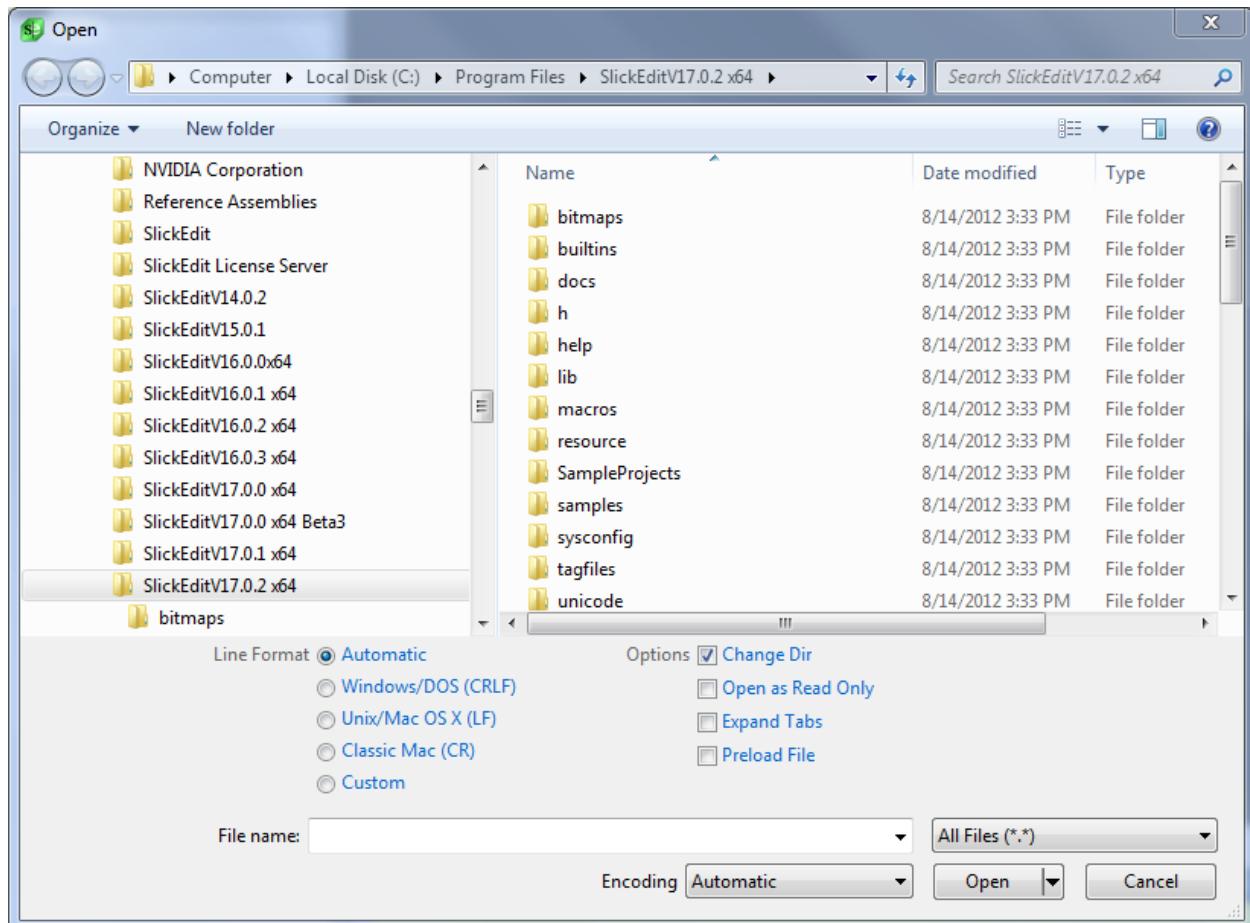
1. Write the **Ibutton_up** event code for the DOS radio button.
2. Copy the DOS radio button to the clipboard.
3. Paste it back onto the dialog box within the frame.
4. Set the **p_caption** property for the new radio button to **MAC**.

Either of these methods can be used to fill an event table. When the **ctlopendos.Ibutton_up()** function gets called, it gets and sets the properties of controls that exist on this dialog box.

Clipboard Inheritance® Example

For the Open dialog, Clipboard Inheritance® was created by copying controls to the clipboard and pasting them.

Dialog Box Inheritance Order



The Open File dialog box has the form name `_edit_form`. This dialog box is created by copying the `_open_form` dialog box (code links and all) to the clipboard, pasting it, and then adding the **Find File** button and the advanced controls. The `_open_file` form can be thought of as the base File Open dialog box class. It is used for all other File Open and Save As operations except for opening files for editing which requires additional controls. The inherited code from the base class File Open dialog required no changes except for the **OK** button. For this, the **OK** button code was replaced with new code. The Find File displays a dialog box which has all of the same advanced controls. The advanced controls were taken from the Open File dialog box (`_edit_form`) and all its related controls, and copied onto the Find File dialog box. The only additional code required was for the **OK** button, which was needed to return the results of the advanced options to the caller.

The following statement highlights the syntax for linking one event table to another:

```
defeventtab dlgbox2.textbox1 _inherit dlgbox1.textbox1
```

Dialog Box Inheritance Order

Each control in Slick-C® has two properties, called `p_eventtab` and `p_eventtab2`. The `p_eventtab` property defines the user level 1 inheritance. User level 1 inheritance permits the modification of the event

handlers for one specific instance of a control without affecting any other (except when Clipboard Inheritance® is used). The dialog editor automatically inserts the necessary function declaration code so that you need to only add statements within the function. After you write the event handler and load the new code, the **p_eventtab** property displayed in the **Properties** list box is updated to reflect that you have defined a user level 1 event table.

The **p_eventtab2** property defines the user level 2 inheritance. User level 2 inheritance is typically used to affect all controls of a specific type. Normally, the dialog editor sets these properties for you when a control is created. For example, when you create a combo box control with the dialog editor, the **p_eventtab2** property is automatically set to **_ul2_combobox**. The **_ul2_combobox** event table defines the default processing used by every combo box. The user level 1 event handler receives an **on_change** event (sent from the user level 2 code) when the text in the combo box changes.

SlickEdit® uses a pre-defined inheritance order called Dialog Box Inheritance Order. When a control receives an event, the following search begins to determine which event handler should get control:

1. IF and ONLY if the event SlickEdit® is searching for is a key event, check the dialog box user level 1 inheritance on the frame of the dialog box.
2. Check current control's user level 1 inheritance.
3. Check current control's user level 2 inheritance.
4. Check automatic inheritance. Only the text box, combo box, and editor window can have any automatic inheritance. This is how your emulation is supported in these controls.
5. Check the dialog box frame user level 1 inheritance.
6. Check the dialog box frame user level 2 inheritance.
7. Check dialog manager inheritance.

As soon as an event handler is found, the search stops and the event handler is executed. Each inheritance level can have up to 20 linked event tables. This limit is only to avoid infinite event table link loops. At run time it is possible, but unusual, to change all inheritance links and event tables for any object. The **eventtab_inherit** function can be used to get or set an event table inheritance link.

Objects and Instances

Every object instance can be uniquely identified by a window id (also called instance handle). Slick-C® treats objects and windows the same. However, some objects, such as image control, have a window id but do not allocate an operating system resource known as a window.

Topics in this section are:

- [Active Object](#)
- [Active Form](#)
- [Instance Expressions](#)

Active Object

When an object receives an event, that object is the active object. More specifically, the **p_window_id** property is set to the instance handle of that object. You can change the active object by setting the **p_window_id** property to the window id of another object. Accessing a property without specifying a control name or instance handle accesses the property of the active object and not the active form.

Note

Changing the active object does NOT change the focus. Use the **_set_focus** method to change the focus.

Active Form

Slick-C® has a **p_active_form** property that returns an instance handle to the current form. The Slick-C interpreter actually does not keep track of what form is active. The active form is found by traversing through the parents (**p_parent**) of the active object until the form is reached.

Instance Expressions

The examples below display common instance expressions.

```
ctltext1.p_text="test"; // Assuming ctltext1 has been declared globally  
or locally,  
                                // lookup the ctltext1 control of the active  
form to get  
                                // the window id, and set the p_text property.  
x=_control ctltext1;    // Put the window id of the "ctltext1" control  
of the active
```

```
// form in the variable x.  
// The variable x does not have to be declared.  
  
There are  
needed. It is  
worry.  
x.p_text="test";  
referenced by the  
(x+1-1).p_text="test";  
any valid  
get the  
x.(x+1-1).x.p_text="test";  
more  
used in an  
form_wid=p_active_form; // Get the window id of the active form.  
form_wid.ctltext1.p_text="test";  
by the  
p_next.p_next.p_prev.p_prev.p_text="test";  
property of  
p_window_id=_control ctltext1;  
p_text="test";  
_cmdline.p_text="test"; // _cmdline is a constant window id defined in  
"slick.sh".  
"test". Cool!!
```

Using Functions as Methods

A command or procedure can be called as a method without any additional declaration data. The sample Slick-C® source below is an example of this feature.

```
#include "slick.sh"
void defmain()
{
    // Call the tbupcase function as a method to operate on the SlickEdit
    // command line. _cmdline is a constant instance handle defined in
slick.sh.
    _cmdline.tbupcase();

}

// This function uppercases the text in a text box or combo box input
field
// and has been written to operate on the current object.
void tbupcase()
{
    // The p_text property is used to get and set the contents of a text
box
    // or combo box input field.
    p_text=upcase(p_text);
}
```

The **tbupcase** is not defined to be a method of a particular class. This feature permits macros written in SlickEdit® text mode to be converted into SlickEdit macros and used as methods. Also, most functions are written to operate on the current object, meaning you have access to many methods. Using functions as methods is useful when writing dialog box event handlers. If a function is called and a statement within the function is not valid for the current object, the macro is stopped, and a dialog box is displayed indicating the error. The **find_error** command (**Macro → Find Slick-C Error**) can then be used to locate the source of the error.

Built-in Controls

Topics in this chapter:

- [Label Control](#)
- [Spin Control](#)
- [Text Box Control](#)
- [Editor Control](#)
- [Frame Control](#)
- [Command Button Control](#)
- [Radio Button Control](#)
- [Check Box Control](#)
- [Combo Box Control](#)
- [List Box Control](#)
- [Scroll Bar Controls](#)
- [Drive List Control](#)
- [File List Box Control](#)
- [Directory List Box Control](#)
- [Picture Box Control](#)
- [Gauge Control](#)
- [Image Control](#)
- [Adding Dialog Box Retrieval](#)

Label Control

The label control is used to display text in any font. A common use of a label control is to place it to the left of a text box to tell the user about what goes in the text box.

Labels can be aligned left or right, or centered horizontally and/or vertically. If you do not need to align the label, set the **p_auto_size** property to TRUE to ensure that the text fits inside the window. To center the label to a text box, select the label control and use the **Up**, **Down**, **Left**, and **Right** arrow keys.

On the Dialog Editor, click the **Insert Label Control** button  to place a label control on a form.

For a complete list of label control properties, methods, and events, from the main menu, select **Help** → **Macro Functions by Category**.

Spin Control

The most common use of a spin control is to increment or decrement a number displayed in a text box. This can be performed WITHOUT writing any code, by making the **tab_index** property of the text box one less than the **tab_index** property of the spin control. An error is displayed if there is no text box with a tab index one less than the spin control, unless the increment property of the spin control is set to zero. To create a spin control, complete the following steps:

1. Create the text box and then create the spin control.
2. Turn off the **auto_size** property of the text box so you can make the height of the text box larger than the font.
3. Use the spin control to increment or decrement the value in a gauge or scroll bar control or increment or decrement a hexadecimal number displayed in a text box. The default increment is 1. Set the **increment** property of the spin control to zero and process the **on_spin_up** and **on_spin_down** events. The **on_change** event is called with a *reason* set to **CHANGE_NEW_FOCUS**, before an **on_spin_up** or **on_spin_down** event, to allow you to return the window ID of the control you want to get focus, after spinning is completed. Return an empty string ("") if you do not want to change the event.

Example:

```
#include "slick.sh"

// This example requires form name form1 with a text box and spin
control.
// The spin control should be named ctlspin1 and the increment property
// should be zero. The tab index of the text box MUST be one less than
// the spin control. This code does not reference the name of the text
box
// so that you can use Clipboard Inheritance(R) to create multiple
working
// copies of a spin control capable of incrementing/decrementing the
value in
// a text box control without writing any new code.
defeventtab form1;
ctlspin1.on_change(int reason)
{
    if (reason==CHANGE_NEW_FOCUS) {
        return(p_prev);
    }
}
ctlspin1.on_spin_up()
```

```
{  
    new_dec_value=hex2dec(p_prev.p_text)+1;  
    p_prev.p_text=dec2hex(new_dec_value);  
}  
ctlspin1.on_spin_down()  
{  
    new_dec_value=hex2dec(p_prev.p_text)-1;  
    p_prev.p_text=dec2hex(new_dec_value);  
}
```

For a complete list of spin control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Text Box Control

The text box control enables the user to enter a single line of text. Editor control determines the number of lines that can be entered. Text boxes support completion with the spacebar and question mark keys. Set the completion property of the text box. The **FILE_ARG** completion type is the most common. It provides completion on file names. New commands can be written that operate in all text boxes, edit windows, and editor controls.

Example:

```
#include "slick.sh"  
_token void upcase_line()  
name_info(' , 'VSARG2_TEXT_BOX|VSARG2_REQUIRES_EDITORCTL)  
{  
    init_command_op();  
    get_line(line);  
    replace_line(upcase(line));  
    retrieve_command_results();  
}
```

Bind the **upcase_line** command in the previous example to **Alt+F12**. This command works in all text boxes, edit windows, and editor controls. The key binding might not work in a text box if you bind the **upcase_line** to one of the CUA keys **Alt+A**, **Alt+Z**, **Ctrl+X**, **Ctrl+C**, or **Ctrl+V**. Use the Redefine Common Keys dialog box (**Tools → Options → Keyboard and Mouse → Redefine Common Keys**) to allow all key bindings to be inherited into text box controls.

For a complete list of text box control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Editor Control

Editor control is used to enter multiple lines, view clipboards, to work with the calculator, and for version

control comments. Almost all of the key bindings for an MDI edit window work in an editor control even when the emulation is changed. Use macro recording to write a new command that works in an edit window and editor control. Mark the **Allow in non-MDI editor control** check box when you finish recording the macro.

For a complete list of editor control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Frame Control

Frame control is used to group a set of related controls. Radio buttons are placed inside of a frame control to indicate to the dialog manager that only one of the radio buttons in the group can be turned on at a time. There are two ways to place a control inside of a frame control:

- Click the left mouse button on the bitmap in the Properties dialog box of the control that you want to place inside the frame. Click and drag with the left mouse button inside the frame control to create the control with the size of the rectangle displayed.
- Copy or cut the control you want to place inside the frame to the clipboard. Select the frame control and press **Ctrl+V** to paste the control inside the frame control.

For a complete list of frame control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Command Button Control

The command button control is most typically used to create an **OK**, **Cancel**, or **Help** button.

For a complete list of command button control properties, methods, and events, from the menu, select **Help → Macro Functions by Category**.

Radio Button Control

Radio buttons must be grouped. When one radio button is enabled, the other radio buttons in the same group are not available. Radio buttons are considered in the same group if they have the same parent. Usually, radio buttons are grouped by placing them inside a picture box or frame control. A picture box can have its **border_style** property set to **BDS_NONE** to display that the picture box control does not exist. Use one of the methods described under [Frame Control](#) to place a radio button inside a frame.

For a complete list of radio button control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Check Box Control

A check box is used to set up a true or false option. Check boxes can be displayed to the left or right of

the caption.

For a complete list of check box control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

List Box Control

A list box provides a way to select from a fixed set of items. Multiple items from the list can be selected at one time by setting the **multi_select** property to MS_SIMPLE_LIST or MS_EXTENDED (used by Open dialog box). A list box receives an **on_change** event, with a reason argument set to CHANGE_SELECTED, when items are selected or deselected because of a key press or mouse event. None of the _lbxxx functions cause an **on_change** event. Use the **_find_longest_line()** function to find the longest line in a list box.

The following example requires a form named "form1", a command button named "ok", and a list box named "ctllist1":

```
#include "slick.sh"
defeventtab form1;
ctllist1.on_change(int reason)
{
    // Check the reason value. In the future we may add more reason
values
    // for the list box.
    if (reason==CHANGE_SELECTED) {
        // IF any items in the list box is selected.
        if (p_Nofselected) {
            ctlok.p_enabled=true;           // Enable the OK button.
        } else if(!ctlok.p_enabled){
            ctlok.p_enabled=false;         // Disable the OK button.
        }
    }
}
```

The following example illustrates how to resize a dialog box based on the longest item in a list box:

```
#include "slick.sh"
defeventtab form1;
ctllist1.on_create()
{
    _lbadd_item("Line1");
    _lbadd_item("This is a longer line2");
    _lbadd_item("This is the longest item in the list box");
    longest=_find_longest_line();
```

```
// Add on a little to account for the left and right borders of the
// list box. Have to convert client width because it's in pixels.
list_width=longest+ p_width-_dx2lx(p_xyscale_mode,p_client_width);
form_wid=p_active_form;

// Again we have to account for the left and right borders.
// Multiply p_x of list box by two to show equal amounts of spacing
on
// each side of the list box.
form_width=2*p_x + list_width+ form_wid.p_width -
_dx2lx(form_wid.p_xyscale_mode,form_wid.p_client_width);

p_width = list_width;
form_wid.p_width = form_width;

// Now make sure the whole dialog box can be seen on screen.
form_wid._show_entire_form();
}
```

The example below illustrates adding pictures to a list box.

```
#include "slick.sh"
#define PIC_LSPACE_Y 60      // Extra line spacing for list box.
#define PIC_LINDENT_X 60    // Indent before for list box bitmap.

defeventtab form1;
ctllist1.on_create()
{
    // Add some extra line height.
    p_pic_space_y = PIC_LSPACE_Y;
    // _pic_xxx arguments are global variables defined in "slick.sh"
which are
    // name table indexes to pictures. You can create and load your own
pictures.
    // All the bitmaps are shipped with the editor. Use the bitmap file
    // "_drremov.bmp" as a template for creating your own bitmap for a
list box.
    // You can load your own bitmap files with the _update_picture
function.
    _lbadd_item("a:",PIC_LINDENT_X,_pic_drremov);
    _lbadd_item("b:",PIC_LINDENT_X,_pic_drremov);
    _lbadd_item("c:",PIC_LINDENT_X,_pic_drfixed);
    // The p_picture property must be set to indicate that this list box
is
    // displaying pictures and to provide a scaling picture for the
    // p_pic_point_scale property. The p_pic_point_scale property allows
the
```

```
// picture to be resized for fonts larger or smaller than the value  
of the  
// p_pic_point_scale point size. If p_pic_point_scale is 0, the  
picture is  
// not scaled.  
p_picture = picture;  
p_pic_point_scale = 8;  
}
```

Finally, the example below illustrates how to disable a list box and make the items in the list box appear grayed.

```
#include "slick.sh"  
defeventtab form1;  
ctllist1.on_create()  
{  
    _lbadd_item("item1");  
    _lbadd_item("item2");  
    p_no_select_color=true;  
    p_enabled=false;  
    p_forecolor=_rgb(80,80,80);  
}
```

For a complete list of list box control properties, methods, and events, from the main menu, select **Help** → **Macro Functions by Category**.

Combo Box Control

A combo box is used in place of a text box for combo box retrieval, when only a fixed set of responses is permitted, or when a common set of responses are known and a different response may be typed in. Combo box retrieval is a mechanism in that the combo list box displays the previous responses entered in the text box of the combo box. The combo box has two style properties:

- The **PSCBO_NOEDIT** style is used when only a fixed set of responses are allowed. Combo boxes support completion with the spacebar and question mark keys. Set the **completion** property of the combo box if there is an existing completion type that suits the needs.
- The **FILE_ARG** completion type is the most common. It provides completion on file names.

The following example illustrates combo box retrieval. The example requires a form named "form1", an OK button named "ctllok", and combo box named "ctlcombo1":

```
defeventtab form1;  
ctllok.lbutton_up()  
{
```

```
// When the OK button is pressed,  
// you need to save combo box retrieve information.  
_append_retrieve(_control ctlcombo1,ctlcombo1.p_text);  
}  
ctllok.on_create()  
{  
    // Fill in the combo box list.  
    ctlcombo1._retrieve_list();  
}
```

A combo box consists of four controls: the root window, text box, picture box, and list box. The properties and methods of the sub-controls may be accessed individually with the **p_cb**, **p_cb_text_box**, **p_cb_picture**, **p_cb_list_box** instance handle properties. The **p_cb_picture** property is only available when the control is displayed.

Example:

```
defeventtab form1;  
ctlcombo1.on_create()  
{  
    // To make the loop a little more efficient,  
    // activate the list box of the combo box control  
    p_window_id=p_cb_list_box;  
    for (i:=1;i<=100;++i){  
        // Add an item to the active list box.  
        _lbadd_item("line="i);  
    }  
    // Activate the root window of the combo box.  
    p_window_id=p_cb;  
}
```

Example:

```
#include "slick.sh"  
defeventtab form1;  
ctlcombo1.on_create()  
{  
    // Show a picture which indicates that clicking on the picture box  
    // button displays a dialog box. _pic_cbdots is a global  
    // variable defined in "slick.sh" which is a handle to a picture.  
    vp_cb_picture.p_picture=_pic_cbdots;  
}  
ctlcombo1.llbutton_down()  
{  
    // Check if the left mouse button was clicked inside the picture box
```

```
// of the combo box.  
if (p_cb_active==p_cb_picture) {  
    result=show( "-modal form2");  
    // Process result here.  
    return("");  
}  
// Skip user level 1 inheritance and execute the default event  
handler  
// defined by user level 2 inheritance.  
call_event(p_window_id,LBUTTON_DOWN,2);  
}
```

The following example requires a form named "form1", command button named "ctllok", a combo box named "ctlcombo1", and another command button named "ctlcommand1":

```
#include "slick.sh"  
defeventtab form1;  
ctllok.lbutton_up()  
{  
    // Check if text in combo box text is valid. You might think you  
could  
    // use a non-editable style combo box. However, many users prefer  
typing  
    // in names using completion rather than using the mouse to select  
an item  
    // out of a list box.  
    status=ctlcombo1._cbi_search( "", "$");  
    if (status) {  
        _message_box("Combo box contains invalid input");  
        return("");  
    }  
    // Have valid input.  
}  
ctlcommand1.lbutton_up()  
{  
    // Add some items to the combo box list.  
    ctlcombo1.p_cb_list_box._lbadd_item("Hello")  
    ctlcombo1.p_cb_list_box._lbadd_item("Open")  
    ctlcombo1.p_cb_list_box._lbadd_item("New")  
    // Make the correct item in the combo box list current so combo box  
    // retrieval works better. _cbi_search searches for p_text in the  
combo  
    // list box. The "$" specifies that an exact match should be found  
and  
    // not a prefix match.  
    int status=ctlcombo1._cbi_search( "", "$");  
    if (!status) {
```

```

        messageNwait("Found it!");
        // Select the line in the combo box so that an up or down arrow
        // selects the line above or below and not the current line.
        ctlcombo1.p_cb_list_box._lbselect_line();
    }
}

```

A combo box receives an **on_change** event with a *reason* argument under the circumstances listed in the table below.

Reason	Description
CHANGE_OTHER	The p_text property changed, probably because of typing.
CHANGE_CLINE	The p_text property changed because selected line in list box changed and the list was visible.
CHANGE_CLINE_NOTVIS	The p_text property changed because a key was pressed which scrolls the list (Up , Down , PgUp , PgDn) while the list was invisible.
CHANGE_CLINE_NOTVIS2	Same as CHANGE_CLINE_NOTVIS . Sent to user level 2 inheritance only. User level 2 inheritance will receive the CHANGE_CLINE_NOTVIS reason as well if the user level 1 inheritance does not catch the on_change event.

