SlickEdit Core v4.3 for Eclipse
Introduction

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

SlickEdit Core is a plug-in for Eclipse that contributes a code editor and several views. The code editor can be used to write code in C/C++, Java, or any of the over 50 other languages we support. The SlickEdit Core Views augment the standard Eclipse views, providing information that helps you better understand and navigate your code.

SlickEdit Core is based on our standalone editor, SlickEdit. So, the capabilities provided by the editor and views are nearly the same in both products. Our goal was to integrate them into Eclipse, making them work as seamlessly as possible.

Wherever SlickEdit offered a capability that already existed in Eclipse we chose to exclude it, except in those cases where our version offers better functionality or performance. For example, we included our SlickEdit Search feature due to its color-coded search functionality and its speed.

SlickEdit Core uses the workspace, project, and build system of Eclipse. Therefore, you cannot use a SlickEdit workspace or project with SlickEdit Core. To migrate from one environment to the other you'll have to redefine the workspace.
Getting the Most Out of SlickEdit® Core

SlickEdit Core brings the rich history of excellence of SlickEdit to the Eclipse framework. It delivers the power programming capabilities needed to write more code faster and more accurately, making it an indispensable plug-in for developers.

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® Core contains the most powerful and comprehensive set of features available in any editor. Many are unique to SlickEdit Core. To see a list of features we think are particularly cool, click Help → SlickEdit Cool Features from the Eclipse 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** - SlickEdit Core 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 Core 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 Core command line. For more information, see [Using the Mouse and Keyboard](#).

- **Type as little as possible** - SlickEdit Core 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.

- **Rapidly navigate code** - Instantly jump from a symbol to its definition or view a list of references. Preview definitions for the current symbol without having to open the file. Use bookmarks to mark important locations in the code. SlickEdit Core includes powerful browsers and search capabilities, allowing you to quickly find the code you want. See [Navigation](#) and [Symbol Browsing](#) for more information.

- **Access information quickly** - SlickEdit Core uses visual indicators to provide you with information about your code, including syntax highlighting and color coding. Special views are also available for
viewing information about files, classes, symbols, definitions, and more. To learn more, see SlickEdit Core Views, Symbol Browsing and the Editing Features chapter.

- **Let SlickEdit Core do the formatting** - Syntax indenting, SmartPaste®, and code beautifiers are just a few of the automatic formatting features in SlickEdit Core. For more information, see the topics in the Editing Features chapter.

- **Time-saving utilities** - SlickEdit Core provides many utilities for working with your code, such as DIFFzilla®, 3-Way Merge, Spell Check, FTP, a RegEx Evaluator, math commands, and even a calculator. See the topics in the Tools and Utilities chapter for more information.

## Quick Start

SlickEdit Core offers a Quick Start Configuration Wizard to help configure the most commonly changed options. Use this to get up and running with SlickEdit Core as quickly as possible.

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.

## Register Your Product

Registering your product allows SlickEdit Inc. to provide you with automatic notification of free updates and new releases, and enters your name into a weekly drawing for a free gift pack. To register your product, visit our Web site at [www.slickedit.com/register](http://www.slickedit.com/register).
New Features and Enhancements

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

64-bit Version for Windows

A native 64-bit version of SlickEdit Core for Eclipse is now available for Windows. This version requires a 64-bit JVM and Eclipse. SlickEdit Core still retains the 2 gigabyte file size limit.

Improved PL/I and COBOL Support

• All supported Context Tagging operations for PL/I and COBOL are improved in this version, with upgraded or newly implemented parsers.

• Includes/copybooks are now tagged at parse-time. In order for SlickEdit Core to recognize the location of your PL/I includes and COBOL copybooks, set the macro variables def_pl1_include_path and def_cobol_copy_path to the location of your PL/I includes and COBOL copybooks, respectively.

• Background tagging is supported for both languages.

• A new PL/I formatting option for defining the margins of your PL/I code, Code Margins, is located at the PL/I formatting options dialog at Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Formatting. Setting these margin values will enable SlickEdit Core to accurately detect your PL/I code. If some of your source files have different margin settings, you can either change this value or add a preprocessor statement to the code (*PROCESS MARGINS(X,Y)), and SlickEdit Core will automatically adjust the margin settings for that particular file from the preprocessor statement. The default margin settings are 2 and 72.

Workspace Tagging Excludes

SlickEdit Core will automatically tag all source files in your Eclipse workspace. The Workspace Tagging Excludes feature allows you to specify absolute paths or partial path components in your workspace which you want to be excluded from automatically being tagged. Available at Window → SlickEdit Preferences → Editing → Workspace Tagging Excludes.
Documentation

The SlickEdit Core documentation only covers the features contributed by the SlickEdit Core plug-in. For information about working with Eclipse, including setting up workspaces, doing builds, and integrating with source control; refer to the documentation provided by Eclipse.

Documentation for SlickEdit® Core consists of:

- A built-in Help system. See The Help System for more information.

- The SlickEdit® Core User Guide, which provides the same information as the Help system in PDF format for viewing and printing as a manual.

- Emulation charts in PDF format for the following editors: BBEdit, Brief, CodeWarrior™, CodeWright®, CUA (SlickEdit Core’s default emulation), Epsilon, GNU Emacs, ISPF, SlickEdit (Text Mode edition), Vim, Visual C++® 6, Visual Studio® default, Xcode®, and Eclipse.

- 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.

After SlickEdit Core is installed on your computer, these documents are located in the docs subdirectory of [EclipseInstallDir]/plugins/com.slickedit.windows.libs_[VERSION]/slickedit.


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.

- "Hello World” the SlickEdit Developer Blog - Read thoughts, tips, and tricks from the developers at SlickEdit at http://blog.slickedit.com.

- The SlickEdit® book - After years of developing code with SlickEdit, expert John Hurst brings his

**Documentation Conventions**

CUA is the default editor emulation mode. 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.

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

**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, `Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Indent` is a quick way to write "select Window → SlickEdit Preferences 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 Indent 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 Views 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.
- Code samples and file names are displayed in a **monospaced** font.
- File extensions and environment variables are written with an **UPPERCASE** font.
- SlickEdit® Core 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 Core, so you can use whichever style you prefer.

**The Help System**

A searchable Help system is integrated into the Eclipse Help for SlickEdit® Core. To display the Help,
from the main menu, click Help → Help Contents. SlickEdit Core Help appears at the bottom of the Contents list. You can use the Eclipse Search Scope feature to create a new scope, so that only matches in the SlickEdit Core Help are displayed when searching.
Supported Languages and Environments

This section outlines the languages and file types supported by each SlickEdit® Core feature, as well as supported emulations.

## Supported Languages and File Types

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionScript</td>
<td>2.0</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ada</td>
<td>95</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ant</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ANTLR</td>
<td>v2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AppleScript</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Assembly Language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bourne shell scripts</td>
<td>bash</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>C, C++</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For more information on supported emulations, please refer to the SlickEdit® Help documentation.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C#</td>
<td>1.0, 1.2, 2.0</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>C Shell</td>
<td>BSD, tcsh</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CFScript</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Ch</td>
<td>*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CICS</td>
<td>2.x</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>COBOL</td>
<td>74, 85, 2002</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>D</td>
<td>1.0</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DB2</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DTD</td>
<td>ISO 8879:1986 SGML</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Erlang</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>F#</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fortran</td>
<td>Fortran 77</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Supported Languages and File Types

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>High Level Assembler</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HTML, CFML</td>
<td>HTML 4.0</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IDL</td>
<td>OMG</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>INI, config files</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>InstallScript</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J#</td>
<td>2.0</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Java</td>
<td>5.0</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Java Script</td>
<td>ECMAScript ECMA-262 (1997)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JCL</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>JSP</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
### Supported Languages and File Types

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lex</td>
<td>System V and BSD Unix</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Lua</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Makefile</td>
<td>Unix make, nmake, and gmake</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Modula-2</td>
<td>Wirth/ PIM</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Objective C</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Pascal</td>
<td>ETH and Delphi</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Perl</td>
<td>Perl 5</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PHP</td>
<td>5</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PL/I</td>
<td>ANSI 1976</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PL/SQL</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PowerN P Assembler</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Progress</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>--------------</td>
<td>----------------------------------</td>
<td>-----------------</td>
<td>------------------</td>
<td>---------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>4GL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PV-WAVE</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>2.0, 3.0</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>REXX</td>
<td>ANSI X3.274 1996, Object Rexx</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Ruby</td>
<td>1.8</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>SAS</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Slick-C</td>
<td>*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>System Verilog</td>
<td>IEEE 2009</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Tcl</td>
<td>8</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Transact-SQL</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VBScript</td>
<td>2.0</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Vera</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Verilog</td>
<td>2005</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Visual Basic</td>
<td>VBA, QBasic, VB 6.0</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Supported Languages and File Types

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Basic .NET™</td>
<td>9.0</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>VHDL</td>
<td>IEEE 1076, VHDL-2006</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Window s batch files</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Windows PowerShell</td>
<td>2.0</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>x86 Assembly</td>
<td>Nasm</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>XML, XSD</td>
<td>1.0, 1.1</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yacc</td>
<td>System V and BSD Unix</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

### Embedded Languages

SlickEdit® Core 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 view displays function names if any exist. Embedded language colors are user-defined.

Embedded Languages in HTML

SlickEdit® Core 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:

```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.gig";
      window.event.cancelBubble = true;
   }

<!-- -->
</SCRIPT>

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

   }

}</java>

Embedded Languages in Perl and Other Scripting Languages

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

SlickEdit® Core provides keyboard emulations for the following editors:

• BBEdit
• Brief
• CodeWarrior
• CodeWright
• CUA (SlickEdit Core's default)
• Epsilon
• GNU Emacs
• ISPF
• SlickEdit (Text Mode edition)
• Vim
• Visual C++ 6
• Visual Studio default
• Xcode
• Eclipse

See [Emulations](#) for more information.
Installation

System Requirements

SlickEdit® Core runs on Microsoft® Windows® and Linux® operating systems with the following requirements:

**Windows:**

- Microsoft Windows 8.1, 8, 7, Vista®, XP, or 2000
- 170 MB available hard disk space

**Linux:**

- Linux 32-bit and 64-bit, kernel 2.6 or later, and LSB compliant (64-bit requires glibc 2.12 or later)
- 240 MB available hard disk space
- GTK 2

In addition, each installation of SlickEdit Core on any platform requires:

- A minimum of 256 MB of memory
- Eclipse 4.3 - 4.4
- CDT, JDT, and PDT are required for building, running, and debugging C/C++, Java, and PHP, respectively.
- Java 5 or later (32-bit or 64-bit on Windows, 32-bit or 64-bit on Linux) from Sun or IBM. SlickEdit Core will not run using gcj.

Installing SlickEdit® Core

SlickEdit Core is provided as an annual subscription that includes support. Installation is performed by using the Eclipse Install Wizard, either with the SlickEdit Core Update Site or with a downloaded ZIP archive.

Installing with the Update Site

**Warning**

The instructions below are for installing version 4.3 of the SlickEdit Core plug-in for Eclipse version 4.3, Kepler. Use of this version of SlickEdit Core with any other version of Eclipse is unsupported. If you are using an earlier version of Eclipse, contact product support for information.
To install SlickEdit® Core from the SlickEdit Core Update Site:

1. In Eclipse, select Help → Install New Software, check Group items by category, and click the Add... button.

2. Enter the SlickEdit Core Update Site URL: http://www.slickedit.com/update/secore/kepler in the Location field and click OK.

3. Expand the Code Editing category and mark the check box for SlickEdit Core for Eclipse, as shown below.

4. SlickEdit Core for Eclipse is comprised of 4 features, only one of which is required. The other 3 features provide integration with other Eclipse plug-ins used for working with C/C++ (CDT), Java (JDT), and PHP (PDT). This prevents users from being forced to install unnecessary plug-ins into their Eclipse environment.

   If you want Eclipse to automatically download all optional features for SlickEdit Core, along with any required plug-ins, mark the Contact all update sites during install to find required software check box. Without this check box marked, the only optional features that Eclipse will install are those which have dependencies satisfied by your current IDE. In other words, if you do not have CDT installed, the SlickEdit Core CDT Integration feature will not be downloaded. If you do have CDT installed, the Install Wizard will download the optional feature.

5. Click Next and continue through the wizard to complete the install process. You will be prompted to restart Eclipse after the installation is finished.

Installing with ZIP Archive

SlickEdit also provides a ZIP file for installing SlickEdit Core, which downloaded at http://www.slickedit.com/trial/slickedit-core. There is link at the bottom of the page for downloading the ZIP file. The page provides instructions for downloading the Trial version, but it is the same installation file as the full version. The license file used determines whether it is a trial or not.

Installing using the ZIP file is almost the exact same process as installing using the Update Site, except that instead of specifying the Update Site URL on the Add Site dialog, you click the Archive... button and browse to the ZIP file. The Install Wizard will walk you through the install, as detailed above.

Licensing

SlickEdit Core 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 Core and depends on your platform. However, if you are setting up a portable installation of Eclipse with SlickEdit Core, to run on a USB drive...
for example, you should manually copy the license file to the `win` subdirectory of your SlickEdit Core installation on Windows or the `bin` subdirectory on Linux.

When SlickEdit Core 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 → SlickEdit License Manager. The SlickEdit License Manager provides the following options:

- **Request a Trial License** - To try out SlickEdit Core, click the option to obtain a Trial License. This will take you to a Web page where you can request a license file. The license file will be sent to you in an e-mail.

- **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 Core.

- **Request a license for a purchased product** - If you have already purchased a license for SlickEdit Core, you can download a license file to this computer from the SlickEdit Web site.

- **Install a license file** - Copies a downloaded license file to the location SlickEdit Core uses to store the file.

- **Renew your subscription** - Select this option to renew your SlickEdit Core subscription.

### Uninstalling SlickEdit® Core

To uninstall SlickEdit® Core:

1. In Eclipse, select Help → Install New Software..., and click the already installed link at the bottom of the dialog.

2. On the Installed Software tab select SlickEdit Core for Eclipse from the list, and click the Uninstall... button at the bottom of the dialog.

3. Click Finish on the Uninstall dialog to complete the uninstall.

**Note**

Uninstallation from the Eclipse Install Wizard will not remove the SlickEdit plug-ins from disk. If you wish to completely remove SlickEdit Core for Eclipse from your computer, after uninstalling from Eclipse you must remove all `com.slickedit.*` directories from `[EclipseInstallDir]/plugins`.  

19
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].

SlickEdit Core 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 Core 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 → SlickEdit 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 Core 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 Core program information, which is automatically provided if you use Help → SlickEdit Product Support. If you initiate a report from the website, select Help → About SlickEdit Core, 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 1.919.473.0100. 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 **Window → SlickEdit Preferences → Application Options → Product Improvement Program**. For more information about these preferences, 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 Core was designed with speed in mind. Most operations perform nearly instantaneously. However, the size and location of your codebase can affect SlickEdit Core performance along with various settings within SlickEdit Core. 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 Core 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 and configuration files are stored locally. SlickEdit Core 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 Core. 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.

SlickEdit Core Configuration Files

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

By default, SlickEdit Core stores your config files in \My Documents\My SlickEdit Core Config on Windows and in $HOME/.secore on Linux. These are typically on a local drive. You can specify a different location for your config using the -vsconfig option when Eclipse is launched:

eclipse -vsconfig /dev/coreconfig

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

Along with making sure that your tag files are stored locally, you should make sure that SlickEdit Core 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 **Window → SlickEdit Preferences → 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 Core. 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 Core 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 project tag files, 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.

Tuning Context Tagging

After you've checked the items above, the next optimizations to try are the various control settings for Context Tagging. SlickEdit Core's Context Tagging system provides many of the advanced features that make using SlickEdit Core 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, 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 **Window → SlickEdit Preferences → Editing → Context Tagging**. This screen contains a number of parameters you can use to control the performance of Context Tagging.

**Background Tagging**

If you are experiencing sporadic pauses in SlickEdit Core, 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 Core, 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 Core 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 Core won't know about those changes until you retag the workspace. For normal size projects, SlickEdit Core 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 select **Tools → Retag Workspace** from the Eclipse main menu. You will have to wait while SlickEdit Core 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.

**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

```cpp
foo();
```

After typing the open parenthesis, SlickEdit Core 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 Core 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 Core 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 -vsconfig option on the command line:

```
eclipse -vsconfig config
```

This will launch Eclipse, putting the configuration in a "config" directory below your Eclipse install
directory. Be sure to use a new location or delete that directory before launching Eclipse 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 Core does not open each candidate file to
eliminate invalid references. So, you get your answer more quickly, but it may not be fully accurate.

**Profiling**

SlickEdit Core 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 Core 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.
SlickEdit Core is one of the most powerful programming editors available today, and one of the most flexible. SlickEdit Core 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 Core 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 Core Options dialogs, by selecting **Window → SlickEdit Preferences** 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 Core will emulate.
- **Colors** - set your color scheme.
- **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.
- **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 Core.

**Emulation**

CUA is the default emulation, which provides key bindings familiar to Microsoft Windows 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 is best.

You can change your emulations at any time by selecting **Window → SlickEdit Preferences** from the main menu. Then expand **Window → SlickEdit Preferences → Keyboard → Emulation**. For more information, see **Emulations**.
Colors

Many users are particular about the colors they use. On this form, you can select a color scheme that will be used to color all editor windows. You can also select your symbol coloring scheme, which defines how different symbols are detected and colored. Preview your selections in different languages using the preview window and the language combo box.

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

Fonts 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. Use the preview windows to view your selections in the languages of your choice.

To set fonts later, select Window → SlickEdit Preferences → Appearance → Fonts. For more information on setting fonts, see Fonts.
Coding

SlickEdit Core 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 Core 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.
Context Tagging

Context Tagging creates a database for all the symbols in your code. This allows SlickEdit Core to rapidly jump from a symbol to its definition or show a list of references. Other features, such as completions, also use this information. To properly work with your code, SlickEdit Core 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 Core 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.
The final step in the Quick Start Configuration Wizard, allows you to do three things:

- **Export Preferences** - 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 Core 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 Core 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 `Window → SlickEdit Preferences` 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 `Window → SlickEdit Preferences → Editing → Cursor Movement`, then set the option `Click past end of line` to True.

- **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 `Window → SlickEdit Preferences → Editing → Cursor Movement`, then set the option `Cursor up/down places cursor in virtual space` to True.

- **Changing the line insert style** - In code, a line of text is a meaningful unit of functionality. SlickEdit® Core 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 `Window → SlickEdit Preferences → 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 `Window → SlickEdit Preferences → Keyboard → 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 Core. These options are accessed from the SlickEdit Preferences dialog (`Window → SlickEdit Preferences → 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 **Format → [Language] Options** menu item (or the `setupext` command).

- **Setting symbol navigation** - For C and C++, by default, with each attempt to navigate to a definition (Ctrl+Dot or **Navigate → 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.

- **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++ preprocessing** - For C and C++, your source code base will typically include preprocessor macros that you use in your code for portability or convenience. For performance considerations, Context Tagging® does not do full preprocessing, so preprocessing that interferes with normal C++ syntax can cause the parser to miss certain symbols. To configure your preprocessing to avoid these omissions, see **C/C++ Preprocessing**.
Start Coding

After settings have been configured, you are ready to start coding. See the Editing Features chapter to learn more about how SlickEdit® Core can help in your everyday work.

If you’re not ready to get to work just yet, you may want to configure even more preferences. For information, see the User Preferences chapter.
User Interface

This chapter describes the SlickEdit Core user interface. Much of the power of SlickEdit Core comes from using the keyboard to invoke operations. See Using the Mouse and Keyboard for more information.
Screen Layout

Note

SlickEdit® Core does not modify the Eclipse screen layout, so this information is intended only as a brief overview. See “Workbench User Guide” in the Eclipse Help for more details about the Eclipse layout.

The Workbench

The workbench is the area where the workspace, projects, and programs are contained. Use the workbench to manage and edit all projects in various perspectives, views, or editors. Work may only occur in one workbench at a time.

The Workspace

The workspace is a collection of projects. A project contains all resources such as source files, subfolders, icons, and generated code.
Perspectives

A perspective in Eclipse is a set of views and editors. For example, the Java perspective has a much different set of views than the Debug perspective. You can customize each perspective's layout by dragging and dropping.

The SlickEdit® Core Editor

Use the SlickEdit Core editor to create and change projects, folders, files, and classes. To edit files and classes in SlickEdit Core, first associate those files or classes with the editor or open the source file using the editor.

To associate files to edit, complete the following steps:
1. From the main menu, click **Window → Preferences**.

2. Expand **General**, then expand **Editors** and select **File Associations**.

3. From the **File Types** list, select the desired file type. Or, to add an extension, click **Add**.

4. From the **Associated Editors** list, select the desired editor.

5. To make this the default editor for this file type, click the **Default** button.

6. Click **OK**.

### Opening Files

Source files can be opened with the SlickEdit Core editor in the following ways:

- To open a source file in the workspace, from the Navigator view, select the desired file. Right-click the file, then select **Open**.

- To open a file using a specific editor, from the Navigator view, select the desired file. Right-click the file, then select **Open With**.

- To open a file that is outside of the workspace, from the main menu, click **File → Open**.
Overview Ruler

To the right of the editor is the **Overview Ruler** for showing the location and type of various annotations in the current file. The Overview Ruler will show almost any annotation contributed to Eclipse by any plug-in. This includes build errors, tasks, bookmarks, breakpoints, and many others.

The Overview Ruler can be used to quickly jump to an annotation, either by using the vertical scroll bar of the editor, or simply by clicking on the annotation marker. Scrolling to an annotation may not be accurate if **Soft Wrap** is on. To see a preview of an annotation you can hover your mouse over the marker in the ruler. The Overview Ruler is customizable by right-clicking on the ruler and selecting **Preferences**, or by going to **Window → Preferences → General → Editors → Text Editors → Annotations**. You can select which annotations get shown in the ruler, and also what colors are used for the markers.

Though the Overview Ruler is shown by default, you can disable it by right-clicking on the ruler and selecting **Hide**. You can also do this by un-checking **Display → Overview Ruler**.

Switching Between Editors

SlickEdit® Core provides the ability to switch from the SlickEdit Core editor to several other Eclipse editors, for the current buffer. These commands are provided in the right-click context menu of the editor:

- **Switch to Java Editor** - Changes to the JDT editor. Visible only for Java files.
- **Switch to C/C++ Editor** - Changes to the CDT editor. Visible only for C/C++ files.
- **Switch to PHP Editor** - Changes to the PDT editor. Visible only for PHP files.
- **Switch to Ant Editor** - Changes to the Eclipse Ant `build.xml` editor. Visible only for `build.xml` files.
- **Switch to Plug-in Manifest Editor** - Changes to the Eclipse plug-in `plugin.xml` editor. Visible only for `plugin.xml` files.

Similarly, you can also switch from any Eclipse editor to the SlickEdit Core editor for the current buffer. To
do this, use the **Switch To SlickEdit**, accessible from the right-click context menu in the Eclipse editor.

### Menus

SlickEdit Core adds some new menus to the main menu. It also adds new items to existing menus. Some SlickEdit Core menus, such as the **Format** menu, are not visible unless the active editor is SlickEdit. For more information on the menu items added by SlickEdit Core, see [Menus, Dialogs, and Views](#).

### Views

Views are windows that show various types of information that you can move around and dock within Eclipse. See [SlickEdit Core Views](#) for information.

### Status Area

The status area for a perspective and an editor displays text messages. It indicates if the current mode is insert, overwrite, or replace, and if a file is read-only. The editor status area also displays the line and column number for the cursor location.
Dialogs

Although SlickEdit® Core shares a heritage with our stand-alone editor, SlickEdit, dialogs within SlickEdit Core may contain options that are not available when the functionality is not applicable to the Eclipse environment. By the same token, some SlickEdit commands may not be available.

See Menus, Dialogs, and Views for descriptions of the menus and dialog specific to SlickEdit Core.

Command Line

To activate the SlickEdit® Core command line, press Esc in CUA emulation, Ctrl+A in Vim emulation, or Alt+X in GNU Emacs emulation.

See SlickEdit® Command Line for more information.
SlickEdit® Core Views

Views complement the file opened in the editor. You can move, resize, and customize views easily. All the views and perspectives have live connectivity, meaning that if a file name or property in one view is modified, then that change stays true for that item in every area of the workspace.

Views support editors and provide alternative presentations as well as ways to navigate the information in your workbench.

Views also have their own menus. To open the menu for a view, click the icon at the left end of the view's title bar. Some views also have their own toolbars. The actions represented by buttons on view toolbars only affect the items within that view.

A view might appear by itself, or stacked with other views in a tabbed notebook. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the workbench window.

Opening SlickEdit® Core Views

Perspectives offer pre-defined combinations of views and editors. To open a view that is not included in the current perspective, from the main menu, click **Window → Show View**. To open a SlickEdit Core view, click **Window → Show View → Other**, expand **SlickEdit** and double-click the view you want to open.

**Tip**

- You can create *fast views* to provide a shortcut to views that you use often.
- After adding a view to the current perspective, you may want to save your new layout by clicking **Window → Save Perspective As**.

For more information on views and the multiple operations they allow, see the Eclipse online Help.

Available SlickEdit® Core Views

The views below are made available by SlickEdit Core.

**Class**

Provides an outline of both the members of the current class as well as any visible inherited members. This view also shows the inheritance hierarchy of the current class, useful for object-oriented programming languages such as Java. See **Class View** for more information.

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

**FTP Client**

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

**Preview**

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 [Preview View](#) for more information.

**References**

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). See [References View](#) for more information.

**SlickEdit Output**

Displays output from various operations within the editor, such as errors.

**SlickEdit Search**

Displays the results of multi-file searches, or when the option **List all occurrences** is selected on the Find and Replace View. See [Find and Replace](#) for more information about searching and replacing.

**Symbols**

Contains the symbol browser, which lists the symbols from all of the tag files. See [Symbols View](#) for more information.
SlickEdit® Core Command Line

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

**Tip**

- SlickEdit® Core 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 Core, so you can use whichever style you prefer.

- Although SlickEdit Core shares a heritage with our stand-alone editor, SlickEdit, some SlickEdit commands are not available when the functionality is not applicable to the Eclipse environment.

**Activating the Command Line**

To activate or toggle the SlickEdit® Core 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 Core's default emulation) - **Esc**
- Epsilon - **Alt+X** or **F2**
- GNU Emacs - **Alt+X** or **F2**
- ISPF - **Esc**
- SlickEdit (Text Mode edition) - **Esc**
- Vim - **Ctrl+A**
- Visual C++ - **Esc**
- Visual Studio default - **Esc**
- Xcode - **Esc**
• Eclipse - Esc

See Emulations for more information.

Command Line History

The SlickEdit® Core 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® Core command line, a list of matching completions is displayed, including any command line arguments used in a previous command. Use the Down arrow to move to the next command in the list, and 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 Core 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 Core. For instance, to find all operations that begin with "find", type find in the command line, and SlickEdit Core 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 (Window → SlickEdit Preferences → Keyboard → 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 Window → SlickEdit Preferences → 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® Core 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 Core, see [Key Shortcuts in Text Boxes](#).

### Using the Command Line to View Key Binding Associations

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

**Tip**

Alternatively, you can use the Key Bindings options page (Window → SlickEdit Preferences → Keyboard → 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® Core command line (by pressing Esc) and type `what_is` (or type `what` and press the spacebar for auto-completion), then press Enter.

2. The status area will prompt with the text What is key. Enter the key sequence in question. The status area will then update with the appropriate 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® Core 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® Core 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.

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® Core 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 Core 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 Core 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 **Window** → **SlickEdit Preferences** → **Keyboard** → **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.

<table>
<thead>
<tr>
<th>Graphical Command</th>
<th>Command Line Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>gui_open</td>
<td>edit</td>
</tr>
</tbody>
</table>
Common SlickEdit® Core Commands

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

<table>
<thead>
<tr>
<th>Graphical Command</th>
<th>Command Line Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>gui_find</td>
<td>find</td>
</tr>
<tr>
<td>gui_replace</td>
<td>replace</td>
</tr>
<tr>
<td>gui_write_selection</td>
<td>put</td>
</tr>
<tr>
<td>gui_append_selection</td>
<td>append</td>
</tr>
<tr>
<td>gui_margins</td>
<td>margins</td>
</tr>
<tr>
<td>gui_tabs</td>
<td>tabs</td>
</tr>
<tr>
<td>gui_find_proc</td>
<td>find_proc</td>
</tr>
</tbody>
</table>

1. e file
2. sa file
3. number
4. f symbol
5. / search_string / options
6. c/ search / replace / options
7. gt/ search / options
8. sb name
9. gb name
10. man command
11. cd directory
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12.</td>
<td><code>del filename</code></td>
</tr>
<tr>
<td>13.</td>
<td><code>pushd directory</code></td>
</tr>
<tr>
<td>14.</td>
<td><code>popd</code></td>
</tr>
<tr>
<td>15.</td>
<td><code>set env=value</code></td>
</tr>
<tr>
<td>16.</td>
<td><code>dos command</code></td>
</tr>
<tr>
<td>17.</td>
<td><code>math expr</code></td>
</tr>
<tr>
<td>18.</td>
<td><code>eclipse_open_resource</code></td>
</tr>
<tr>
<td>19.</td>
<td><code>eclipse_next_view</code></td>
</tr>
<tr>
<td>20.</td>
<td><code>eclipse_maximize_part</code></td>
</tr>
<tr>
<td>21.</td>
<td><code>eclipse_show_outline</code></td>
</tr>
</tbody>
</table>
Using the Mouse and Keyboard

SlickEdit® Core provides three ways to launch operations: commands, menu items, and key bindings. 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 Core command line.
- Click Edit → Find/Replace in the main menu.
- Press the key binding Ctrl+F.

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.

Key Shortcuts in Text Boxes

Most keyboard shortcuts for basic text operations can be used inside any text box in SlickEdit® Core, including the SlickEdit Core 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 Window → SlickEdit Preferences → Editing → General, then set the option CUA text box to False.

<table>
<thead>
<tr>
<th>Text Box Editing Operation</th>
<th>Key Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Append Cut</td>
<td>Ctrl+Shift+X</td>
</tr>
<tr>
<td>Append to Clipboard</td>
<td>Ctrl+Shift+C</td>
</tr>
<tr>
<td>Copy Word to Clipboard</td>
<td>Ctrl+K</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Cut Line</td>
<td>Ctrl+Backspace</td>
</tr>
</tbody>
</table>

52
<table>
<thead>
<tr>
<th>Text Box Editing Operation</th>
<th>Key Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut to End of Line</td>
<td>Ctrl+E</td>
</tr>
<tr>
<td>Cut Word</td>
<td>Ctrl+Shift+K</td>
</tr>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
</tr>
<tr>
<td>Delete Character Under Cursor or Selection</td>
<td>Delete</td>
</tr>
<tr>
<td>Delete Previous Character or Selection</td>
<td>Backspace</td>
</tr>
<tr>
<td>Expand Alias</td>
<td>Ctrl+Shift+O</td>
</tr>
<tr>
<td>Expand Partially Typed Parameter or Insert Space</td>
<td>Spacebar</td>
</tr>
<tr>
<td>Extend Selection to Mouse Position</td>
<td>Shift+Click</td>
</tr>
<tr>
<td>Insert Mode Toggle</td>
<td>Insert</td>
</tr>
<tr>
<td>List Clipboards</td>
<td>Ctrl+Shift+V</td>
</tr>
<tr>
<td>List Matches to Partially Typed Parameter</td>
<td>?</td>
</tr>
<tr>
<td>Lowercase Word</td>
<td>Ctrl+Shift+L</td>
</tr>
<tr>
<td>Move Cursor Left</td>
<td>Left arrow</td>
</tr>
<tr>
<td>Move Cursor Right</td>
<td>Right arrow</td>
</tr>
<tr>
<td>Move Cursor to Beginning of Line</td>
<td>Home</td>
</tr>
<tr>
<td>Move Cursor to End of Line</td>
<td>End</td>
</tr>
<tr>
<td>Next Word</td>
<td>Ctrl+Right arrow</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
</tr>
<tr>
<td>Previous Word</td>
<td>Ctrl+Left arrow</td>
</tr>
<tr>
<td>Select Line</td>
<td>Triple-click</td>
</tr>
<tr>
<td>Select Text Between Cursor and Beginning of Line</td>
<td>Shift+Home</td>
</tr>
<tr>
<td>Select Text Between Cursor and End of Line</td>
<td>Shift+End</td>
</tr>
</tbody>
</table>
### 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 `Window → SlickEdit Preferences → Keyboard → 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](#).
Printing

Printing within SlickEdit Core varies slightly between Windows and Linux.

Printing on Windows

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

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

Windows Printer Configuration

To configure printing options, from the main menu, click File → Print (or use the gui_print command). The Print dialog is displayed, as pictured below.
To print immediately using the current settings, click Print. To select the font used for printing, click Font. For printer setup, click Setup.

**Header/Footer Print Settings**

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

**Print Margin Settings**

To define margin spacing, use the Margins tab of the Print dialog. In the After header and Before footer fields, enter the amount of spacing to come between the header and the first line on a page, and the amount of spacing to come between the last line on a page and the footer.
In the top, bottom, left, and right margin fields, enter the amount of spacing in inches to come between the outer edge of the paper and the printed text.

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

**Print Schemes**

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

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

System schemes are stored in the `print.ini` file located at

```
[EclipseInstallDir]/plugins/com.slickedit.windows.libs_[VERSION]/slickedit
```

User-defined schemes or modified system schemes are stored in the `uprint.ini` file.

---

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

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

**Linux Printer Configuration**

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

![Text Mode Print Dialog](image)

This dialog contains some of the same options as the Windows Print dialog. The available options.

**Inserting a Formfeed**

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

This chapter describes how to set key preferences that control the look and behavior of SlickEdit Core. For a complete description of all preferences screens, see Options Dialog.
Introduction to User Preferences (Options)

SlickEdit Core 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 SlickEdit Preferences dialog, which can be displayed from the main menu by clicking Window → SlickEdit Preferences. For more information including a detailed breakdown of options, see Options Dialog.

SlickEdit Core contains two kinds of preferences:

• **Global Options** - affect all languages.

• **Language Specific Options** - affect only the specified language.

**Tip**

If you are using SlickEdit Core in a multiple user environment, each user must define a VSECLIPSECONFIG environment variable that refers to a local directory. This allows each user to have their own configuration. If making modifications to vslick.ini, make a local copy of this file and place it in the VSECLIPSECONFIG directory file. See Vim Tutorial 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)

Other global preferences, such as default search options and selection styles are also available for setting through the SlickEdit Preferences dialog (Window → SlickEdit Preferences). These options are described in the documentation on a contextual basis. For a listing of preferences 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®, and other code-style settings are all language-specific. These settings are located on the SlickEdit Preferences dialog (Window → SlickEdit Preferences → Languages → [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 Format → [Language] Options menu item (or the setupext command). This will open the SlickEdit Preferences 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 Core records all state information and option settings in your config directory. Some items are saved in the state file, and others are stored in specific files. For more information, see Configuration Directories and Files.

Periodically, you should make a backup of your config directory. Some errors in SlickEdit Core may cause your configuration to be corrupted. Having a backup of this directory allows you to fall back to a known, good configuration, rather than having to start over with a clean, default configuration.

Options Export and Import

SlickEdit Core also provides the capability to Export and Import your options settings. You can create an export package of all or part of your SlickEdit Core 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 is not supported. Also, options should not be exported from one version of SlickEdit Core and then imported into another. While these operations may work for some options, we cannot predict when this will cause a problem.

For more information see Export/Import Options.
Emulations

*Emulation* is the process of imitating another program. SlickEdit® Core 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 to 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

```
[EclipseInstallDir]/plugins/com.slickedit.windows.libs_[VERSION]/slickedit
```

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® Core command line in each emulation, see *Activating the Command Line*.

- **BBEdit**
- **Brief** - This emulation relies heavily on `Alt` key bindings. In addition to Brief emulation support, SlickEdit Core also supports Brief regular expressions. See *Brief Regular Expressions* for more information.
- **CodeWarrior**
- **CodeWright**
- **CUA** - CUA is an acronym for Common User Interface, a standard set of user interface guidelines similar to those used in Microsoft products. *This is the default emulation mode used by SlickEdit Core.*
- **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*).
- **Mac OS X**
- **SlickEdit® Core (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 `Ctrl` 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 Core 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 Core command, the corresponding Visual C++ key binding is used.

- **Xcode**

**Changing Emulations**

After SlickEdit® Core is installed, you are prompted to choose an emulation. CUA is the default emulation mode for SlickEdit Core. 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 `Window` → `SlickEdit Preferences`, expand `Keyboard` in the tree, and select `Emulation`.

---

63
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® Core 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.

## Vim Tutorial

SlickEdit Core 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 Core users.

To use the command, open the SlickEdit command line, then type `vimtutor` and press `Enter`. The practice file is displayed in the editor. Each time you use the command, SlickEdit Core creates a fresh copy of this file.

**Note**

- 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 Core 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.
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® Core 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 Window → SlickEdit Preferences → Keyboard → 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.
Managing Bindings

Create and manage key bindings using the Key Bindings option screen. This displays a list of all SlickEdit® Core 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 **Window → SlickEdit Preferences → Keyboard → 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.
Bindings are based on the editor emulation mode (CUA is the default). To change the emulation mode, click **Window → SlickEdit Preferences → Keyboard → 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 Core 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 Core 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® Core 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 (**Window → SlickEdit Preferences → Keyboard → 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® Core 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**. If you have advanced knowledge of SlickEdit® Core, you can edit the Slick-C® key binding source directly. See **Editing the Key Binding Source** for more information.

**Removing Bindings**

To remove a binding:

1. Using the Key Bindings 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® Core and imported in, useful for creating backups, sharing with other team members, or taking with you should you switch computers.

**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.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® Core 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 Core 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 Core:

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® Core defaults, from the main menu, click **Window → SlickEdit Preferences → Keyboard → 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.
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 Window → SlickEdit Preferences → Keyboard → 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® Core main menu displays the key bindings for commands associated with each menu entry. These bindings can be condensed for non-CUA emulations. See Menus for more information.
Executing Eclipse Commands From the SlickEdit® Editor

Use the Slick-C® command `eclipse_execute_command` to execute Eclipse commands from the Slick-C environment. You must provide the identifier of the Eclipse command, as well as any parameters you wish to pass as arguments. Note that some commands in Eclipse are specific to a certain editor, and thus, will not work when launched from the SlickEdit environment.

The argument information for `eclipse_execute_command(_str id, _str paramNames, _str paramValues)` is as follows (from Javadoc):

- **id** - identifier of the Eclipse command.
- **paramNames** - comma delimited String of the parameter names for 'id'. Can be null or empty.
- **paramValues** - comma delimited String of the parameter values for 'id'. Can be null or empty.

One of the best ways to use this command is to write a wrapper function to bind a key to the Eclipse command of your choice. Here is an example that shows how to execute the **Open Resource** command, with no arguments:

```c
_command void open_resource() name_info(',') {
  _eclipse_execute_command("org.eclipse.ui.navigate.openResource";"","");
}
```

`open_resource` can then be bound to Ctrl+Shift+R, as it is by default in Eclipse.

Here is an example of a command that takes some arguments:

```c
_command void show_problems() name_info(',') {
  _str id = "org.eclipse.ui.views.showViewMenu"
  _str params = "org.eclipse.ui.views.showView.viewId,org.eclipse.ui.views.showView.makeFast"
  _str values = "org.eclipse.ui.views.ProblemView,false"
  _eclipse_execute_command(id, params, values);}
```

This command will activate the Eclipse Problems view, showing it in the default location (not in the fast view bar).

Tips on Finding Eclipse Command Information

Finding Eclipse command identifiers and parameter information for programmatic execution can sometimes be a challenging task. This section provides some tips on gathering all the information you need.
1. Use the Eclipse Keys preference screen to find the name of a command.

2. Search a Cheat Sheet for the command name.

On the Eclipse Keys preference screen, found at Window → Preferences → General → Keys, you can find the names of all the Eclipse commands you may want to execute from the SlickEdit® editor. You will need this information to track down the command identifier. The following example tracks down the identifier for the Open Search Dialog command, which opens the Eclipse Search dialog.

Other useful information on this screen includes the Category and When fields. As previously stated, some commands require certain active editors to function, and thus, will not launch from SlickEdit. If the Category for a command is "C/C++ Source", that command is specific to the CDT editor. Similarly, if the When for a command is something like "PDE Editor", then that command is specific to one of the PDE editors. If you have found that a certain command won’t execute as expected, use the Keys screen to first make sure that it is not specific to a different editor.

Now that you have the name of the command, you can search an Eclipse Cheat Sheet in order to discover the command identifier.

1. Go to File → New → Other, then expand User Assistance and select Cheat Sheet.

2. Enter any name you wish for the Cheat Sheet and use the Simple Cheat Sheet template. Click Finish.

3. Open the Cheat Sheet and select Title → Item in the Content pane. The Command pane will appear. Now click the Browse button.
4. Type the command name in the text box to locate the command in the tree. Select the appropriate item in the tree to display the identifier for the command.
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 Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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 Core. To turn these on or off, from the main menu, click Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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. The SlickEdit Core editor and views do not use the Eclipse color and font settings.

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

Tip

Xft fonts are supported on Linux by default.

Setting Fonts for Screen Elements

To configure font settings for screen elements, use the Fonts options screen (Window → SlickEdit Preferences → Appearance → Fonts). For a description of each option, see Font Options.

Tip

"Regular" editor windows that do not use Unicode fonts are controlled by the SBCS/DBCS Source Windows element.
Recommended Fonts for Elements

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

**Note**

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

**Command Line Fonts**

The following table contains recommendations, based on the operating system, for the Command Line font element:
## Colors, Color Coding, and Symbol Colors

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

### Selection List Fonts

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

### Dialog Box Fonts

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

### Text Box Fonts

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

### SBCS/DBCS Source Window Fonts

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

### Unicode Source Window Fonts

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

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

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 (Window → SlickEdit Preferences → Appearance → Colors or the color command) to set the color for different screen elements in SlickEdit Core. This includes syntactic elements in the editor window, like keywords, comments, strings, etc. as well as other user interface elements like the message area or the status line. Window colors and backgrounds are set using the facilities provided by the operating system.

Setting Colors for Screen Elements

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

1. From the main menu, click Window → SlickEdit Preferences → Appearance → Colors (or use the color command). The Colors option screen is displayed.
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 Core 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 Core 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 scheme 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 Schemes**

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

To use a color scheme, from the **Scheme** drop-down, select a color scheme and click **Apply** or **OK**.

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

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 lexer names to get embedded language color coding. The following is an example for Perl:

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

Symbol Coloring

Use the Symbol Coloring options screen (Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → View and checking Symbol Coloring. You can also enable Symbol Coloring for a specific file by selecting Display → 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 scheme, along with the Color Coding lexer, 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 Core 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.

Symbol Coloring under some circumstances can cause SlickEdit Core to pause momentarily while you type. If you experience these pauses, please turn off Symbol Coloring by unchecking Display → 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 Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → View and unchecking Symbol Coloring.

Note

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

Symbol Coloring can be used to highlight unidentified symbols. These are symbols for which the SlickEdit Core Context Tagging engine can not find a definition. 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 Window → SlickEdit Preferences → 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 Schemes

A symbol coloring scheme 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 scheme 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 schemes, 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:
Symbol Coloring

- 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 it's 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 scheme. For a symbol to match a rule, it must be the first rule in the symbol coloring scheme 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.

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

### Color Scheme Compatibility

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

In addition, each base color scheme has a designated, default symbol coloring scheme preferred for that
scheme. This allows you to switch color schemes and automatically get a corresponding symbol coloring scheme which is compatible. See Colors for more information.

**Selecting a Symbol Coloring Scheme**

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

1. From the main menu, click **Window → SlickEdit Preferences → Appearance → Symbol Coloring**. The Symbol Coloring options screen is displayed.

2. Using the **Scheme** combo box, select a scheme name. If your current scheme is modified, those modifications will be automatically saved to that scheme. If you select a scheme which is incompatible with your current base color scheme, you will be prompted to confirm that you really want to use the selected color scheme. See Color Scheme Compatibility for more information.

3. The list of rules will be shown for the selected symbol coloring scheme. You can get a quick overview of the scheme 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 scheme.

**Editing a Symbol Coloring Scheme**

To edit the current symbol coloring scheme, from the main menu, click Window → SlickEdit Preferences → 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® Core.

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 scheme. By default a rule will inherit from the Window Text color defined in your base color scheme.

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 its 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 its 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 Scheme

New symbol coloring schemes can be created by selecting an existing scheme and adding or subtracting rules, then saving the scheme under a new name. To create a new symbol coloring scheme from scratch, start with the **None** scheme and then add your own custom rules.

![Enter Scheme Name](image)

### Selecting a Symbol Coloring Scheme for the Current File

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

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

Selecting a specific scheme for the current file will not change the symbol coloring scheme for any other files, nor will it change the default symbol coloring scheme. The selected symbol coloring scheme 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 scheme you selected, as long as your base color scheme does not change.

### Language-Specific Symbol Coloring Settings

Certain Symbol Coloring features can be disabled on a per-language basis. To edit language-specific symbol coloring options, from the main menu, click **Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → View**. The Language View options are displayed.
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 with are not found by Context Tagging®. This can serve effective 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 decreases the accuracy of Symbol Coloring, especially with respect to detecting misspelled symbols.

### Symbol Coloring Performance Settings

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 Window → SlickEdit Preferences → Editing → Context Tagging. The Context Tagging® options are displayed.

From this dialog, you can configure the following settings.

- **Update 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 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.

**Color Coding**

For information on how to set up colors for various entities in the editor, see Colors. This section describes
how to configure the Color Coding engine, which identifies entities to color.

**Adding Color Coding Keywords to Supported Languages**

To add color-coded keywords to a supported language, complete the following steps:

1. From the main menu, click **Window** → **SlickEdit Preferences** → **Languages**, expand your language
category and language, then select **Color Coding**.

2. Click **New**.

3. Enter the new keywords separated with a space character.

4. Click **OK**.

5. Click **OK** on the Color Coding options screen.

For more information, see **Color Coding Configuration**.

**Creating Color Coding for a New Language**

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

1. From the main menu, click **Window** → **SlickEdit Preferences** → **Languages**, expand your language
category and language, then select **Color Coding**.

2. Select the **Color Coding Tokens Tab**, then click **New**. The Enter New Keywords dialog box is
displayed.

3. Enter the new lexer name. Usually this is a language name such as **C** or **Java**. Click **OK**.
4. On the **Tokens** tab, make sure the new keyword is selected, then correct the **ID start characters**. These are valid characters which can be the start of an identifier.

5. Correct the **ID follow characters**. These are additional characters which are valid after the start ID character. For example, digits are usually allowed in identifiers, but not as the first character of an identifier.

6. Select the **Color Coding Comments Tab**. This lists the comments currently defined and allows you to define new multi-line and line comments. For each comment, click **New** to add a line or multi-line comment.

7. Select the **Color Coding Numbers Tab** to display various numeric style options.

8. Select the **Color Coding Strings Tab** to display various string literal options.

9. If you have not found all the options you need, click the **Color Coding Language Tab**. This displays even more advanced language-specific options.

1. Click **OK** on the Color Coding options screen.

**Color Coding Configuration**

To configure color coding, from the main menu, click **Window → SlickEdit Preferences → Languages**, expand your language category and language, then select **Color Coding**. This screen lets you change the list of tokens and other information that is color coded in the editor. The options on each tab are described in the section, **Language-Specific Color Coding Options**.

Click **Colors** at the bottom of the dialog to display the Color Settings dialog, which allows you to specify the color for color coding elements and other editor elements (see **Setting Colors for Screen Elements**).

**Advanced Color Coding Configuration**

The **vslick.vlx** file defines language-specific coloring support. For information about modifying this file, and how to create a new lexer name, see **VLX File and Color Coding**.
This chapter describes SlickEdit Core Context Tagging® system used for symbol analysis, the features that use Context Tagging, and how to manage tag files.
Context Tagging Features

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

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

List Members

When typing a member access operator (Dot, Comma, ":", and ":=" for C++; Dot for Java; IN and OF for COBOL), members are automatically listed. You can access this feature on demand by pressing 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 for 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 **Window → SlickEdit Preferences → Languages**, expand your language category and language, then select **Context Tagging**.

2. Clear the **Auto-list members** check box.

The following example shows the results of what is displayed after typing a **Dot** when entering Java source. Notice that the Javadoc comments are displayed in a mini-HTML browser. To view documentation for Java APIs, you must install the source files as part of the JDK. If clicking on a URL, the default HTML browser starts. Clicking on other hypertext links navigates within the comment window. The equals method in the example below has two occurrences, one in the String class and one in the Object class. Press **Ctrl+PgDn** or **Ctrl+PgUp** to select the next or previous occurrence.

The example below shows the display after typing a **Dot** when entering C++ source code. The stack class is one of the C++ standard template library classes.

![C++ stack class example](image)

**Parameter Information**

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

The following example shows the result of pressing **Alt+Comma** inside the argument list of the Java API String method **startsWith**. The Javadoc comments are displayed in a mini-HTML browser. To view documentation for Java APIs, you must install the source files as part of the JDK. If clicking on a URL, the default HTML browser starts. Clicking on other hypertext links will navigate within the comment window. The **startsWith** method has two overloads that accept different arguments. Press **Ctrl+PgDn** or **Ctrl+PgUp** to select the next or previous occurrence.
Auto List Compatible Parameters

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