The **on_drop_down** event is sent to a combo box with a *reason* argument. The *reason* argument specifies one of the conditions listed in the table below.

Reason	Description
DROP_UP	After combo list box is made invisible.
DROP_DOWN	Before combo list box is made visible.
DROP_INIT	Before retrieve next/previous. Used to initialize list box before it is accessed.
DROP_UP_SELECTED	Mouse released while on valid selection in list box and list is visible.

Example:

```
#include "slick.sh"
defeventtab form1;
ctlcombol.on_drop_down(int reason)
{
    if (reason==DROP_INIT) {
        if (p_user=="") {
            p_user=1; // Indicate that the list box has been filled.
            // Insert a lot of items.
            p_cb_list_box._insert_name_list(COMMAND_TYPE);
            p_cb_list_box._lbsort();
            p_cb_list_box._lbtop();
        }
    }
}
```

For a complete list of combo box control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Scroll Bar Controls

There are two scroll bar controls that operate similarly: **vscroll** and **hscroll** (vertical and horizontal, respectively). The scroll bar controls are used to provide the user an avenue for selecting an integer that has a fixed range or a way for displaying the completion status of a process. Set the min, max, **small_change**, and **large_change** properties to define the minimum integer value, maximum integer value, increment/decrement that occurs when arrows are pressed, and increment/decrement that occurs when you click the left button between the arrow and thumb box respectively.

The **on_change** event is sent after dragging the thumb box is completed. The **p_value** property contains the new scroll position and will be in the range **p_min..p_max**.

The **on_scroll** event is sent while you click and drag the thumb box of a scroll bar.

Example:

```
#include "slick.sh"
defeventtab form1;
ctlvscroll1.on_scroll()
{
    message("on_scroll p_value="p_value);
}
ctlvscroll1.on_change()
{
    message("on_change p_value="p_value);
}
```

For a complete list of scroll bar control properties, methods, and events, from the main menu, select **Help** → **Macro Functions by Category**.

Drive List Control

The drive list is a combo box that allows selection of different disk drives. The Open dialog box uses this control.

The drive list control receives an **on_change** event with a *reason* argument of **CHANGE_DRIVE** when the drive is changed by selecting a different drive from the combo list box.

Example:

```
#include "slick.sh"
defeventtab form1;
ctlcombo1.on_change(int reason)
{
    if (reason==CHANGE_DRIVE) {
        message("Item selected from list. Current drive is now
":+_dvldrive());
    }
}
```

For a complete list of drive list control properties, methods, and events, from the main menu, select **Help** → **Macro Functions by Category**.

File List Box Control

The file list box control displays a list of files. Multiple files can be selected by setting the **multi_select** property to **MS_SIMPLE_LIST** or **MS_EXTENDED** used by Open dialog box. A file list box receives an **on_change** event with a *reason* argument under the circumstances listed in the table below.

Reason	Description
CHANGE_SELECTED	Occurs when items are selected or cleared because of a key press or mouse event. None of the _lb??? functions cause an on_change event.
CHANGE_FILENAME	The _filename() function was called which changed the file names listed.

Example:

```
#include "slick.sh"
defeventtab form1;
ctlcommand1.lbutton_up()
{
    ctllist1._flfilename("*.bat", "c:\\\\");
}
ctllist1.on_change(int reason)
{
    if (reason==CHANGE_FILENAME) {
        message("File list display directory " _flfilename());
    }
}
```

For a complete list of file list box control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Directory List Box Control

The directory list box control displays a list of directories. A file list box receives an **on_change** event with one of the *reason* arguments listed in the table below.

Reason	Description
CHANGE_SELECTED	Occurs when items are selected or cleared because of a key press or mouse event. None of the _lb??? functions cause an on_change event.
CHANGE_PATH	The _dlfilename() function was called which changed the file names listed, the left mouse button was double-clicked, or Enter was pressed.

The following example requires a form named "form1", a text box named "ctltext1", and a directory list box named "ctllist1":

```
#include "slick.sh"
defeventtab form1;
ctllist1.on_change(int reason)
{
    if (reason==CHANGE_PATH) {
        // Set the text in the text box to current directory. Changing
        // directories with the directory list box control changes the
        // editor's current directory.
        ctltext1.set_command(_dpath(),1);
    }
}
```

}

For a complete list of directory list box control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Picture Box Control

The picture box is used to place other controls inside of it, like the frame control. The picture box is capable of displaying bitmaps, displaying bitmap buttons, and all the features of the image control. To display bitmaps and bitmap buttons, use the image control feature described in the topic [Image Control](#).

For a complete list of picture box control properties, methods, and events, from the menu item select **Help → Macro Functions by Category**.

Gauge Control

Gauge control is typically used to indicate the completion status of a process.

Example:

```
// Create a form with a command button named ctlcancel, and gauge named
// ctlgauge1. Set the cancel and default properties of the command
button
// to true.
#include "slick.sh"

static bool gcancel;
_command test()
{
    // Need to tell compiler ctlgauge1 is a control because the
    // form1_wid.ctlgauge1 is too ambiguous.
    _control ctlgauge1;

    // Show the form modeless so there is no modal wait.
form1_wid=show("form1");
// Disable all forms except form1_wid.

disabled_wid_list=_enable_non_modal_forms(0,form1_wid);
gcancel=0;
for (i:=1;i<=100;++i) {
    // Read mouse, key, and all other events until none are left or
until
    // the variable gcancel becomes true.
    process_events(gcancel);
    if (gcancel) {
        break;
```

```
        }
        // Do work here. Replace the delay below with the operation you
want to do.
        // The delay makes this example look more real.
        delay(10);

        form1_wid.ctlgauge1.p_value=i;
    }
    // Enable all forms that were disabled.
    _enable_non_modal_forms(1,0,disabled_wid_list);
    form1_wid._delete_window();
}
defeventtab form1;
ctlcancel.lbutton_up()
{
    gcancel=1;
}
```

For a complete list of gauge control properties, methods, and events, from the main menu, select **Help → Macro Functions by Category**.

Image Control

Image control is for creating bitmap buttons or toolbar buttons. The image control performs a subset of the features of the picture box control.

Adding a Bitmap Command Button or Check Box

Perform the steps below to add a bitmap button to a dialog box. The same steps can also be used to add a check box.

1. Create a new form for editing. From the main menu, select **Macro → New**.
2. Create an image control. Double-click the **Image Control** bitmap.
3. Set the **p_picture** property to `bbfind.bmp`. Make sure that you specify the full path (the default path used by the installation program is `c:\vslick\bitmaps` on Windows or `/usr/lib/vslick/bitmaps` on UNIX). In this step you enter the `bbfind.bmp` bitmap as an example.
4. Set the **p_command** property to **gui_find**. The **Down** arrow of the combo box displays all the editor commands.
5. Set the **p_message** property to **Searches for a string you specify**.
6. Set the **p_style** property to **PSPIC_FLAT_BUTTON** or **PSPIC_BUTTON**.

Tip

The **bb** prefix indicates that this is a bitmap that can be used by a toolbar. You can edit the `bbfind.bmp` file with Paintbrush (`pbrush.exe`). Use `bbblank.bmp` as a template for creating your own bitmap buttons.

The following example illustrates how to load your own picture like a toolbar button:

```
#include "slick.sh"
defeventtab form1;
ctlimage1.on_create()
{
    index=_update_picture(-1,bitmap_path_search("bbfind.bmp"));
    if (index<0) {
        if (index==FILE_NOT_FOUND_RC) {
            _message_box("Picture bbfind.bmp was not found");
        } else {
            _message_box("Error loading picture
bbfind.bmp\n\n":+get_message(index));
        }
        return("");
    }
    p_picture=index;
    p_command="gui_find";
    p_message="Searches for a string you specify";
    p_style=PSPIC_FLAT_BUTTON;
}
```

The following example illustrates how to give the appearance of a button being pushed in. While you can do this by setting styles, here you can see how some other functions accomplish this task. For this example, create a form named "form1" and an image control named "ctlimage1".

```
#include "slick.sh"
defeventtab form1;
ctlimage1.on_create()
{
    index=_update_picture(-1,bitmap_path_search("bbfind.bmp"));
    if (index<0) {
        if (index==FILE_NOT_FOUND_RC) {
            _message_box("Picture bbfind.bmp was not found");
        } else {
            _message_box("Error loading picture
bbfind.bmp\n\n":+get_message(index));
        }
        return("");
    }
    p_picture=index;
```

```
p_command="gui_find";
p_message="Searches for a string you specify";
p_style=PSPIC_BUTTON;
}
ctlimage1.lbutton_down()
{
    // Reset the button counter so we don't get double and triple
click events.
    get_event('B');
    mou_mode(1)
    mou_capture();
    done:=false;
    event:=MOUSE_MOVE;
    for (;;) {
        switch (event) {
            case MOUSE_MOVE:
                mx=mou_last_x("m"); // "m" specifies mouse position in
current scale mode
                my=mou_last_y("m");

                if (mx>=0 && my>=0 && mx<=p_width && my<=p_height) {
                    if (!p_value) {
                        p_value=1; // Show the button pushed in.
                    }
                } else {
                    if (p_value) {
                        p_value=0; // Show the button up.
                    }
                }
                break;
            case LBUTTON_UP:
            case ESC:
                p_value=0; // Restore the button state.
                done=true;
                break;
        }
        if (done) break;
        event=get_event();
    }
    mou_mode(0);
    mou_release();
    say('out');
    return("");
}
```

Adding Dialog Box Retrieval

Dialog box retrieval enables previous responses for check boxes, radio buttons, spin boxes, text boxes, and combo boxes to be retrieved. Press **F7** to retrieve the previous response, and **F8** to retrieve the next response. For example, the Insert Literal dialog box contains a spin box that is used to enter the character code of the character to insert. If you use it to enter a Hex value of **0xAE** (to insert a registered trademark symbol), then later use it to enter a Hex value of **0x99** (to insert an unregistered trademark symbol), the next time you use the dialog you can press **F7** to retrieve the previous entry of **0xAE**, and then **F8** to retrieve the next entry of **0x99**.

The responses to dialog boxes are saved for the next session when you exit the editor and auto-restore is enabled.

The example below illustrates how to add dialog box retrieval to your own dialog boxes. Create a form named "form1", a text box (any name), a check box (any name), and a command button named "ok".

```
#include "slick.sh"
defeventtab form1;
ctlok.on_create()
{
    // Retrieve the previous response to this dialog box.
    _retrieve_prev_form();
}
ctlok.lbutton_up()
{
    _save_form_response();
    p_active_form._delete_window(1);
}
```

Menus

You can create a new menu and change or add menu items by using the Menu Editor dialog box (**Macro** → **Menus**, select a menu to edit or click **New**). Or, to create a new menu, use the Open Menu dialog box (**Macro** → **Menus**) and click **New**. A quick way to bind a pop-up menu to a mouse click is to use the **Show** button on the Open Menu dialog box while recording a macro. When you are finished recording the macro, the Key Bindings option screen (**Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**) is displayed which enables the binding of the new macro to a mouse click.

This section describes macro programming details about menus for advanced menu item enabling and for writing macros that manage menus. Topics are:

- [Menu Editor Dialog Box](#)
- [Menu Item Alias Dialog Box](#)
- [Auto Enable Properties Dialog Box](#)
- [Creating and Editing Menu Resources](#)

Menu Editor Dialog Box

The Menu Editor dialog is used for editing menu resources. Use the Menu Editor to modify the SlickEdit MDI menu bar or an existing menu resource which can be displayed as a pop-up or menu bar.

To access this dialog, from the main SlickEdit® menu, select **Macro** → **Menus**, then click **Open** to open a menu for editing, or **New** to create a new menu. The **New** button on the Open Menu dialog box creates a new menu resource and places you in the Menu Editor so you can add menu items. After creating a menu, you can use the **Show** button on the Open Menu dialog box while macro recording to create a command which runs a menu by displaying it as a pop-up. If you bind the recorded command to a left or right button mouse event, the menu will be displayed at the cursor position.

You DO NOT need to specify key bindings for menu items because our Menu Editor automatically determines the key bindings for you. Use the Advanced Appearance option screen (**Tools** → **Options** → **Appearance** → **Advanced**) to choose between short and long key names.

For information about each field and option on the Menu Editor dialog, see "Menu Editor dialog" in the SlickEdit **Help** → **Index**. See [Creating and Editing Menu Resources](#) for information on creating forms with menu bars or advanced information.

Menu Item Alias Dialog Box

When you click the **Alias** button on the [Menu Editor Dialog Box](#), the Menu Item Alias dialog is displayed. This dialog box allows you to define aliases (similar commands) for the command that is being executed. Enter each alias command on a separate line. If one of the alias commands is 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

Auto Enable Properties Dialog Box

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. See also "aliases" in the SlickEdit **Help → Index** for more information.

Auto Enable Properties Dialog Box

For convenience, SlickEdit® has some predefined enable/disable attributes 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/disable state of the command. See [Creating and Editing Menu Resources](#) for information on enabling and disabling menu items with your own callback. For information about each field and option on the dialog, see "Auto Enable Properties dialog" in the SlickEdit **Help → Index**.

Creating and Editing Menu Resources

Modified menus are stored in the state file `vslick.sta` file. The easiest way to create or change a menu is to use the Open Menu dialog box (**Macro → Menus**). After you select the menu, the [Menu Editor Dialog Box](#) is displayed and you can edit the menu resource. After the menu is created, use the **show**, **mou_show_menu**, or **_menu_show** function to run the menu by displaying it as a pop-up window. The **_menu_set** method may be used to create a menu bar on a form. Another way to create or change a menu is to define or modify a menu resource. See the following topics:

- [Defining a Menu Resource](#)
- [Predefined Attributes for Auto-Enabling Commands](#)
- [Macro Callbacks for Enabling Commands](#)
- [Placing a Menu Bar on a Form](#)
- [Displaying a Menu as a Pop-Up](#)

Defining a Menu Resource

Use the **insert_object** command to insert macro source code for a menu into the current buffer. Edit the resource properties and then run the macro to apply the resource changes. Ignore the message **No main entry point** if it is displayed. Changing a menu resource does not change any menu bars. Menu bars represent menu resources that have been loaded. A menu definition has the following format:

```
_menu menu_name {
    submenu menu_item, help_command, help_message, categories {
        menu_item, command, categories, help_command, help_message
    }
    submenu
}endsubmenu
}
```

Creating and Editing Menu Resources

The table below contains the menu items and their definitions:

Menu item	Definition
menu_item	Menu item name in double quotes. Use & to choose selection character.
command	Any editor command. Places the cursor on the command line and press ? to list all editor commands.
help_command	Command to be executed when F1 is pressed. Usually it is a help or popup_imessage command.
categories	Specifies zero or more help categories in double quotes. Multiple help categories are separated with (pipe).
help_message	A single line message in double quotes displayed on message line.

Example of a menu definition:

```
_menu mymenu {
    submenu "&File", "Help file menu", "Displays File drop-down menu",
"ncw" {
        "&New", "new", "ncw" , "help new", "Creates a new file to edit";
        "&Open\tCtrl+O", "gui_open", "help gui_open", "Open a file";
    }
    submenu "&Edit", "Help edit menu", "Displays Edit drop-down
menu", "ncw" {
        "Cu&t", "cut", "sel|nrdonly", "help cut", "Deletes the selection
and copies it to the clipboard";
    }
}
```

Predefined Attributes for Auto-Enabling Commands

Predefined enabling or disabling attributes can be specified for any command. Specify these attributes in the **name_info** of a command definition. Auto-enabling attributes affects the enable/disable state for a command placed in a menu or in a toolbar. The following command is disabled when there is no editor control on which to operate:

```
#include "slick.sh"
```

```
_command void top_of_buffer()
    name_info(' ', VSARG2_READ_ONLY| VSARG2_REQUIRES_EDITORCTL)
{
}
```

Macro Callbacks for Enabling Commands

If the auto-enable attributes do not provide the features that you want, you can define the enable and disable callback for the command. The name of the callback function you define is based on the name of the command as shown in the following example:

```
#include "slick.sh"

static bool gSomeOtherState;
/*
   This function gets called if your command is used in a menu or
toolbar.
   You must return a combination of the MF_ flags ORed together.

BEWARE: If an _OnUpdate callback causes a Slick-C run-time error,
you
may not see the error. In addition, the timer used for toolbars,
Context Tagging(R), AutoSave, and some other features may be
automatically terminated. Exit and restart the editor to restart
this timer. Use the "say" function to debug your _OnUpdate
callback.
*/
int _OnUpdate_mycommand(CMDUI &cmdui,int target_wid,_str command)
{
    //say('h1');
    // Lets assume this command requires the target to be an editor
control
    // with a selection.
    // IF the target is not an editor control:
    if ( !target_wid || !target_wid._isEditorCtl() ) {
        //say('disabled at h2');
        return(MF_GRAYED);
    }
    //say('h3');
    // IF the editor control does not have a selection:

    if ( !target_wid.select_active2() ) {
        //say('disabled at h4');
        return(MF_GRAYED);
    }
    //say('h5');
```

```
if (gSomeOtherState) {
    //say('disabled at h6');
    return(MF_GRAYED);
}
//say('enabled at h7')
return(MF_ENABLED);
}
_command void mycommand() name_info('VSARG2_REQUIRES_EDITORCTL')
{
    // Some code here...
}
// This command affects the enable/disable of mycommand.
_command void mycommand2(_str argument="0")
{
    gSomeOtherState=(argument)?1:0;

    // Indicate that the enable state of the toolbar buttons must be
updated.
    // The _tbSetRefreshBy function is very fast. Toolbars will be
updated
    // after the macro terminates and the user stops typing fast.
    _tbSetRefreshBy(VSTBREFRESHBY_USER);
}
```

Placing a Menu Bar on a Form

The following sample code shows how to add a menu on a form as a menu bar:

```
#include "slick.sh"

// Create a form called form1 and set the border style to anything BUT
// BDS_DIALOG BOX. Windows does not allow forms with a dialog box
style
// border to have menu bars.
defeventtab form1;
form1.on_load()
{
    // Find index of MDI menu resource.
    index=find_index(def_mdi_menu,oi2type(OI_MENU));
    // Load this menu resource.
    menu_handle=p_active_form._menu_load(index);
    // _set_menu will fail if the form has a dialog box style border.
    // Put a menu bar on this form.
    _menu_set(menu_handle);
    // You DO NOT need to call _menu_destroy. This menu is destroyed
when
    // the form window is deleted.
```

```
        }
        form1.on_init_menu()
        {
            // Gray out all menu items that are not allowed when there are no
            child windows.
            _menu_set_state(p_menu_handle,!ncw, MF_GRAYED,C);
        }
    }
```

Displaying a Menu as a Pop-Up

If the **show** or **mou_show_menu** function meets your needs, use one of them. The following sample code shows how to display a menu as a pop-up:

```
#include "slick.sh"
defmain()
{
    // Low-level code to display menu bar as pop-up.
    // Could just use show or mou_show_menu function.
    index=find_index(_mdi_menu,oi2type(OI_MENU))
    if (!index) {
        message("Can't find _mdi_menu");
    }
    menu_handle=_menu_load(index,P);
    // Display this menu in the menu of the screen.
    x=_screen_width()/2;y=_screen_height()/2;
    flags=VPM_CENTERALIGN|VPM_LEFTBUTTON;
    _menu_show(menu_handle,flags,x,y);
    _menu_destroy(menu_handle);
}
```

Common Macro Dialog Boxes

There are several important macro dialog box forms and functions that you can use in your own macros. The table below lists the general purpose forms and dialog box functions.

Form	Description
<code>_textbox_form</code>	Displays a variable number of text boxes or combo boxes.
<code>_sellist_form</code>	Displays a list box, an optional combo box, and a variable number of command buttons.
<code>_select_tree_form</code>	Displays a list using a tree control, an optional combo box, and a variable number of command buttons. Invoked using the <code>select_tree()</code> function.
<code>_open_form</code>	Used to open and save files that does not have the advanced controls.
<code>_edit_form</code>	Used to open and save files that has the advanced controls used for the File → Open dialog box.
<code>_font_form</code>	Used to prompt for a font.
<code>_choose_font</code>	(Non-UNIX platforms only) Dialog box built-in to operating system used to prompt for a font.
<code>_printer_setup</code>	(Non-UNIX platforms only) Dialog box built-in to operating system used for printer setup.

If a key displays a dialog box, you can find out the command the key executes by using the Key Bindings options screen (**Tools → Options → Key Bindings**).

String Functions

The table below describes commonly used string functions. See **Help → Macro Functions by Category → String Functions** for a complete list.

See also documentation for the [parse Statement](#).

Function	Description
<code>_str center (_str string ,int width [,-str pad_ch])</code>	Returns <i>string</i> padded evenly on left and right with spaces or a character you choose with the optional argument <i>pad_ch</i> .
<code>_dec2hex (long number [,int base])</code>	Returns <i>number</i> converted to <i>base</i> specified.
<code>_str expand_tabs (_str string [,int start [,int count [,_str option]]])</code>	Very similar to substr function except that this function supports tab characters very well.
<code>_str field(_str string ,int width)</code>	Returns <i>string</i> padded with trailing spaces to <i>width</i> characters.
<code>long hex2dec(_str number [,int base])</code>	Returns <i>number</i> converted to <i>base</i> specified.
<code>_str indent_string(int width)</code>	If indent with tabs is on, a string of tabs of length <i>width</i> is returned. Otherwise, a string of spaces of length <i>width</i> is returned.
<code>bool isalnum(_str ch)</code>	Returns non-zero value if <i>ch</i> is a numeric or alphabetic character.
<code>bool isalpha(_str ch)</code>	Returns non-zero value if <i>ch</i> is an alphabetic character.
<code>bool isdigit(_str ch)</code>	Returns non-zero value if <i>ch</i> is a numeric character.
<code>bool isinteger(_str string)</code>	Returns non-zero value if <i>string</i> is a valid int . If <i>string</i> is floating point number, 0 is returned.
<code>bool isnumber(_str string)</code>	Returns non-zero value if <i>string</i> is a valid double (floating point number).
<code>_str last_char(_str string)</code>	Returns last character of <i>string</i> . If <i>string</i> is null, the space character is returned.
<code>int lastpos(_str needle [_str haystack [,int start [,_str options]]])</code>	Returns the position (1..length(haystack)) of the last occurrence of <i>needle</i> in <i>haystack</i> . If <i>needle</i> is

String Functions

Function	Description
	not found, 0 is returned. Regular expressions are supported.
int length(_str string)	Returns the number of characters in <i>string</i> .
_str lowercase(_str string)	Returns <i>string</i> converted to lowercase.
_str number2onoff(_str number)	Returns off if <i>number==0</i> . Otherwise on is returned.
_str number2yesno(_str number)	Returns N if <i>number==0</i> . Otherwise Y is returned.
parse expr with template	Breaks apart the expression <i>expr</i> given into variables that appear in <i>template</i> , and much more. See parse Statement for more information.
bool parseoption(_str & cmdline ,_str option_ch)	Strips + or - option from <i>cmdline</i> . Returns non-zero number if <i>option_ch</i> was found.
int pos(_str needle [, _str haystack [,int start [, _str options]]])	Returns the position (1..length(haystack)) of the first occurrence of <i>needle</i> in <i>haystack</i> . If <i>needle</i> is not found, 0 is returned. Regular expressions are supported.
bool setonoff(_str & name ,_str value)	Sets <i>name</i> to 1 or 0 corresponding to <i>value=on</i> or <i>value=off</i> . Returns 0 if input value is valid. Displays message if <i>value</i> is not on or off.
bool setyesno(int & name ,_str value)	Sets <i>name</i> to 1 or 0 corresponding to <i>value=Y,Yes</i> or <i>value=N,No</i> . Returns 0 if input value is valid. Displays message if <i>value</i> is not Y or Yes , N or No .
_str translate(_str string , _str replace_string ,_str search_string , _str search_options)	Returns <i>string</i> with all occurrences of <i>search_string</i> replaced with <i>replace_string</i> .
_str strieq(_str string1 ,_str string2)	Returns true if <i>string1</i> matches <i>string2</i> when case is ignored.
_str strip(_str string ,_str ltb [, _str strip_char])	Returns <i>string</i> stripped of leading and/or trailing <i>strip_char</i> .
_str strip_filename(_str filename , 'P' 'D' 'E' 'N')	Returns <i>filename</i> with part stripped. P=Path, D=Drive, E=Extension, N=Name.
_str strip_last_word(_str & line)	Returns the last space delimited word in <i>line</i> . The

Search Functions

Function	Description
	last word and trailing spaces are deleted from <i>line</i> .
<code>_str strip_options(_str cmdline ,_str & options)</code>	Returns <i>cmdline</i> without words that start with the characters <code>-</code> , <code>+</code> , or <code>[</code> . <i>options</i> variable is set to stripped option words.
<code>_str substr(_str string ,int start [,int length [, _str pad]])</code>	Returns <i>length</i> characters of <i>string</i> beginning at <i>start</i> . By default, <i>length</i> defaults to rest of <i>string</i> . If <i>length</i> is greater than length of <i>string</i> , the return string is padded with blanks or <i>pad</i> character if specified.
<code>_str translate(_str string [, _str output_table [, _str input_table [, _str pad]]])</code>	Returns <i>string</i> with characters translated according to arguments.
<code>_str upcase(_str string)</code>	Returns <i>string</i> converted to uppercase.
<code>int verify(_str string , _str reference [, M [,int start]])</code>	Returns the position (<code>1..length(string)</code>) of first character not matching or matching a character in <i>reference</i> . 0 is returned on failure.
<code>_str word(_str string ,int Nth)</code>	Returns the <i>Nth</i> space or tab-delimited word in <i>string</i> . Is returned if the <i>Nth</i> word does not exist.

Search Functions

Two levels of search functions exist: high level functions that provide user interfacing and multiple file searching, and built-in functions that are used without affecting the high level search commands such as the **find_next** command. The built-in functions are not affected by the global editor search options.

The table below shows a list of commonly used search functions. For a complete list, see **Help → Macro Functions by Category → Search Functions**.

Function	Description
gui_find	Displays Find and Replace tool window open to the Find tab, and performs search using the find or _mffind functions.
gui_replace	Displays Find and Replace tool window open to the Replace tab, and performs search using gui_replace2 or _mfreplace functions.
gui_replace2	Performs a search and replace based on arguments given. This function is very similar to the replace function, except that this function uses a dialog box to prompt the user where to replace.
find_next	Searches for next occurrence of search string used by any of these high-level search functions. This function is not affected by previous searches done with low-level built-in functions.
find	Performs search based on arguments given.
replace	Performs a replace based on arguments given. The user is prompted where to replace through the message line.
_mffind	Performs a multiple file and buffer search based on the arguments given.
_mfreplace	Performs a multiple file and buffer search based on the arguments given.
search	Performs a search, or search and replace, based on arguments given. Does not support wrapping to top or bottom of file. When performing a replace, the user is not prompted at all.

Function	Description
repeat_search	Searches for the next occurrence of search string used by last call to the search built-in.

The following example searches for lines that contain a particular search string and places the lines in another window and buffer:

```

void defmain()
{
    orig_wid := p_window_id;

    // The +w option forces a new window to be created. The +t options
    // force a new buffer to be created.
    status := edit("+w +t");
    if (status) {
        _message_box("Unable to create temp window and buffer\n\n":+
                    get_message(status));
    }

    delete_line();           // Delete the blank line.
    output_wid := p_window_id;

    p_window_id = orig_wid;
    top();                  // Place the cursor at the top in column 1.
    status=search("if","w@"); // Case-insensitive word search for if @
specifies
                    // no string not found message.

    loop
    {
        if (status) {
            break;
        }

        // Place the cursor at the end of the line so no
        // more occurrences can be found on this line.
        get_line(auto line);

        _end_line();
        output_wid.insert_line(line);
        status=repeat_search();
    }

    // Make the output window active so we can see the results.

```

```
    p_window_id = output_wid;
}
```

The next example is very similar to the example above except that the output data is placed in a view and buffer. The only advantage in using a view and buffer is that the output can be displayed in a list box without the user having to see a new window created.

```
#include "slick.sh"

void defmain()
{
    // Create a temporary view and buffer within the current window.
    // Each window can store multiple cursor positions (views) to any
buffer.
    orig_view_id := _create_temp_view(auto temp_view_id);
    if (orig_view_id=="") {
        return("");
    }
    activate_view(orig_view_id);

    top();                                // Place the cursor at the top in column
1.
    status := search("if", "w"); // Case sensitive word search for if.

    for (;;) {
        if (status) {
            // Clear the pending message caused by built-in search failing.
            clear_message();
            break;
        }
        get_line(auto line);

        // Place the cursor at the end of the line so no more occurrences
        // can be found on this line.
        _end_line();

        activate_view(temp_view_id);
        insert_line(' 'line);    // Insert a space at the beginning of the
line
                                // because this will be inserted into a
listbox.

        activate_view(orig_view_id);
        status=repeat_search();
    }
}
```

```
// Display the buffer in a list box.  
// The _sellist_form dialog box will delete the temp view and buffer.  
// The original view must be activated before showing the  
_sellist_form or  
// the dialog box will operate strangely.  
activate_view(orig_view_id);  
result=show("_sellist_form -mdi -modal",  
           "Sample Selection List",  
           // Indicate next argument is view_id.  
           SL_VIEWID|SL_SELECTCLINE,  
           temp_view_id,  
           "OK",  
           "", // Help item.  
           "", // Use default font.  
           "" // Call back function.  
           );  
  
if (result) {  
    message("Selection list cancelled");  
} else {  
    message("Item selected is "result);  
}  
}
```

Selection Functions

SlickEdit® supports multiple selections; however, only one selection can be active or visible. Selections are specified by handles. Most selection functions accept a selection handle. A handle of "" specifies the active selection or selection showing, that is always available.

The table below describes some common selection functions.

Function	Description
<code>_alloc_selection</code>	Returns a handle to a selection.
<code>_free_selection</code>	Frees a selection associated with the selection handle given. Note that it cannot free the active selection. To free the active selection, use <code>_show_selection</code> first.
<code>_show_selection</code>	Used to make another selection the active selection.
<code>_duplicate_selection()</code>	Returns the actual handle number of the active selection.

Example:

```
// Duplicate the current line.
mark_id=_alloc_selection();
if (mark_id<0) {
    message(get_message(mark_id));
    return(rc);
}
_select_line(mark_id);
_copy_to_cursor(mark_id);

// This selection can be freed because it is not the active selection.
_free_selection(mark_id);

// This code copies selected text and keeps the
// resulting selection on the source text instead of
// the destination text.
if (_select_type() == "") {
    message(get_message(TEXT_NOT_SELECTED_RC));
    return(1);
}
mark_id=_duplicate_selection()
```

```
// Make a copy of the active selection.  
_copy_to_cursor();  
  
// Save the selection id.  
old_active_mark_id= duplicate_selection();  
  
// Must make another mark active before the old active mark can be freed.  
show_selection(mark_id));  
  
// Make copy of visible mark active.  
free_selection(old_active_mark_id));
```

For more information about selection functions, from the main menu, select **Help** → **Macro Functions by Category**, then click **Selection Functions**.

Writing Selection Filters

The module `markfilt.e` provides the procedure `filter_selection` for filtering selected text. Define a global procedure that accepts a string and returns a string. Then pass the name of the procedure to the `filter_selection` procedure.

The following batch program converts the marked text into hexadecimal ASCII codes. Each hexadecimal ASCII code is separated by a comma. One possible use of this function could be to convert a binary font file into hexadecimal ASCII codes to be compiled into a C program.

```
#include "slick.sh"

_str hex_filt(_str string);

void defmain()
{
    if (_select_type() == " ") {
        message(get_message(TEXT_NOT_SELECTED_RC));
        return(TEXT_NOT_SELECTED_RC);
    }
    // Underscores must be converted to dashes.
    return(filter_selection(hex_filt));
}

_str hex_filt(_str string)
{
    line:@"";
    for (i :=1;i<=length(string);++i) {
        line=line:+dec2hex(_asc(substr(string,i,1))):+",";
    }
    return(line);
}
```

Unicode and SBCS or DBCS Macro Programming

The following information applies for Unicode users only. When the code editor is running in UTF-8 mode (by default, `vs.exe` for Windows runs in this mode), buffers can contain either SBCS/DBCS data or UTF-8 data depending on how a buffer is loaded. To make it easier for macros to support these two buffer data formats, almost all macro functions accept and return UTF-8 strings. This allows most macros to automatically work. Macros that use or set column positions often do not work correctly for both buffer data formats. The solution is to call raw functions.

Example:

```
// This will not work if the current buffer is an SBCS/DBCS buffer,  
// word is a UTF-8 string (that this example assumes), and word  
// contains characters above 127.  
p_col=p_col+length(word);  
// This will work.  
p_col=p_col+_rawLength(word);  
// This works too.  
word=_rawText(word);  
p_col=p_col+length(word);
```

Example:

```
// This will not work if the current buffer is an SBCS/DBCS buffer and  
// the current line contains characters above 127.  
get_line(line);  
  
string=expand_tabs(line,p_col);  
// This works.  
get_line_raw(line);  
  
string=expand_tabs(line,p_col);  
// This works too, but is less efficient if all operations on line  
// can support raw data.  
get_line(line);  
string=expand_tabs(_rawText(line),p_col);
```

The `_UTF8()` macro function indicates if the code editor is in UTF-8 mode. The `p_UTF8` property tells you whether the current buffer contains UTF-8 data. The `p_encoding` property indicates what format the buffer will be saved in by default.

Like typical programming languages (Java, C++), Slick-C® source files are code page dependant. Strings are converted from the current code page to UTF-8. This is important if you enter characters above 127. All of the macro functions and properties accept and return UTF-8. The Slick-C functions in the table below DO NOT accept or return UTF-8 data.

**Unicode and SBCS or DBCS
Macro Programming**

Function	Definition
<code>_default_option(VSOPTIONZ_SPECIAL_CHAR_XLAT_TAB)</code>	All other options for this function are UTF-8.
All seek functions: <code>goto_point()</code> , <code>_QROffset()</code> , <code>_GoToROffset</code> , <code>_nrseek()</code> , <code>point()</code> , and <code>seek()</code>	All seeking is done on raw data. Buffers need to be loaded in the same raw format so that seek functions work.
All <code>_rawXXX()</code> or <code>XXX_raw()</code> functions	Unlike the C API, the Slick-C functions <code>get_text()</code> and <code>_expand_tabs()</code> return UTF-8 data.

The `p_display_xlat` Slick-C property DOES NOT accept or return UTF-8 data.

The following are the Slick-C raw functions:

- `_expand_tabs_raw()`
- `get_line_raw()`
- `get_text_raw()`
- `insert_line_raw()`
- `_insert_text_raw()`
- `replace_line_raw()`
- `_rawLength()`
- `_rawSubstr()`
- `_rawText()`

The table below shows the raw functions that optionally support raw data.

Function	Description
<code>pos()</code>	When <code>p_rawpos</code> appended to <code>options</code> argument.
<code>lastpos()</code>	When <code>p_rawpos</code> appended to <code>options</code> argument.
<code>upcase()</code>	When <code>p_UTF8</code> property given as second argument.
<code>lowcase()</code>	When <code>p_UTF8</code> property given as second argument.
<code>parse</code>	When <code>p_rawpos</code> appended to <code>options</code> of search argument.

Function	Description

The following are the Slick-C new UTF-8 functions:

- **_MultiByteToUTF8()**
- **_UTF8()**
- **_UTF8Asc()**
- **_UTF8Chr()**
- **_UTF8ToMultiByte()**

The following C API functions DO NOT accept or return UTF-8 data:

- The functions **vsGetText()**, **vsGetRText()**, **vsExpandTabsC()**, **vsQSelectedTextLength()**, **vsGetSelectedText()** - These functions always return raw data. Use the **vsUTF8()** function or check the **VSP_XLAT** property to determine if you need to translate the buffer data. Since these API functions assume that the maximum buffer length is the same as the read length, it would be useless for these functions to return translated data.
- All seek functions (**vsQOffset**, **vsQROffset**, **vsGoToPoint**, and **vsGoToROffset**) - All seeking is done on raw data. Since the Context Tagging® database stores seek positions, buffers need to be loaded in the same raw format so that seek works.
- All **vsXXXRaw()** functions.

Shelling Programs from a Slick-C® Macro

To execute another program from a Slick-C macro, use the **shell** built-in, the **dos** command, or the **execute** built-in. The latter method is similar to executing a command on the command line, and enables the creation of expressions that execute Slick-C internal commands, Slick-C batch programs, or external programs. If you are only interested in executing an external program, use the **shell** built-in or the **dos** command.

Example:

```
// Capture the output of Slick GREP and process the error messages.  
dos("-e sgrep DEBUG *.c");  
// Redirect the output of sgrep to a file.  
shell("sgrep DEBUG *.c >junk");  
// Run the DOS dir command and wait for a key to be pressed before  
// closing command shell window.  
shell("dir *.c >junk", "w");  
// Display the Calculator dialog box. Show is an internal command.  
execute("show _calc_form");
```

Interfacing With Other Languages (DLL)

SlickEdit products have a DLL interface for Windows. Use the Slick-C® macro language instead of the DLL interface except when you need an interface to the DLL in another program, when better speed is needed, or when the Slick-C macro language is missing a function that you want.

After a DLL function is added, call it from a Slick-C macro just like any other Slick-C function. DLL functions can be used for timer call backs and any place a Slick-C function is used.

To get started using the DLL interface, edit the `simple.c` file located in the `samples\simple` subdirectory of your installation directory. The **VSAPI** functions have the prefix **vs**.

Command Line Interface

This section describes how to write macros using the command line interface.

Command Line Arguments

When a command is invoked, the expression **arg(1)** contains the rest of the command line after the name with leading spaces removed. Alternatively, the command can declare a named argument whose value is the same as **arg(1)**. For example, invoking the edit command **e file1 file2** calls the **e** command with **file1** **file2** in **arg(1)**. The **parse** built-in is an excellent function for parsing a command line string. When another macro calls a command, more than one argument string can be passed. Calling the **arg** function with no parameters returns the number of parameters with which the command or procedure was called.

Example:

```
#include "slick.sh"
// This command supports completion on a filename followed by an
// environment variable argument.
_command void test1() name_info(FILE_ARG , "ENV_ARG")
{
    parse arg(1) with file_name env_name;
    message("file_name="file_name" env_name="env_name );
}
```

The string constant expression given to the **name_info** keyword is used for argument completion, restricting when the command can be executed, and a few other options.

get_string Procedure

The **get_string** procedure reads a single argument from the user.

Example:

```
#include "slick.sh"
_command void test2()
{
    if (get_string(file_name,"Filename: ",FILE_ARG ;Help message )) {
        return(1); // Cancel key pressed.
    }
    if (get_string(env_name,"Environment variable name: ",
                  ENV_ARG ;Help message , "PATH" ) ) {
        return(1); // Cancel key pressed.
    }
}
```

Single Argument Prompting with Support for Prompt Style

```
    message( "file_name="file_name" env_name="env_name) ;  
}
```

Single Argument Prompting with Support for Prompt Style

Use the **prompt** procedure to write a command that accepts one command line argument, or prompts for the argument if it is not given. If the user presses **Esc** while being prompted for the argument, file execution does not continue.

Example:

```
// This command supports completion on an environment variable argument.  
  
#include "slick.sh"  
_command void test3() name_info(ENV_ARG)  
{  
    // If the user selects to abort, the prompt procedure stops  
    // execution.  
    env_name=prompt(arg(1),"Environment variable name: ");  
    message("env_name="env_name);  
}
```

Hooking Startup and Exit

Invoking a Macro on Startup

To invoke any macro command defined by typing **_command** or an external program when the editor initializes, use the **-#** invocation option. For example, invoking the command **vs makefile -#bottom_of_buffer** loads the file **makefile** and executes the **bottom_of_buffer** command. To invoke a command with parameters, place the command and parameters inside double quotes. Another method for getting macro code to start without changing any invocation options is to create a module with a definite entry point.

Invoking a Macro on Exit

If you want a function to be invoked when the editor exits, create a macro procedure with a name that has the prefix **_exit_**. To automatically invoke a macro when exiting SlickEdit®, use the following code:

```
void _exit_cleanup_stuff()
{
    messageNwait( "Got here" );
}
```

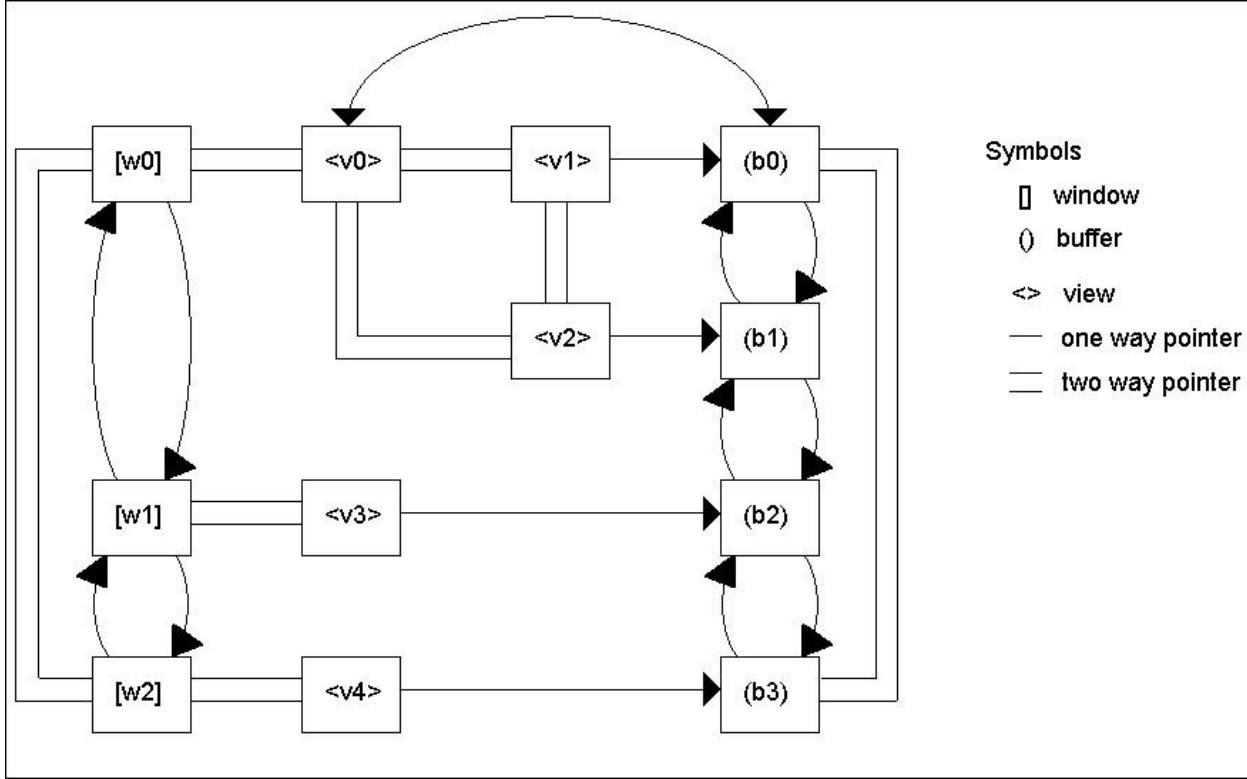
State File Caching

By default, a module, event, dialog box template, or picture from `vslick.sta` is not loaded until it is referenced. Using the **definit** primitive forces a module to be loaded when the editor is invoked. The default state file cache is about 200 K. You can set this size with the **-st** invocation option or with the Virtual Memory option screen (**Tools** → **Options** → **Application Options** → **Virtual Memory**). When the state file cache becomes full, the least recently used module, dialog box template, event table, or picture is removed from memory to reduce the cache size.

You might have critical modules that you want permanently stored in memory. Place the **no_code_swapping** keyword at the top of the module to force the module to be loaded and permanently stored in memory on startup; then, if a critical disk failure occurs while reading the state file, the product is protected. A few modules that provide basic editing capabilities remain permanently in memory.

Windows Data Structure

The following diagram shows startup with two files loaded (buffers b2 and b3) and two windows (w1 and w2) viewing those files:



The extra window, w0, is a hidden window used to allow quick switching to system buffers such as **.command** and **.killed**. If you attempt to leave the hidden window active, another window is made active when the editor refreshes the screen. Since window w1 is active, you currently see window w1 of buffer b2. You might be able to see window w2 of buffer b3 if the window w1 does not overlap window w2.

A ring of buffers and a ring of windows are maintained, where each window may contain a ring of views. However, by convention, all windows except the hidden window contain one view. Some macros temporarily create extra views in other windows, but they delete them before they terminate.

The tables in the following sections show some of the buffer and windowing built-ins that are available.

The built-ins **_next_buffer** and **_prev_buffer** activate the next and previous buffers. **_next_window** and **_prev_window** move around the window ring. **_next_view** and **_prev_view** move around the active view ring. The built-in function **load_files** inserts views, windows, and/or buffers. The command **_delete_buffer** removes the active buffer from the buffer ring and activates the previous non-hidden buffer. **_quit_view** removes the active view from the active windows view ring. The previous view becomes the new active view. When **_quit_view** is executed and only one view exists in the active window, the window is removed and the previous window becomes active. The hidden window cannot be deleted.

A view holds the information necessary for the editor to remember the location and scroll position in a

buffer. A view also contains a window id and a buffer id. Activating a view with the **activate_view** built-in activates the window and buffer specified by the view as well as selecting the cursor location/scroll position.

Each buffer maintains a non-active view. When a buffer is activated by one of the built-ins **_next_buffer**, **_prev_buffer**, **_delete_buffer** or **load_files** (assuming you do not use an option that overrides this), the active view information is saved in the non-active view of the buffer, and the buffer's new non-active view information is copied into the active view.

The following sections describe the contents of each structure.

Window Properties

Window Property	Description
p_window_x , p_window_y	Top left coordinates of window.
p_tile_id	Indicates that windows are part of a tile window group and whether a window is zoomed. Windows of a tiled window group have the same tile_id . A zoomed window has a negative tile_id .
p_x	The top left x position of window.
p_y	The top left y position of window.
p_height	Window height.
p_width	Window width.
p_view_id	Pointer to active view.
p_next (_next_window)	Window id of next window.
p_prev (_prev_window)	Window id of previous window.
p_child	Window id of child window.

View Properties

View Property	Description
block,line within block	Accessible via point and goto_point .

Buffer Properties

View Property	Description
p_line	Line number of current line.
p_col	Column position within current line (1..2 billion).
p_left_edge	Column scroll position.
p_cursor_x	Text cursor x position.
p_cursor_y	Text cursor y position.
p_window_id	Window id.
p_buf_id	Buffer id.

Buffer Properties

Buffer Property	Description
p_buf_name	Name of buffer.
p_buf_flags	Indicates whether a buffer is hidden and may specify other buffer options.
p_Noflines	Number of lines in file.
p_modify	Non-zero indicates buffer has been modified.
p_margins	String containing left, right, and new paragraph margins (1..2 billion).
p_tabs	String containing up to 2 billion tab stops.
p_mode_name	Name of current mode.