```c
hwnd=CreateWindowEx(0,
    szAppName,
    "The Hello Program",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
    NULL,
    HWND WINAPI IsolationAwareCreateWindowEx(...__n int X, __n int Y, __n int nWidth, __n int nHeight,
      __in_opt HWND hWndParent, __n_opt HMENU hWndMenu, __n_opt HINSTANCE hInstance, __n_opt LPVOID lpParam)
```

Auto List Compatible Parameters

When typing a function operator such as the open parenthesis, "(", a list of compatible variables and expressions for the current argument is displayed. Auto List Compatible Parameters can also be used instead of List Members, in assignment statements (x=<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 Window → SlickEdit Preferences → 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.
Completions

Completions save keystrokes as you are typing code by providing a way to automatically complete partially-typed text. Press Ctrl+Space for SlickEdit® Core 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.

Symbol Browsing

SlickEdit® Core gives you the ability to browse and view symbols in your files or workspaces. There are several views and dialogs that display information as you work to help you find what you need exactly when you need it:

• **Class** - This view 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 view is docked as a tab on the left side of the editor by default.

• **Outline** - The Outline view provides an outline view of symbols in the current workspace.

• **Find Symbol** - This dialog 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 view 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 view 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.

For more detailed information about these views and how SlickEdit Core can help you browse symbols, see Symbol Browsing. For information about how to navigate between symbols in files, see Symbol Navigation.

**Statement Level Tagging**

Statement Level Tagging is a feature of Context Tagging that provides a more detailed view of items in the Defs view for C/C++, Java, Python, and Visual Basic .NET. Along with definitions, view constructs like if, while, and for statements. It also displays every non-comment line of code. To see this feature in action, from the Outline View, right-click and select Show Statements.
Building and Managing Tag Files

Context Tagging® creates tag files to store information about symbols and, optionally, cross-reference information from your source code. Many of the most powerful features of SlickEdit® Core use this information to speed your coding.

Tag File Categories

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

- **Project tag files** contain the symbols in the files that are part of your projects. Your project tag files are listed under “Workspace Tag Files”. This will change when you switch workspaces in SlickEdit Core.

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

- **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".

- **Auto-Updated tag files** are designed to be shared by multiple users of SlickEdit® Core on a network. You can use the vsmktags utility to rebuild these tag files as part of your nightly build process. When SlickEdit Core 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 “Auto-Updated Tag Files”.

Building Tag Files

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

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 either of the following:
Building Tag Files

- Use the Projects view - right-click on the root node and select Retag Workspace.

- Use the Context Tagging - Tag Files dialog - select Tools → Tag Files from the main menu. Select the tag file listed under Workspace Tag File and click the Rebuild Tag File button.

Creating Tag Files for Compiler-Specific Libraries

The Tag Compiler Libraries dialog is used to tag libraries associated with the most commonly used languages in SlickEdit Core. This dialog appears as part of the Quick Start Configuration Wizard, when SlickEdit® Core 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).

![Tag Compiler Libraries Dialog](image)
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 Core 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**

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.

**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 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.

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 this workspace and not in libraries. If *Generate References* is not checked, you will still be able to jump from a symbol to its definition in a library using Ctrl+Dot (push_tag).
This option is off by default since most programmers do not want to see library functions shown in the reference list.

- **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. If you are tagging the Boost library, you would name the file "Boost". 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 (colons on UNIX, e.g. `*.c;*.cpp;*.h` for Windows, `*.c:* .cpp:* .h` for UNIX). 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 wildcards. To specify multiple patterns, separate them with semicolons (colons on UNIX). No files are searched in a path that is excluded, including any files in sub-directories beneath.

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

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

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#, InstallShield, JavaScript, Perl, PV-WAVE, Slick-C®, Tornado, TCL, and Visual Basic .NET, and usually it is not necessary to build tag files for the run-times of these languages. If you already built a tag file for run-times during installation, you can skip this section. If you are using Perl, Python, or TCL, and the compiler cannot be found in PATH (or registry for Windows), you need to build tag files for these run-time libraries.

Managing Tag Files

The Context Tagging - Tag Files Dialog (Tools → Tag Files) is used to manage your tag files.

The left pane of the dialog lists all of your tag files, separated into categories (see Tag File Categories below). A tag file having a File 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

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. Project tag files, providing it contains other C/C++ source files.
2. Auto-updated tag files containing other C/C++ source files.
3. The "C" Compiler Configuration tag file corresponding to your default C compiler configuration as specified in your project (see C/C++ Compiler Settings), or global default.
4. 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. Project tag files, providing it contains other Java source files.
2. Auto-updated tag files, containing other Java source files.
3. Language-specific Java tag files, in the order that they are listed in the Context Tagging - Tag Files Dialog.

Rebuilding Tag Files

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.
The following settings are available:

- **Retag modified files only** - If checked, SlickEdit® Core will incrementally rebuild the tag file, only retagging files that have been modified since the last time they were tagged. If not checked, SlickEdit Core 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 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 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.

**Workspace Tagging Excludes**

SlickEdit Core will automatically tag all source files in your workspace. The Workspace Tagging Excludes dialog allows you to specify absolute paths or partial path components in your workspace, which you want to be excluded from automatically being tagged. This feature is located at **Window → SlickEdit Preferences → Editing → Workspace Tagging Excludes**. For more information see, [Workspace Tagging Excludes](#).

**Context Tagging® Options**

**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 **Window → SlickEdit Preferences → 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 (Window → SlickEdit Preferences → Application Options → Virtual Memory). See Virtual Memory Options for more information.

---

**Language-Specific Context Tagging® Options**

You can activate and deactivate various Context Tagging features on a per-language basis. To access these options, from the main menu, select Window → SlickEdit Preferences → Languages, expand your language category and language, then select Context Tagging®. See Language-Specific Context Tagging® Options for more information.
Editing Features

This chapter describes the editing features of SlickEdit Core that are not specific to a particular language.
Notifications

To help you better understand what's going on in the editor, SlickEdit Core 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 Core is performing operations in the background. This helps you correlate high levels of system activity with otherwise invisible operations in the editor.

- **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.

For information on configuring **Notifications**, including a description of the notification levels, see [Notification Options](#).

Feature Notifications

SlickEdit Core 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 View

Regardless of the notification level selected, all feature notifications are added to the **Notification** tool window. To view this tool window, select **Window** → **Show View** → **Other**, then expand **SlickEdit** and select **Notifications**.
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.
Basic Editing

Overview

SlickEdit® Core 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 Core attempts to match these features in each emulation. For example, the Brief emulation provides a **brief_isel ect_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 Core, 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 Core 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 Core and Selections

Many editing operations are performed on selected text, so you need to know how selections work in SlickEdit Core 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 Core handles line selections differently than most other editors. See [Selections](#) for more information.

SlickEdit® Core Clipboards

Most text editing operations involve clipboards. Clipboards in SlickEdit Core 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 Core, by default, keeps a stack of the 50 most recently used clipboards. You can see a list of your clipboards by using the Clipboards view (**Ctrl**+**Shift**+**V**, **Edit** → **List Clipboards** or **list_clipboards** command). See [Clipboards](#) for more information.

Insert/Replace Editing Mode

By default, SlickEdit® Core 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 **Window** → **SlickEdit Preferences**, 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 (Window → SlickEdit Preferences → Keyboard → 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® Core 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 (Window → SlickEdit Preferences → Appearance → Colors). See Setting Colors for Screen Elements for more information.

Selection Types

There are three selection types in SlickEdit® Core: 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.

<table>
<thead>
<tr>
<th>Selection Type</th>
<th>Description</th>
<th>Creation Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character selection</td>
<td>This is created when one or more individual characters are selected. This is also known as a &quot;char&quot; selection.</td>
<td>Use the mouse to drag or use the select_char command (F8 or Edit → Select → Char). See also Character Selections.</td>
</tr>
<tr>
<td>Line selection</td>
<td>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</td>
<td>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</td>
</tr>
</tbody>
</table>

---

117
### Selections

<table>
<thead>
<tr>
<th>Selection Type</th>
<th>Description</th>
<th>Creation Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>selection type depends on how the selection was created.</td>
<td>(when the mouse pointer changes to point right). See also Line Selections.</td>
</tr>
<tr>
<td>Block selection</td>
<td>This is created when columns of text are selected, also known as a &quot;column selection&quot;.</td>
<td>Right-click and drag or use the select_block command (Ctrl+B or Edit → Select → Block). See also Block Selections.</td>
</tr>
</tbody>
</table>

### 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
---|---
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+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® Core 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 (Window → SlickEdit Preferences → 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, including the function heading, 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.

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® Core 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 (Window → SlickEdit Preferences → Editing → Selections). See Selection Options for more information.

**Selection Indicator**

SlickEdit® Core 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 (+) to indicate if there are "extra" characters selected, or a minus sign (-) to indicate if there are fewer characters selected, depending on the start and end columns of the selection. For example, if the selection spans one entire line and part of the subsequent line, the indicator displays "1 Line+".

- 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 Core 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® Core 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.

<table>
<thead>
<tr>
<th>Selection Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Numbers in Selection</td>
<td>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.</td>
<td>Use the <code>add</code> command.</td>
</tr>
<tr>
<td>Add Numbers to Selection (Enumeration)</td>
<td>Automatically adds incrementing numbers to a selection of code.</td>
<td>Use the <code>enumerate</code> command to auto-add numbers or use the <code>gui enumerate</code> command (Edit → Other → Enumerate) to display the Enumerate Dialog, where you can specify options.</td>
</tr>
<tr>
<td>Align Block Selection Center</td>
<td>Centers text in a block selection within the selected area.</td>
<td>Use the <code>align_selection_center</code> command.</td>
</tr>
<tr>
<td>Align Block Selection Left</td>
<td>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.</td>
<td>Use the <code>align_selection_left</code> command.</td>
</tr>
<tr>
<td>Align Block Selection Right</td>
<td>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.</td>
<td>Use the <code>align_selection_right</code> command.</td>
</tr>
<tr>
<td>Append Selection to Clipboard</td>
<td>Appends selected text to the clipboard.</td>
<td>Use the <code>append_to_clipboard</code> command ( Ctrl+Shift+C or Edit → Append to Clipboard )</td>
</tr>
<tr>
<td>Selection Operation</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Beautify Selection</td>
<td>Beautifies the selected text according to the beautification settings for the current language. See Beautifying Code for more information.</td>
<td>Use the <code>beautify_selection</code> command, or check the Restrict to selection box on the Beautifier dialog (Format → Beautify).</td>
</tr>
<tr>
<td>Cancel Selection</td>
<td>Cancels the selection.</td>
<td>Use the <code>deselect</code> command or press Ctrl+U.</td>
</tr>
<tr>
<td>Casing: Lowercase Selection</td>
<td>Translates characters within a selection to lowercase letters. See Case and Capitalization of Text for more casing options.</td>
<td>Use the <code>lowcase_selection</code> command (Ctrl+Shift+L or Edit → Other → Lowcase).</td>
</tr>
<tr>
<td>Casing: Toggle Selection Casing</td>
<td>Toggles the characters within a selection between lowercase and uppercase.</td>
<td>Use the <code>togglecase_selection</code> command.</td>
</tr>
<tr>
<td>Casing: Uppercase Selection</td>
<td>Translates characters within a selection to lowercase letters. See Case and Capitalization of Text for more casing options.</td>
<td>Use the <code>upcase_selection</code> command (Ctrl+Shift+U or Edit → Other → Upcase).</td>
</tr>
<tr>
<td>Copy Selection by Dragging</td>
<td>Drag/copies selected text.</td>
<td>Press and hold Ctrl while clicking inside a selection and then dragging with the mouse to the desired location (Ctrl+LButtonDn).</td>
</tr>
<tr>
<td>Copy Selection to Clipboard</td>
<td>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).</td>
<td>Use the <code>copy_to_clipboard</code> command (Ctrl+C or Edit → Copy).</td>
</tr>
<tr>
<td>Copy Selection to Cursor</td>
<td>Copies 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 (Window → SlickEdit Preferences → Editing →)</td>
<td>Use the <code>copy_to_cursor</code> command or press Ctrl+Shift while holding the right mouse button (Ctrl+Shift+RButtonDn).</td>
</tr>
<tr>
<td>Selection Operation</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Cut Selection</strong></td>
<td>Deletes a selection and copies it to the clipboard.</td>
<td>Use the cut command (Ctrl+X or Edit → Cut).</td>
</tr>
<tr>
<td><strong>Delete and Append to Clipboard</strong></td>
<td>Deletes a selection and appends it to the clipboard.</td>
<td>Use the append_cut command (Ctrl+Shift+X or Edit → Append Cut).</td>
</tr>
<tr>
<td><strong>Execute Selection</strong></td>
<td>Executes each line or sub-line of a selection as if entered on the command line.</td>
<td>Use the execute_selection command or press Alt+=.</td>
</tr>
<tr>
<td><strong>Files: Append Selection to File</strong></td>
<td>Appends selected text to the specified file.</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Files: Write Selection to File</strong></td>
<td>Writes selected text to the specified file.</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Fill Selection</strong></td>
<td>Fills a selection with the specified key character.</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Hide Selection</strong></td>
<td>Hides all lines in a selection by collapsing as a Selective Display unit.</td>
<td>Use the hide_selection command (Display → Hide Selection). Use the show_all command (Display → Show All) to redisplay lines.</td>
</tr>
<tr>
<td><strong>Indenting: Indent Selection</strong></td>
<td>Indents 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</td>
<td>Use the indent_selection command (Tab or Edit → Indent).</td>
</tr>
<tr>
<td>Selection Operation</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Selection Operation</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>Indenting: Unindent Selection</td>
<td>Uninds the selected text according to the Syntax Indent settings or by one tab stop, depending on the <strong>Indent with tabs</strong> 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.</td>
<td>Use the <code>unindent_selection</code> command (<strong>Shift+Tab</strong> or <strong>Edit → Unindent</strong>).</td>
</tr>
<tr>
<td>List Clipboards</td>
<td>Allows you to view and insert a clipboard. See <a href="#">Clipboards</a> for more information.</td>
<td>Use the <code>list_clipboards</code> command (<strong>Ctrl+Shift+V</strong> or <strong>Edit → List Clipboards</strong>).</td>
</tr>
<tr>
<td>Move Selection by Dragging</td>
<td>Drag/moves selected text.</td>
<td>Press and hold the left mouse button while clicking on a selection and then dragging with the mouse to the new location.</td>
</tr>
<tr>
<td>Overlay Block Selection</td>
<td>Overlays a block selection at the current cursor location.</td>
<td>Use the <code>overlay_block_selection</code> command (<strong>Edit → Other → Overlay Block</strong>).</td>
</tr>
<tr>
<td>Overlay/Adjust Block Selection</td>
<td>Overlays a block selection at the current cursor location, and fills the source selection with blanks.</td>
<td>Use the <code>adjust_block_selection</code> command (<strong>Edit → Other → Adjust Block</strong>).</td>
</tr>
<tr>
<td>Paste</td>
<td>Inserts the most recent clipboard at the current cursor location. To insert another clipboard, see the List Clipboards operation or <a href="#">Clipboards</a>.</td>
<td>Use the <code>paste</code> command (<strong>Ctrl+V</strong> or <strong>Edit → Paste</strong>).</td>
</tr>
<tr>
<td>Reflow Selection</td>
<td>Reflows text within a selection according to the margin settings</td>
<td>Use the <code>reflow_selection</code> command (<strong>Format → Format</strong>).</td>
</tr>
<tr>
<td>Selection Operation</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Selection</td>
<td>specified on the Word Wrap option screen (see Language-Specific Word Wrap Options). Block selections are wrapped within the columns of the block. Char selections are not supported for this operation.</td>
<td>Selection.</td>
</tr>
<tr>
<td>Reverse Selection</td>
<td>Reverses the characters in a selection.</td>
<td>Use the reverse_selection command.</td>
</tr>
<tr>
<td>Shift Text in Selection Left</td>
<td>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.</td>
<td>Use the shift_selection_left command (Shift+F7 or Edit → Other → M2 Left).</td>
</tr>
<tr>
<td>Shift Text in Selection Right</td>
<td>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.</td>
<td>Use the shift_selection_right command (Shift+F8 or Edit → Other → M2 Right).</td>
</tr>
<tr>
<td>Sort Lines Within Selection</td>
<td>Sorts lines in a selected area in ascending order.</td>
<td>Use the sort_within_selection command or select the Sort within selection option on the Sort dialog (Format → Sort).</td>
</tr>
<tr>
<td>Sort Selected Lines</td>
<td>Sorts lines in a selected area in ascending order, by comparing only the first columns.</td>
<td>Use the sort_on_selection command or select the Sort on selection option on the Sort dialog (Format → Sort).</td>
</tr>
<tr>
<td>Spell Check Selection</td>
<td>Checks the spelling of selected text according to the Spell Option settings. See Spell Checking for more information.</td>
<td>Use the spell_check_selection command (Format → Spell Check → Check Selection).</td>
</tr>
</tbody>
</table>
Cut, Copy, Paste, and Move

The main menu item Edit provides access to commonly used editing features. 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® Core uses to recognize words, on a per-language basis. To do this, use the Word chars option on the Language-Specific General Options screen (Window → SlickEdit Preferences → 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.

<table>
<thead>
<tr>
<th>Cut Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Deletes the selection and copies it to the clipboard.</td>
<td>Use the cut command (Ctrl+X or Edit → Cut).</td>
</tr>
<tr>
<td>Append Cut</td>
<td>Deletes the selection and appends it to the clipboard.</td>
<td>Use the append_cut command (Ctrl+Shift+X or Edit → Append Cut).</td>
</tr>
<tr>
<td>Cut Word</td>
<td>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.</td>
<td>Use the cut_word command (Ctrl+Shift+k or Edit → Delete → Word).</td>
</tr>
<tr>
<td>Cut Line</td>
<td>Deletes the current line and copies it to the clipboard. Invoking this operation from the keyboard multiple times in succession creates one clipboard.</td>
<td>Use the cut_line command (Ctrl+Backspace or Edit → Delete → Line).</td>
</tr>
</tbody>
</table>
### Cut Operation

<table>
<thead>
<tr>
<th>Cut Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut to End of Line</td>
<td>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.</td>
<td>Use the <code>cut_end_line</code> command (Ctrl+E or Edit → Delete → To End of Line).</td>
</tr>
<tr>
<td>Cut Code Block</td>
<td>Prompts to delete the current code block statement and copies the lines to the clipboard.</td>
<td>Use the <code>cut_code_block</code> command or press Ctrl+Del.</td>
</tr>
</tbody>
</table>

### Copying Text

The table below shows some common Copy operations.

<table>
<thead>
<tr>
<th>Copy Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy to Clipboard</td>
<td>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).</td>
<td>Use the <code>copy_to_clipboard</code> command (Ctrl+C or Edit → Copy).</td>
</tr>
<tr>
<td>Copy Word</td>
<td>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.</td>
<td>Use the <code>copy_word</code> command (Ctrl+K or Edit → Copy Word).</td>
</tr>
<tr>
<td>Copy to Cursor</td>
<td>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 (Window → SlickEdit Preferences → Editing → General).</td>
<td>Use the <code>copy_to_cursor</code> command or press Ctrl+Shift while holding the right mouse button (Ctrl+Shift+RbuttonDn).</td>
</tr>
</tbody>
</table>
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 (`Display → 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 (Window → SlickEdit Preferences → 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 view (Edit → List Clipboards), or use the `list_clipboards` command to display the Select Text to Paste dialog. Both the view and dialog show a list of your clipboards and let you select the clipboard to insert. The only difference is that the view 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® Core allows this capability by default. To disable it, from the main menu, click Window → SlickEdit Preferences, expand Editing and click General, then set the Allow drag drop text option to False.

### Clipboards

Use SlickEdit® Core clipboards to copy and move text in files, the SlickEdit Core command line, dialog text boxes, or any other application that supports text clipboards, such as a word processor. Clipboards in SlickEdit Core are internal to the editor and separate from the system clipboard provided by the operating
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 view, or by using the modal Select Text to Paste dialog. Both provide the same information, except the Clipboards view is dockable and contains a color-coded Preview area for previewing clipboard contents.

• To display the Clipboards view, 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 view 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 (Window → SlickEdit Preferences → 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 view 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 view 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 view, 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 view, the View menu item on the right-click context menu:
  - Auto - When this is selected, the Clipboards view 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® Core 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 view (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® Core 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 Window → SlickEdit Preferences, expand Editing and click General, then enter a value in the Maximum clipboards box.

**Other Operations**

**Inserting Lines**

SlickEdit® Core provides several ways to start a new line or split a line, as described in the table below.

<table>
<thead>
<tr>
<th>Line Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split Line at Cursor</td>
<td>Splits the line at the cursor and appends enough blanks at the</td>
<td>Use the <code>split_insert_line</code> command or press Enter.</td>
</tr>
</tbody>
</table>
### Other Operations

<table>
<thead>
<tr>
<th>Line Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>beginning of the new line to align it with the first non-blank character of the original line.</td>
<td>Use the nosplit_insert_line command or press Ctrl+Enter.</td>
<td></td>
</tr>
<tr>
<td>No Split Insert Line (After)</td>
<td>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.</td>
<td>Use the nosplit_insert_line_above command or press Ctrl+Shift+Enter.</td>
</tr>
<tr>
<td>No Split Insert Line (Before)</td>
<td>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.</td>
<td>Use the nosplit_insert_line_before command or press Ctrl+Shift+Enter.</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Casing Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase Selection</td>
<td>Translates characters within a selection to lowercase letters.</td>
<td>Use the lowercase_selection command (Ctrl+Shift+L or Edit → Other → Lowcase).</td>
</tr>
<tr>
<td>Lowercase Word</td>
<td>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.</td>
<td>Use the lowercase_word command.</td>
</tr>
<tr>
<td>Uppercase Selection</td>
<td>Translates characters within a selection to lowercase letters.</td>
<td>Use the uppercase_selection command (Ctrl+Shift+U or Edit → Other → Uppcase).</td>
</tr>
<tr>
<td>Uppercase Word</td>
<td>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 characters.</td>
<td>Use the uppercase_word command.</td>
</tr>
</tbody>
</table>
### Casing Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle Casing</td>
<td>Toggles the case of letters within a selection.</td>
<td>Use the <code>togglecase_selection</code> command.</td>
</tr>
<tr>
<td>Uppercase Mode</td>
<td>Toggles Uppercase mode on and off. Uppercase mode means that letters you type are automatically upperscased, 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).</td>
<td>Use the <code>caps</code> command.</td>
</tr>
<tr>
<td>Capitalize Selection</td>
<td>Capitalizes the first letter of each word in a selection.</td>
<td>Use the <code>cap_selection</code> command (Ctrl+Shift+A or Edit → Other → Capitalize).</td>
</tr>
<tr>
<td>Capitalize Word</td>
<td>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.</td>
<td>Use the <code>cap_word</code> command.</td>
</tr>
</tbody>
</table>

### 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 Display → Hex or Display → 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.

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® Core 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.

<table>
<thead>
<tr>
<th>Hex/Text View Operation</th>
<th>Key Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Byte to Left of Cursor and Move Cursor Left</td>
<td>Backspace</td>
</tr>
<tr>
<td>Delete Byte Under Cursor</td>
<td>Delete</td>
</tr>
<tr>
<td>Move Cursor to Beginning of Hex Line</td>
<td>Home</td>
</tr>
<tr>
<td>Move Cursor to Last Character of Hex Line</td>
<td>End</td>
</tr>
<tr>
<td>Toggle Cursor Between Hex Data on Left, Text Data on Right</td>
<td>Tab and Shift+Tab</td>
</tr>
</tbody>
</table>
Navigation

There are two types of navigation in SlickEdit® Core: 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 Core 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 Core's code navigation features, you aren't getting the most out of the editor.

Symbol Navigation

Symbol Navigation allows you to jump from a symbol to its definition or to a reference with a single keystroke. A pushed bookmark is set, allowing you to return to the symbol with another keystroke. You can chain a series of these navigation operations together, creating a stack of locations. Then pop your way back to the starting location.

To navigate between symbols use the following operations:

- **Go to Definition** - To quickly move the cursor from a symbol to its definition, pushing a bookmark in the process, press Ctrl+Dot. Alternatively, click Navigate → Go to Definition or use the push_tag command.

- **Go to Reference** - To create a list of references and optionally jump to the first one, pushing a bookmark in the process, press Ctrl+/ . Alternatively, click Navigate → Go to Reference or use the push_ref command.

- **Pop Bookmark** - To pop the bookmark and return to the previous location, press Ctrl+Comma. Alternatively, click Search → Pop Bookmark or use the pop_bookmark command. See Pushed Bookmarks for more information about working with bookmarks.

When you first call these operations, if a tag file does not exist for the current file, it will be built (see Building Tag Files).

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. You can tell SlickEdit® Core 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 Window → SlickEdit Preferences → Languages, expand your language category and language, then select Context Tagging®. 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 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 Window → SlickEdit Preferences, 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 (Edit → Find Next or find_next command). To go to the previous occurrence, press Ctrl+Shift+G (Edit → Find Previous 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 View

The Find Symbol View (Search → Find Symbol or gui_push_tag command) is used to locate symbols (tags) which are declared or defined in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match. See Find Symbol View for descriptions of the options that are available.

More Symbol Navigation Methods

There are several other methods for navigating to symbols:

• The Symbols View shows the symbols for all tag files. Right-click in the view 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 view.

• At the SlickEdit® Core 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 (Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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 Navigate → Go to Matching Parenthesis). The find_matching_paren command supports matching parenthesis pairs { }, [ ] and ( ).

**Tip**

For Python, SlickEdit® Core 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 Window → SlickEdit Preferences → 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 (Window → SlickEdit Preferences → 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 **Window → SlickEdit Preferences**, 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 **Window → SlickEdit Preferences → 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 view.
- **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 Core 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 (\textit{Ctrl} + \textit{Alt} + \textit{Left}).
- **next_sexp** - Moves to the next S-expression (\textit{Ctrl} + \textit{Alt} + \textit{Right}).
- **backward_up_sexp** - Navigates to the start of the immediately enclosing block (\textit{Ctrl} + \textit{Alt} + \textit{Up}).
- **forward_down_sexp** - Drills down into the next block (\textit{Ctrl} + \textit{Alt} + \textit{Down}).
- **select_prev_sexp** - Extends a character selection from the cursor to the start of the previous S-expression (\textit{Ctrl} + \textit{Alt} + \textit{Shift} + \textit{Left}).
- **select_next_sexp** - Extends a character selection from the cursor to the start of the next S-expression (\textit{Ctrl} + \textit{Alt} + \textit{Shift} + \textit{Right}).
- **cut_prev_sexp** - Deletes the S-expression to the left of the cursor and copies it to the clipboard (\textit{Ctrl} + \textit{Alt} + \textit{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).

**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 \texttt{Cursor right/left wraps to next/previous line} (Window → SlickEdit Preferences → 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 \texttt{Cursor right/left wraps to next/previous line} (Window → SlickEdit Preferences → 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 \texttt{Cursor up/down places cursor in virtual space} (Window → SlickEdit Preferences → 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 (Window → SlickEdit Preferences → 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 (Window → SlickEdit Preferences → 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 **Navigate → Go to Line**. Enter the line number and click **OK**. Alternatively, you can use the **goto_line** command in the syntax **goto_line linenum**.

**Navigating to an Offset**

To seek to a byte offset in the current buffer, from the main menu click **Navigate → Go to Offset**, or use the **gui_seek** command. This function is the same as the C **lseek** function. However, if you have opened the file with tab expansion, the seek position on disk may be different.

When the Seek dialog appears, enter the position to seek for. You may specify a C syntax expression. In addition, you may prefix the expression with a plus or minus sign (+ or -) to specify a relative seek position.

Some examples are:

- **0x10+10** - Seek to offset 26
- **+8+4** - Seek to current offset + 12
- **-8+4** - Seek to current offset - 12
Select the **Decimal** option to enter the seek position in decimal number format. Select the **Hex** option to enter the seek position in hexadecimal number format. You can type an "x" as the first character in the **Position to seek for** text box and this option will automatically be selected.

## Navigating to URLs

SlickEdit® Core 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 Core.

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 **Window** → **SlickEdit Preferences**, 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 **Window** → **SlickEdit Preferences** → **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.
Symbol Browsing

- **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® Core 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® Core 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 views that display information as you work to help you find what you need at exactly the time you need it:

- **Class View**
- **Outline View**
- **Find Symbol View**
- **Preview View**
- **References View**
- **Symbols View**
- **Symbol Properties View**

See also [Symbol Navigation](#) for information about how to navigate between symbols in files.

**Class View**

The Class view, 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 view also shows the inheritance hierarchy of the current class. This is useful for object-oriented programming languages such as Java.

Display of the Class view can be toggled on/off by clicking **Window → Show View → Other** , then expand SlickEdit and select **Class** or by using the **toggle_tclass** command. To display the view on demand, use the **activate_tclass** command.
If you are coding within a class, the top pane (hierarchy pane) of the view 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 view.
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 view.

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 view. 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 only) Organizes import statements in Java files. See [Organize Imports](#) for more information.

- **Go to Tag** - Moves the cursor to the selected tag. See [Symbol Navigation](#) for more information.

- **References** - Brings the References view into focus, displaying the references for the symbol. See [References View](#) 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 View](#) for more information.
• **Increase/Decrease Listed Members Limit** - Controls the number of members displayed in the members pane. When this option is selected, the command line will prompt you for a variable value. The default is 400.

• **Sort Classes By Hierarchy** and **Sort Classes By Name** - These options toggle the display of classes sorted either by hierarchy or alphabetically by name.

• **Sort Members By Line Number** and **Sort Members By Name** - These options toggle the display of members sorted either by line number or alphabetically by name.

• **Organize Members By Class** - Groups the members in the members pane by their class (or interface). When this option is selected, all "Sort" options are available. When this option is not selected, visible members in this pane will not be grouped at all. They will instead be displayed in one list, sorted by name.

• **Auto Expand All Top Level Classes** - Expands all top level class nodes in the members pane whenever the current class changes. The default behavior is to only auto-expand the node of the current class.

• **Auto Expand All Structs/ Enums/ Inner Classes** - Expands all struct, enum, and inner class nodes displayed in the members pane whenever the content is refreshed. By default this option is turned off, and these nodes are collapsed.

• **Quick Filters** and **Scope Filters** - Quick filters allow you to display only certain items in the members pane, such as functions, prototypes, etc. Scope filters allow you to display members only in certain scopes, such as public or global, private, protected, etc.

---

**Outline View**

The Outline View contains the defs (definitions) browser, which provides an outline view of symbols in the current file.

By default, the Outline View is docked as a tab on the right side of the editor. Display can be toggled on/off by clicking **Window → Show View → Outline** or by using the **eclipse_show_outline** command.
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.

Outline View Options

Right-click on any symbol in the Outline View 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 Nesting** - Organizes symbols by their scope within the current file. Clear this option to display all of the symbols in one flat list.
- **Show Statements** - (C/C++, Java, Visual Basic only) This option controls the Statement Level Tagging feature. When selected, the view shows an outline of all statements in each function within the current file. This allows you to see a primitive function flowchart or to navigate to a specific statement within a function.
- **Display Files (disabled)** - Displays the names of the files that are open in the editor. Clear this option to only show symbols in the current file, allowing you to use the window as a true outline view.
- **Auto Expand** - Automatically expands all levels within the current file. If this option is cleared, you will need to click to expand items manually.
- **Expand All** - Expands all symbols or levels in the current file.
- **Expand 1 Level** - Expands everything one level below the current symbol.
- **Expand 2 Levels** - Expands everything two levels below the current symbol.
- **Display Non-taggable Files** - Displays files that are open in the editor that are not taggable, such as text files.
- **Properties** - Displays the Symbol Properties View, showing the properties of the selected item, such as visibility, whether it's static or final, etc. Note that you cannot use this window to change the properties.
- **Arguments** - Displays the return type and arguments for functions/methods in the Symbol Properties View.
- **References** - Displays the list of references for the selected symbol, just as if you pressed Ctrl+/ in the editor window. See Symbol Navigation for more information.
- **Show Call Tree** - Displays a tree of symbols used by the selected symbol, for example, other functions called by the current function. See Viewing Symbol Uses with the Calling Tree for more information.
Find Symbol View

- **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 Outline view.

**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 View

The Find Symbol view (Search → Find Symbol or gui_push_tag command) is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, Substring, or fast prefix match.

Searching for a symbol is faster than a normal text search because it is executed against the Context Tagging® database, rather than searching through your source files. Find Symbol also avoids false hits in comments or string literals. Though Syntax-Driven Searching in the regular Search Dialogs and Views provides this same capability, it cannot match the speed of Find Symbol.

See Find Symbol View for information about the options that are available on the view.

Preview View

The Preview view 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 Window → Show View → Other, then expand SlickEdit and select Preview or by using the toggle_preview command. To display the view on demand, use the activate_preview command.
The Preview view 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.
Preview View

• **Editor preview window** - Shows the contents of the actual source file at the line number of the selected symbol. Double-click to open the code in the editor. Right-click to adjust symbol search filtering options.

• **Size bars** - Use the size bars to adjust the width of the symbol list and/or the height of the documentation comments area.

• **Buttons** - The following buttons are found along the right edge of the Preview window:
  
  - **Back and Forward** - Allow you to navigate among the hypertext links that you have traversed in the documentation comments.
  
  - **Go to definition** - Opens the selected symbol in the editor.
  
  - **Go to reference** - Finds references to the selected symbol.
  
  - **Show in symbol browser** - Locates the selected symbol in the Symbols View.
  
  - **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.

<table>
<thead>
<tr>
<th>Editor Element in Use</th>
<th>Preview Window Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any source file open in the editor</td>
<td>The Preview window shows the definition or declaration of the symbol under the cursor, along with the symbol's documentation comments, if any exist.</td>
</tr>
<tr>
<td>The Defs, Symbols, Class, Current Context, and Find Symbol views</td>
<td>Single-click on a symbol and the Preview window displays the selected symbol and its documentation comments, if any exist. See Outline View, Symbols View, Class View, and Find Symbol View for more information.</td>
</tr>
<tr>
<td>Call Tree dialog and References view</td>
<td>The Preview window shows the location of the symbol references or use.</td>
</tr>
<tr>
<td>The Base Classes and Derived Classes symbol browser dialogs</td>
<td>Single-click on a symbol and the Preview window displays the selected symbol and its documentation comments, if any exist. See Symbols View for more information.</td>
</tr>
<tr>
<td>The Search Results view</td>
<td>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</td>
</tr>
</tbody>
</table>
References View

The References view 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 Window → Show View → Other, then expand SlickEdit and select References or by using the toggle_refs command. To display the view on demand, use the activate_refs command.

The References view automatically comes into focus when you use the Go to Reference feature or when you select References from the right-click menu of the Class, Defs, or Symbols view.

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 view 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 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 view to toggle the preview pane on and off. Because source can also be previewed in the **Preview View**, you may find it more efficient to use the References window with the preview pane off.

**References View Options**

Right-click on a symbol or file in the left pane of the References window to display the following options:

- **Contents** - Displays the following menu of save and print operations for the references browser tree:
  - **Save** - Writes the items displayed in the references browser to a text file, prompting you for a file name and directory location. The text file will then be displayed in the editor.
  - **Print** - Displays the Print dialog, where you can configure options for printing the tree.
  - **Save Subtree** and **Print Subtree** - These options function similarly to the above except they apply to the selected subtree.
  - **Quick filters, Scope, Functions, Variables, Data Types, Statements, and Others** - All of these items are for filtering the data displayed in the References view.

**Symbols View**

The Symbols view contains the symbol browser, which lists symbols from all of the tag files.
By default, the Symbols view is docked as a tab on the left side of the editor. Display can be toggled on/off by clicking Window → Show View → Other, then expand SlickEdit and select Symbols or by using the toggle_symbols command. To display the view on demand, use the activate_symbols command.

The top part of the window contains an option and combo boxes that are used for filtering. The bottom part of the window lists the symbols grouped by category. Symbols in your workspace are listed in the top group labeled "Workspace." The rest of the symbols are grouped by language or compiler.

Hover the mouse over the bitmap of a symbol to see a 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 View**

The symbols listed in the symbol browser can be filtered using the Class and Member combo boxes. The Class combo box filters the items listed under the Classes folder. The Member combo box filters the items listed under any displayed classes or under any of the other folders, like Global Variables, Static Variables, Defines, etc. Enter multiple words in either combo box to search for items containing either word.

For example:

- Enter person into the Class combo box to find all classes containing the word "person".

- Enter person manager into the Member combo box to find all members, variables, etc. containing the word "person" or "manager".

**Note**
The filters are case-sensitive, so be sure to type the values in the same case.

The items listed under the Classes folder are global classes that are not part of a namespace or package.

To clear the filters and see all items again, select the Show all tags option.

For non-object-oriented languages, use the Member combo box to search, since there are no classes. You can hide the combo boxes to save space by right-clicking and selecting Filters, then unchecking the corresponding check box.

**Symbols View Options**

Right-click on a symbol in the Symbols view to access the following additional filtering options as well as code management options:

- **Go to Definition** - Moves the cursor to the symbol's definition (proc). See Symbol Navigation for more information.

- **Go to Declaration** - Moves the cursor to the symbol's declaration (proto). See Symbol Navigation for more information.

- **Quick Refactoring** - Offers two Quick Refactorings: Rename and Modify Parameter List. See Quick Refactoring for more information.

- **Organize imports** - (Java only) Organizes import statements in Java files. See Organize Imports for more information.

- **Set Breakpoint** - Sets a debugging breakpoint. See Setting Breakpoints for more information.

- **Find Tag** - Searches for symbols and displays them in the symbol browser. Note that the Find Symbol view also provides this functionality.

- **Manage Tag Files** - Displays the Context Tagging - Tag Files Dialog for use in managing your tag files.

- **Expand** and **Collapse** options - Expands/collapses symbols as specified.

- **Sort by** - Sorts symbols displayed by tag name, line number, or containers to top, which puts classes, structs, etc. at the top of the list.

- **Filters** - Filter by class or member, or select Filtering Options to display the Symbol Browser Filter Options dialog. See Symbol Browser Filter Options for information on the available options.

- **Contents** - Displays the following menu of save and print operations for the symbol browser tree:
  
  - **Save** - Writes the items displayed in the symbol browser to a text file, prompting you for a file name and directory location. The text file will then be displayed in the editor.
  
  - **Print** - Displays the Print dialog, where you can configure options for printing the tree.

- **Save Subtree** and **Print Subtree** - These options function similarly to the above except they apply to
the selected subtree.

- **Base Classes** - Displays the Base Classes dialog, which shows a list of base classes for the selected class on the left with the list of that class's members on the right. Base classes are displayed in a tree view, allowing you to explore up the inheritance hierarchy. See Viewing Base and Derived Classes for more information. Note that the Class View 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 View, 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 View.

- **References** - Displays the list of references for the selected symbol in the References View, just as if you pressed Ctrl+/ in the editor window. See Symbol Navigation for more information.

- **Calls/Uses** - Displays a tree of symbols that are used by this symbol or called by this function. See Viewing Symbol Uses with the Calling Tree for more information.

**Viewing Symbol Uses with the Calling Tree**

View symbol uses to see what symbols (variables, functions, methods, classes, etc.) are used by a specific function or method.

To view the symbols that a particular function or method uses, first create a project or open an existing project. Then from the Symbols view, 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 Outline View by right-clicking on a symbol and selecting Show Call Tree.
Right-click in this tree to display/modify the symbol filters. Items in the tree can be expanded to view uses recursively. Double-click or press the spacebar on an item in the tree list to go to an item. Double-click and **Space** are the same except when the item is a prototype that has a corresponding code section. Double-clicking will then go to the prototype's corresponding code section.

If the focus is in the Symbol Uses/Calling Tree dialog, the selected item will be shown in the **Preview View** view, just as it is in the **Symbols View**.

**Viewing Base and Derived Classes**

To see what classes are inherited by a particular class, right-click on the class in the Symbols view and select **Base Classes**. To see what classes are derived from a particular class, right-click on the class in the Symbols view and select **Derived Classes**. Both dialogs have the same interface.
The left pane of each dialog contains a tree showing the class inheritance hierarchy (the class list). The right pane shows a list of the members of the selected class (the member list).

If the focus is in the class list, the selected class will be displayed in the member list, if it can be resolved. If the focus is in the member list, the selected item will be shown in the Preview window, and is the name as it appears within the class definition.

To jump to the symbol in the code, pushing a bookmark in the process, double-click on a symbol in either pane. Press Ctrl+Comma to go back. Right-click on a symbol for filtering options.

**Symbol Browser Filter Options**

To access symbol browser filter options, right-click in the Symbols view 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 View** (right-click in the Symbols view and choose **Arguments**, or from the main menu click **Window → Show View → Other**, then expand SlickEdit and select **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.

- **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 view.
  - **Member Filter** - When selected, the member filter is displayed in the Symbols view.

**Symbol Properties View**

The Symbol Properties view displays detailed information (properties and arguments) for the symbol at the cursor location. Note that this window is read-only, so you can't use it to change the properties.

Display can be toggled on/off by clicking **Window → Show View → Other**, then expand **SlickEdit** and select **Symbol Properties**. To display the view on demand, right-click a symbol in the Symbols view and select **Properties** or **Arguments**, or use the **activate_tag_properties_toolbar** command.
Viewing and Displaying

SlickEdit® Core 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 Core provides comprehensive capabilities to color the text in the editor window. For more information, see Colors, Color Coding, and Symbol Colors.

Current Line

SlickEdit Core 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 (Window → SlickEdit Preferences), 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 Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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 Core 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 Core 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 **Window → SlickEdit Preferences → 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 **Window → SlickEdit Preferences → Appearance → Colors**, then select **Modified Line** or **Inserted Line** under the **Modifications** node. For more information about setting colors, see [Colors](#).

SlickEdit Core can clear the modified and inserted line color when you save a file. To activate this feature, from the main menu, click **Window → SlickEdit Preferences**, 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® Core 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 **Display → 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.

- **For a specific language** - Using the Options dialog (**Window → SlickEdit Preferences**), 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 Display → 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 Window → SlickEdit Preferences, expand Appearance and select Special Characters. Enter the character codes that you wish to use.

To view the differences between a DOS format text file and another format when Display → New Line Chars is active, choose something other than a space for the End-Of-Line (2) character. Under Windows, the recommended choices are 13 for End-Of-Line (1) and 10 for End-Of-Line (2).

**Changing the Color of Special Characters**

To change the colors and styles of special characters, use the Color options screen (Window → SlickEdit Preferences → 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. 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 Display → 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 (Window → SlickEdit Preferences), 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 **Window → SlickEdit Preferences → 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 **Display → Soft Wrap**, or use the `softwrap_toggle` command. This toggles Soft Wrap on and off.

- **For a specific language** - Using the Options dialog (**Window → SlickEdit Preferences**), 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:
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 Display → Selective Display, or by using the selective_display command. For more information, see View Dialogs and Views. 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.

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 Display → Expand/Collapse Block, press Ctrl+\, or use the plusminus command. See Expanding/Collapsing Code Blocks for more details.

- **Copy visible text to the clipboard** - Click Display → 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 Display → Show All (show_all command).

To define the type of information to show/hide, see [Selective Display Regions](#).

### Expanding/Collapsing Code Blocks

SlickEdit® Core 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 Window → SlickEdit Preferences → 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 View** - 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** - Select this option to set multiple levels of Selective Display based on braces or indent.

• **Paragraphs** - Displays the first line of each paragraph. A paragraph is defined by a group of lines followed by one or more blank lines.

• **Hide Selection** - Select this option to hide the lines in the current selection.

The Selective Display dialog also contains static options for expanding/collapsing sub-levels. See View Dialogs and Views for more information and details about the available settings.

### Hex/Line Hex View

SlickEdit® Core 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 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 **Window → SlickEdit Preferences**, 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® Core 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 Window → SlickEdit Preferences → Languages, expand your language category and language, then select Indent.
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 Window → SlickEdit Preferences → Languages, expand your language category and language, then click Indent. Select the Indent with tabs option.

Setting Tab Spacing

The default value of the Tab key is eight spaces. You can change this value in the Tabs text box. In general, the Tabs setting should match the Syntax indent value. For example, by default for the C language extension, the Syntax indent value is set to 4, and the Tabs setting is set to +4. The plus sign (+) indicates that the editor will automatically expand the stops by four. By default, the Tabs setting is "+4", which indicates that the default tab setting is eight spaces.

To work properly with the Sun Java API source code, the tab stops need to be in increments of eight, but the syntax indent must be set to four. The Syntax Indent affects not only the Tab key, but also the number of spaces to indent for each code block level.

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
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 (Format → 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 Window → SlickEdit Preferences, 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 Window → SlickEdit Preferences. Expand Languages in the tree, select the language category and language, then click **Indent**. 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 Window \(\rightarrow\) SlickEdit Preferences \(\rightarrow\) Application Options \(\rightarrow\) 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 Window \(\rightarrow\) SlickEdit Preferences \(\rightarrow\) Languages \(\rightarrow\) Application Languages \(\rightarrow\) [Language] \(\rightarrow\) 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® Core should scan.

You can also toggle Adaptive Formatting on and off for a language by using the menu item Format → 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.

The individual formatting style settings are located on various screens in the language-specific section of the Options dialog. A label, linked to the Adaptive Formatting options screen, appears next to each setting to indicate if Adaptive Formatting is on or off for that setting. Adaptive Formatting does not change the value set in these options; rather, it overrides the value on a document-by-document basis. For example, notice how Adaptive Formatting is enabled for settings on the C/C++ Indent options screen:

![C/C++ Indent Options Screen]

The values specified by Adaptive Formatting will be used instead of the values shown on the options screen. Click on the Adaptive Formatting hyperlinks to jump to the Adaptive Formatting options screen for that language.

**Recognized Settings**

The table below shows each option recognized by Adaptive Formatting, the path to the option in SlickEdit® Core, 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 Window → SlickEdit Preferences → Languages, expand your language category and then select your language.
<table>
<thead>
<tr>
<th>Formatting Option</th>
<th>Path to Option in SlickEdit Core</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax indent</td>
<td>Formatting (Language-Specific Formatting Options)</td>
<td>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.</td>
</tr>
<tr>
<td>Tabs</td>
<td>Formatting (Language-Specific Formatting Options)</td>
<td>Specifies tab stops, which can be in increments of a specific value or at specific column positions. See Indenting with Tabs for information.</td>
</tr>
<tr>
<td>Indent with tabs</td>
<td>Formatting (Language-Specific Formatting Options)</td>
<td>Determines whether the Tab key, Enter key, and paragraph reformat commands indent with spaces or tabs. See Indenting with Tabs for information.</td>
</tr>
<tr>
<td>Begin/End style</td>
<td>Formatting Options (Language-Specific Formatting Options)</td>
<td>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.</td>
</tr>
<tr>
<td>Indent CASE from SWITCH and Indent constant from case</td>
<td>Formatting Options (Language-Specific Formatting Options)</td>
<td>Specifies Syntax Expansion indentation. See the section for your language in the Language-Specific Editing chapter. For example, Language-Specific Formatting Options.</td>
</tr>
<tr>
<td>No space before parenthesis</td>
<td>Formatting Options (Language-Specific Formatting Options)</td>
<td>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.</td>
</tr>
<tr>
<td>Insert padding between parenthesis</td>
<td>Formatting Options (Language-Specific Formatting Options)</td>
<td>When enabled, a space is placed after the open parenthesis and</td>
</tr>
</tbody>
</table>
### Formatting Option | Path to Option in SlickEdit Core | Description
--- | --- | ---
| | | 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 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® Core 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.
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
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® Core:

- **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.
- **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 Core automatically closes block comments. For example, in C++ when you type "/**
SlickEdit Core automatically inserts ”*/”. This is not part of the Auto-Close feature. To configure
this, go to Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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
Window → SlickEdit Preferences → 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:

```c
if (pWindowView->pBuffer->LineNum>100) {
    pW<Cursor is Here>
}
```

Press **Ctrl+Shift+Comma, Ctrl+Shift+Space, Ctrl+Shift+Space** to obtain the following result:

```c
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.

Configuring Completion Settings

To configure Auto-Complete settings, from the main menu, click **Window → SlickEdit Preferences → 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® Core:

• **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 command. This command will expand the alias only if there are no matching completions. Otherwise, it will show a list of completions.

• If Syntax Expansion is enabled (Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → 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 alias.slk, located in the SlickEdit® Core 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 Window → SlickEdit Preferences → 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 view 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 Core 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 Mac OS X, 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 **Window → SlickEdit Preferences → Languages**, expand your language category and language, then select **Aliases**. As an example, the Java 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.

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.

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%\a</td>
<td>Inserts the authors name.</td>
</tr>
<tr>
<td>%\c</td>
<td>Places the cursor. This sequence can be used multiple times in the same alias value in order to create a series of &quot;hot spots&quot; within the alias. After the alias is expanded, press Ctrl+{ (next_hotspot command) to jump to the next cursor stop.</td>
</tr>
<tr>
<td>%\d</td>
<td>Inserts the date (locale-dependent).</td>
</tr>
<tr>
<td>%\e</td>
<td>Inserts the date in MMDDYY format.</td>
</tr>
<tr>
<td>%\t</td>
<td>Inserts the time (locale-dependent).</td>
</tr>
<tr>
<td>%%</td>
<td>Inserts a percent character.</td>
</tr>
<tr>
<td>%\f</td>
<td>Inserts the current file name.</td>
</tr>
<tr>
<td>%\g</td>
<td>Inserts a file separator character. This is the backslash on Windows () platforms and slash (/) on UNIX/Mac.</td>
</tr>
<tr>
<td>%\w</td>
<td>Outputs the line number.</td>
</tr>
<tr>
<td>%\n</td>
<td>Inserts the current function name.</td>
</tr>
<tr>
<td>Escape Sequence</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>%\o</td>
<td>Inserts the current function name with signature.</td>
</tr>
<tr>
<td>%\j</td>
<td>Inserts the current class name.</td>
</tr>
<tr>
<td>%\j+</td>
<td>Inserts the current class name, fully-qualified.</td>
</tr>
<tr>
<td>%\i</td>
<td>Indents.</td>
</tr>
<tr>
<td>%\b</td>
<td>Unindents.</td>
</tr>
<tr>
<td>%\s</td>
<td>Preserves trailing spaces. This should be placed at the end of a line.</td>
</tr>
<tr>
<td>%\l</td>
<td>Preserves leading spaces.</td>
</tr>
<tr>
<td>%\x ColumnNumber</td>
<td>Moves the cursor to the specified column number.</td>
</tr>
<tr>
<td>%\x+ ddd</td>
<td>Increments column by ddd.</td>
</tr>
<tr>
<td>%\x- ddd</td>
<td>Decrements column by ddd.</td>
</tr>
<tr>
<td>%\m MacroName ArgumentList %</td>
<td>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).</td>
</tr>
<tr>
<td>%\m sur_text%</td>
<td>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.</td>
</tr>
<tr>
<td>% EnvironmentVariable %</td>
<td>Inserts the value of the environment variable specified.</td>
</tr>
<tr>
<td>%( ParameterName )</td>
<td>Parameter Prompting replacement. See Parameter Prompting for more information.</td>
</tr>
<tr>
<td>%\p</td>
<td>Inserts parameters from the function that is located beneath it. See Doc Comments for more information.</td>
</tr>
<tr>
<td>%\q</td>
<td>Insert the type for the parameters (double, integer, string, etc.) from the function that is located beneath</td>
</tr>
<tr>
<td>Escape Sequence</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>%p</td>
<td>it. Typically used in conjunction with the <code>%p</code> escape sequence. See <a href="#">Doc Comments</a> for more information.</td>
</tr>
<tr>
<td>%r</td>
<td>Inserts the return type from the function that is located beneath it. See <a href="#">Doc Comments</a> for more information.</td>
</tr>
<tr>
<td>%u</td>
<td>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 <a href="#">Doc Comments</a> for more information.</td>
</tr>
<tr>
<td>%v</td>
<td>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 <a href="#">Doc Comments</a> for more information.</td>
</tr>
<tr>
<td>%un</td>
<td>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 <a href="#">Doc Comments</a> for more information.</td>
</tr>
<tr>
<td>%vn</td>
<td>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 <a href="#">Doc Comments</a> for more information.</td>
</tr>
<tr>
<td>%()</td>
<td>Used to separate identifier characters. For example, <code>%\u%()n</code> has the effect of the <code>%u</code> option followed by a literal &quot;n&quot;. 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 <code>%\dx</code>. Write <code>%d%()x</code> instead.</td>
</tr>
</tbody>
</table>

### Escape Sequence Examples

The following table contains some examples of using escape sequences in alias values:
<table>
<thead>
<tr>
<th>Alias Name and Description</th>
<th>Value</th>
</tr>
</thead>
</table>
| Sample formatted dates     | %\mprinttime #b. #d, #Y% ==> Apr. 07, 2008  
%\mprinttime #A, #B #d, #Y% ==> Monday, April 07, 2008  
%\mprinttime #m/#d/#Y% ==> 04/07/2008 |
| comment - A header comment to have the date and time inserted. | /* Date: \d Time: \t */  
/*******************************************************/ |
| if - A simple if statement, with indenting, support for surround, and a cursor position. | if(%\c){  
%\i// Comment goes here  
%\i%m sur_text%  
} |
| ifdef - An if/else statement with indenting and several cursor hot spots. | if(%\c){  
%\i%c  
} else {  
%\i%c  
} |
| wmain - A WinMain function template with indenting and a cursor position. | int APIENTRY WinMain(HANDLE hInstance,  
HANDLE hPrevInstance,  
LPSTR lpszCmdParam,  
int nCmdShow)  
{  
%\i%c  
} |
| /** - A Javadoc comment. | /**  
*  
* %\c  
*  
* @author %\a (%\d)  
* %\u  
* @param %\p %\c  
* %\v  
* @return %\c%\v  
*/ |
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.
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+[) to jump the cursor to the next part of the statement. In the case of the for loop above, Ctrl+[ would move the cursor from the group in parentheses to the code block.

The structures loop, if, and switch or case are also expanded. You do not have to type the entire keyword for Syntax Expansion to occur. If there is more than one keyword that matches what you type, a list of possible keyword matches is displayed. To get the C template displayed above, type "f" followed by pressing Space.