Tutorials

This chapter contains the following sections:

- [Defining Stack Routines](#)
- [Searching for a String Within a Current Function](#)
- [Reading and Modifying Buffers](#)
- [Working with Existing Macros](#)

Defining Stack Routines

These examples show you what can be done in a language that supports typed variables and untyped container variables. The following example code shows how to define a set of stack routines in Slick-C® that support any type of element:

```
void stacknew(typeless &stack)
{
    stack._makeempty(); // Destroy current contents of stack.
    stack[0]=0;         // Make an array and use first element as top
count.
}

void stackpush(typeless &stack, typeless &value)
{
    stack[++stack[0]]=value;
}

typeless stackpop(typeless &stack)
{
    if (stack[0]<=0) return(" ");

    // Make a copy of the element.
    result=stack[stack[0]--];

    // Free space allocated by value and delete array element. _deleteel
is a
    // built-in method which operates on arrays and hash tables.
    stack._deleteel(stack[0]+1);
    return(result);
}

defmain()
{
```

```
// The above routines can handle variables of any type, including
// string constants.
struct RECORD {
    int i;
    _str s;
};

// You can't make a limit on the number of elements in an array.
// We will add support for initially allocating a specific number of
elements.
RECORD arecord[];
arecord[0].i=4;
arecord[0].s="element 0";

RECORD symboltable[];           // Declare a hash table/associative
array.
symboltable:["name1"].i=1;
symboltable:["name1"].s="element 0";

stacknew(stack);
stackpush(stack,arecord);      // Push an array onto the stack.
stackpush(stack,symboltable); // Push a hash table/associative array
onto
                           // the same stack.

stackpush(stack,"string");    // Push a string constant onto the
same stack.
}
```

The following example shows how a container variable can access structure members as an array:

```
/*
Read lines of a file which contains tab-delimited data into an array
of
structures. Each line represents an array structure element.

The tab-delimited data on each line represents fields in the
structure.
We will assume the file contains valid data for filling this
structure.
*/
int ReadTable(_str filename,typeless (&table)[])
{
    // Use an editor buffer to open and cache the file. Data is read
    // in blocks from the file only. We don't need this much power, but
    // Slick-C needs a few more non-editor file I/O functions.
    status=_open_temp_view(filename,temp_view_id,orig_view_id);
```

```
if (status) return(status);

top();up(); // Place cursor on line 0 before first line of file.
for (j=0; ;++j) {
    if (down()) break;
    get_line(line);
    if (line== "") continue;

    rest=line;
    p= &table[j]; // Make p point to this structure element.
    // Here we access structure members as an array of elements.
    p->[0]="";

    // Note that loop supports fields which are strings of length 0 .
    for (i=0; ;++i) {
        if (rest=="" && i) break;

        // Parse is similar to REXX. We were unable to come up with a
        // satisfactory function syntax so went with a REXX-style
syntax.
        // Place text up to but not including tab character into value
variable.
        // Place tab character and rest of data in rest variable.
        parse rest with value "\t" +0 rest;
        if (substr(rest,1,1)=="\t") {
            rest=substr(rest,2);
        }
        p->[i]=value;
    }
}

_delete_temp_view(temp_view_id);
activate_view(orig_view_id);
return(0);
}

struct TABLE_ENTRY {
    _str name;
    int value;
};

// defmain is the main entry pointer for a Slick-C batch/script macro.
defmain()
{
    // Table file should exist.
    // NOTE: (TABLE_ENTRY []) is type compatible with (typeless []).
    TABLE_ENTRY table[];
    status=ReadTable("table",table);
```

Searching for a String Within a Current Function

```
if (status) {
    _message_box("Failed to read table file");
    return(1);
}

_message_box("First record: name=":+table[0].name:+"
value=":+table[0].value);
}
```

Searching for a String Within a Current Function

This macro can be used with many languages. It searches the current procedure or function for a specified string, with specified options. Use this macro in cases where references do not work, such as searching for a partial identifier name.

Several useful aspects of this macro, aspects that can be reused in other macros, are that it prompts the user for a string, it selects the current procedure, and it performs a search within the selection.

See the following sections:

- [Creating the Macro](#)
- [Analyzing the Macro](#)
- [Command Line Search Options](#)

Creating the Macro

Complete the following steps:

1. Enter the macro code below into a file called `procsearch.e`.
2. To load the module, from the main menu, select **Macro → Load Module**.
3. Bind the command **proc_search** to a key. To use the macro, press the appropriate key.
4. In the **Search string** text box, enter the text to search for, and in the **Options** text box, enter the search options (see [Command Line Search Options](#)).

Contents of `procsearch.e`:

```
#include "slick.sh"
_command int proc_search(...) name_info('VSARG2_READ_ONLY|
                                         VSARG2QUIRES_EDITORCTL|
                                         VSARG2_MARK)
{
    // Save the original cursor position to restore later.
    save_pos(auto original_position);
```

Searching for a String Within a Current Function

```
// Prompt the user for a search string, and search options.
_str result = show('-modal _textbox_form',
                    'Search Function', // Dialog box caption.
                    TB_RETRIEVE_INIT, // Flags.
                    " ", // Use default text box width.
                    " ", // Help item.
                    " ", // Button list.
                    'procsearch', // Retrieve name.
                    'Search string:', // First prompt.
                    'Options:ixcs'); // Second prompt and default.

if ( result==" " ) {
    // If the user clicked the Cancel button, just return.
    return(COMMAND_CANCELLED_RC);
}

// The results from the text boxes.
_str search_string=_param1;
_str search_options=_param2;

int status=select_proc(); // Select the current proc.
if ( status ) {
    // In rare cases select_proc can fail if a procedure is too
complex.
    // If select_proc failed, show an error messages, return the
cursor to the
    // original position, and return.
    _message_box(nls("select_proc failed"));
    restore_pos(original_position);
    message(get_message(status));
    return(status);
}

lock_selection(); // Lock the selection.
begin_select(); // Move the cursor to the beginning of the
selection.

status=find(search_string,'m':+search_options); // Find the text
that the
                                                // user specified
using the
                                                // options
specified. We
                                                // prepend the
'm' option
                                                // since we know
we are
                                                // searching in a
```

Searching for a String Within a Current Function

```
selection.  
    if ( status ) {  
        // If the search string was not found, deselect and return the  
        cursor to  
        // the original position.  
        deselect();  
        restore_pos(original_position);  
    }  
  
    // Just return the status. This will leave the proc selected so  
    that  
    // find_next works.  
    return(status);  
}
```

Analyzing the Macro

The **save_pos()** call at the beginning of the macro saves the current cursor position information. This function places the cursor in its original position if necessary.

The **show()** function launches a dialog box. In this case, the **show()** function launches a general purpose dialog box named **_text_box_form**. The dialog box **_text_box_form** prompts the user for one or more strings. After the first argument, the remaining arguments to **show()** pass to the **on_create** dialog box. In this case, there are several arguments.

The second argument to **show()** is the caption for the **on_create** dialog box.

The next argument is a set of flags. In this case, the only flag specified is **TB_RETRIEVE_INIT**. The **TB_RETRIEVE_INIT** flag tells the dialog box to initialize itself by retrieving the last values filled in for this dialog box.

Use the next three arguments to specify text box width, help, and a button list. These particular arguments are unused in this example, which is why they are shown here as **".**

The retrieve name is a unique name used to retrieve the values that were previously filled in for this dialog box. Any remaining arguments are interpreted as prompts for the user. Default values can be given by specifying the prompt as **prompt:defaultvalue**. The first prompt is the search string, and the second is for search options. The options have default **ixcs**, meaning case-insensitive, and exclude comments and strings. See the following section for a list of command line search options.

After the call to **show**, verify that the result is **".** If so, then the user clicked the **Cancel** button, so we return. Otherwise, SlickEdit® must obtain the values that the user provides. These values are returned in global variables **_param1.._param N**. In this case, our search string is returned in **_param1**, and the search options are in **_param2**. These are saved in local variables.

SlickEdit calls **select_proc** to select the current function. If **select_proc** returns a non-zero status, then it failed, so it is returned. In rare cases, **select_proc** can fail if a function is too long, or has preprocessing that keeps it from correctly identifying the end of the function.

Next, **lock_selection()** is called, and then **begin_select()** is called to move to the beginning of the

Searching for a String Within a Current Function

selection.

Now, we can call **find()** with the search string and the search options from the user. Insert **m** at the beginning of the options string to specify search only in the selection.

Finally, check the status from **find**. If the string is not found, clear the function and restore the original cursor position.

Command Line Search Options

Command line search options include the characters listed in the table below.

Option	Description
+	(Default) Forward search.
-	Reverse search.
<	(Default) Place cursor at beginning of string found.
>	Place cursor after end of string found.
E	(Default) Case-sensitive search.
I	Case-insensitive search.
M	Search within visible mark.
H	Find text in hidden lines.
R	Search for SlickEdit® regular expression.
L	Interpret string as a Perl regular expression.
~	Interpret string as a Vim regular expression.
U	Interpret string as a Perl regular expression. Unix syntax regular expressions are no longer supported.
B	Interpret string as a Perl regular expression. Brief syntax regular expressions are no longer supported.
N	(Default) Do not interpret search string as a regular search string.
@	No error message.

Option	Description
W	Limits search to words such as variable names.
,	Delimiter to separate ambiguous options.

Reading and Modifying Buffers

Slick-C® includes the Slick-C API. The API covers many actions normally performed in a code editor, including navigating and modifying buffers.

Topics in this section:

- [Functions for Reading and Modifying Buffers](#)
- [Common Functions for Navigating Buffers](#)
- [Escape Backslashes Example](#)
- [Comment Out Debug Print Lines Example](#)

Functions for Reading and Modifying Buffers

The table below contains functions for reading and modifying buffers. This table focuses on one particular category of the API, those functions that allow you to programmatically traverse and modify buffers. These powerful functions enable you to take tasks that you can do manually, and create a macro to perform the same tasks in seconds.

Function	Action
<code>_str cur_word(int & start_col [, _str from_cursor])</code>	Gets the current word at cursor.
<code>int delete_line()</code>	Deletes the current line.
<code>void _delete_text(int len)</code>	Delete <i>len</i> bytes starting from the cursor position.
<code>void get_line(_str & line)</code>	Retrieves current line.
<code>_str get_text([int count [,int seek_pos]])</code>	Gets a stream of text starting at current line.
<code>void keyin(_str string)</code>	Inserts string of characters as if typed from the keyboard.

Function	Action
void insert_line(_str line)	Inserts <i>line</i> after current line.
void _insert_text(_str string)	Inserts <i>string</i> at cursor position.
void replace_line(_str line)	Replaces current line.

Common Functions for Navigating Buffers

The table below contains functions that can be used for navigating buffers.

Function	Action
int up([int num])	Moves cursor up <i>num</i> lines, or one line if no value passed in.
int down([int num])	Moves cursor down <i>num</i> lines, or one line if no value passed in.
void left()	Moves cursor one position to the left.
void right()	Moves cursor one position to the right.
void top()	Places cursor at first line and first column of buffer.
void bottom()	Places cursor at end of last line of buffer.
void _begin_line()	Places cursor at the beginning of the current line.
void _end_line()	Places cursor after the end of the current line.

Escape Backslashes Example

Escape backslashes if, for every slash in a directory name, you actually need two for the compiler to handle the directory name or string properly.

Example:

```
_command escape_slash() {
    // Set string szLine to the current line.
    _str myLine;
    get_line(myLine);
```

```
// Replace slash with double slashes.  
myLine = strtranslate(myLine, "\\\\", "\\");  
  
// Replace the line in the buffer.  
replace_line(myLine);  
}
```

The above command accepts the following line of code:

```
myDirectory = "C:\Data\Corporate\Internal";
```

and replaces it with:

```
myDirectory = "C:\\Data\\\\Corporate\\\\Internal";
```

Comment Out Debug Print Lines Example

Print or debug statements can be used to debug. These statements need to have supporting comments or they must be deleted. The following example shows a simple function that loops through your entire file. It contains supporting comments for all of the lines that have a **printf** statement:

```
_command comment_printf() {  
    typeless p;  
    save_pos(p);           // Save the original position in the buffer  
    top();                 // Go to top of buffer  
    up();                  // Get to the top line  
  
    for (;;) {  
        int status=search("printf","wxcs");   // search for printf as a  
whole  
                                         // word, but exclude  
comments and  
                                         // strings  
  
        if (status) break;                    // If no other instances  
are found,  
                                         // stop  
  
        _begin_line();                      // If printf exists, move  
cursor to  
                                         // the first column  
  
        _insert_text("//");                // Add a comment  
  
        _end_line();                      // Move cursor to the end
```

of the

```
// line
}

restore_pos(p); // Restore the original position in the buffer
}
```

The function uses many of the buffer modifications and navigation macros. Starting at the top of the file, it searches for printf lines and adds a comment when necessary. Modify this macro to meet your needs. For example, if you want the lines deleted instead of commented, replace the `_insert_text()` call with `delete_line()`.

Working with Existing Macros

Every time you select a menu, click a button, or enter a key, a Slick-C® macro is called to perform an action. More than half of the code in SlickEdit products is written in Slick-C and this source is provided to you when you install, so you can tweak the product or use the Slick-C source as an example to help write your own macros. By default, the Slick-C source is located in the `macros` subdirectory of your SlickEdit® installation folder.

To make a macro change, or to recycle existing code, you need to know how to find a name to a particular command and how to find its location in the source code. These examples will walk you through the steps:

- [Example: Turning on Line Numbers for All Files](#)
- [Example: Counting Lines of Code](#)

Example: Turning on Line Numbers for All Files

SlickEdit® includes a line number toggle option to turn line numbers on and off for each edit window. This option is located on the **View** menu (**View** → **Line Numbers**). By default, all files are displayed without line numbers. When you enable them, they are enabled throughout sessions until you disable them.

SlickEdit also provides an option to enable line numbers on a language-specific basis (**Tools** → **Options** → **Languages** → **[Language Category]** → **[Language]** → **View**).

To automatically turn on line numbers for all files that are opened or created in SlickEdit regardless of the language, you will need to write a macro, as outlined in the subsequent sections:

- [Find the Command Definition](#)
- [Create the New Macro](#)
- [Load the Macro](#)
- [Results](#)

Find the Command Definition

You need to find the command that is associated with **View → Line Numbers** in order to view its source code, so that you can obtain the function you'll be using in your new macro.

To determine the command that is associated with **View → Line Numbers**:

1. Close any open files.
2. From the main menu, select **Macro → Menus**. The dialog box contains a list of all menus. To view the main menu, select **_mdi_menu** and click **Open**. The Menu Editor dialog is displayed.
3. Navigate to **View → Line Numbers**. When you select **Line Numbers**, certain fields in the dialog box are populated. The **Command** field is populated with the Slick-C® command that is invoked when this menu item is selected. In this case, the command is **view-line-numbers-toggle**. Every time that you click **View → Line Numbers** from the main menu, **view-line-numbers-toggle** is called.

To view the source code for the **view-line-numbers-toggle** command:

4. From the main menu, click **Macro → Go to Slick-C Definition**.
5. Start typing **view**, and select **view_line_numbers_toggle()** from the drop-down list, then click **OK**.
6. By viewing the source, it is a simple "if on then off, else on" algorithm, using bitwise logic. Note that you will need to use **p_LCBufFlags|=VSLCBUFFLAG_LINENUMBERS** in your new macro to enable the display of line numbers.

Create the New Macro

1. Create a new empty file named **DisplayAllLines.e**.
2. Copy and paste or type the following code into the file:

```
#include "slick.sh"

void _buffer_add_ViewLineNumbers()
{
    p_LCBufFlags |=VSLCBUFFLAG_LINENUMBERS;
    p_line_numbers_len = _default_option(VSOPTION_LINE_NUMBERS_LEN);
}
```

Any Slick-C macro that starts with **_buffer_add_** is called when a new edit window is displayed. To enable the numbers for every file, use the logic from Step 5 above.

Load the Macro

The new macro needs to be loaded. To load the macro, from the main menu, select **Macro → Load Module → DisplayAllLines.e**.

If the macro was loaded properly, the message **Modules loaded** is displayed in the SlickEdit® message line. If an error message is displayed, the macro did not load and the change did not take effect. Correct

the error and load the macro again.

Results

Now every new file opened has line numbers. If any files were left open at the beginning, close and reopen them and they will all have line numbers.

To remove the functionality that turns on line numbers for all files, you need to unload `DisplayAllLines.e`: From the main menu select **Macro > Unload Module**. Select `DisplayAllLines.ex` from the list and click **OK**. The list shows a `.ex` extension on the module instead of a `.e` because you are actually compiling the source file into a binary file (`.ex`) and loading it, not the actual source code.

Example: Counting Lines of Code

The number of lines of code in your workspace, projects, or files is often used to measure and analyze performance, and can be determined by using a macro.

This example describes a macro, `linecount.e`, that loops through all projects in the current workspace and all files within each project in the current workspace, and then displays a report in a new editor window.

You can obtain `linecount.e` from the SlickEdit Web site at www.slickedit.com in the Slick-C® Documentation section. Line numbers referenced in the subsections below:

- [Gather Workspace, Project, and File Information](#)
- [Loop and Count](#)
- [Create the Report](#)
- [Load the Macro](#)
- [Run the Macro](#)

Gather Workspace, Project, and File Information

Get a list of all projects and files in the workspace. `_GetWorkspaceFiles()` (Line 88) gets the list of all projects in a workspace and places the list in a temporary buffer. The loop following (Lines 93-95), parses through the buffer and stores the information in a temporary array for later reporting. This array, defined in Line 67, is a three-dimensional array to store multiple projects, and multiple files per project.

Loop through each project, starting at Line 98, and fill the array with all file names for each project. `GetProjectFiles()` does this by placing the list in a temporary buffer. Grab the names from the buffer and put them in the array (Lines 109-124).

Loop and Count

For each project, open up a temporary buffer for each file in the project. Think of it as an invisible buffer where you can move the cursor programmatically to check whether it is in a comment.

- `_open_temp_view` (Line 139) opens it.

- **up()** and **top()** (Line 158) places the cursor at the top to start.
- **down()** (Line 161) will move the cursor down one line at a time.

Loop through the file to read one line at a time, as mentioned above (Lines 161-202). This validates whether the current line is in a comment (Line 171), and if not, it increments the counter. If the current line is in a comment, the next step is to jump to the end of the comment or comment block (Line 168). Another check is made to see if the current line is in a comment and count it if it is not a comment.

Create the Report

All of the information is now stored in an array, so the next task is to generate a report and loop thru the array to display the results. This is done in Lines 220-263.

The **displayResultsInBuffer** flag can be changed to false to only display the total lines in the entire workspace.

Now that you understand the macro, the next steps are to load and run it.

Load the Macro

To load `linecount.e`, be sure to save it to your local hard drive, then from the main menu, click **Macros** → **Load Module**. Find `linecount.e` and click **Open**.

Run the Macro

You can now run the macro. There are several ways to run macros: from the command line, through a menu item, or by using a keyboard shortcut.

To run the macro from the command line:

1. Open the command line by pressing **Esc** or by clicking in the message line area.
2. Type **linecount** and press **Enter**.

To associate the macro with a menu item:

1. Select **Macro** → **Menus**, then select menu on which you want to add the macro. For example, to add the macro to the right-click context menu, select `_ext_menu_default`.
2. Click **Open**.
3. In the Menu Editor dialog, click **Insert** to add a new menu item.
4. Type a new **Caption**, set the **Command** to `linecount`. Use the **Up** and **Down** buttons to move the new item to the desired location in the list. Type "Menu Editor dialog box" in the Help Index (**Help** → **Index**) for more information about using the Menu Editor.

To associate the macro with a key or key sequence:

1. From the main menu, click **Tools** → **Options** → **Keyboard and Mouse** → **Key Bindings**.

2. Find a key sequence that is not used®do not bind keys that are bound. To determine if a key or key sequence is already in use, place the focus in the **Search by key sequence** field and press the key/ key sequence you want to check. For example, press **Enter** and the table will be filtered to show all commands bound to the **Enter** key.
3. After determining the key or key sequence you want to use for the new binding, close the Options dialog.
4. From the main menu, click **Macro → List Macros**.
5. Select **linecount**, then click **Bind to Key**. The Key Bindings option screen is displayed with **linecount** selected.
6. Click **Add** and when the Bind Key dialog appears, type the key sequence to bind.
7. Click **Bind**, then **OK**.

Events

This section contains the topics:

- [Event Names](#)
- [Keys](#)

Event Names

Event names are used as arguments to the **def** primitive. Event names are also used when comparing events returned by the **get_event** or **test_event** built-in functions or when defining an event handler function. An event name is a string literal of a length of one or more. An event name string of a length one specifies an ASCII character. To keep the macro source compatible, some event names do not have to be enclosed in quotes as long as the _ (underscore) character is used instead of the - (dash) character. The following sections list the acceptable constants.

Keys

This section contains the following topics:

- [ASCII Characters](#)
- [Function Keys](#)
- [Extended Keys](#)
- [Miscellaneous Keys](#)
- [Key Name Examples](#)
- [Mouse Events](#)
- [on Events](#)

ASCII Characters

Acceptable ASCII characters are \0..\255. Backslash is used for non-displayable keys.

You may also quote displayable characters such as "a" or "4". The keys \1..\29 are also represented by the following keys:

- C-A
- C-B..C-Z
- C-[,C-\],C-]

- **C-^**
- **C-_**

The ASCII keys **\129..1255** are the same key binding as **\128**.

Function Keys

Acceptable function keys are **F1**, **F2**, and **F12**.

Extended Keys

Acceptable extended keys are the following:

- **Backspace**
- **Delete**
- **Down**
- **End**
- **Enter**
- **Escape**
- **Home**
- **Insert**
- **Left**
- **Pad_5**
- **Pad_Minus**
- **Pad_Plus**
- **Pad_Slash**
- **Pad_Star**
- **PageDown**
- **PageUp**
- **Right**
- **Tab**
- **Up**

Miscellaneous Keys

Acceptable miscellaneous keys are

- **C-A-Enter**
- **C-A-Tab**
- **C-A-Esc**
- **C-A-Backspace**
- **C-PrtScn**
- **C-Ctrl**
- **A-Alt**

Key Name Examples

The following are examples of uses for key names in the Slick-C® language:

```
// Note that "A-a" is different than "A-A" which requires
// the Alt and Shift keys to be pressed.
def "A-x"=safe_exit;
def "A-?"=help;
def "C-X" "b"=list_buffers;
def \0 - \255= nothing;
ctlcombol.on_change()
{
}
ctlcombol."c-s-a"() // Define event handler for Ctrl+Shift+A
{
}
ctlcombol."a"- "z" , "A- "Z"()      // Define event handler for characters
A-Z upper-
                                         // and lowercase.
{
}
void p()
{
    for (;;) {
        key=get_event();
        if (key==name2event("ESC") break;
        if (key==name2event("UP")) {
            // ...
        } else if (key==name2event("DOWN") ) {
            // ...
        }
    }
}
```

}

Mouse Events

The following are acceptable mouse events.

- **lbutton_double_click**
- **lbutton_down**
- **lbutton_triple_click**
- **lbutton_up**
- **mbutton_double_click**
- **mbutton_down**
- **mbutton_triple_click**
- **mbutton_up**
- **rbutton_double_click**
- **rbutton_down**
- **rbutton_triple_click**
- **rbutton_up**

on Events

Below is a list of the **on** events. The acronyms "hsb" and "vsb" stand for horizontal and vertical scroll bar, respectively.]

- **on_change**
- **on_change2**
- **on_close**
- **on_create**
- **on_create2**
- **on_destroy**
- **on_destroy2**
- **on_drop_down**
- **on_got_focus**

- **on_hsb_bottom**
- **on_hsb_line_down**
- **on_hsb_line_up**
- **on_hsb_page_down**
- **on_hsb_page_up**
- **on_hsb_thumb_pos**
- **on_hsb_thumb_track**
- **on_hsb_top**
- **on_load**
- **on_lost_focus**
- **on_resize**
- **on_scroll**
- **on_spin_down**
- **on_spin_up**
- **on_sscroll_lock**
- **on_vsb_bottom**
- **on_vsb_line_down**
- **on_vsb_line_up**
- **on_vsb_page_down**
- **on_vsb_page_up**
- **on_vsb_thumb_pos**
- **on_vsb_thumb_track**
- **on_vsb_top**

Miscellaneous Events

on_select is an acceptable miscellaneous event.

Reserved Words and Keywords

The following keywords are reserved in the Slick-C® language:

- **_command**
- **_metadata**
- **_notinit**
- **_reinit**
- **_str**
- **arg**
- **auto**
- **bool**
- **boolean**
- **break**
- **case**
- **class**
- **const**
- **continue**
- **default**
- **defexit**
- **defined**
- **definit**
- **defload**
- **defmain**
- **do**
- **double**
- **else**
- **enum**
- **enum_flags**

- **extern**
- **false**
- **for**
- **foreach**
- **if**
- **in**
- **instanceof**
- **int**
- **intdiv**
- **interface**
- **long**
- **loop**
- **namespace**
- **no_code_swapping**
- **null**
- **parse**
- **private**
- **protected**
- **public**
- **return**
- **short**
- **static**
- **struct**
- **switch**
- **this**
- **true**
- **typedef**
- **typeless**

- **union**
- **using**
- **var**
- **void**
- **while**
- **with**

The following keywords are reserved for built-in functions:

- **_a2e**
- **_asc**
- **_assert**
- **_callmethod**
- **_callmethod**
- **_chr**
- **_construct**
- **_delete_unused**
- **_deleteel**
- **_dllexport**
- **_dllload**
- **_e2a**
- **_el**
- **_fieldindex**
- **_fieldindex**
- **_fieldname**
- **_fieldname**
- **_findmethod**
- **_findmethod**
- **_get_var**

- **_getfield**
 - **_getfield**
 - **_indexin**
 - **_insertel**
 - **_instanceof**
 - **_instanceof**
 - **_isempty**
 - **_isfunptr**
 - **_length**
 - **_length**
 - **_load**
 - **_load_template**
 - **_make**
 - **_makeempty**
 - **_maybe_e2a**
 - **_nextel**
 - **_set_var**
 - **_setfield**
 - **_setfield**
 - **_sort**
 - **_typename**
 - **_typename**
 - **_update_template**
 - **_varformat**
 - **_write_state**
 - **call**
 - **call_event**
 - **call_index**
-

- **call_key**
- **center**
- **delete_name**
- **dsay**
- **env_match**
- **error_pos**
- **event2index**
- **event2name**
- **eventtab_index**
- **eventtab_inherit**
- **exit**
- **file_match**
- **find_index**
- **get_env**
- **index_callable**
- **index2event**
- **insert_name**
- **isinteger**
- **isnumber**
- **keyin**
- **last_index**
- **lastpos**
- **length**
- **list_bindings**
- **lowcase**
- **name_index2funptr**
- **name_info**
- **name_match**

- **name_name**
- **name_type**
- **name2event**
- **nls**
- **pos**
- **pow**
- **prev_index**
- **replace_name**
- **say**
- **set_env**
- **set_eventtab_index**
- **set_name_info**
- **signal_handler**
- **stop**
- **translate**
- **strappend**
- **strcmp**
- **stricmp**
- **strieq**
- **strip**
- **strrev**
- **substr**
- **togglecase**
- **trace**
- **translate**
- **upcase**
- **verify**

The following keywords are reserved for future use:

- **abstract**
- **catch**
- **final**
- **finally**
- **throw**
- **try**
- **unsigned**
- **virtual**
- **template**

The following keywords are reserved, but deprecated. Avoid using them.

- **_notinit**
- **bigint**
- **bigfloat**
- **bigstring**

The following keywords are reserved and used for event, dialog, and menu programming.

- **_check_box**
- **_combo_box**
- **_command_button**
- **_control**
- **_editor**
- **_form**
- **_frame**
- **_gauge**
- **_hscroll_bar**
- **_image**
- **_inherit**

- **_label**
- **_list_box**
- **_menu**
- **_minihtml**
- **_nocheck**
- **_picture_box**
- **_print_preview**
- **_radio_button**
- **_spin**
- **_sstab**
- **_sstab_container**
- **_text_box**
- **_tree_view**
- **_vscroll_bar**
- **def**
- **defeventtab**
- **endsubmenu**
- **submenu**

All identifiers starting with **p_** are reserved to be used as Slick-C property names.

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.

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.

Code block

A syntactical set of code that is delimited by a specific begin and end. These include **if**, **for**, function defines, areas between braces, etc.

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. Thirteen 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.

FLEXnet® Publisher

Licensing option for multiple users on a server.

Function heading

A term that refers to both the function signature and the comment above it. A function signature is the first line (typically) of a function that contains the function name and the parameters. This can spread across multiple lines, but is still considered one Line of Code (LOC).

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.

List Members

A SlickEdit Context Tagging® feature that automatically lists members when you type a member access operator. Also access this feature by pressing **Alt+Dot** to invoke the **list-symbols** command.

List Symbols

A SlickEdit Context Tagging® feature that lists symbols visible in the current context. Access this feature by pressing **Alt+Dot** to invoke the **list-symbols** command.

List Parameters

A SlickEdit Context Tagging® feature that displays a list of compatible variables and expressions for the current argument when you type a function operator. 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**.

Symbol Completion

A SlickEdit Context Tagging® feature that attempts to complete or correct the symbol under the cursor. Access this feature by pressing **Ctrl+Space** to invoke the **codehelp-complete** command.

Auto-Complete

A SlickEdit feature for saving keystrokes that 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.

Parameter Info

Automatically displays the prototype for a function when you type a function operator, and highlights the current argument within the displayed prototype.

Symbol Information Help

Displays the information about the symbol under the cursor, including documentation comments and its return type.

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 defines a set of projects and retains the settings for an editing session.

Index

Symbols

#endregion, 1436
#pragma, 1430
#region, 1436
(Standard only) XML Formatting Options, 556
(Windows only) Remote proxy port, 992
(Windows only) Use remote proxy, 992
.ext wildcard name match style, 1070
.ext<Enter> in File name, 1069
.BAK, 1079
.bz2
 Decompress .bz2 files on open, 1068
 Look in zip/word/excel files, 783
.cfg.xml File Format, 1164
.editorconfig, 107
.editorconfig support, 961
.gz
 Decompress .gz files on open, 1068
 Look in zip/word/excel files, 783
.seeditorconfig.xml support, 961
.tar
 Look in zip/word/excel files, 783
.xz
 Decompress .xz files on open, 1068
 Look in zip/word/excel files, 783
.Z
 Look in zip/word/excel files, 783
/, 579
/ command, 579
3 Way Merge Dialog, 876
<Tab> in File name, 1069
[Language] File Extensions dialog, 1003
[Language] Referenced in Languages dialog, 1004
_command, 1418
_control, 1473
_inherit, 1443
_str center, 1501

A

Abbreviate similar files, 298, 755
About SlickEdit, 44, 1125
ActionScript Formatting Options, 895
Activate Preview tool window from other windows, 947
Activate tool window, 1086
Activating the XML Outline View, 570
Active form, 1473
Active object, 1473
active project, 168
Ada Beautifier, 497, 499
Ada Formatting Options Standard Edition, 496
Adaptive Formatting, 376
Adaptive Formatting Options (Language-Specific), 1017
Adaptive Formatting Results dialog, 379
Adaptive Formatting Statistics dialog, 382
Add #include, 507
Add (Code Templates Add New Item dialog box), 419
Add Dialog Box, 609
Add File dialog box (Code Templates), 417
Add file to project upon Save As, 1077
Add FTP Profile Advanced Tab, 764
Add FTP Profile General Tab, 763
Add import, 515
Add New Item dialog box (Code Templates), 418
Add New Language dialog, 489
Add opened files to Recent Files, 1068
Add Parameter dialog box (Code Templates), 418
Add source files from working directory, 736
Add Tag File dialog, 218
Add to current workspace, 736
Add to Project, 735
Add to shelf, 623
Add Tree dialog box, 891
Add using statement, 542
Adding a File Type Filter, 1082
Adding completion to command, 1420
adding files to a project, 176
Additional Options (Export/Import Options), 1116
Advanced Appearance Options, 949
Advanced File Mappings, 1000
Advanced Keyboard and Mouse Options, 956
advanced options for C/C++, 504
Advanced tab (FTP Options), 1094
Advanced Tool Options, 839
Alert Icons, 70
alias
 Surround With, 441
Alias Options (Language-Specific), 1026
aliases, 392
 creating from selection, 402
 directory aliases, 393

escape sequences, 396
expansion, 392
expansion option, 1030
language-specific aliases, 394
parameter prompting, 401
Aligning Controls, 1456
All Languages, 1001
All Languages Options, 1001
Allow drag/drop of text, 962
Allow edit and continue (hot swap) where available, 992
Allow editing of source files during debugging, 992
Allow mixed language references, 975
Alt menu hotkeys, 957
AND operator, 1396
android, 287, 287
Annotation Editor dialog, 462
Annotation File Manager dialog, 465
Annotation Type Conflicts dialog, 464
Annotation Types dialog, 463
anonymous e-mail address, 1094
anonymous login, 763
Anonymous Unions, 1386
ANSI-C Formatting Options, 895
Ant Options, 498
Ant targets, 237
Ant-like wildcards
 Ant-like exclusions examples, 893, 893
Apache Ant, 236
API Help, 403
 language-specific, 403
Appearance Options, 906
Appearance Options (Language-Specific), 1011
Append EOF character, 1077
Application Options, 1085
Application theme
 Dark theme, 908
Applying Formatting Rules to XML Files, 571
arg built-in, 1417
argument completion, 389
Argument completion
 options for, 1030
Argument Declarations, 1416
Array length method, 1380
Arrays, 1380
Assignment Operator, 1405
Associate File Types (Quick Start), 58
Associate File Types dialog, 148
Associate File Types Options, 1082
Associate with language, 492
associating workspaces, 149
Associative array variables, 1392
Asynchronous message duration(s), 992
At left margin, 1020
At level of indent, 1020
attach debugger, 249
 core file, 250
 GDB, 250
 Java, 250
 LLDB, 250
 Mono, 250
 running process, 249
attach debugger (GDB), 250
attach debugger (Java), 250
attach debugger (LLDB), 250
attach debugger (Mono), 250
attach debugger (WinDbg), 256
Attach Debugger Menu, 846
auto, 1007
auto caps mode, 1006
auto deselect, 981
Auto Enable Properties dialog box, 868
Auto escape regular expression, 985
Auto exit build window, 962
Auto Folder, 153
auto hide toolbar, 77
auto increment search results window, 783
Auto List Compatible Parameters, 210
auto list members, 208
auto merge, 877
auto parameter information, 209
Auto read only, 1076
auto refresh, 764
Auto reload, 1075
Auto reload all files if current file changed, 1076
Auto reload current file only, 1076
Auto reload timeout (ms), 1076
Auto restore
 Auto restore supported options per monitor configuration, 1088
 Auto restore build window, 1087
 Auto restore clipboards, 1087
 Auto restore files, 1087
 Auto restore line modify, 1087
 Auto Restore Options, 1086
 Auto restore projects tree, 1087
 Auto restore selective display, 1087
 Auto restore symbol browser tree, 1087

Auto restore working directory, 1087
Auto restore workspace, 1087
Auto restore workspace files, 1087
Auto Symbol Translation, 567
Auto-Close Options
 Options, 987
Auto-Close Options (Language-Specific)
 Options, 1038
Auto-Complete, 385
Auto-Complete History, 1029
Auto-Complete Options (Language-Specific), 1028
Auto-display parameter, 1042
Auto-insert matching parameter, 1043
Auto-list compatible parameters, 1043
Auto-list compatible values, 1033
Auto-list members, 1033
auto-restore, 147
Auto-Surround Options (Language-Specific)
 Options, 1040
autodecl, 1431
autodeclctls, 1431
autodeclvars, 1431
automatic directory mapping, 597
Automatic mode, 734
automatic width (comment wrapping), 1024
Automatically build references stack, 975
Automatically close visited files, 987
Automatically correct breakpoint scope, 992
Automatically expand doc comments, 1020
Automatically pop references stack, 975
Automatically pop when no more references, 975
Autos tool window, 84
AutoSave activated, 1080
AutoSave directory, 1081
AutoSave File Options, 1080
Avoid updating context if average time exceeds (ms), 978
AWK Formatting Options, 895

B

Background Process, 292
Background tagging of open files, 971
Background tagging of other files, 971
Background Tagging Options, 969
Backspace in replace mode, 955
Backspace key, 955
Backspace over tab, 955
Backup dialog box, 204
Backup directory path, 1079

Backup File Options, 1078
backup files, 1078
Backup History, 198
Backup History Browser dialog, 199
Backup History tool window, 79
Backup location, 1078
backups, 198
backward, 780
Base Classes dialog box, 360
base file, 600
basic editing, 307
 cut, copy, paste, move, 318
Batch Formatting Options, 895
batch macro, 1372
BBEdit emulation keys, 1212
 clipboard, 1217
 command line and text box editing, 1215
 compiling and programming, 1218
 cursor movement, 1212
 debugging, 1219
 deleting, 1213
 files and buffers, 1216
 inserting, 1213
 macros, 1220
 miscellaneous, 1220
 searching, 1215
 selecting, 1214
 windowing, 1217
beautifier, 479
Beautifier Settings
 Export/Import Options, 1117
Beautifiers, 499
Beautify Menu, 874
Beautifying code, 479
begin end structure matching, 333
 setting match style, 334
 viewing/defining, 333
binary character searching, 677, 707
binary files (editing), 186
Bind Key dialog, 954
bind macro to key, 720
bitmaps (adding to image control), 1491
bitmaps (adding to list box), 1481
Bitwise AND, 1396
Bitwise OR, 1396
Bitwise XOR, 1396
block cursor option, 908
Block insert mode, 325
block selection

inclusive option, 981
block selections, 311
block statement, 1406
Bookmark Stack dialog, 798
bookmarks, 446
 named, 446
 pushed, 450
 relocatable code markers, 449
Bookmarks, 947, 986
Bookmarks dialog box, 799
Bookmarks options, 986
Bookmarks tool window, 796
boolean, 1405
Bounds, 1183
Bounds (Language-Specific), 1009
breadcrumbs, 450
break (primitive), 1410
breakpoints, 251
 conditional, 251
 exceptions, 253
 relocatable code markers, 253
Breakpoints, 947
breakpoints tab, 251
Breakpoints tool window, 80
Brief emulation keys, 1221
 clipboard, 1224
 command line, 1224
 compiling and programming, 1227
 cursor movement, 1221
 debugging, 1228
 deleting, 1223
 files and buffers, 1225
 inserting, 1222
 macros, 1229
 miscellaneous, 1230
 searching, 1223
 selecting, 1228
 text box editing, 1224
 windowing, 1226
Brief regular expressions, 709
Buffer cache size, 1088
buffers, 294
 closing, 306
 listing open, 303
 switching between, 301
build and compile, 228
 auto-build on save, 230
build errors, 239
build methods, 235
 build on save, 230
 commands, 230
 GNU C/C++, 235
 makefiles, 235
 operations, 228
 parsing errors, 240
 projects, 229
 setting shortcuts for Ant targets, 238
 Xcode, 236
build command
 escape sequences, 230
build errors, 239
 listing, 240
 navigation, 240
 parsing with reg expressions, 240
 viewing, 239
Build Menu, 842
build methods, 235
 GNU C/C++, 235
 Xcode, 236
build options, 175, 175
build settings (projects), 172
Build tab (Project Properties), 827
Build tag file (C++ Compiler), 504
Build tool window, 80
Build Window, 909
Build window
 Erased lines, 962
 Show change directory commands in build window, 963
 Show VSLICKERRORPATH in build window, 963
build window (auto exit), 962
Build workspace tag file with references, 974
Building CTag Based Tag Files, 221
building running android, 288
Building Tag Files, 216
building tag files, 218
built-in functions, 1425
byte offset navigation, 336

C

c command, 583
C# Beautifier, 499
C# Organize Imports, 542
C# Organize imports options, 543
C# tutorial, 1131
C++ Formatting Options
 Community Edition, 895

Standard Edition, 895
C/C++ advanced options, 504
C/C++ Beautifier, 499
C/C++ Compiler Properties, 502
C/C++ Formatting Options, 500
C/C++ preprocessing, 505
cache size (for buffers), 1088
cache size (for tagging), 1088
calculating (see mathematics), 638
calculator dialog box, 638
Call Stack tool window, 84
call tree, 358
callers tree, 359
Cancel button (adding), 1460
capture groups for Perl, 649
capture groups for SlickEdit, 652
capture groups for Vim, 654
Cargo workspaces, 164
case sensitive, 634
case statement, 1413
casting types, 1401
Categories (Code Templates Add New Item dialog box), 418
Categories (Template Manager dialog box), 416
CFML Formatting Options, 559
CFScript Formatting Options, 895
Ch Formatting Options, 895
Change directory, 909
Change Directory dialog box, 756
changing emulations, 109
character selection, 309
 inclusive option, 981
characters (identifiers), 1048
characters (inserting literal), 325
Check Box control, 1479
Check In Dialog Box, 609
Check Out Dialog Box, 609
Checkin/Checkout Files dialog, 1109
child backup directory, 1079
CICS Formatting Options, 508
Clang (attaching to remote process), 250
Class, 947
Class Exclusion Manager dialog, 341
class properties, 84
Class tool window, 339
 exclusion manager, 341
 filtering hierarchy pane, 341
 filtering/sorting members pane, 342
Classes (debug) tool window, 845
clear bookmark, 449
Clear Modified Lines, 366
Clear text box on Enter, 1069
Click past end of line, 969
Clipboard Inheritance(R), 1468
clipboards (how to use), 320
Clipboards tool window, 80
Close after find/replace, 984
Close deletes pushed bookmarks, 986
close workspace, 161
Closing a dialog box, 1460
closing files, 201
 automatically, 331
closing SlickEdit (see exiting SlickEdit), 43
COBOL Formatting Options, 508
Code Annotations, 458, 947
 Annotation Editor dialog , 461
 filtering, 461
 go to annotation, 461
code annotations
 code marker, 459
 copying, 462
 creating, 460
 deleting, 462
 editing, 462
 for code reviews, 467
 for recording tasks, 467
 handling type conflicts, 464
 locations, 458
 managing files, 464
 moving, 462
 private and shared, 459
 types, 458
 user-defined, 465
 viewing, 461
Code Annotations tool window, 460
code folding, 369
code navigation, 329
code reviews with annotations, 467
Code Templates, 409
 Add New Item dialog box, 418
 Creating a multi-file template, 421
 Creating templates, 411
 Export/Import Options, 1117
 Global Substitution Parameters, 417
 Locating Templates, 419
 Manually creating a template, 419
 Metadata file reference, 422
 Organizing templates, 414

Pre-defined substitution parameters, 412
Substitution parameters, 412
Template Manager dialog box, 415
Template Options dialog, 417
User templates (Locating), 419
CodeWarrior emulation keys, 1231
 clipboard, 1235
 command line and text box editing, 1235
 compiling and programming, 1237
 cursor movement, 1232
 debugging, 1238
 deleting, 1233
 files and buffers, 1236
 inserting, 1232
 macros, 1239
 miscellaneous, 1240
 searching, 1233
 selecting, 1234
 windowing, 1237
CodeWright emulation keys, 1241
 clipboard, 1244
 command line and text box editing, 1244
 compiling and programming, 1246
 cursor movement, 1241
 debugging, 1247
 deleting, 1242
 files and buffers, 1246
 inserting, 1242
 macros, 1248
 miscellaneous, 1248
 searching, 1242
 selecting, 1243
 windowing, 1246
Coding (Quick Start), 57
collapse code block, 370
Color Coding
 advanced configuration, 145
 creating support for a new language, 139
General Tab , 1047
How to add a interpolation to string, 142
How to add a line comment, 140
How to add a multi-line comment, 140
How to add a string, 141
How to add words, 139
How to define color coding for numbers, 141
introduction , 126
Language Tab , 1057
More Tab , 1052, 1053
Numbers Tab , 1054
Options (Language-Specific) , 1046
Settings Tab , 1051
Tags Tab, 1059
Tips on using regular expressions matching, 143
Tokens Tab , 1049
use , 139
Color Coding General Tab, 1047
Color Coding Language Tab, 1057
Color Coding More Tab, 1052, 1053
Color Coding Numbers Tab, 1054
Color Coding Options (Language-Specific), 1046
Color Coding Profile
 creating a new color coding profile, 1168
 modifying a color coding profile, 1168
Color Coding Profile Format, 1168
Color coding profile name, 491
Color Coding Profiles, 1168
Color Coding Search Options dialog, 779
Color Coding Settings Tab, 1051
Color Coding Tags Tab, 1059
Color Coding Tokens Tab, 1049
Color Inheritance, 129
Color Options, 911
Color Picker dialog, 129
Color Profile, 760
Color Settings dialog box, 911
colors, 126
 Color Coding, 126
 customizing, 126
 editor windows, 145
 for embedded languages, 130
 of special characters, 367
 profiles, 129
 Symbol Coloring, 126, 130
Colors (Quick Start), 55
column indicators, 69
comapring with version control
 compare, 621
Combo Box control, 1482
Command Button control, 1479
command line
 (see SlickEdit command line), 89
 invocation options, 37
 Launching SlickEdit from OS command line, 37
command line parameters, 37
Command line prompting, 957
Command Line Switches, 91
commands, 93
 binding to keys, 115

defining with Slick-C, 1418
see also SlickEdit command line, 89
viewing associated key bindings, 91

Commands (Version Control Options), 1105

Commands, Built-ins, Defs (Differences), 1425

Comment Options (Language-Specific), 1017

comment styles (C++ language constructs), 1373

Comment width, 858

Comment Wrap Options (Language-Specific), 1022

comment wrapping, 457

- enable, 1023
- reflowing comments, 457
- sync vertical line, 1024
- width settings, 1023

comments, 452

- block comments, 452
- commenting lines, 452
- configuration, 452
- configuring block comments, 1018
- configuring doc comments, 1020
- configuring line comments, 1019
- creating doc comments, 453
- doc comment examples, 453
- Doxygen, 455
- editing options, 1021
- Javadoc, 453
- line comments, 452
- reflowing, 457
- removing, 452
- string editing, 456
- wrapping, 457

XMLDoc, 454

Common Formatting Options for Brace-style Languages, 895

common keys (redefining), 99

Compare file contents before auto reload, 1076

Compare Options dialog box, 602

comparing files, 592

comparing folders (directories), 593

comparing parts of files, 594

comparing symbols, 594

compile, 228

- projects, 229
- Visual C++, 229
- vsbuild, 229

Compile/Link tab (Project Properties), 830

Compiler Properties (Language-Specific), 1064

compilers.xml, 1160

Compiling macros, 1447

completion, 385

- Adding to command, 1420
- Auto-Close, 387
- word/variable, 388

Completion (Context Tagging), 211

Completion (CTags Based Tagging), 214

Completion choice, 1032

Completions in dialogs, 388

Concatenation, 1398

concurrent process buffer (Build window), 80

conditional breakpoints, 251

confidence level, 382

config (see configuration), 1159

configuration, 1159

- changes not on menu, 1151
- configuration variables, 1154
- environment variables, 1149
- of build settings, 172
- of project directories, 170
- of project tools, 170
- of projects, 169
- options for saving, 1089
- set command, 1152
- system config files, 1162
- table of sys config files, 1162
- table of user config files, 1160
- user config directory, 1159
- user.cfg.xml, 1151