To override the insertion of braces immediately for one line if, for, or while statements, type a semicolon immediately after the keyword. For example:

```
if; => if ( <cursor here> ) <next hotspot>;
```

To override non-insertion of braces immediately for if, for, while, foreach, with, lock, fixed, and switch statements, type an open brace immediately after the keyword. For example:

```
if{ => if ( <cursor here> ) { <next hotspot> }
```

If the default behavior of Syntax Expansion does not match your coding style, for most languages, it can be customized. From the main menu, click Window → SlickEdit Preferences → 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 Window → SlickEdit Preferences → Languages, expand your language category and language, then select Indent.

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® Core 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 Window → SlickEdit Preferences → 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:

```plaintext
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

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 Core 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.

```java
/*
 * $copyright$
 */

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® Core 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 → SlickEdit Template Manager**.

**Instantiating a Template**
You can add an item to your current project by clicking **File → Add New Item from SlickEdit 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 SlickEdit Template**. The Add New Item dialog box is shown below.

Instantiating a Template

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® Core 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 Core 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
Use the Template Manager to Create a New Template

Click File → SlickEdit 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 → SlickEdit 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:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$itemname$</td>
<td>Name of item entered, as on the Add New Item dialog.</td>
</tr>
<tr>
<td>$fileinputname$</td>
<td>Name of item entered, as on the Add New Item dialog, without file extension.</td>
</tr>
<tr>
<td>$safeitemname$</td>
<td>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 <strong>My Custom Class</strong>, then the $safeitemname$ would evaluate to My_Custom_Class for a C++ source code file.</td>
</tr>
<tr>
<td>$upcasesafeitemname$</td>
<td>Same as $safeitemname$ with all characters upercased.</td>
</tr>
<tr>
<td>$lowcasesafeitemname$</td>
<td>Same as $safeitemname$ with all characters lowercased.</td>
</tr>
<tr>
<td>$tempdir$</td>
<td>Location of operating system temp directory. No trailing file separator.</td>
</tr>
<tr>
<td>$rootnamespace$</td>
<td>Root namespace or package for the current project. This is typically used for C# and Java projects to find the namespace containing <strong>Main()</strong> (or <strong>main()</strong> in the case of Java).</td>
</tr>
<tr>
<td>$ampmtime$</td>
<td>Time of day in the form <em>hh:mm</em>[am</td>
</tr>
<tr>
<td>$localtime$</td>
<td>Time of day in locale-specific format.</td>
</tr>
<tr>
<td>$time$</td>
<td>Time of day in the form <em>hh:mm:ss</em>.</td>
</tr>
<tr>
<td>$localdate$</td>
<td>Current date in locale-specific format.</td>
</tr>
<tr>
<td>$date$</td>
<td>Current date in the form <em>mm/dd/yyyy</em>.</td>
</tr>
<tr>
<td>$projectname$</td>
<td>Current project name (no path, no extension).</td>
</tr>
<tr>
<td>$safeprojectname$</td>
<td>Current project name (no path, no extension), with all unsafe characters replaced with safe characters.</td>
</tr>
</tbody>
</table>
### 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:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$workspacename$</td>
<td>Current workspace name (no path, no extension).</td>
</tr>
<tr>
<td>$safeworkspacename$</td>
<td>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.</td>
</tr>
<tr>
<td>$projectworkingdir$</td>
<td>Current project working directory. No trailing file separator.</td>
</tr>
<tr>
<td>$projectbuilddir$</td>
<td>Current project build (output) directory. No trailing file separator.</td>
</tr>
<tr>
<td>$projectconfigname$</td>
<td>Current project configuration name.</td>
</tr>
<tr>
<td>$workspaceconfigname$</td>
<td>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.</td>
</tr>
<tr>
<td>$projectdir$</td>
<td>Location of current project file. No trailing file separator.</td>
</tr>
<tr>
<td>$workspacedir$</td>
<td>Location of current workspace file. No trailing file separator.</td>
</tr>
<tr>
<td>$username$</td>
<td>Operating system login name.</td>
</tr>
</tbody>
</table>
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 **File → New Item from SlickEdit Template**.

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 → SlickEdit 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 Core, 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.
Add Parameter 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 **File → New Item from SlickEdit Template** or **File → New Item from SlickEdit 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 current project** - When selected, the new item is added to the current project.

• **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:

```
[EclipseInstallDir]/plugins/com.slickedit.windows.libs_[VERSION]/slickedit/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 Core**.

### Manually Creating a Template

SlickEdit® Core 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 Core.
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 Core.

   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 File → New Item from SlickEdit Template.

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
<?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>

</TemplateDetails>
</SETemplate>
```

213
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 → SlickEdit 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

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:

<table>
<thead>
<tr>
<th>Element</th>
<th>Child Elements</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Description</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>File</td>
<td>-</td>
<td>ReplaceParameters,TargetFilename</td>
</tr>
<tr>
<td>Files</td>
<td>File</td>
<td>-</td>
</tr>
<tr>
<td>Name</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Parameter</td>
<td>-</td>
<td>Name,Value</td>
</tr>
<tr>
<td>Parameters</td>
<td>Parameter</td>
<td>-</td>
</tr>
</tbody>
</table>
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
<?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>
    </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
<?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>
  <TemplateName>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateName>
</SETemplate>
```

File

File is an optional child element of Files. Specifies a file for the template item. See the example below.

- **Attributes**
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplaceParameters</td>
<td>Optional. Specifies whether parameter substitution takes place on the file contents when the file is created from the template. Note that parameter substitution always takes place on the TargetFilename attribute value (example: \texttt{TargetFilename}=&quot;$\texttt{fileinputname$.cpp}$&quot;). Possible values are &quot;1&quot; (true) or &quot;0&quot; (false). Defaults to &quot;1&quot; (true).</td>
</tr>
<tr>
<td>TargetFilename</td>
<td>Optional. Specifies the actual name of the item that is created from the template. This attribute is especially useful when creating a multi-file template where file names of files created from the template are assembled by parameter substitution.</td>
</tr>
</tbody>
</table>

- **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
<?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>
    <!-- Code template content goes here -->
  </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
<?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
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/setemplate.dtd">
<SETemplate Version="1.0" Type="Item">
  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>
</SETemplate>
```

**Parameter**

**Parameter** is an optional child element of **Parameters**. Specifies a custom substitution parameter for the template item. For a list of pre-defined substitution parameters, see [Predefined Substitution Parameters](#).

See the example below.

- **Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Parameter name. This is the name of the substitution parameter WITHOUT delimiters. For example, if the delimiter is &quot;$&quot; (the default), then a substitution parameter that inserts a copyright string would be defined as &quot;copyright&quot; and NOT as...</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value. This is the value that the substitution parameter evaluates to when a string or File is created from the template.</td>
</tr>
</tbody>
</table>

- **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
<?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
<?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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Template version number. The current version is</td>
</tr>
</tbody>
</table>
### Attribute | Description
---|---
"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
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM
"http://www.slickedit.com/dtd/vse/setemplate/1.0/setemplate.dtd">
<SETemplate Version="1.0" Type="Item">

<TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
</TemplateDetails>

<TemplateContent>
    <Files>
        <File>MyClass.java</File>
    </Files>
</TemplateContent>

</SETemplate>
```

**SortOrder**

**SortOrder** is an optional child element of TemplateDetails. Specifies an order number that is used to sort the template item in relation to other template items in a list. Used to sort template items in a category on the Add New Item dialog box.

If no **SortOrder** is specified for a template item, then the **SortOrder** value defaults to "0".

- **Attributes** - None.
- **Child elements** - None.
• **Parent elements** - TemplateDetails.

• **Value** - Text value is required. An integer that is greater than or equal to "0". When sorting in relation to other template items, low SortOrder values are placed ahead of higher values in a sorted list.

**Example**

The following example illustrates the metadata for an item template for a custom Java class.

```
<?xml version="1.0" ?>
<!DOCTYPE SETemplate SYSTEM "http://www.slickedit.com/dtd/vse/setemplate/1.0/setemplate.dtd">
<SETemplate Version="1.0" Type="Item">
  <TemplateDetails>
    <Name>My Java Class</Name>
    <Description>My custom Java class</Description>
    <DefaultName>MyClass.java</DefaultName>
    <SortOrder>100</SortOrder>
  </TemplateDetails>
  <TemplateContent>
    <Files>
      <File>MyClass.java</File>
    </Files>
  </TemplateContent>
</SETemplate>
```

**TemplateContent**

TemplateContent is a **required child element of** SETemplate. Specifies the contents of a template item.

• **Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delimiter</td>
<td>Optional. Delimiter used when replacing substitution parameters in content. Defaults to &quot;$&quot;.</td>
</tr>
</tbody>
</table>

• **Child elements** - Files, Parameters.

• **Parent elements** - SETemplate.
Example

The following example illustrates the metadata for an item template for a custom Java class.

```xml
<?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
<?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>
  </TemplateDetails>
</SETemplate>
```
Quick Brace/Unbrace

    <TemplateName>MyClass.java</TemplateName>
    </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® Core 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:

```cpp
if ( cond ) doSomething();
```

Hanging single line statements are often broken across two lines in the editor:

```cpp
if ( cond )
  doSomething();
```

**Using Quick Brace/Unbrace**

When you use Quick Brace, SlickEdit Core 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:

```cpp
if ( cond ) <cursor here> doSomething();
```

or

```cpp
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:

```cpp
if ( cond ) {
```
doSomething();
}

**Tip**

TIP Indentation and brace style settings are specified on the **Formatting Options** screen specific to your language (Window → SlickEdit Preferences → 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:

```plaintext
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 a language-specific basis. To access this option, from the main menu, click **Window → SlickEdit Preferences → 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® Core 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 Window → SlickEdit Preferences → Languages, expand your language category and language, then click Indent and uncheck Use Dynamic Surround.

SlickEdit® Core 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 Core command line, then type dynamic-surround. If SlickEdit Core 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:
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;
    }
```
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;
}
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 Window → SlickEdit Preferences → 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 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).

**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 with the **Surround With** option selected. As an example, the 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
Surround With

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:

```c
VSARG2QUIRESEDITORCTL | VSARG2MARK | VSARG2REQUIRESAB_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.

### Deleting Code Blocks

Unsurround is also associated with the **cut_line** (Ctrl+Backspace) and **delete_line** (Ctrl+Del) commands. When one of these commands is invoked while the cursor is on the first line of a block statement, the Delete Code Block dialog appears, from which you can choose to delete the line, delete the entire block, or unsurround the block.

Each of these operations copies the removed text to the clipboard. This is useful if you want to paste the structure into a different location, because as soon as the text is pasted, SlickEdit® Core enters Dynamic Surround mode, allowing you to pull statements into the pasted block.

The Delete Code Block dialog also contains an option to **Always just delete line** when **cut_line** or **delete_line** operations are invoked. Selecting this option will prevent the dialog from appearing when these operations are used. To see the dialog again, use the **cut_code_block** command.
Bookmarks

Bookmarks are used to save the current edit location, so you can quickly return to it later. There are two types of bookmarks:

- **Named Bookmarks** - 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.

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.

**Note**

While the SlickEdit stand-alone editor provides its own Bookmarks view, the bookmark functionality in SlickEdit Core integrates with the Eclipse Bookmarks view.

See the following sections for more information about working with named bookmarks:

- Setting Named Bookmarks
- Navigating Named Bookmarks
- Deleting Named Bookmarks
- Bookmark Stack Dialog

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 (Window → SlickEdit Preferences → 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:

• From the main menu, click **Search → Set Bookmark**. The Eclipse **Add Bookmark** dialog is opened. Type the name of the bookmark in the text box, then click **OK**.

• On the SlickEdit® Core 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 Eclipse **Add Bookmark** 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® Core 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 → 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
Named Bookmarks

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 → Previous Bookmark or Search → 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.

- Use the Go to Bookmark feature: From the main menu, click Search → 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.

**Tip**

When you use the Go to Bookmark dialog to jump to a named bookmark, SlickEdit® Core 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 Core 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 Eclipse Bookmarks view, select the bookmark to delete and 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 → 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® Core 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 Eclipse Bookmarks view, select the bookmark to remove and click Delete.

To remove all named bookmarks at once:

- Using the Eclipse Bookmarks view, select all bookmarks and press the Delete key.

- On the SlickEdit® Core command line, use the clear_bookmarks command.

**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® Core.

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 → 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 → 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 Core 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® Core 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 (Window → SlickEdit Preferences → Editing → Bookmarks).

• Use the Bookmark Stack dialog - SlickEdit Core 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 → 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® Core opens a file that was not previously open and you navigate away from it, SlickEdit Core prompts to close the visited, unmodified file. To remove the prompt and specify the default action, set the option Automatically close visited files (Window → SlickEdit Preferences → 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 (Window → SlickEdit Preferences → Editing → Bookmarks).
Core automatically deletes pushed bookmarks when the editor is closed.

- SlickEdit Core 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 (Window → SlickEdit Preferences → Editing → Bookmarks). Note that this option corresponds to the def_top_bottom_push_bookmark configuration variable.
Setting Breakpoints

The quickest way to set or clear a breakpoint is to press F9. This toggles the breakpoint for the current line.

The breakpoints functionality in SlickEdit® Core integrates with the Eclipse Breakpoints view. This view displays a list of breakpoints and lets you easily add, remove, and activate them. To display the Breakpoints view, click Window → Show View → Other, expand Debug and double-click Breakpoints. See "Breakpoints view" in the Eclipse online Help for more information.

Setting Conditional Breakpoints

For help on setting conditional breakpoints, see the Eclipse Help on "Managing conditional breakpoints".

Setting Java Exception Breakpoints

For help on setting Java Exception breakpoints, see the Eclipse Help section "Add Java Exception Breakpoint".
Commenting

SlickEdit® Core 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 Core 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 Format → 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 Format → 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 Format → Uncomment Lines (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® Core 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 Format → 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 Window → SlickEdit Preferences → 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® Core 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 Core 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 **Format → 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 **Window → SlickEdit Preferences → 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 Core 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:

```java
/**[CURSOR_HERE]*/
int setDimensions(int length, int width, int height) {
    ...
}
```

Pressing **Enter** at the "cursor here" location results in the following automatic completion:

```java
/**
 * [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:

```java
///[CURSOR_HERE]
int setDimensions(int length, int width, int height) {
    ...
}
```

Pressing **Enter** at the "cursor here" location results in the following automatic completion:

```java
/// <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:

```c
/*![CURSOR_HERE]*/
int setDimensions(int length, int width, int height) {
    ...
}
```

Pressing **Enter** at the "cursor here" location results in the following automatic completion:

```c
/*!
 * [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 **Window → SlickEdit Preferences → Languages**, expand your language category and language, and 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® Core 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 **Format → 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 **Window → SlickEdit Preferences → 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 **Format → Reflow Comment**.
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 **Format → Beautify** (or use the `gui_beautify` command). A dialog box is displayed with functions specific to the type of project that is active. If an HTML project is active, then the HTML Beautifier dialog appears with options. If a GNU C/C++ project is active, then the C/C++ Beautifier dialog opens, and so on. Beautifying is supported for the languages listed below. Follow the cross-reference links to learn more about working with each beautifier.

- Ada - See [Ada Beautifier](#).
- C/C++, Objective-CC - These languages use beautifiers accessible through the Language options. See [C/C++ Beautifier](#).
- Java, JavaScript, Slick-C® - These beautifiers contain the same options and settings. See [Java Beautifier](#).
- CFML, HTML - These beautifiers contain the same options and settings. See [HTML Beautifier](#).
- Javadoc - See [Javadoc Beautifier Options Dialog](#).
- XML, XSD - These beautifiers contain the same options and settings. See [XML Beautifier](#).

Reflowing Text

To reflow text in the current paragraph according to your margin settings, click **Format → Format 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 (**Window → SlickEdit Preferences → 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

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 views.
- **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

To access Quick Refactoring, from the main menu, click **Format → Quick Refactoring**. Quick Refactoring menu can also be accessed from the right-click context menus in the Symbols and Defs views.

Quick Rename

Quick Rename uses the Context Tagging® to rename a symbol under the cursor or any symbol selected in the Defs or Symbols views. 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 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

When a refactoring finishes, the Refactoring results dialog box is displayed, allowing you to review the changes.
There are three panes in this window:

- The left pane is read-only and shows the original file(s).
- The right pane shows the refactored file(s). For convenience, this pane can be edited.
- The bottom pane lists all files that have been modified by the refactoring. Clicking on any file in this list brings that file into view, where it can be reviewed and edited.

Click **Save All** at the bottom of this window to save all the refactoring and editing changes that were made on all files. Click **Cancel** to discard changes and have all files remain the way they were before the refactoring process.

Click **Next Diff** or **Prev Diff** to advance to the next or previous change made by the refactoring. Click **File>>** to restore the contents of the current selected file to its original contents.
Click **Block>>** to restore an entire block of changes to the original contents. Click **Del Block** to remove a block of code inserted by the refactoring. Click **Line>>** to restore the current line to its original contents.

Some refactorings, in particular **Quick Modify Parameter List**, may require further user input. In this case each input will be displayed under the file it is in, and there will be two additional buttons: **Next Input** and **Prev Input**. You will not be able to save the refactoring results until you have resolved all of the input requests.

### Java Refactoring

The **Eclipse JDT Refactor** main menu item for Java refactoring disappears when you are not using the JDT Editor. SlickEdit Core has made several of these refactorings available in the editor's right-click menu, under the **Source** menu item:

- **Override/Implement Methods**
- **Implement Getters and Setters**
- **Generate Delegate Methods**
- **Add Constructors from Superclass**
- **Generate Constructor Using Fields**
- **Externalize Strings**

There are also other Java refactorings from the JDT accessible from the **Refactor** menu item of the editor's right-click context menu:

- **Move**
- **Change Method Signature**
- **Convert Anonymous Class to Nested**
- **Move Member Type to New File**
- **Pull Up**
- **Push Down**
- **Extract Interface**
- **User Supertype Where Possible**
- **Inline**
- **Introduce Factory**
- **Encapsulate Field**
See the Eclipse Help for descriptions and other information regarding Java refactoring (Java Development User Guide → Reference → Refactoring).
Language-Specific Editing

This chapter describes the language-specific editing features of SlickEdit Core.
Introduction to Language-Specific Editing

Many features in SlickEdit® Core 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 Preferences

Preferences for language-specific features can be set through the SlickEdit Preferences dialog (Window → SlickEdit Preferences → Languages → [Language Category] → [Language] or config command). A shortcut method to access language options for the current buffer is to use the Format → [Language] Options menu item, or the setupext command. This will open the SlickEdit Preferences dialog to the General language-specific option screen for that language. See Language Options for more information.

Language Editing Mode

SlickEdit® Core 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 Format → 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 (Window → SlickEdit Preferences → 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 **Associated 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: `chcccp`). 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
4. In the **Color coding lexer name** field, specify the language identifier so that SlickEdit® Core knows what elements to color. Use the drop-down list to select the lexer.

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 (**Format** → **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 (**Window** → **SlickEdit Preferences** → **Languages** → **File Extension Manager**) is used to add and work with file extensions in SlickEdit® Core.
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.

- **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 (Window → SlickEdit Preferences → 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.

### Adding and Removing File Extensions

If SlickEdit® Core does not provide a file extension that you need to use, you can add it. If you have added a new language to SlickEdit Core 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).

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 the Extensionless File Manager to associate languages to files that do not have extensions. There are two mechanisms used to map files to languages: file mapping and pattern mapping. Each type has its own list of mappings in the Extensionless File Manager, which can be accessed by going to Window → SlickEdit Preferences → Languages → Extensionless File Manager and is seen 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 features and options that are available for Ada, including language-specific options and the Ada Beautifier.

Ada Formatting Options

To access the Ada Formatting Preferences, from the main menu, click Window → SlickEdit Preferences → 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 preferences 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

You can beautify Ada files and change the beautify settings by using the Ada Beautifier dialog box. This dialog box can be accessed from the main menu by clicking Format → Beautify, or by using the gui_beautify command.

To instantly beautify Ada code according to the settings that are selected on the Ada Beautifier dialog box, use the ada_beautify or ada_beautify_selection commands.

The following settings and operations are available on the Ada Beautifier:

- **Restrict to selection** - When checked, only lines in the selection are beautified.
- **Sync extension options** - When checked, the language options are updated to reflect any changes that these dialogs have in common.
- **Beautify** - Beautifies current selection or buffer and closes the dialog box.
- **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.
- **Save Settings** - Saves beautify options in the uformat.ini file. These settings are used by the ada_beautify command.

The tabs on the Ada Beautifier are described in the sections below.

### Indent Tab

The following settings are available:

- **Indent with tabs** - When checked, tab characters are used for leading indent of lines. This value defaults to the Tabs text box on the language-specific Formatting options screen (see Language-Specific Formatting Options).

- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level. The words "Syntax indent" are in parenthesis to help indicate that this field has the same meaning as the Syntax indent text box on the language-specific Formatting options screen (see Language-Specific Formatting Options). By default, this text box is initialized with the current language setup setting.