Configurations (Debugging Options), 996

Configurations (New Project Tool Wizard), 838

Configure Error Parsing, 241

Configure Error Parsing Options, 241

Configuring Interactive Profiles, 643

Configuring Single File Project Profiles, 184

constants (defining), 1376

constants (numeric), 1375

context menus, 86

Context Tagging, 208

- Building tag files, 216
- configuring COBOL, 221
- Configuring other languages, 221
- Creating compiler-specific tag files , 216
- Creating language-specific tag files, 218

Features, 208

managing tag files, 222

options for, 224

Quick Start, 60

Tag Files dialog, 888

workspace files, 216, 221

Context Tagging Features
Auto List Compatible Parameters, 210
Completions, 211
List Members, 208
Parameter Information, 209
Statement Level Tagging, 213
Symbol Browsing, 212
Tag-Driven Navigation, 208
Context Tagging Options, 972
Context Tagging Options (Language-Specific), 1041
Context Tagging result cache maximum, 976
Context Tagging toolbar, 79
continue (primitive), 1410
Continue bullet list on Enter (comment wrapping), 1024
control characters (inserting), 325
conventions, 14
 code syntax, 14
 menus and dialogs, 14
convert HTML symbols, 567
convert Unicode to UCN, 1147
Cool Features, 1126
Copy, 500
Copy Code Block dialog, 445
Copy dialog box, 204
Copy source to template directory (Code Templates Add File dialog box), 417
Copy Unicode As Menu, 769
copying code annotations, 462
copying code blocks, 445
Copyrights Tab (about SlickEdit), 1125
core file, 250
Count number of lines, 1075
CR, 749, 751, 1061
CR w/o LF erases line in build window, 962
CR/LF, 749, 751, 1061
Create file outline, 1011
Create New Configuration dialog, 169
Create new workspace, 736
Create project directory from project name, 736
create workspace, 162
creating a file from a selection, 189
Creating a new category (Template Manager dialog box), 415
Creating a new template (Template Manager dialog box), 415
creating a shelf, 621, 621
creating custom project types, 168
creating directory aliases, 393
creating extension-specific aliases, 395
creating files, 189
Creating new configurations (compiler), 503
creating project files, 181
creating projects, 167
CTags Based Tagging, 214
 Building CTags Based tag files, 221
 Features, 214
CTags Based Tagging Features
 Completions, 214
 Tag-Driven Navigation, 214
CTags Tagging Options, 1110
Ctrl+/ (go to reference), 330
Ctrl+Alt+Dot (go to declaration), 329
Ctrl+Comma (pop bookmark), 330
Ctrl+Dot (go to definition), 329
Ctrl+Shift+V, 321, 962
CUA emulation keys, 1203
 clipboard, 1206
 command line and text box editing, 1206
 compiling and programming, 1208
 cursor movement, 1203
 debugging, 1210
 deleting, 1204
 files and buffers, 1208
 inserting, 1204
 macros, 1210
 miscellaneous, 1211
 searching, 1204
 selecting, 1205
 windowing, 1208
CUA text box, 962
current character, 70
Current class, 1030
Current Context toolbar, 79, 343
Current file, 1030
Current line box color, 910
Current line column color, 910
Current line highlight, 910
Current paragraph, 858
Cursor and Mouse, 908
Cursor left/right in leading spaces, 969
Cursor Movement, 968
cursor navigation, 335
Cursor page up/down, 1183
Cursor right/left wraps to next/prev line, 968
Cursor style, 908
Cursor up/down places cursor in virtual space, 969

-
- Cursor up/down within soft-wrapped lines, 969
Custom Parameters tab (Template Manager dialog box), 416
custom project types, 168
Custom View, 153
Customize, 736
Customize Project Types dialog, 168
CVS, 619
CVS Options, 619
- D**
- D Formatting Options, 895
Data Set Utilities dialog box, 1494
Date display style, 909
dBASE Formatting Options, 549
Debug, 844
debug (see also debugging), 247
debug key bindings, 248
Debug Menu, 844
Debug Sessions toolbar, 84
debug tab (FTP Options), 1097
Debug toolbar, 78
Debug Windows Menu, 845
debugger (see also debugging), 247
Debugger Options dialog, 253
debugging, 247
 attach to core file, 250
 attach to Java Virtual Machine, 250
 attach to Mono Virtual Machine, 250
 attach to remote process, 250
 attach to running process, 249
 C/C++ with GNU debugger, 254
 C/C++ with WinDbg debugger, 255
 GDB, 250, 250
 generate debug, 253
 Google Go, 284
 hot-swap debugger, 992
 mixed mode view, 248
 multiple sessions, 249
 named sessions, 249
 Perl, 273
 PHP, 260
 Python, 268
 Ruby, 278
 setting breakpoints, 251
 setting options for, 253
 Slick-C, 1450
 tool windows, 254
debugging android, 289
- Debugging Configurations Options, 996
Debugging Directories Options, 995
Debugging General Options, 990
debugging gwt, 286
Debugging Numbers Options, 993
Debugging Options, 989
Debugging Runtime Filters Options, 994
Decimal ruler, 910
def, 1441
def primitive, 1441
Default Arguments, 1417
Default layout applied to dragged out document tabs, 299
default local directory, 1094
DefaultName element (Code Templates metadata file reference), 423
defeventtab, 1441
defeventtab primitive, 1441
defining constants, 1376
defining event tables, 1443
defining functions, 1415
defining keys, 1441
defining procedures, 1415
defining project dependencies, 169
Defining Special Characters, 367
definit, 1445
defload, 1445
defmain, 1426
Defs, 948
Defs tool window, 344
Defs Tool Window
 options, 345
 Outline View for XML, 569
Delete, 501
delete bookmark, 449
Delete Code Block dialog, 444
Delete key, 955
Delete Menu, 767
delete selection, 981
Deleting a template (Template Manager dialog box), 416
deleting code annotations, 462
deleting code blocks, 444
dependencies (defining for projects), 169
Dependencies tab (Project Properties), 833
Dependency of, 736
deploying gwt, 286
deprecation, 1431
Derived Classes dialog, 360

Description element (Code Templates metadata file reference), 424
deselect after paste, 981
Details tab (Template Manager dialog box), 416
Dialog Box Retrieval, 1492
dialog boxes (forms for macros), 1500
Dialog Editor, 1454
Dialog Editor Properties dialog box, 1455
Diff columns, 1008
Diff dialog box, 590
diff options tab, 883
Diff Setup dialog box, 883
Differences Between SlickEdit Vim and gvim, 1310
diffing files, 590
diffing symbols, 594
diffmap.ini, 1160
diffsessions.xml, 1160
DIFFzilla, 590
 backup history, 597
 comparing folders (directories), 593
 comparing parts of files, 594
 comparing symbols, 594
 comparing two files, 592
 directory mapping, 597
 dynamic difference editing, 590
Files tab, 879
 generating file lists, 595
 options, 881
 path info, 879
DIFFzilla Dialog, 878
Directories (Debugging Options), 995
directories (for projects), 170
Directories tab (Project Properties), 822
directory (auto-change), 909
directory aliases, 393
 creating, 393
 embedding env variables, 394
 using, 393
Directory List Box control, 1488
directory mapping, 597
Directory Project Options, 1085
Directory View, 153
Disable auto-loading of scripts, 993
Dismiss on select, 1069
Display auto-complete after idle, 979
Display parameter information after (ms) idle (0 implies no delay), 977
display tool tips, 120
Displaying Dialog Boxes, 1461
do (loop), 1407
do not upload, 1094
Doc Comment Editor dialog, 455
doc comments, 453
 configuration, 1020
docking toolbars, 76
document math, 640
Document Menu, 849
Document Mode, 734
Document Overview Bar, 472
Document Tabs, 295
 Abbreviate file tab captions, 967
 abbreviate similar files, 298
 Document tab title, 967
 Hide known file extensions, 967
 Hiding the document tabs with Zoom Toggle, 298
 New file tab position, 967
 Show close buttons on document tabs, 967
 sort order options, 966
 Zoom (hide tabs) when one window, 965
documentation, 13
documentation conventions, 14
documentation feedback, 13
Doxygen comments, 455
drag and drop text, 320
drag/drop, 962
Draw box around current line, 948
Draw box only, 910
Drive List control, 1487
DTD caching, 556
dual monitors, 95
dynamic difference editing, 590
Dynamic Surround, 436
dynamic-language projects, 167

E

e/edit command Smart Open, 1068
Eclipse emulation keys, 1250
 clipboard, 1253
 command line and text box editing, 1253
 compiling and programming, 1255
 cursor movement, 1250
 debugging, 1256
 deleting, 1251
 files and buffers, 1254
 inserting, 1251
 macros, 1257
 miscellaneous, 1257

searching, 1251
selecting, 1252
windowing, 1254

Edit, 500

Edit 'A B C' start on file A, 1068

Edit (File Manager), 204

Edit Alias Parameter dialog, 1028

Edit Menu, 765

edit other copy unicode as ucn menu, 1147

Edit Other Menu, 767

Edit Select Menu, 766

Edit toolbar, 78

Editing an existing template (Template Manager dialog box), 415

editing code annotations, 462

Editing Options, 959

Editing Options (Language-Specific), 1006

editing recorded macros, 721

Editing XMLDoc Comments, 544

Editor Control, 1478

Editor Window Options, 964

editor windows, 294

- closing, 306
- duplicating, 301
- left margin width, 300
- linking, 305
- splitting, 300
- tiling, 301

Elements (Code Templates metadata file reference), 423

embedded language color, 130

embedded languages , 23

- HTML , 23
- Perl , 24
- UNIX , 24

embedding env variables in aliases, 394

Emulation (Quick Start), 54

Emulation Options, 951

emulations, 108

- changing, 109
- determining keys/functions, 110
- supported, 108
- tables of keys, 1203

Emulations (list), 108

Enable auto-completion, 1029

enable comment wrap, 1023

Enable Python pretty printing, 993

Enable soft wrap, 1026

encoding, 1144

SBCS/DBCS, 1144

Unicode, 1145

UTF-8, 1144

Encoding (global option), 1074

END command saves the file, 1183

End key, 955

end of line, 367

Enhanced Scroll Bar, 472

Enter Alias Parameter dialog, 401

Enter key, 955

Enter New Alias Name dialog, 395

Enter New Profile Name dialog, 139

Enter places cursor in prefix area, 1183

Entire block comment (Comment Wrap tab), 858

entity reference translation, 567

Enumerate dialog box, 771

Enumeration, 313

environment variables, 1149

- set command, 1152
- setting in user.cfg.xml, 1151
- VSЛИCKINCLUDE, 1437
- VSЛИCKPATH, 1437
- VST, 1430

Epsilon emulation keys, 1270

- argument and repeating a key, 1281
- clipboard, 1274
- command line, 1278
- compiling and programming, 1275
- cursor movement, 1270
- debugging, 1276
- deleting, 1272
- files and buffers, 1274
- inserting, 1271
- macros, 1277
- miscellaneous, 1279
- searching, 1272
- selecting, 1273
- text box editing, 1278
- windowing, 1275

Error File dialog box, 240

error parsing, 240

- configuration, 241
- exclusions, 243
- sample expression, 244
- testing expressions, 245

Error Regular Expressions dialog box, 241

escape sequences for aliases, 396

Escape sequences for build commands, 230

Event driven dialog (event tables), 1443

Event Names, 1538
Event tables, 1441
event-driven dialog boxes, 1368
events and event tables, 1441
examples of Perl reg expressions, 676
examples of SlickEdit reg expressions, 691
examples of Vim reg expressions, 706
Excel
 multi-file find, 783
exception breakpoints, 253
Exceptions tool window, 81
Exclusions, 1080
 exclusions (error parsing), 243
Executable name, 736
execute Ant target, 238
execute NAnt target, 238
Executing programs from macro, 1514
Exit confirmation prompt, 1090
Exit Options, 1089
exit process, 962
Exit SlickEdit on AutoSave, 1080
exiting SlickEdit, 43
 default options, 43
 with modified buffers, 43
expand code block, 370
Expand tabs, 746
Expand tabs to spaces, 1077
Expanded text, 1031
Explorer Open dialog box, 743
Explorer Standard Open dialog box, 743
Export Groups, 1114
export key bindings, 117
export to HTML, 558
Export/Import Options, 1112
expressions (unary operators in), 1395
extend file history, 1156
Extend line comments, 1021
extend workspace history, 1157
extension-specific aliases
 creating, 395
 escape sequences, 396
 parameter prompting, 401
Extract method (Quick Refactoring), 483

F

f command, 331
failed saves, 196
Fast auto read only, 1076
Fast line count on partial load, 1073

Feature Notification, 292
Feature Notifications
 options, 1090
features (new), 5
file associations, 148
File element (Code Templates metadata file reference), 425
File Exclusions
 Ant-like exclusions examples, 893
File Extension Manager, 491, 999
file filters, 1081
file history (File menu), 1156
file history (Project menu), 1157
File List Box control, 1487
file lists, 962
file lists (generating), 595
File locking, 1074
File Manager (Using), 202
File Manager Files Menu, 733
File Manager Menu, 731
File Manager Select Menu, 732
File Menu, 728
File Navigation, 908
File Open dialog box, 743
File Options, 1065
File Options (Language-Specific), 1060
File Options dialog box, 195
file search order, 1167
File Sort dialog box, 204
File Tabs, 753
 Abbreviate file tab captions, 967
 abbreviate similar files, 755
 Hide known file extensions, 967
 New file tab position, 967
 Show close buttons on document tabs, 967
 sort order options, 966
File Tabs tool window, 81
file types, 148
fileman select menu, 204
files, 186
 auto-close, 331
 autosave options, 197
 Backup History, 198
 backups, 198
 closing, 201
 comparing parts of, 594
 comparing two files, 592
 configuration files and directories, 1159
 creating from selections, 189

diffing file history, 597
encoding, 1144
failed saves, 196
file filters, 1081
finding files to open, 193
FTP, 637
generating lists, 595
inserting into buffers, 194
listing open files/buffers, 303
merging, 599
navigating between, 335
opening, 191
quick create/open, 189
save as, 196
saving, 196
setting associations, 148

Files, 948
Files element (Code Templates metadata file reference), 426
files menu, 728
Files of Type Filters, 1081
Files tab (DIFFzilla), 879
Files tab (Project Properties), 820
Files tab (Template Manager dialog box), 416
Files tool window, 82
Files Tool Window, 189
Fill Selection, 315
Filtering marked text, 1510
filters, 764
filter_selection procedure, 1510
find
 see also searching, 574
Find all references immediately, 974
Find and Replace tool window, 776
find command, 579
Find completions which fix minor typos, 1037
Find File dialog box, 193
Find Files tab (Find and Replace), 789
Find in Files tab (Find and Replace), 780
Find references in background, 974
Find references incrementally, 974
Find references search strategy, 974
Find Symbol, 948
Find Symbol tool window, 792
Find tab (Find and Replace), 778
finding files, 193
find_error command, 1449
find_proc command, 1448
Finish (New Project Tool Wizard), 841

firewall proxy tab, 1095
First line is top, 1019
fixed right margin (comment wrapping), 1024
fixed width (comment wrapping), 1023
float toolbar, 77
Floating point numbers, 1375
Folder View, 153
follow characters, 1048
Font Configuration dialog box, 938
Font dialog box, 124
Font Options, 938
fonts, 122
 changing, 122
 editor windows, 124
Fonts (Quick Start), 56
for (loop), 1407
foreach (loop), 1408
foreground search, 785
Formatting Options (Language-Specific), 1015
Formatting Rule Set Configuration, 570
Fortran Formatting Options, 508
forums, 13
fp command, 331
Frame control, 1479
FTP, 635
 configuration options, 637
 connecting, 636
 disconnecting, 637
 FTP Profile Manager, 635
 opening files, 637
 tool windows, 635
ftp connection profile, 635
FTP Default Options, 1093
FTP Default Options Advanced Tab, 1094
FTP Default Options Debug Tab, 1097
FTP Default Options Firewall/Proxy Tab, 1095
FTP Default Options SSH/SFTP Tab, 1097
FTP files, 635
FTP Menu, 730
FTP Options Advanced Tab, 1094
ftp options firewall/proxy tab, 1095
ftp options general tab, 763
ftp options ssh/sftp tab, 1097
FTP tool window, 635
full screen mode, 95
Function Prototypes, 1424
functions (defining), 1415
functions (finding), 1425

G

Gauge control, 1489
GDB (attaching to remote process), 250
GDB (multiple session debugging), 249
GDB (newer version), 996
GDB (setting configurations), 996
GDB projects, 166
General Appearance Options, 906
General Editing Options, 960
General Options (Language-Specific), 1002
Generate debug, 253
generate file list, 595
generate references, 350
Get Dialog Box, 609
getting started android, 287
getting started gwt, 286
get_string procedure, 1516
Git, 618
Git Options, 618
Global Alias Options, 989
global aliases, 393
global directory, 1079
Global Find dialog, 205
global nested directories, 1079
Global Replace dialog box, 205
globs
 Ant-like exclusions examples, 893
GNU C/C++ build methods, 235
GNU C/C++ projects, 165
GNU Emacs emulation keys, 1258
 argument and repeating a key, 1270
 clipboard, 1262
 command line, 1266
 compiling and programming, 1264
 cursor movement, 1258
 debugging, 1265
 deleting, 1260
 files and buffers, 1263
 inserting, 1260
 macros, 1266
 miscellaneous, 1268
 searching, 1261
 selecting, 1261
 text box editing, 1266
 windowing, 1263
Go to Bookmark dialog box, 797
Go to Bookmark feature, 448
go to declaration, 329

go to definition, 329
 navigation option, 1043
Go to import, 515
go to reference, 329
Google Go, 284
Google Go Formatting Options, 895
Goto using statement, 542
Grid Settings dialog, 866
Groovy Beautifier, 499
GUID Generator, 629
gwt, 286, 286

H

hash tables, 1382
hash tables editing, 864
header files (Slick-C), 1360
Header Files, Including, 1437
Help Menu, 1123
help resources, 13
help system, 14
Hex
 Bytes per column, 1010
 Language-Specific View Options, 1010
 Number of columns, 1010
Hex character code, 1375
hex view, 371
hexadecimal display, 326
hexadecimal numbers, 1375
Hide maximized child window titlebars, 965
Hide mouse pointer, 908
hide toolbar, 77
Highlight current line, 948
Highlight current line (per Language), 948
Highlight matches, 780
Highlight matching blocks, 963
Highlight matching blocks after (ms) idle, 963
Highlight matching blocks timeout (ms), 963
Highlight matching symbols under cursor, 1045
Highlight references in editor, 975
highlight symbols, 589
Highlight tool window, 474
Highlighting, 474
History, 1029
history (increase File menu), 1156
history (increase Project menu), 1157
History dialog box, 193
History Options, 1083
History retrieval, 984
Home key, 956

HOME Key Configurations, 99
Home places cursor on command line, 1183
Horizontal scroll bar, 909
host name of firewall, 1096
host name of FTP server, 763
host type, 763
hot fixes, 32
 automatic installation, 33
 list of installed, 34
 manual installation, 32
 unloading, 34
hot key, 87
hot-swap debugger, 992
hotfixes, 32
hotkeys, 957
Hotspot Options, 988
HTML, 557
 beautifying, 560
 browser configuration, 558
 exporting current file, 558
HTML and XML Formatting Options Standard Edition, 558
HTML Beautifier Case and Quoting tab, 564
HTML Beautifier Comments & Languages tab, 566, 566
HTML Beautifier dialog box, 560
HTML Beautifier Indent tab, 560
HTML Beautifier Tags tab, 562
HTML Formatting Options, 559
HTML symbol translation, 567
HTML toolbar, 558

|

iconized windows, 862
Identifiers, 1048
IDL Formatting Options, 895
if statement, 1405
Ignore forward class declarations, 1044
ignore spaces, 878
Image control, 1490
Immediately update context if maximum observed time is less than (ms), 978
Import Color Profile, 922
Import File List dialog, 181
import key bindings, 117
Import Options, 1115
Import options (C# Imports), 543
Import options (Java Imports), 529
Import options (Refactoring), 515, 542
Importing Color Profiles, 922
importing files to projects, 181
importing makefiles, 182
Imports, 515, 542
Imports Menu, 874
Include options (Refactoring), 507
Includes, 507
Including macro header files, 1437
increase file history (File menu), 1156
increase workspace history (Project menu), 1157
Incremental search highlighting, 985
incremental searching, 575, 576
Indent by syntax indent, 1007
indent style, 1007
indicator (macro recording), 70
indicator (named bookmark), 447
indicator (pushed bookmark), 451
Inheritance, 1468
Inherited Code Found dialog box, 1459
initial directory, 764
Initial NumLock state, 958
Initialize search string, 984
Initialize with default options, 985
initializing macro modules, 1445
Insert File dialog box, 194
Insert Form Source dialog box, 1458
Insert keyword before parameter when required, 1043
Insert Literal dialog box, 325
insert mode, 962
Insert open parenthesis for functions, 1032
insert real indent, 1008
Insert space after comma, 1043
insert toggle, 70
Inserting control characters, 325
inserting files into buffers, 194
Installed templates (Locating), 419
installing SlickEdit, 26
Installing Xdebug, 261
Instances, 1473
Interactive Profiles
 Configuring, 643
Interactive tool window, 84, 643
Internet Protocol (IP), 1093
Invocation Options, 37
invoking Ant targets, 237
invoking NAnt targets, 237
ISPF Copy Lines, 1190
ISPF Delete Lines, 1191

-
- ISPF Emulation, 1182
 - ISPF emulation keys, 1281
 - clipboard, 1286
 - command line and text box editing, 1287
 - compiling and programming, 1289
 - cursor movement, 1281
 - debugging, 1290
 - deleting, 1283
 - files and buffers, 1288
 - inserting, 1283
 - macros, 1291
 - miscellaneous, 1292
 - selecting, 1284
 - selective display, 1291
 - windowing, 1289
 - ISPF Emulation Options, 1182
 - ISPF Exclude Lines, 1197
 - ISPF Expose First Lines, 1191
 - ISPF Expose Last Line, 1192
 - ISPF Expose Next Level of Code, 1194
 - ISPF Insert After, 1189
 - ISPF Insert Before, 1189
 - ISPF Insert Bounds Ruler, 1190
 - ISPF Insert Columns Ruler, 1190
 - ISPF Insert Lines, 1191
 - ISPF Insert Mask Line, 1193
 - ISPF Insert Tabs Ruler, 1195
 - ISPF Insert Text, 1195
 - ISPF Join Lines, 1196
 - ISPF Line Command A, 1187
 - ISPF Line Command B, 1187
 - ISPF Line Command Delete, 1187
 - ISPF Line Command Exclude, 1188
 - ISPF Line Command First, 1187
 - ISPF Line Command Uppercase, 1188
 - ISPF Line Commands, 1188
 - ISPF Line Labels, 1188
 - ISPF Lowercase Lines, 1192
 - ISPF Make Data Lines, 1193
 - ISPF Move Lines, 1192
 - ISPF Overlay Lines, 1194
 - ISPF Repeat Lines, 1194
 - ISPF Select Lines, 1197
 - ISPF Shift Lines Left or Right, 1188
 - ISPF Split Line, 1196
 - ISPF Unsupported Primary Commands, 1198
 - ISPF Uppercase Lines, 1196
 - J# Formatting Options, 895
 - Java, 510
 - Java Beautifier, 499
 - Java compiler, 515
 - Java Compiler Properties, 518
 - Java Formatting Options, 499
 - Java Live Errors, 515
 - Java Options Android tab, 519
 - Java Options AppletViewer tab, 519
 - Java Options Classpath tab, 519
 - Java Options Compiler tab, 519
 - Java Options dialog, 519
 - Java Options GWT tab, 519
 - Java Options J2ME tab, 519
 - Java Options Jar tab, 519
 - Java Options Javadoc tab, 519
 - Java Options JRE tab, 519
 - Java Options Live Errors tab, 519
 - Java Organize Imports, 515
 - Java Organize imports options, 529
 - Java projects, 166
 - Javadoc Beautifier dialog box, 531
 - Javadoc comments, 453
 - Javadoc Editor dialog box, 531
 - JavaScript Beautifier, 499
 - JavaScript Formatting Options, 499
 - JCL Formatting Options, 508
 - Join comments, 1022
 - JSP TagLib Formatting Options, 559
 - Jump over tab characters, 969
 - Jump to first item when finding references, 975
 - jumping to a tag, 329
 - JUnit, 516
 - Justification dialog box, 857
 - justified, 1026
 - justify style, 1025
 - K
 - keep alive default FTP, 1095
 - keep alive FTP connection, 764
 - Key Binding Options, 951
 - Key Binding Profile Format, 1181
 - key bindings, 111
 - binding macros, 720
 - creating, 115
 - definitions of terms, 111
 - emulations, 108
 - export/import, 117
 - key message delay, 118

J

unbinding, 116
used in debugging, 248
viewing associated commands, 91

key definition, 91
Key message delay, 957
key names, 87
key names (option), 950
key shortcuts in text boxes, 97
Keyboard and Mouse Options, 950
Keyboard Options
 advanced, 956
Keys Help, 110
Keyword case, 497
Keywords, 1030

L

I command, 579
Label control, 1476
Language Constructs, 1373
Language Manager, 488, 998
Language Options, 997
Language Setup, 492
language support , 16
Language tab (Color Coding Setup), 1057
language-specific aliases, 394
 creating from selection, 402
language-specific API help, 403
language-specific projects, 176
Largest file to AutoSave, 1081
Last line is bottom, 1019
Launching SlickEdit, 36
Layout applied to dragged out document tabs, 299
Layouts menu, 299
Icase, 324
Leave selected, 985
left and respace, 1025
Left and Right, 1019
LF, 749, 751, 1061
libraries, 160
License Agreement Tab (about slickedit), 1125
licensing, 27
Light bulb, 1031
Limit size of backup, 1079
Limitations with sharing your configuration with multiple instances, 106
line endings, 749, 751, 1061
Line endings for new files (global option), 1074
line feed, 749, 751, 1061
Line format, 749, 751, 1061

line hex, 371
line indicators, 69
Line insert style, 962
line navigation, 336
line numbers, 367, 1010
line selections, 310
Line separator char, 746
Line wrap, 969
Link Window dialog box, 305
Linking to a window, 305
List all occurrences, 780
List Box control, 1480
list clipboards, 320
list clipboards dialog box, 320
List command line completions, 908
List errors, 240
List Files dialog box, 202
List files of type, 745
list hot fixes, 34
List include files after typing #include, 1033
List Members, 208
List of matches, 1031
list open files/buffers, 303
list project files, 851
list workspace files, 851
Lists shelves, 624
literal characters (inserting), 325
Live Error Profiles (Language-Specific), 1061
Live errors, 480
live errors, 515
Live Errors tab (Project Properties), 834
LLDB (multiple session debugging), 249
LLDB projects, 166
load (for different drives), 196
Load as Binary, 1060
load command, 1447
Load entire file, 1075
Load File, 501
Load File Options, 1072
Load Module dialog box, 724
load partial if larger than, 195
Load partially for large files, 1075
Load partially when files are larger than, 1075
Loaded Classes tool window, 84
Loading macros, 1447
loading project files, 183
Localization, 909
Locals, 1030
Locals tool window, 85

-
- Locating templates, 419
 - Location, 1020
 - Location (Code Templates Add New Item dialog box), 419
 - locations for code annotations, 458
 - locked files, 195
 - look in, 778
 - loop (loop), 1409
 - Loops, 1406
- M**
- Mac Option/Alt key behavior, 957
 - Mac resize borders, 965
 - macOS, 96
 - format buffer, 42
 - special features, 22
 - writing selection filters, 1510
 - macOS emulation keys, 1293
 - clipboard, 1298
 - command line and text box editing, 1296
 - cursor movement, 1293
 - deleting, 1295
 - files and buffers, 1298
 - inserting, 1294
 - macros, 1299
 - miscellaneous, 1299
 - searching, 1296
 - selecting, 1295
 - windowing, 1298
 - macOS Installation, 26
 - macOS style Browse for Folder dialog, 950
 - Macro Menu, 859
 - macro prompting, 1516
 - macros
 - programmable, 723
 - recorded, 718
 - macros (batch), 724
 - Maintenance and Support, 3
 - Make backup files, 1078
 - makefile (build with auto), 235
 - makefile (building without), 235
 - makefile (custom build command), 235
 - makefile import, 182
 - makefiles, 169
 - managing code annotation files, 464
 - Managing Extensionless Files, 493
 - managing projects, 164
 - managing projects in workspace, 163
 - managing workspaces, 160
 - Margins, 908
 - margins (setting), 1025
 - Margins dialog box, 856
 - Match block comment setting, 858
 - Match case, 779
 - Match case (default), 984
 - match highlighting, 589
 - Match whole word, 779
 - Match whole word (default), 984
 - math (see mathematics), 638
 - Mathematical Operators, 1395
 - mathematics, 638
 - document math, 640
 - expressions with mixed bases, 638
 - math commands, 639
 - math commands (examples), 640
 - overflow/underflow, 640
 - prime numbers, 641
 - using the Calculator, 638
 - Max files, 1074
 - Max undo steps, 1074
 - Maximize editor in full screen mode, 950
 - Maximum candidates for list parameters, 976
 - Maximum class/struct members shown, 976
 - Maximum clipboards, 962
 - Maximum completion history matches, 979
 - Maximum distance for Highlight matching blocks (KB), 964
 - Maximum files for cache updating, 980
 - Maximum functions found by parameter help, 976
 - Maximum items found in references search, 977
 - Maximum nesting level, 911
 - Maximum nesting level for Highlight matching blocks, 963
 - Maximum number of auto-complete history to store, 979
 - Maximum number of memory allocators, 1089
 - Maximum number of tags per file, 977
 - Maximum response time for list members (ms), 976
 - Maximum response time for list parameters (ms), 977
 - Maximum search results output (KB), 985
 - Maximum size of files for building token list, 977
 - Maximum size of files for statement tagging, 977
 - Maximum size of files to tag, 977
 - Maximum size of files to tag (slow files), 977
 - Maximum size of in-line deltas in archive files (KB), 1079
 - Maximum size to backup, 1079

Maximum stack depth, 987
Maximum suspended update time(ms), 992
Maximum symbols, 979
Maximum tags found in symbol search, 977
Maximum time for parsing current file (ms), 977
Maximum word completion, 979
Members tool window, 85
Memory tool window, 85
Menu Customizations
 Export/Import Options, 1117
Menu Editor dialog box, 866
menu if no selection, 86
menu if selection, 86
Menu Item Alias dialog box, 1202
menus (changing or adding), 1494
Mercurial Options, 620
Merge dialog box, 599
merging files, 599
Message line, 69
Message List, 469, 948
Message list colors, 911
Message List tool window, 469
Message modified color, 911
Message visited color, 911
metadata, 1436
Methods, 1475
Mini Find and Replace Dialog, 576
minimap, 100
Minimap
 Language-Specific View Option, 1011
Minimum level to collapse, 911
Minimum prefix, 1032
Minimum running update time(ms), 991
Minimum size for fast delta creation (KB), 1079
Minimum wildcard cache update time (ms), 980
Minutes before restarting, 971
mixed mode view (debugging), 248
mn_flags, 1179
Modal and modeless dialog boxes, 1465
mode name, 1003
Modified Buffers dialog box, 43
modified lines
 reset, 1077
 viewing, 366
Modify parameter list (Quick Refactoring), 483
modifying doc comment templates, 455
modifying selected text, 313
Module Initializations, 1445
modules (Slick-C), 723
Mono, 533
Mono (.NET) Options dialog, 536
Mono Options Assemblies tab, 536
Mono Options Compiler tab, 536
Mono Options dialog, 536
Mono Options Interpreter tab, 536
Mono Options Path tab, 536
Mono projects, 167
More Information (Quick Start), 61
Mouse Options
 advanced, 956
mouse over symbol, 211
mouse pointer option, 908
mouse selection, 981
Move dialog box, 204
moving code annotations, 462
Moving Controls, 1456
multi-file diff output dialog box, 595
multi-file undo/redo, 589
multiple cursors and selections, 328
multiple file search and replace, 780
multiple instances, 36
multiple monitors, 95
multiple monitors (configuring), 41

N

Name (Code Templates Add New Item dialog box), 418
Name (Code Templates Add Parameter dialog box), 418
Name element (Code Templates metadata file reference), 427
Named Arguments, 1417
named bookmarks, 446
 Bookmarks tool window, 796
 deleting/clearing, 449
 indicator, 447
 navigating (go to), 448
 setting, 446
 toggle, 447
 workspace bookmarks, 449
named sessions (debugging), 249
namespaces, 504, 505, 553, 554
name_info, 1420
NAnt, 236
NAnt targets, 237
navigate to named bookmark, 448
navigate to pushed bookmark (popping), 450
navigation, 329

between buffers/windows, 301
between symbols, 329
between words, 332
cursor movements, 335
go to offset, 336
in pages and files, 335
in statements and tags, 334
subword navigation, 332
symbol browsing, 339
to a specific line, 336
URLs, 337

Network and Internet Options, 1092
Network Settings, 1093
never, 1007

New (File) dialog box, 189
New Annotation dialog, 460
new configurations, 503
New dialog, File tab, 733
New Extension dialog, 492
new features, 5
New Field dialog, 463
New File Tab, 733
New file tab position, 967
New Folder Name dialog, 160
New Project dialog, 168
New Project Tab, 735
New Project Tool (New Project Tool Wizard), 838
New Project Tool Wizard, 837
New Workspace Tab, 737
Newline characters, 1010
next word, 332
Next word style, 962
no window reordering, 966
Non wildcard name match style, 1071
non-mdi editor control, 862
none, 1007
Normalized Profile, 1164
Notification Options, 1090
Notifications, 292
notifications
 options, 1090
no_code_swapping, 1519
null, 1381
number lines every, 760
Number of backups to keep for each file, 1079
number of copies, 760
Number of elements to expand in arrays, 991
Number of lines to color above and below the current page, 978, 978

Number of lines to scan for Autos, 991
Number of off-page lines to color per pass (chunk size), 978, 978
Number of recent language modes to store, 1085
Number of recent project types to store, 1085
Numeric constants, 1375
numeric overflow or underflow, 640
numeric sort, 633

O

Object Instances, 1473
Objective-C Formatting Options, 499
offset navigation, 336
OK button (adding), 1460
one file per window, 965
Only highlight current set of references, 975
Open Application, 492
Open dialog box, 743
Open File Options, 1065
Open files using, 1068
Open Form dialog, 1458
Open Makefile as Workspace dialog, 836
Open Menu dialog box, 1201
Open Other Workspace Menu, 816
open site, 1096
Open tab (Project Properties), 836
Open tool window, 82
Open Tool Window, 738
Open URL dialog box, 193
open workspace, 161
opening files
 quick open, 189
Opening Files, 191
opening Unicode files, 1146
opening URLs, 193
Operators, 1395
Options
 auto-restore, 147
 Export/Import, 1112
 Export/Import Overview, 105
 Overview , 104
 Quick Start Configuration Wizard, 54
 Saving, Restoring, and Backing Up, 105
options (using macros to discover/control), 722
Options dialog, 897
Options dialog shortcuts, 902
Options Export/Import, 1112
 Export Groups, 1114
options for common keyboard keys, 99

-
- options for pushed bookmarks, 451
 - Options History, 1111
 - Options Search, 901
 - Options tab (DIFFzilla), 881
 - OR operator, 1396
 - order, 633
 - Organize All Workspaces dialog, 162
 - Organize C# Imports, 542
 - Organize Imports, 515, 542
 - Organize Imports Options, 529, 543
 - Organize Java Imports, 515
 - organize_imports_options, 515, 542
 - organizing projects, 157
 - orientation, 760
 - OS prompt, 92
 - OS/390 Assembler Formatting Options, 508
 - Other wildcard name match style, 1070
 - output file, 877
 - output style, 878
 - Output tool window, 82
- P**
- Package View, 153
 - Pad parentheses, 1043
 - Parameter element (Code Templates metadata file reference), 427
 - Parameter info, 1042
 - Parameter Information, 209
 - parameter prompting, 401
 - Parameters element (Code Templates metadata file reference), 429
 - paren style, 334
 - Parenthesis matching style, 963
 - parse statement, 1410
 - parsing options for C/C++, 504
 - parsing options for Verilog, 554
 - partial load (fast line count), 1073
 - Partial word wrap, 1026
 - Pascal Formatting Options, 546
 - passive transfers, 1097
 - password, 763
 - paste, 320
 - pedantic, 1431
 - profile.xml, 1160
 - Perforce Options, 613
 - Perl
 - Running and Debugging, 273
 - Perl capture groups, 649
 - Perl Formatting Options, 895
 - Perl regular expressions, 662
 - compatibility issues with old syntax, 712
 - examples, 676
 - perl regular expressions
 - compatibility issues, 711
 - Perl tagged expressions, 649, 649
 - personal.sca, 1161
 - PHP
 - project, 263
 - Running and Debugging, 260
 - PHP Beautifier, 499
 - PHP Formatting Options, 559
 - Picture Box control, 1489
 - pictures (adding to image control), 1491
 - pictures (adding to list box), 1481
 - PL/I Formatting Options, 549
 - PL/SQL Formatting Options, 549
 - Place cursor at end, 984
 - Place cursor on focus click, 966
 - platform-specific notes, 96
 - Pointer Variables, 1392
 - pop bookmark, 450
 - port, 764
 - Preferences
 - see Options, 104
 - Prefix area width, 1182
 - Prefix matching (Options dialog), 902
 - preprocessing for C/C++, 505
 - preprocessing for Verilog, 553
 - Preserve column on top/bottom, 962
 - Preserve trailing, 1032
 - preserve width on existing (comment wrapping), 1024
 - Preview (Code Templates Add File dialog box), 417
 - Preview All shows modified file(s) on the left, 984
 - preview symbol information, 211
 - Preview tool window, 346
 - what is displayed, 348
 - Preview tool window options, 946
 - Preview window symbol lookup timeout (ms), 974
 - prime numbers, 641
 - print background color, 759
 - print color, 759
 - print color coding, 759
 - Print dialog box, 757
 - print hex, 759
 - printing
 - insert formfeed, 758

Prioritize navigation to symbols in the current project, 1044
procedures (defining), 1415
process buffer, 962
Process recognized xterm color output in Build Window, 909
procs and prototypes, 331
Product Improvement Program, 45
Product Improvement Program Options, 1092
product registration, 3
product support, 44
Product Support
 Contacting, 44
Product Updates, 1127
Product Updates Menu, 1125
Profile Combo Box, 500
profile name, 763
profiles (colors), 129
profiling (Slick-C), 1449
Program Info Tab (about SlickEdit), 1125
programmable macros, 723
 loading, 724
Project Build Commands tab, 827
Project Configuration Settings dialog, 169
project directories tab, 822
project files
 adding and removing, 176
 creating, 181
 importing, 181
 listing, 851
 loading, 183
 makefiles, 169
Project Files Tab, 820
Project Live Errors tab, 834
Project Menu, 814
Project name, 736
Project properties (Tools), 823
Project Properties dialog, 819
Project tab (New dialog), 735
Project Templates
 Export/Import Options, 1117
Project Tools Tab, 823
Project Tools toolbar, 78
Project type, 735
project types, 165
 dynamic languages, 167
 GNU C/C++, 165
 Java, 166
 Mono, 167
 other compilers, 166, 166
 Perl, PHP, Python, 167
 Ruby, 167
 Visual Studio, 165
project wildcards, 160
project.vpe, 1161
projects, 152
 build output options, 175
 build settings, 172
 build system options, 175
 command line execution, 172
 configurations, 169
 configuring directories, 170
 configuring tools, 170
 creating, 167
 creating custom types, 168
 defining dependencies, 169
 defining language-specific, 176
 importing files, 181
 libraries, 160
 loading files, 183
 makefiles, 169
 managing, 164
 managing source files, 176
 managing within a workspace, 163
 organizing, 157
 Other project type, 165
 project types, 165
 project wildcards, 160
 setting active, 168
 sharing between workspaces, 164
 specifying command directory, 172
 version control, 160
 with one file, 176
Projects tool window, 83
Prompt for value (Code Templates Add Parameter dialog box), 418
prompt procedure, 1517
prompt replace dialog box, 785
Prompt string (Code Templates Add Parameter dialog box), 418
Prompt to undo past last save, 1074
prompting from macros, 1516
Properties dialog box, 1455
protect read-only, 962
Protect read-only mode, 962
prototypes, 1424
Proxy Settings, 1099
pushed bookmarks, 450

-
- bookmark stack, 451
 - indicator, 451
 - options, 451
 - popping, 450
 - pushing, 450
 - viewing, 451
 - PVCS, 619
 - Python
 - Running and Debugging, 268
 - Python Beautifier, 499
- ## **Q**
- Quick Brace, 434
 - Quick brace/unbrace one line statements, 896
 - quick create, 189
 - quick extract method, 483
 - quick modify parameter list, 483
 - quick open, 189
 - quick refactoring, 482
 - undo/redo, 482
 - Quick Refactoring Menu, 873
 - quick rename, 482
 - quick replace, 575
 - quick replace literal with constant, 484
 - quick search, 574
 - Quick Start, 54
 - Quick Start Configuration Wizard, 54
 - quote_key command, 758
- ## **R**
- Radio Button Control, 1479
 - rc variable, 1452
 - Read List dialog box, 205
 - read only indicator, 70
 - read only mode (macros), 862
 - read only mode (protect), 962
 - rebuild, 238
 - rebuilding tag files, 223
 - recognizing Unicode files, 1146
 - Record width, 746
 - recorded macros, 718
 - binding to keys, 720
 - deleting, 722
 - execute_last_macro_key, 720
 - operations, 718
 - recording a new macro, 719
 - running, 721
 - saving and editing, 721
 - using to discover/control options, 722
 - recording tasks with annotations, 467
 - redeclvars, 1432
 - Redefine Common Key Options, 955
 - redo (multi-file), 589
 - redo replacement, 589
 - refactoring, 482
 - Refactoring Options, 1111
 - Refactoring results, 484
 - References, 948
 - references (options), 974
 - References tool window, 349
 - options, 353
 - References view, 349
 - reflow comment dialog box, 857
 - Reflow next, 962
 - reflowing comments, 457
 - Reflowing Text, 481
 - Regex Evaluator, 660
 - Regex Evaluator tool window, 660
 - entering expressions, 661
 - entering test cases, 660
 - options, 661
 - Registers tool window, 85
 - registration, 3
 - Regular Expressions,
 - An Overview,
 - Brief, 709
 - capture groups for Perl, 649
 - capture groups for SlickEdit, 652
 - capture groups for Vim, 654
 - minimal versus maximal matching, 647
 - Perl, 662
 - Perl examples, 676
 - search and replace with, 649, 652, 654
 - SlickEdit, 678
 - SlickEdit examples, 691
 - syntax option, 984
 - tagged search expressions, 649, 654
 - tagged search expressions for SlickEdit, 652
 - testing, 660
 - Unicode category specifications, 714
 - UNIX, 710
 - Vim, 693
 - Vim examples, 706
 - Reinsert after current, 1074
 - Release Notes Tab (about SlickEdit), 1125
 - Reload on switch buffer, 1076
 - reload prompt, 1075

Reload With Encoding dialog, 747
remote JVM, 250
remote Mono, 250
remote process (debugging), 250
remote to local directory mapping, 764
Remove Dialog Box, 609
remove duplicate lines, 634
Remove EOF character, 1077
removing files from a project, 176
removing SlickEdit, 34
Rename (Quick Refactoring), 482
reorder windows, 966
repeat command on selected dialog box, 205
replace
 see also replacing, 574
replace command, 583
Replace in Files tab (Find and Replace), 787
Replace literal with constant (Quick Refactoring), 484
replace mode, 962
Replace params in target file (Code Templates Add File dialog box), 417
Replace tab (Find and Replace), 785
replace toggle, 70
replacing, 574
 Find and Replace tool window, 585
 minimal vs maximal matching, 647
 quick replace, 575
 Regular Expressions,
 replace and c commands, 583
 see also searching, 574
 tagged search expressions for SlickEdit, 649, 652, 654
 undo/redo, 589
Reset, 501
Reset modified lines, 1077
resolve links, 764
Response timeout(s), 991
Restore cursor after replace, 985
Restore Default Options, 1116
retagging workspace, 216, 221
return statement, 1416
revision one, 877
revision two, 877
REXX Formatting Options, 549
Right CTRL = Enter/Send, 1183
router, 1096
Ruby
 Running and Debugging, 278
Ruby Formatting Options, 895
rule, symbol coloring, 132
Run-time error finding, 1449
Run/Debug tab (Project Properties), 832
running a program, 247
Running and Debugging Perl, 273
Running and Debugging PHP, 260
Running and Debugging Python, 268
Running and Debugging Ruby, 278
running recorded macros, 721
Running SlickEdit, 36
running SlickEdit, 36
 multiple instances, 36

S

safe exit, 43
Same name, 1081
Same name different extension, 1081
Save after period of inactivity, 1080
Save after period of time, 1080
Save all prompts to name unnamed files, 1078
Save and Bind to Key, 862
save as (using), 196
Save As dialog box, 196
Save configuration, 1089
Save Failed dialog box, 196
Save File Options, 1076
Save files on loss of focus, 1077
save macro dialog box, 861
Save Multi-File Diff Output dialog box, 596
save password, 763
Save to, 1080
Save to different directory, 1080
Save/restore file position, 1074
saving files, 196
saving recorded macros, 721
SBCS/DBCS, 1144
sc.lang.IHashIndexable, 1383
sc.lang.IIndexable, 1381
sc.lang.IToString, 1390
SCC, 611
 configuration, 612
 opening a project, 612
Scoping and Declaring Variables, 1400
screen layout, 68
screen management, 95
scroll bars, 120
Scroll bars, 909
Scroll Markers, 472

Scroll style, 909
Scroll when, 909
search
 see also searching, 574
search (go to offset), 336
search and replace, 574
Search backward, 984
Search Engines Options, 1101
search for, 778
Search for word matches if symbol is not found, 975
Search hidden text, 984
Search Menu, 774
Search Options, 981
 Default regular expression syntax, 984
Search Options dialog box, 779
search order, 1167
 executable files, 1167
Search results, 948
Search Results node, 901
Search results output, 587
Search Results tool window, 83
searching, 574, 984
 classes, 80
 Default Find and Replace GUI, 985
 Excel files, 783
 Find and Replace tool window, 585
 find and slash commands, 579
 Find Symbol tool window, 588
 finding all symbols in file, 589
 history retrieval, 984
 incrementally, 575
 initialization options, 984
 Maximum buffer size for incremental search (KB), 984
 Maximum occurrence matches, 984
 members, 80
 Mini window options, 984
 minimal vs maximal matching, 647
 multiple files, 780
 quick search, 574
 regular expression syntax option, 984
Regular Expressions,
 syntax-driven, 587
tagged search expressions for Perl, 649
tagged search expressions for SlickEdit, 652
tagged search expressions for Vim, 654
undo/redo, 589
Word files, 783
Seek dialog, 336
seek offset position, 336
Select a Buffer dialog box, 305
Select Files With Attribute dialog, 204
Select Files With Extension dialog, 204
select first, 86
Select Mode dialog box, 488
Select Mouse Event dialog, 114
Select Symbol, 948
Select Symbol dialog, 801
Select Text to Paste dialog, 770
Selected text (if exists), 985
selecting a code block, 441
Selecting a mode, 488
Selecting Controls, 1454
selecting text, 308
 block insert mode, 325
 blocks (columns), 311
 characters, 309
 commands, 313
 counting lines/characters, 311
 drag-and-drop, 320
 enumeration, 313
 keyboard shortcuts, 309
 lines, 310
 modifying a selection, 313
Selection (Comments), 858
selection (creating alias from), 402
selection color, 308
Selection Functions Overview, 1508
Selection indicator, 69, 311
selection only, 759
Selection Options, 980
selection styles, 980
Selective Display, 369
Selective Display bracketing, 910
Selective Display dialog box, 806
Selective Display Function headers, 808
Selective Display line color, 911
Selective Display Multi-level, 810
Selective Display Preprocessor directives, 810
Selective Display Search Text, 807
Selective Display toolbar, 78
Selective Display, Expand/collapse, 958
server type, 763
service name, 1097
Set Attributes dialog, 205
set command, 1152
Set current directory when switching buffers, 1068

set named bookmark, 446
set pushed bookmark, 450
set scroll style, 120
Set Variable dialog box, 864
SETemplate element (Code Templates metadata file reference), 430
setting
 fonts, 122
setting active project, 168
Setting Properties, 1455
SFTP, 635
sftp tab, 1097
sharing code annotations, 459
sharing projects, 164
Sharing Your Configuration, 106
shell prompt, 92
shell scripts , 23
shelling, 92
Shelling from macro, 1514
shelving, 621
Short key names, 950
shortcuts for build and rebuild, 238
shortcuts in Options dialog, 902
shortcuts in text boxes, 97
Show all folders in directory panel, 1069
Show auto reload timeout notifications, 1076
Show bitmap in margin for each reference, 975
Show categories, 1031
Show change directory commands in build window, 963
show changes, 877
Show close buttons on document tabs, 967
Show comments, 1043
Show dot files, 909
Show EOF character, 1074
Show extra line after last newline, 963
Show file name twice, 1085
Show files in Add Source Files dialog, 1072
Show folders in file list, 1069
show hidden files (Save As dialog), 749
show hidden files (Save Copy As dialog), 751
Show history, 1031
Show icons, 1031
Show indicator for read-only files, 1069
Show indicator for writable files, 1070
Show info for symbol under mouse, 1044
Show info for symbol under mouse after (ms), 974
Show MDI menu in full screen mode, 950
Show new workspace dialog, 1086
Show number base popups, 908
Show parameters, 1031
Show pictures, 753
Show preview for symbol under cursor, 1045
Show preview of symbols in tool windows after (ms), 974
Show preview of symbols in tool windows on mouse-over, 974
Show pushed bookmarks, 986
Show set bookmarks, 986
Show value of symbol under mouse, 992
Show VSLICKERRORPATH in build window, 963
Show/match files in current file directory, 1069
Show/match files in history, 1069
Show/match open files, 1069
Show/match workspace and project files, 1069
Simple Variables, 1400
Single File Projects, 184
single line statements, 434
Size limit for comparing contents (KB), 1076
Sizing Controls, 1456
Slick-C Batch Files, Writing, 1426
Slick-C Beautifier, 499
Slick-C Debugger, 1450
Slick-C Formatting Options, 895
Slick-C headers, 725
Slick-C module, 723
Slick-C Profiler dialog, 1449
Slick-C profiling, 1449
Slick-C Stack, 83
Slick-C Stack tool window, 83
Slick-C variable, 724
SlickEdit book, 13
SlickEdit capture groups, 652
SlickEdit command line, 89
 activating, 89
 common commands, 93
 history, 90
 keyboard shortcuts, 91
 prompting, 92
 prompting (option), 957
 shelling, 92
SlickEdit emulation keys, 1300
 clipboard, 1304
 command line, 1304
 compiling and programming, 1306
 cursor movement, 1300
 debugging, 1307
 deleting, 1302

files and buffers, 1305
inserting, 1301
macros, 1308
miscellaneous, 1309
searching, 1302
selecting, 1302
text box editing, 1304
windowing, 1306
SlickEdit File Manager, 202
SlickEdit regular expressions, 678
 compatibility issues with old syntax, 713
 examples, 691
SlickEdit tagged expressions, 652
SLICKEDITCONFIG, 1149
smart next window, 302
Smart next window style, 966
Smart Open, 192
 e command, 189
 e command options, 190
 Open Tool Window, 738
SmartPaste, 374
Smooth horizontal scroll, 909
Smooth vertical scroll, 909
snippets, 392
soft wrap, 1026
SoftWrap, 368
sort (selection), 633
Sort buffer, 633
sort commands, 634
sort dialog box, 633
Sort on selection, 633
Sort within selection, 633
Sort workspaces and projects on the All Workspaces menu, 1085
sorting text, 633
 commands, 634
SortOrder element (Code Templates metadata file reference), 430
sort_on_selection command, 634
Source Diff, 590
Source file name (Code Templates Add File dialog box), 417
Space always inserts space, 1032
space between, 760
Space inserts longest unique prefix, 1032
Special Character Options, 948
Special Characters, 1010
special characters tab, 948
specifying tabs for a file extension, 374
spell check, 630
 in multiple files, 631
operations, 630
running, 630
Spell Check Files dialog box, 631
Spell Check Menu, 875
Spell Check Options, 1102
spelling dialog box, 630
spill file (and buffer cache), 196
Spill file path, 1088
Spin control, 1477
Split line comments, 1021
Split strings, 1022
SQL Server Formatting Options, 549
ssh executable, 1097
ssh tab, 1097
st command, 1447
stack class, 209
Standard HTML and XML Case and Quoting Tab, 559
Standard HTML and XML General Tab, 558
Standard HTML and XML Tags Tab, 559
Standard Open dialog box, 743
Standard property, 1164
Standard toolbar, 79
Start after minutes idle, 971
Start after seconds idle, 971
start characters, 1048
Start in column, 1020
Start mode, 962
start on file, 1068
Start up options, 37
Start wrapping on line, 1023
starting FTP connection, 636
state file, 725
state file (alternate), 42
Statement Level Tagging, 213
statement navigation, 334
status line, 69
stopping FTP connection, 637
strict, 1432
strict2, 1432
strictarglists, 1432
strictboolean, 1433
strictellipsis, 1434
strictenums, 1433
strictincludes, 1434
strictnames, 1434
strictnumbers, 1434

strictparens, 1435
strictpointers, 1433
strictprotos, 1435
strictreturn, 1435
strictsemicolons, 1435
strictstrings, 1436
string concatenation, 1398
String editing, 1022
string editing (comments), 456
string literals, 1373
string operators, 1379
Strip trailing spaces, 1077
struct, 1383
structure matching, 333
 setting match style, 334
 viewing/defining, 333
submenus, 86
subsystem, 1097
Subversion, 614
Subversion Options, 616
subword navigation, 332
summary of keys, 108
support (for SlickEdit), 44
Suppress prompt unless modified, 1075
surrogate support, 1147
Surround With, 440
 alias, 441
 commands, 442
Surround With Dialog, 440
surrounding, 436, 440
 unsurrounding, 444
surrounding text, 436
SVN History Dialog, 614
switch statement, 1413
Symbol Arguments tool window, 363
Symbol Browser Filter Options dialog box, 360
symbol browsing, 339
 base/derived classes, 360
 Class tool window, 339
 Current Context Toolbar, 343
 Defs tool window, 344
 Find Symbol tool window, 346
 Preview tool window, 346
 References tool window, 349
 symbol refs/callers tree, 359
 symbol uses/call tree, 358
 Symbols tool window, 354
Tag Arguments tool window, 363
Tag Properties tool window, 363
symbol callers, 359
Symbol Coloring, 126, 130
 Options , 923
 rule, 132
 Unidentified Symbols, 132
 View Menu , 805
Symbol Coloring Options, 923
Symbol declaration, 1031
symbol highlighting, 589
symbol information, 211
symbol navigation, 329
 between multiple instances, 331
 browsing symbols, 339
 Find Symbol tool window, 331
 more methods, 331
Symbol Properties tool window, 363
Symbol Refs/Callers tree dialog box, 359
Symbol Translation Editor, 568
symbol use, 358
Symbol Uses/Calling tree dialog box, 358
Symbol view, 354
symbols
 comparing, 594
Symbols, 948, 1030
Symbols tool window, 354
 base/derived classes, 360
 filter options, 360
 filtering symbols, 356
 options, 356
 refs/callers tree, 359
 uses/calling tree, 358
Sync background colors, 914
Sync current directory, 1069
sync vertical line (comment wrapping), 1024
syntax expansion, 406
Syntax Expansion, 1008
Syntax expansion, 1030
Syntax expansion (Overview), 406
Syntax indent, 1015
Syntax indent ruler, 910
syntax-driven search, 587
system configuration files, 1162
system state file, 725
SystemVerilog Beautifier, 499, 553

T

Tab cycles through choices, 1032
tab groups, 76
Tab inserts longest, 1032

Tab key, 373, 1007
Tab key aligns multiple cursors, 1007
Tabs, 1016
tabs dialog box, 374
Tabs dialog box, 856
Tabs ruler, 910
tabular lists, 74
Tag Compiler Libraries Dialog, 216
Tag file cache maximum, 976, 1089
Tag file cache size, 975, 1088
Tag file on save, 971
Tag file on switch buffer, 971
tag files, 504
 building, 216, 221
 categories, 215
 language-specific libraries, 218
 rebuilding, 223
 search order, 222
 workspace files, 216, 221
Tag Files (Language-Specific), 1045
tag navigation, 334
tagged expressions for Perl, 649
tagged expressions for SlickEdit, 652
tagged expressions for Vim, 654
tagging, 208, 214
 compiler libraries, 216
 workspace files, 216, 221
tagging (identifiers), 1048
tagging (minutes before retagging), 971
tagging cache, 1088
Tagging Excludes, 894
Tags tab (Color Coding Setup), 1059
Target file name (Code Templates Add File dialog box), 417
Tcl Options, 895
Template file (Template Manager dialog box), 416
Template Manager dialog box (Code Templates), 415
 Template Options dialog, 417
 TemplateContent element (Code Templates metadata file reference), 431
 TemplateDetails element (Code Templates metadata file reference), 432
Templates list (Code Templates Add New Item dialog box), 418
Templates list (Template Manager dialog box), 416
Terminal tool window, 84
testing regular expressions, 660
Text Box control, 1478
text box editing keys, 97
text compare, 590
text editing
 block insert mode, 325
 inserting characters, 325
 selections, 308
 sorting, 633
text selection, 981
third-party workspaces, 164
Threads tool window, 85
three way merge editing, 599
Throw away file lists, 962
Time display style, 909
timeout, 764
Timeout (s), 1080
Timeout after (ms), 978, 978
toggle, 70
toggle bookmark, 447
Tool Options, 1102
Tool Window & Toolbar Options, 944
Tool Window, Toolbar Options, 944
Tool Windows
 Customizing, 78
tool windows (overview), 76
Tool Windows Options, 943
Tool Windows Options dialog, 943
Toolbar and Tool Window Layout
 Export/Import Options, 1117
Toolbar Control Properties dialog, 812
Toolbar Customization, 942
Toolbar Customization dialog, 942
Toolbar Customizations
 Export/Import Options, 1117
Toolbar Options, 941
Toolbar update delay(ms), 992
toolbars, 76
 displaying, 76
 docking, 76
Toolbars
 Customizing, 77
Toolbars dialog, 941
tools (for project configuration), 170
Tools Menu, 871
Tools tab (Project Properties), 823
Tools toolbar, 79
Top of file line, 908
Track active project in workspace along with workspace history, 1085
transfer type, 763

-
- Truncate file at EOF, 1075
truncation, 1008
Turn off Highlight matching blocks when file larger than (KB), 964
tutorials, 1130
 C#, 1131
 C/C++, 1130
 Java, 1141
two up, 759
twopass, 1436
Type Casting, 1401
typeless, 1393
typeless variable declaration, 1401
types of code annotations, 458
- U**
- ucase, 324
UCN, 1147
unattended installation, 27
unbinding a key, 116
Unbrace, 434
uncommenting, 452
Underline URLs, 909
undo (multi-file), 589
undo replacement, 589
Unicode, 1145
 category specs for reg expressions, 714
 converting to UCN, 1147
 file recognition, 1146
 implementation, 1147
 limitations, 1147
 opening files, 1146
 surrogate support, 1146
Unidentified Symbols, 132
uninstalling SlickEdit, 34
union, 1385
Unit Test, 948
Unit testing
 JUnit , 516
Unit Testing tool window, 83
UNIX regular expressions, 710
UNIX temp environment, 1088
Unlist Files with Attributes dialog, 204
Unlist Files With Extension dialog, 204
Unlist Search dialog box, 205
unload hot fix, 34
unload module, 724
Unlock Dialog Box, 609
Unnormalized Profile, 1164
Unshelve dialog box, 625
unsurround block, 444
unsurrounding text, 436
unxcpp.h, 1161
Update auto-complete after idle, 979
Update highlighting after (ms) idle, 978
update manager, 31
Update Manager Options dialog, 1127
Update symbol coloring after (ms) idle, 978
Update wildcard cache, 980
Update workspace tag file on activate, 971
Update workspace tag file on open, 971
updates (to SlickEdit), 31
upgrading SlickEdit, 31
upload filename case, 764
URI Scheme Options, 1098
URL (opening), 193
URL Mapping Options, 1097
url mappings, 557
URL navigation, 337
Use background tagging, 970
Use background tagging threads, 971
Use Clear key as NumLock, 958
Use Command+key for dialog hotkeys, 957
Use Command+key for menu drop-downs, 957
Use file association, 492
use firewall/proxy, 764
use hanging indent (comment wrapping), 1024
use smart merge, 878
Use strict case-sensitivity rules, 1033, 1034
Use subword matching rules, 1034
Use Syntax Expansion on space, 1008
Use timeout, 1079
Use undo, 1074
Use workspace bookmarks, 986
user configuration files, 1159
user id, 763
User interface, 68
 tabular lists, 74
user preferences, 104
user state file, 725
User-Created Forms
 Export/Import Options, 1117
User-Created Menus
 Export/Import Options, 1117
User-Created Toolbars
 Export/Import Options, 1117
User-Loaded Modules dialog, 860
User-Recorded Macros

-
- Export/Import Options, 1117
 - user.cfg.xml, 1161, 1164
 - user.cfg.xml File Format, 1164
 - usercpp.h, 1161
 - usersystemverilog.svh, 1161
 - userverilog.v, 1161
 - Using Version Control, 605
 - usrprjtemplates.vpt, 1161
 - UTF-8, 1144
- V**
- Value (Code Templates Add Parameter dialog box), 418
 - variable (Slick-C), 724
 - Variable editor, 864
 - Variable Editor dialog box, 864
 - Variable Initializations, Details, 1401
 - VBScript Beautifier, 499
 - VBScript Formatting Options Standard Edition, 546
 - Vera Options, 895
 - Verilog Beautifier, 499, 553
 - Verilog Beautifiers, 553
 - Verilog parsing options, 554
 - Verilog preprocessing, 553
 - version control, 604
 - Add to shelf, 623
 - advanced settings, 610
 - configuration, 610
 - CVS, 619
 - Git, 618
 - List shelves, 624
 - overview, 604
 - PVCS, 619
 - SCC, 611
 - selecting, 604
 - setting up command line systems, 611
 - shelving, 621, 621
 - Subversion, 614
 - using, 605
 - workspaces and projects, 160
 - Version Control Advanced Settings, 1106
 - version control command setup dialog box, 610
 - Version Control Commands Setup Dialog box, 1105
 - Version Control Menu, 872
 - Version Control Options, 1103
 - Version Control Providers, 1105
 - Version Control Setup dialog box, 610
 - Version Control Setup Options, 1103
- Vertical line color, 908
 - Vertical line columns, 908
 - Vertical scroll bar, 909
 - vi-!: 1326
 - vi-\$: 1325
 - vi-%: 1325
 - vi-': 1326
 - vi-':<, 1326
 - vi-':>, 1326
 - vi-.., 1325
 - vi-/, 1326
 - vi-:<, 1326
 - vi-:>, 1326
 - vi-?: 1326
 - vi-b, 1326
 - vi-bdelete, 1326
 - vi-bnext, 1326
 - vi-bprevious, 1326
 - vi-bufo, 1326
 - vi-buffer, 1326
 - vi-buffers, 1326
 - vi-cd, 1327
 - vi-close, 1327
 - vi-copy, 1327
 - vi-delete, 1327
 - vi-edit, 1327
 - vi-file, 1327
 - vi-g, 1327
 - vi-gl!, 1327
 - vi-global, 1327
 - vi-global!, 1327
 - vi-help, 1327
 - vi-join, 1327
 - vi-k, 1327
 - vi-list, 1327
 - vi-move, 1327
 - vi-n, 1327
 - vi-next, 1327
 - vi-nohlsearch, 1327
 - vi-number, 1327
 - vi-p, 1328
 - vi-print, 1328
 - vi-put, 1328
 - vi-q, 1328
 - vi-ql!, 1328
 - vi-qall, 1328
 - vi-quit, 1328
 - vi-quit!, 1328
 - vi-r, 1328

vi:-read, 1328
vi:-redo, 1328
vi:-registers, 1328
vi:-rewind, 1328
vi:-s, 1328
vi:-sbuffer, 1328
vi:-set, 1329
vi:-shell, 1330
vi:-split, 1330
vi:-substitute, 1328
vi:-t, 1330
vi:-tag, 1330
vi:-undo, 1330
vi:-version, 1330
vi:-vglobal, 1330
vi:-vsplit, 1330
vi:-w, 1330
vi:-wall, 1330
vi:-wq, 1330
vi:-wqall, 1330
vi:-write, 1330
vi:-x, 1330
vi:-xall, 1330
vi:-yank, 1330
vi:-z, 1330
view
 hex/line hex, 371
 line number options, 1010
 line numbers, 367
 special characters, 366
View Menu, 803
View Options (Language-Specific), 1009
view special characters, 366
View text symbols (Special Characters tab), 948
viewing
 current line, 365
 modified lines, 366
viewing accessible members, 362
viewing symbol callers, 359
viewing symbol properties, 357
viewing symbol references, 349
viewing symbol uses, 358
Vim capture groups, 654
Vim emulation keys, 1310
 clipboard, 1316
 command line and text box editing, 1317
 compiling and programming, 1321
 cursor movement, 1310
 debugging, 1322
deleting, 1314
files and buffers, 1318
inserting, 1313
macros, 1322
miscellaneous, 1323
searching, 1314
selecting, 1315
windowing, 1320
Vim emulation options, 958
Vim EX commands, 1326
Vim EX range specifiers, 1325
Vim regular expressions, 693
 examples, 706
Vim tagged expressions, 654, 655
Vim tutorial, 1142
Virtual Memory Options, 1088
visible lines only, 759
Visual C++ emulation keys, 1331
 clipboard, 1334
 command line and text box editing, 1334
 compiling and programming, 1336
 cursor movement, 1331
 debugging, 1337
 deleting, 1332
 files and buffers, 1335
 inserting, 1332
 macros, 1338
 miscellaneous, 1338
 searching, 1332
 selecting, 1333
 windowing, 1336
Visual C++ Setup dialog box, 229
Visual Studio emulation keys, 1339
 clipboard, 1342
 command line and text box editing, 1343
 compiling and programming, 1345
 cursor movement, 1339
 debugging, 1346
 deleting, 1341
 files and buffers, 1344
 inserting, 1340
 macros, 1347
 miscellaneous, 1347
 searching, 1341
 selecting, 1341
 windowing, 1344
Visual Studio projects, 165
Visual Studio workspaces, 164
VPW extension, 149

vpwhist file extension, 1162
vrestore.slk, 1161
vs, 37
vsbuild to compile files, 229
Vscroll Bar control, 1486
vsdebugio connection port, 992
vsdiff, 598
vsdiff Invocation Options, 598
VSLICK, 1149
vslick.sta, 40, 1161
VSLICKBACKUP, 1150
VSLICKBIN, 1150
VSLICKBITMAPS, 1150
VSLICKINCLUDE, 1437
VSLICKLOAD, 1150
VSLICKMACROS, 1150
VSLICKMISC, 1150
VSLICKPATH, 1149
VSLICKRESTORE, 1149
VSLICKTAGS, 1150
VSLICKXNOPLUSNEWMSG, 1151
VSLICKXTERM, 1150
VSNet emulation keys, 1339
VST, 1151
vstw program, 1447
vstw.exe program, 1447
vusrobs.e, 1162

W

Watch tool window, 85
watches, 252
watchpoints, 252
Web Browser Setup, 558
Web Browser Setup Options, 1100
web development, 555
welcome, 2
what is key, 91
When tab key reindents, 1007
where is command, 91
while (loop), 1410
wid_expression, 1473
Wildcards, 708
Window Color dialog box, 145
Window Font dialog box, 124
window font dialog box, 1120
Window left margin, 908
Window Menu, 1119
window ordering, 966
windows, 294

Windows PowerShell Options, 895
Windows style Browse for Folder dialog, 950
Windows temp environment, 1088
Windows to color, 978, 979
Word
 multi-file find, 783
word boundary, 1026
word chars, 1005
Word completion, 1030
word navigation, 332
word wrap, 1026
Word Wrap Options (Language-Specific), 1024
word wrap while typing, 1026
workspace bookmarks, 449
workspace files (listing), 851
workspace history list, 1157
Workspace Properties dialog, 817
Workspace tag file only, 971
workspaces, 152
 creating, 162
 managing, 160
 managing projects, 163
 opening and closing, 161
 sharing projects, 164
 third-party, 164
workspaces and projects, 152
Workspaces and Projects (Quick Start), 59
wrap, 780
 SoftWrap, 368
Wrap at beginning/end, 984
Wrap line length, 1074
wrapping (comments), 457
Write List dialog box, 205
Write Selection dialog box, 189

X

Xcode build methods, 236
Xcode emulation keys, 1349
 clipboard, 1353
 command line and text box editing, 1352
 cursor movement, 1349
 deleting, 1350
 files and buffers, 1353
 inserting, 1350
 macros, 1354
 miscellaneous, 1354
 searching, 1351
 selecting, 1350
Xcode workspaces, 164

xcom command, 1427
XEDIT line commands, 1183
XEDIT Line Commands, 1197
XML, 555
 beautifying, 560
 DTD caching, 556
 toggle between begin/end tags, 557
 turn off auto-validate, 1158
 XMLDoc Editor, 556
XML Beautifier Case and Quoting tab, 564
XML Beautifier Comments & Languages tab, 566
XML Beautifier dialog box, 560
XML Beautifier Indent tab, 560
XML encoding, 1144
XML Formatting Options, 556
XML property, 1164
XML symbol translation, 567
XML toolbar, 555
XMLDoc Beautifier dialog box, 545
XMLDoc comments, 454
XMLDoc Editor dialog box, 544
XMLDOC Editor dialog box, 556
XOR operator, 639
Xterm colors, 909

Z

Zip extensions, 1068
Zoom (hide tabs) when one window, 965