- **Tab size** - Specifies output tab size. The output tab size is only used if the Indent with tabs check box is enabled on the language-specific Formatting options screen (see Language-Specific Formatting Options).
Options). This value defaults to the Syntax indent text box on the options screen.

- **Original tab size** - Specifies what the original file's tab expansion size was. It is necessary to know the tab expansion size of your original file to handle reusing indent amounts from your original file. Currently the beautifier only reuses the original source file's indenting for comments. This option has no effect if the original file has no tab characters.

- **Continued Lines**
  - **Max line length** - Specifies the maximum length a statement line can be before it is wrapped to a new line. Set this value to 0 to preserve line breaks.
  
  - **Continuation indent** - Specifies how much to indent lines of statements which continue to the next line. This has no affect on assignment statements or parenthesized expressions. Lines which are a continuation of an assignment statement are indented after the first equal sign (=). Lines which are a continuation of a parenthesized expression are indented after the open paren.

- **Operator position** - Specify where the operator should be positioned when breaking a statement across multiple lines. For example, given the statement:

  ```plaintext
  Seconds := Days * Hours_Per_Day * Minutes_Per_Hour * Seconds_Per_Minute ;
  ```

  An operator position setting of **End of same line** would result in:

  ```plaintext
  Seconds := Days *
  Hours_Per_Day *
  Minutes_Per_Hour *
  Seconds_Per_Minute ;
  ```

  An operator position setting of **Beginning of next line** would result in:

  ```plaintext
  Seconds := Days
  * Hours_Per_Day
  * Minutes_Per_Hour
  * Seconds_Per_Minute ;
  ```

## Statements/Declarations Tab

The following options are available on the **Statements/Declarations** tab:

- **Reserved word case** - Specifies the case for reserved words. For example, if you choose **UPPER**, then the Ada reserved word "procedure" would be beautified to "PROCEDURE".

- **One statement per line** - When checked, only one statement is allowed per line of code.

- **One declaration per line** - When checked, only one declaration is allowed per line of code.

- **One parameter per line** - When checked, only one parameter is allowed per line of code in a formal parameter list of a subprogram specification.
• **One enumeration per line** - When checked, only one enumeration is allowed per line of code in an enumerated type definition.

**Horizontal Spacing Tab**

This tab allows you to specify how certain operators and separators are padded. The following options are available:

• **Item** - Syntactic item to which padding settings get applied.

**Note**

The “Binary operators” item includes: + - * / ** := = /= => <= >= < >

• **Padding Before** - When checked, one space is placed before the item.

• **Padding After** - When checked, one space is placed after the item.

• **Padding Preserve** - When checked, the original padding (or lack of padding) around the item is preserved.

**Vertical Alignment Tab**

The following options are available on the **Vertical Alignment** tab:

• **Align on declaration colon** - When checked, adjacent declaration lines (including parameter specifications) have their colons vertically aligned. For example, before beautify:

  ```ada
  procedure foo ( A_Var : Boolean ;
                  Another_Var : Boolean ) ;
  ```

  After beautify:

  ```ada
  procedure foo ( A_Var , Boolean ;
     Another_Var , Boolean ) ;
  ```

• **Align on declaration in-out** - When checked, the modes of parameter specifications in the formal part of a subprogram declaration are vertically aligned. For example, before beautify:

  ```ada
  procedure foo ( A_Var : in Boolean ;
                 Another_Var : in out Boolean);
  ```

  After beautify:

  ```ada
  procedure foo ( A_Var : in Boolean ;
                 Another_Var : in out Boolean);
  ```
Blank Lines Tab

The following options are available on the Blank Lines tab:

- **Item** - Syntactic item to which blank lines settings get applied.
  - **Subprogram declaration** - Procedure or Function declaration.
  - **Subprogram body** - Procedure or Function body.
  - **Type declaration** - Any declaration that begins with the reserved word "TYPE".
  - **for-use** - Aspect clause. For example:
    ```ada
    for Medium Size use 2*Byte;
    ```
  - **Subunit comment header** - The comment block that appears just before a subunit (e.g. Procedure body, etc.).
  - **begin/end** - Any line that starts with the reserved words "begin" or "end."
  - **if/elsif/else** - The if, elsif, and else parts of an if statement.
  - **return** - Any line that starts with the reserved word "return."
  - **Loops** - Loop statements (e.g. loop, while, for).
  - **Nested paren list item** - A parenthesized item that is itself enclosed in a larger parenthesized list. For example, before beautify:
    ```ada
    Default_Data : constant Data_Type :=
      ( A_Set => ( others => ( Item1 => false , Item2 => false , Item3 => false ) ) , -- Paren-d item enclosed in larger paren-d list
       B_Set => ( others => ( Item1 => false , Item2 => false , Item3 => false ) ) ) ;
    ```
    After beautify:
    ```ada
    Default_Data : constant Data_Type :=
      ( A_Set => ( others => ( Item1 => false , Item2 => false , Item3 => false ) ) , -- Paren-d item enclosed in larger paren-d list
       B_Set => ( others => ( Item1 => false , Item2 => false , Item3 => false ) ) ) ;
    ```
  - **Before** - Specify how many blank lines are inserted before item.
• **After** - Specify how many blank lines are inserted after item.

• **Between** - Specify how many blank lines are inserted between like items.

**Comments Tab**

The following options are available on the **Comments** tab:

• **Comment lines immediately below a type declaration indented by** - The amount to indent a comment appearing immediately below a TYPE declaration.

• **Trailing comments** - Trailing comments appear at the end of lines which contain statements or declarations. For example:

```
A := B + C ;  -- This is a trailing comment
-- This is not a trailing comment
procedure foo ( A_Var : Boolean ) ;
```

• **Specific column** - When selected, trailing comments are placed at the specified column.

• **Indent by** - When selected, trailing comments are indented by the specified number of columns after the last character of the end of the statement or declaration.

• **Original relative indent** - When selected, trailing comments are indented by reusing the indent after the last character of the end of the statement or declaration of the original source file.

• **Force type declaration comments to next line** - When selected, trailing comments appearing at the end of a TYPE declaration line are forced onto the next line.

**Advanced Tab**

The following options are available on the **Advanced** tab:

• **if-then-else continued lines** - Use these advanced options to customize how multi-line conditional expressions of an if-then-else statement are indented.

• **Force a linebreak on logical operators** - A line break is forced before/after (depending on your Operator position setting) every logical operator in the condition of an if/elseif. For example, before beautify:

```
-- Indent per level = 3
-- Operator position = Beginning of next line

if A = B and C = D then
  null ;
end if;
```

After beautify:
if A = B
    and C = D then
        null ;
end if ;

• **Additional indent for logical operator** - Additional indent amount for a line broken on a logical operator. This amount is in addition to the current indent level. For example, before beautify (Indent per level = 3; Operator position = Beginning of next line; Additional indent for logical operator = 3):

-- Indent per level = 3
-- Operator position = Beginning of next line
-- Additional indent for logical operator = 3

if A = B and C = D then
    null ;
end if ;

After beautify:

if A = B
    and C = D then
        null ;
end if ;

• **Additional indent for logical operator when followed by another line that begins with logical operator** - Additional indent amount for a line broken on a logical operator that is followed by another line that also is broken on a logical operator that is different. This amount is in addition to the current indent level, and in addition to the **Additional indent for logical operator** setting.

For example, before beautify (Indent per level = 3; Additional indent for logical operator = 3; Additional indent for logical operator when followed by another line that begins with different logical operator = 3):

-- Indent per level = 3
-- Operator position = Beginning of next line
-- Additional indent for logical operator = 3
-- Additional indent for logical operator when
-- followed by another line that begins with different logical operator = 3

if A = B and then C = D or else E = F then
    null ;
end if ;

After beautify:
if A = B
    and then C = D
    or else E = F then
    null ;
end if ;

Schemes Tab

To define a new scheme, set the various beautify options then click the Save Scheme button. User-defined schemes are stored in uformat.ini.
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 preferences can be configured by going to Window → SlickEdit Preferences → Languages → XML/Text Languages → Ant → Options and are pictured below.

The following preferences 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® Core for C and C++, including language-specific formatting options, the C/C++ Beautifier, compiler settings, and preprocessing.

SlickEdit Core'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_ansic_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 Core command line, then type `set-var def_ansic_exts "<extensions>"`, where `<extensions>` is the space-delimited list of extensions.

For example:

```
set-var def_ansic_exts "c h"
```

Please note that if you also code in C++ and any of these extensions are used for C++, they will be interpreted as ANSI C.

C/C++ Beautifier

In SlickEdit 2012, the beautifiers for C++ and Objective-C 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 General Beautify Options for more information.

Beautifier Profiles

The C++ Formatting preferences allow you to pick which formatting profile you want to be in effect, edit or delete existing profiles, and create new profiles. To access the C++ Beautifier settings, go to Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → Formatting.
The C++ 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 **Format → 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.

- **Create Copy** - Creates a copy of the currently selected profile, after prompting you for a name. This is how you 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.

- **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 targetted 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 preferences 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

In order to correctly perform full preprocessing, parsing, symbol analysis, and cross-referencing, SlickEdit® Core 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 preferences screen. From the main menu, click Window → SlickEdit Preferences → Languages → Application Languages → C/ C++ → Compiler Properties.

![Compiler Settings Interface]

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 Core ships with header files for each compiler, and the correct header file will appear in the Header File field. The header file configures the parser to emulate the compiler that is chosen in the Compiler Name field.
Creating New Configurations

There are two ways to begin a new configuration. In both cases, a dialog box will be invoked, prompting for the name of the new configuration.

- Click **Copy** to copy the selected compiler configuration. This can be used as a template for creating a new configuration and makes the process of creating similar configurations more convenient.

- Or, click **Add** to create a configuration from scratch or to add a newly installed compiler.

If you wish to remove the selected compiler and associated configuration from the list, click **Delete**. This does not delete any files from disk.

Building the Tag File

The **Build tag file** button on the C/C++ Compiler Properties dialog is used to build tag files from the header file found in the include directories for the selected compiler configuration. This is especially useful when new configurations are created. If you do not build the tag file here manually, it will be built on demand.

C/C++ Preprocessing

Typically your source code base will include preprocessor macros that you use in your code for portability or convenience. For performance considerations, Context Tagging® does not do full preprocessing, so macros that interfere with normal C++ syntax can cause the parser to miss symbols. For example:

```c
MYNAMESPACEDECL(my)
struct MYPACKEDMACRO BinaryTree {
  MTYPELESS data;
  MYPOINTER(BinaryTree) next;
  MYPOINTER(BinaryTree) prev;
};
MYPOINTER(BinaryTree) proot = MYNULL;
MYENDNAMESPACE
```

This example uses the following preprocessor macros:

```c
#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 MTYPELESS 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® Core to be aware of these preprocessor
macros. To do so, complete the following steps:

1. From the main menu, click **Window → SlickEdit Preferences → 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`, located in your configuration directory. Rather than using the dialog, you can add large numbers of `#defines` directly to this file. You may want to make sure that your entire development team has an up-to-date copy of this configuration file once you have added all of your local preprocessor macros.

**Note**

The `usercpp.h` file should only be used for `#defines` and `#undefs` not `#includes`. 
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 Window → SlickEdit Preferences → 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® Core provides a full-featured Java development environment, allowing you to edit, build, and debug Java programs. Topics in this section:

- **Initial Setup** - Read this to configure SlickEdit Core 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, views, and option screens specific to Java programming.

**Initial Setup**

**Context Tagging® for Java**

SlickEdit® Core 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 Core 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**.

![Context Tagging - Tag Files Dialog](image)

2. Click the **Auto Tag** button to open the Create Tag Files for Compiler Libraries dialog. This dialog is displayed automatically when you first run SlickEdit Core.
3. SlickEdit Core 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 Core 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 Core 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 Core will display a progress bar while your libraries are being tagged. When finished,
   2. SlickEdit Core will display the Context Tagging - Tag Files dialog. You can close this if you have no other libraries to tag.

**Java-Specific Features**

SlickEdit® Core 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

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

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 Format → Imports → Organize Imports, or from the
right-click context menu, select Imports → Organize Imports. Alternately, use the
jrefactor_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 Format → Imports → Add Import, or
from the right-click context menu, select Imports → Add Import. Alternately, use the
jrefactor_add_import command.

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-Specific Interfaces

This section provides detailed information about the following dialogs, views, and option screens that are
specific to Java programming:

• Java Formatting Options Interface

• Organize Java Imports Options Interface

• Java Beautifier Dialog

• Javadoc Editor Dialog

• Javadoc Beautifier Options Dialog

Java Formatting Options Interface
Options are available for Java for changing the Syntax Indent and Syntax Expansion style settings. To access these options, from the main menu, click Window → SlickEdit Preferences → Languages, expand Application Languages > Java, then click Java Formatting Options.

**Note**

Languages similar to Java 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:
  
  • **Insert braces immediately** - Specifies whether template should be inserted with braces.
  
  • **Insert blank line between braces** - Specifies whether a blank line should be inserted between braces when a template expands with braces.
  
  • **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® Core 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.

• **Use continuation indent on function parameters** - Determines whether function parameters should always use the continuation indent.

By default, SlickEdit Core formats multi-line function parameters as follows:

```java
myLongMethodName(firstarg,
secondarg,
thirdarg
);
myLongMethodName(
    firstarg,
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            createdButtonFired(buttonIndex);
        }
    },
    thirdarg
);
myLongMethodName(new ActionListener() { // special case anonymous class
```
first argument
   public void actionPerformed(ActionEvent e) {
      createdButtonFired(buttonIndex);
   },
   secondarg,
   thirdarg
);
myLongMethodName(
   secondarg,
   new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         createdButtonFired(buttonIndex);
      }
   },
   thirdarg
);

If Use continuation indent on function parameters is selected, the format will change as follows:

myLongMethodName(firstarg,
   secondarg,
   thirdarg
);
myLongMethodName(
   firstarg,
   new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         createdButtonFired(buttonIndex);
      }
   },
   thirdarg
);
myLongMethodName(new ActionListener() {   // special case anonymous class
   first argument
   public void actionPerformed(ActionEvent e) {
      createdButtonFired(buttonIndex);
   },
   secondarg,
   thirdarg
};

• 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.
• **No 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.

### Java Beautifier

To beautify a Java document, open the file you want to beautify, then from the main menu, click **Format → Beautify** (or use the *gui_beautify* command). The Java Beautifier will be displayed, which allows you to make settings for how the code will be beautified.

The C#, JavaScript, and Slick-C Beautifiers contain the same options and settings as the Java Beautifier.

The following buttons and settings are available on the Beautifier:

• **Beautify** - Beautifies current selection or buffer and closes the dialog box.

• **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.

• **Save Settings** - Saves beautify options in *uformat.ini* file. These settings are used by the *beautify* command.

• **Restrict to selection** - When this option is selected, only lines in the selection are beautified.

• **Sync extension options** - When this option is selected, the language options are updated to reflect any changes that these dialogs have in common. For example, changing the begin-end style to **Style 2** will update your brace style for Syntax Expansion.

The tabs on the Java Beautifier are described in the sections below.

### Begin-End Style Tab

The **Begin-End Style** tab of the Java Beautifier is pictured below.
The following options and settings are available:

- **Do not change brace style** - Select this option if you do not want your brace style changed. This is useful if you are using a brace style that is not supported by SlickEdit® Core.

- **No space before paren** - Determines whether a space is placed between a keyword such as `if`, `for`, or `while` and the open paren.

- **else on same line as }** - When this option is selected, the beautifier will place `else` on the same line. This is typical when using brace Style 1. The following is an example of using Style 1 with an `else` clause:

  ```java
  if (i<j) {
  } else {
  }
  ```

- **Apply to function braces** - When this option is selected, the beautifier will apply your begin/end style to braces for function definition.

**Indenting Tab**

The **Indenting** tab of the Java Beautifier, pictured below, provides indenting parameters that you can use when working with Java files in SlickEdit® Core.
The following options and settings are available:

- **Indent with tabs** - When this option is selected, tab characters are used for the leading indent of lines. This value defaults to the Tabs text box setting on the language-specific **Formatting** options screen (see **Language-Specific Formatting Options**).

- **Indent first level of code** - Do not clear this check box. When this check box is selected, the first level of code inside a function or method definition is not indented.

- **Indent CASE from SWITCH** - When this option is selected, the case and default statements found inside switch statements are indented from the switch.

- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level of code. We have put the words "Syntax indent" in parenthesis to help indicate that this field has the same 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 language setup setting.

- **Tab size** - The value in this field specifies the output tab size. The output tab size is only used if the option **Indent with tabs** is enabled on the language-specific **Formatting** options screen (see **Language-Specific Formatting Options**). This value defaults to the Syntax indent text box on the **Indent** options screen.

- **Original tab size** - The value in this field specifies the size of the original expansion tab. SlickEdit Core uses the expansion size of your original file to handle reusing indent amounts from your original file. Currently the beautifier only reuses the original source files indenting for comments. This option has no effect if the original file has no tab characters.

- **Continuation indent** - The value in this field specifies how much to indent lines of statements that continue to the next line. This has no effect on assignment statements or parenthesized expressions. Lines that are a continuation of an assignment statement are indented after the first equal sign. Lines that are a continuation of a parenthesized expression are indented after the open paren. Given the
following example:

```java
unsigned
int i;
```

The result would be:

```java
unsigned
<Continuation Indent>int i;
```

- **Align on parens** - When this option is selected, the text for parenthesized expressions that spans multiple lines is aligned on the first non-blank after the parenthesis or on the parenthesis itself.

- **Align on equal** - When this option is selected, the text for multi-line assignment is aligned on the first non-blank after the equals sign (=) or on the equal sign itself.

**Comments Tab**

The **Comments** tab on the Java Beautifier, pictured below, contains options for setting the parameters that you want for the trailing comments.

The following options are available:

- **Indent stand alone comments** - Indicates whether comments that appear on lines by themselves with no statement text to the left are indented to the current statement indent level. For example:

  ```java
  /* stand alone
   * comment
  */
  // another stand alone comment
  i=1;  // trailing comment
  ```
• **Indent column 1 comments** - Normally comments that start in column 1 are left alone. Select this option if you want the indent for these comments to be adjusted.

• **Specific column** - This text box specifies the column in which trailing comments should be placed. Trailing comments are comments that appear at the end of lines that contain statements or declarations. For example:

```java
// another stand alone comment
/* stand alone comment
i=1; // trailing comment
if (x) { /* trailing comment.
   */
}
```

• **Original absolute column** - When this option is selected, trailing comments are placed at the same column as the original source file. Trailing comments are comments that appear at the end of lines that contain statements or declarations.

• **Original relative column** - When this option is selected, trailing comments are indented by reusing the indent after the last character of the end of the statement or declaration of the original source file. Trailing comments are comments that are displayed at the end of lines that contain statements or declarations. For example, if the original code is as follows:

```java
if () {
    i=1; // trailing comment
    i=4; /* trailing comment.
   */
}
```

The resulting code would be:

```java
if () {
    i=1; // trailing comment
    i=4; /* trailing comment.
   */
}
```

**Other Tab**

The **Other** tab on the Java Beautifier, pictured below, contains the preprocessing and pad condition options.
The following options are available:

- **Indent preprocessing** - When this option is selected, the indent before the # character of preprocessing is set to indicate the preprocessing nesting level.

- **Indent inside block** - When this option is selected, preprocessing inside brace block is indented when inside preprocessing. Otherwise, preprocessing within a brace block start in column 1.

- **Indent inside special #ifndef** - Many header files starts with the following lines of code:

  ```
  #ifndef myheader_h
  #define myheader_h
  #endif
  ```

  When this option is selected, preprocessing inside this special #ifndef case is indented.

- **Eat spaces after #** - When this option is selected, the spaces after a preprocessor #, but before the keyword (if, ifdef, else, elif, endif, etc.), are removed. This is useful for fixing old C code where the # character had to start in column 1 and spaces were used after the # to indicate the nesting level.

- **Force parens on return** - When this option is selected, parentheses are added to return statements which do not have parentheses.

- **Pad condition** - These options indicate if parenthesized conditional expressions should have their spacing adjusted.

**Schemes Tab**

The Schemes tab of the Java Beautifier is pictured below.
To define a new scheme, set the various beautify options, and press the **Save Scheme** button. User defined schemes are stored in `uformat.ini`.

**Organize Java Imports Options Interface**

Options are available on the Options dialog to configure the Organize Imports feature (**Window** → **SlickEdit Preferences** → **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 **Format** → **Imports** → **Options**
- Select **Format** → **Imports** → **Options** from the right-click context menu in the editor window.
- Use the `jrefactor_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® Core will attempt to automatically add imports as you edit Java code.

- **Automatically add import during code help for JSP:** If selected, SlickEdit Core 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, right-click within the edit window and select Edit Javadoc Comments. Or, with the cursor inside a comment, click Format → Edit Javadoc 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® Core 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 right-clicking within the edit window and selecting Edit Javadoc Comments. Then click the Options button. The Javadoc Beautifier Options dialog is displayed. The following settings are available:

• Align parameter comments to longest parameter name - If checked, the parameters are aligned to the length of the longest parameter name. If the parameter name length is less than the minimum length, the minimum length is used. If the parameter length is longer than the maximum parameter length, the description for the parameter will start on the next line.

• Align exception comments to longest exception name - If checked, the exceptions are aligned to the length of the longest exception name. If the exception name length is less than the minimum length, the minimum length is used. If the exception length is longer than the maximum exception length, the description of the parameter will start on the next line.

• Align return comments - Indicates whether @return comments should be aligned to the first line of comment text. No alignment is performed if tags which are indent-sensitive such as the <pre> tag are used.

• Align deprecated comments - Indicates whether @return comments should be aligned to the first line of comment text. No alignment is performed if tags which are indent-sensitive such as the <pre> tag are used.

• Add blank line after parameter comment - If checked, a blank line is added if a tag follows an @param tag.

• Add blank line after parameter comment group - If checked, a blank line is added if a tag follows an @param group.

• Add blank line after return comment - If checked, a blank line is added if a tag follows the @return tag.
• **Add blank line after description** - If checked, a blank line is added between the description and the first @ tag. This option is ignored if the description contains a custom or unsupported @ tag.

• **Add blank line after example** - If checked, a blank line is added if a tag follows the @example tag.
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 Window → SlickEdit Preferences → 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.
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 Window → SlickEdit Preferences → 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.
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 Core. This setting will be overridden for a particular file should the file contain a preprocessor "MARGINS" statement.
This section describes some of the advanced features that are available for the Python language.

Begin/End Structure Matching for Python

Begin/End Structure Matching moves the cursor from the beginning of a code structure to the end, or vice versa.

To place the cursor on the opposite end of the structure when the cursor is on a begin or end keyword pair, press Ctrl+] (find_matching_paren command or from the menu click Navigate → Go to Matching Parenthesis). The find_matching_paren command supports matching parenthesis pairs {}, [], and ()

For Python, SlickEdit® Core 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.
XML and HTML

Features for XML and HTML are described below. See also XML/HTML Formatting.

XML

XML features in SlickEdit® Core include Context Tagging®, validation, well-formedness checking, a beautifier, Color Coding, URL Mapping, Syntax Expansion, and Syntax Indenting for XML, XSLT, and schemas (DTD or XSD).

XML Validation

If the XML file being edited references a DTD or schema, SlickEdit Core 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 Window → SlickEdit Preferences → Network & Internet Options → URL Mappings. Then add the URL for this DTD/schema and specify a directory where that file can be found.

• If you don't want to validate the document when it is opened, select Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Formatting and uncheck Auto validate on open.

• 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.

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.

XML Formatting Options

Content in XML and HTML files may be set to automatically wrap and format as you edit, through the XML/HTML Formatting feature. See XML/HTML Formatting for complete information.

Other miscellaneous tag and attribute options are provided through the XML Formatting Options screen. To access these options, from the main menu, click Window → SlickEdit Preferences → Languages,
expand XML/Text Languages > XML, then click XML Formatting Options.

Tip

You can also display the XML Formatting Options screen by clicking the XML Options button on the XML/HTML Formatting Scheme Configuration dialog (see Working with Schemes).

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.

- **Auto validate on open** - When this option is selected, XML files are automatically validated when they are opened. The result of the validation is displayed on the Output view.

- **Auto symbol translation** - When selected, Auto Symbol Translation is enabled, a feature that automatically converts a character or sequence of characters to the appropriate entity reference. Click the Settings button to display the Symbol Translation Editor. See Auto Symbol Translation for more information.

- **Auto formatting options** - These settings are used to configure XML/HTML Formatting.

  - **Default formatting scheme** - Specifies the default formatting scheme to use for the selected language. See Default Schemes for more information.

  - **Enable content wrapping** - When selected, content wrapping is enabled, based on the scheme configuration for the selected formatting scheme. See Working with Schemes for more information.
• **Enable tag layout** - When selected, tag layout is enabled, based on the scheme configuration for the selected formatting scheme.

• **Configure Schemes** - Use this button to open to the XML/HTML Formatting Schemes Configuration dialog to configure tags. See [Working with Schemes](#).

### XMLdoc Editor

Use the XMLdoc Editor to generate Microsoft XML syntax comments for C#, C, C++, Java, JavaScript, and Slick-C®. Note that by default, when creating a new comment, the Javadoc Editor is displayed for all file types except C#. To work around this limitation, start an XML comment with "/**" and then right-click in the edit window and select **Edit XML Comments**.

Unknown XML tags are left "as is" and not removed.

### XML Beautifier

To beautify XML source code, from the main menu click **Format → Beautify** (or use the gui_beautify command). The XML Beautifier dialog appears, where you can specify preferences for how the code is beautified.

#### Caution

The XML Beautifier is not affected by XML/HTML Formatting. If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

You can use the commands `xml_beautify` or `xml_beautify_selection` to instantly beautify the file or the selection according to the settings on the Beautifier dialog.

#### Note

The XSD Beautifier contains the same options and settings as the XML Beautifier.

The following buttons and options are available on the Beautifier:

• **Beautify** - Beautifies current selection or buffer and closes the dialog box.

• **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.

• **Save Settings** - Saves beautify options in `uformat.ini` file. These settings are used by the `xml_beautify` command.

• **Restrict to selection** - When on, only lines in the selection are beautified.

• **Sync extension options** - When on, the language options are updated to reflect any changes that these dialogs have in common.
The tabs on the XML Beautifier are described in the sections below.

**Indent Tab**

The **Indent** tab on the XML Beautifier is pictured below.

![XML Beautifier Indent Tab](image)

The following settings are available:

- **Indent for each level (Syntax indent)** - The amount to indent for each new nesting level of tags. We have put the words “Syntax indent” in parenthesis to help indicate that this field has the same value as the **Syntax indent** text box setting on the language-specific Formatting options screen (see Language-Specific Formatting Options). By default, we initialize this text box with your current language setup setting.

- **Indent with tabs** - When on, tab characters are used for leading indent of lines. This value defaults to the Tabs text box setting 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 the **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 an XHTML `<td>` block which was at an indent level of 30, and your max line length was set to 80, then that line would not be wrapped until it reached a total length of 30+80=110 characters. Set this value to 0 if you want
your line breaks preserved.

- **Broken tag lines** - Specify how broken tag lines are treated from the following options:

  - **Indent from tag column by** - Specifies the amount to indent for broken tag lines from the starting column of the tag. Specify 0 to align broken tag lines with the starting column of the tag.

  - **Use original relative indent** - Reindent broken tag lines using the original relative indent amount from the starting column of the tag.

  - **Preserve original indent** - Preserve the original absolute indent amount on broken tag lines.

**Tags Tab**

The *Tags* tab on the XML Beautifier is pictured below.

The *Tags* tab contains the following options and settings:

- **Tag case** - Specifies how you want your tag names cased. For example, if you choose UPPER, then `<tag>` would be beautified to `<TAG>`. Under normal circumstances you will want to preserve the case of your XML tags, but for certain special cases (e.g. XHTML) you may want to change this setting.

- **Tag settings** - The settings in this group box apply to the tag that is selected in the list box. The `<DEFAULT TAG>` tag item in the list of tags specifies settings to use when no settings exist for a tag found during beautification.

- **Add** - Display the Add Tag dialog. This dialog allows you to add a tag definition to the list and specify how it will be beautified.

- **Remove** - Used to remove the currently selected tag.

- **Content** - Specify how to beautify content from the following options:
• **Reformat** - When off, all white space and line breaks are preserved. However, tags are formatted (tag case, attribute case, etc.).

• **Indent** - When on, nested tags will be indented one syntax indent level. Furthermore, if **Reformat** is on, the selected tag's CDATA content (i.e. plain text), bounded by the opening and closing tag, will be indented 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. If **Reformat** is on, then leading indent is adjusted. This option is useful for XHTML.

**Tip**

Some examples of content settings for specific tags are:

• **style** - **Literal** (content is indented to the same level as the `<style>` open tag)

• **style** - **Reformat, Literal** (content is indented one syntax indent level from the `<style>` open tag)

• **pre** - All **Content** check boxes off

• **blockquote** - **Reformat, Indent**

• **End tag** - When on, the selected tag has an end tag. For XML you will normally want this to remain on.

• **End tag required** - When on, the selected tag's ending tag is required. For XML you will normally want this to remain on.

• **Preserve tag body** - When on, all properties of the body of the tag selected will be preserved. This is especially useful for processing instructions like `<?xml ... ?>` where you do not want the embedded text to be beautified.

• **Preserve tag position** - When on, the position of the tag within the document is preserved. This is especially useful with JSP/ASP tags where reindenting the tag would interrupt the flow of the script code.

• **Line breaks** - Select the way lines are broken:

  • **Before open tag** - Specify the number of line breaks before the opening tag. For example, if you were to set the number of line breaks before the opening tag to 3 for the XHTML `<td>` tag, and the original content was:

    ```
    <tr>
    <td>
    </td>
    </tr>
    ```

    The resulting content would be:
Please note that the number of line breaks is not the same as the number of blank lines. If you wanted three blank lines, then you would set the number of line breaks to 4.

• **After close tag** - Specify the number of line breaks after the closing tag. For example, if you were to set the number of line breaks after the closing tag to 3 for the XHTML `<td>` tag, and the original content was:

```
<TR>
<TD>
</TD>
</TR>
```

The resulting content would be:

```
<TR>
<TD>
</TD>
</TR>
```

Please note that the number of line breaks is not the same as the number of blank lines. If you wanted three blank lines, then you would set the number of line breaks to 4.

• **Stand-alone** - When on, the selected tag will always have at least one preceding and trailing line break on both its opening and ending tag when beautified. You can specify that there be more than one line break by setting **Line breaks** for the opening and closing tags.

**Attributes/Values Tab**

The **Attributes/Values** tab of the XML Beautifier is pictured below.
The Attributes/Values tab contains the following settings:

- **Attribute case** - Specifies how you want attributes cased inside the body of a tag. For example, if you choose UPPER, then `<td align="right">` would be beautified to `<td ALIGN="right">`. Under normal circumstances you will want to preserve the case of your XML attributes, but for certain special cases (e.g. XHTML) you may want to change this setting.

- **Word value case** - Not available for XML.

- **Hex value case** - Not available for XML.

- **Quote word values** - Specifies whether you want word values enclosed in double quotes after the `=` of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`. Select **Preserve** if you want word values left alone. Under normal circumstances you will want to preserve your XML values, but for certain special cases (e.g. XHTML) you may want to change this setting.

- **Quote number values** - Specifies whether you want number values enclosed in double quotes after the `=` of an attribute inside the body of a tag. For example, `<td width=590>` would be beautified to `<td width="590">`. Select **Preserve** if you want number values left alone. Under normal circumstances you will want to preserve your XML values, but for certain special cases (e.g. XHTML) you may want to change this setting.

- **Quote all values** - When on, all values will be quoted after the `=` of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`. Under normal circumstances you will want to preserve your XML values, but for certain special cases (e.g. XHTML) you may want to change this setting.

**Comments Tab**

The Comments tab of the XML Beautifier is pictured below.
The Comments tab contains the following options and settings:

- **Indent stand alone comments** - When on, indicates whether comments which appear on lines by themselves with no content to the left are indented to the current content indent level. The following is an example of a stand-alone comment:

  ```xml
  <!-- stand alone comment -->
  ```

- **Indent column 1 comments** - Normally comments that start in column 1 are left alone. Turn this on if you want the indent for these comments to be adjusted as well.

- **Define Comments** - Displays the XML Comments dialog. This dialog allows you to define what the beautifier recognizes as a comment. The sequence `<!-- -->` is defined as the XML comment by default. If you delete all comment definitions then all comments will be parsed as content.

- **Trailing comments** - Specify how trailing comments are treated from the following options:

  - **Specific column** - This text box specifies the column that "trailing comments" should be placed at. By trailing comments, we mean comments which appear at the end of lines which contain tags. An example of a trailing comment is:

    ```xml
    <TD> <!-- trailing comment -->
    ```

  - **Original absolute column** - When on, "trailing comments" are placed at the same column as the original source file. By trailing comments, we mean comments which appear at the end of lines which contain tags.

  - **Original relative column** - When on, "trailing comments" are indented by reusing the indent after the
last character of the end of the statement or declaration of the original source file. By trailing
comments, we mean comments which appear at the end of lines which contain tags.

The following is an example of code before beautifying trailing comments:

```xml
<Outer>
  <Inner><four characters><!-- trailing comment --></Inner>
</Outer>
```

The resulting code would be:

```xml
<Outer>
  <Inner><four characters><!-- trailing comment -->
 </Inner>
</Outer>
```

**Advanced Tab**

The **Advanced** tab of the XML Beautifier is pictured below.

The following option is available on the **Advanced** tab:

- **Remove blank lines** - When on, blank lines are deleted.

**Schemes Tab**

The **Schemes** tab of the XML Beautifier is pictured below.
To define a new scheme, use the Beautifier to set the various beautify options and then press the **Save Scheme** button on the **Schemes** tab. User defined schemes are stored in `uformat.ini`.

**DTD Caching**

When you open an XML document that has a document type definition of (`!DOCTYPE`) that refers to a remote external DTD, the DTD file is downloaded and cached locally. The DTD is processed to provide Context Tagging® and better color coding. Currently, only HTTP (and not FTP) remote files are supported. This automatic caching allows you to work offline and edit XML documents that reference remote DTDs when you do not have an Internet connection. If you want to force re-caching of the DTD for the current XML document, right-click to open the context menu and select **Apply DTD changes**. Applying DTD changes is necessary after you create a new XML document and complete the document type definition (`!DOCTYPE`).

**Opening DTD Files from XML**

To open the external DTD referenced by document type definition (`!DOCTYPE`), place the cursor anywhere on the `!DOCTYPE` tag and press `Alt+1` (or right-click to display the context menu and select **Go to Error/Include File**).

**URL Mapping**

SlickEdit® Core 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 **Window** → **SlickEdit Preferences** → **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+J** 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 languagespecific 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® Core 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**.

**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 **Window** → **SlickEdit**
Preferences → Network & Internet Options → Web Browser Setup. See Web Browser Setup Options for more information.

**HTML Formatting Options**

Content in XML and HTML files may be set to automatically wrap and format as you edit, through the XML/HTML Formatting feature. See XML/HTML Formatting for complete information.

Other miscellaneous tag and attribute options are provided through the HTML Formatting Options screen. To access these options, from the main menu, click Window → SlickEdit Preferences → Languages, expand Web Authoring Languages > HTML, then click HTML Formatting Options.

**Tip**

You can also display the HTML Formatting Options screen by clicking the HTML Options button on the XML/HTML Formatting Scheme Configuration dialog (see Working with Schemes).

**Note**

Languages similar to HTML 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.

- **Case for inserted tags** - This option is where you specify if you want the tag names to be lowercase or uppercase. For example, if you select Uppercase, then <td> becomes <TD>. The hyperlink indicates if Adaptive Formatting is on or off for this setting.

- **Case for inserted attributes** - This option is where you specify if you want attributes to be lowercase or uppercase inside the body of a tag. For example, if you select Uppercase, then <td align=right> would become <td ALIGN=right>. The hyperlink indicates if Adaptive Formatting is on or off for this setting.

- **Case for inserted single word values** - This option is where you specify if you want word values to be uppercase or lowercase when the = of an attribute is inside the body of a tag. For example, if you select Uppercase, then <td align=right> becomes <td align=RIGHT>. The hyperlink indicates if Adaptive Formatting is on or off for this setting.

- **Case for hex values** - This option is where you specify if you want hex values to be uppercase or lowercase after the = of an attribute inside the body of a tag. For example, if you select Uppercase, then <body bgcolor=#ffffff> would become <body bgcolor=#FFFFFF>. The hyperlink indicates if Adaptive Formatting is on or off for this setting.

- **Embedded ASP dialect** - The language that you select here determines the default embedded language mode for ASP files.

- **Use path for file entries** - This option is used by the HTML toolbar. When this attribute is selected, path information is included when inserting file names into the value of an attribute. For example, creating a link with this option turned on might result in the following example.

  ```html
  <A HREF=file://c:/dev/html/index.htm#>sample link</A>
  ```

  When this option is not selected, the result is the following example.

  ```html
  <A HREF= index.htm#>sample link</A>
  ```

- **Use lower case file names when inserting links** - This option is used by the HTML toolbar. When this option is selected, the lowercase file names are used when inserting links into the value of an attribute. See the example for Use paths for file entries.

- **Use quotes for numerical values** - When this option is selected, word values are enclosed in double quotes after the = of an attribute inside the body of a tag.
• **Use quotes for single word values** - When this option is selected, number values are enclosed in double quotes after the = of an attribute inside the body of a tag.

• **Insert colors using color names (if possible)** - When this option is selected, colors are inserted by using the color names if possible. For example, instead of using `#ff0000` to represent the color red, the color name `red` is used.

  <BODY bgcolor=#ff0000>
  <!-- and -->
  <BODY bgcolor=red>
  <!-- are identical -->

• **Use <DIV> tags for alignment** - This option is used by the HTML toolbar. When this option is selected, the <DIV> tag is used for aligning text. For example, rather than using a <CENTER> tag to designate alignment, use the following tagging:

  <DIV ALIGN=CENTER>
  </DIV>

• **Auto symbol translation** - When selected, Auto Symbol Translation is enabled, a feature that automatically converts a character or sequence of characters to the appropriate entity reference. Click the **Settings** button to display the Symbol Translation Editor. See [Auto Symbol Translation](#) for more information.

• **Auto formatting options** - These settings are used to configure XML/HTML Formatting.

  • **Default formatting scheme** - Specifies the default formatting scheme to use for the selected language. See [Default Schemes](#) for more information.

  • **Enable content wrapping** - When selected, content wrapping is enabled, based on the scheme configuration for the selected formatting scheme. See [Working with Schemes](#) for more information.

  • **Enable tag layout** - When selected, tag layout is enabled, based on the scheme configuration for the selected formatting scheme.

  • **Configure Schemes** - Use this button to open to the XML/HTML Formatting Schemes Configuration dialog to configure tags. See [Working with Schemes](#).

### HTML Beautifier

To beautify an HTML document, open the document you want to beautify, then from the main menu, click **Format → Beautify** (or use the `gui_beautify` command). The HTML Beautifier dialog will be displayed, which allows you to make settings for how the code will be beautified.

**Caution**

The HTML Beautifier is not affected by XML/HTML Formatting. If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find
You can use the commands `h_beautify` or `h_beautify_selection` to instantly beautify the file or the selection according to the settings on the Beautifier dialog.

**Note**

The CFML Beautifier contains the same options and settings as the HTML Beautifier.

The following buttons and options are available on the Beautifier:

- **Beautify** - Beautifies current selection or buffer and closes the dialog box.
- **Reset** - Restores the dialog box settings to the values that appeared when you invoked the dialog.
- **Save Settings** - Saves beautify options in `uformat.ini` file. These settings are used by the `h_beautify` command.
- **Restrict to selection** - When on, only lines in the selection are beautified.
- **Sync extension options** - When on, the 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.

![HTML Beautifier Indent Tab](image)

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.

• **Broken tag lines** - Specify how broken tag lines are treated from the following options:
  • **Indent from tag column by** - Specifies the amount to indent for broken tag lines from the starting column of the tag. Specify 0 to align broken tag lines with the starting column of the tag.
  • **Use original relative indent** - Reindent broken tag lines using the original relative indent amount from the starting column of the tag.
  • **Preserve original indent** - Preserve the original absolute indent amount on broken tag lines.

**Tags Tab**

The Tags tab on the HTML Beautifier is pictured below.
The **Tags** tab contains the following options and settings:

- **Tag case** - Specifies how you want your tag names cased. For example, if you choose **UPPER**, then `<td>` would be beautified to `<TD>`.

- **Tag settings** - The settings in this group box apply to the tag that is selected in the list box. The `<DEFAULT TAG>` tag item in the list of tags specifies settings to use when no settings exist for a tag found during beautification.

- **Add** - Display the Add Tag dialog. This dialog allows you to add a tag definition to the list and specify how it will be beautified.

- **Remove** - Used to remove the currently selected tag.

- **Content** - Specify how to beautify content from the following options:
  
  - **Reformat** - When off, all white space and line breaks are preserved. However, tags are formatted (tag case, attribute case, etc.).
  
  - **Indent** - When on, the selected tag's content, bounded by the opening and closing tag, will be indented 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. If **Reformat** is on, then leading indent is adjusted.

### Tip

Some examples of content settings for specific tags are:

- **XMP** - **Literal**
• **PRE** - All Content check boxes off

• **BLOCKQUOTE** - Reformat, Indent

• **STYLE** - Reformat, Literal

• **End tag** - When on, the selected tag has an end tag. For example, the tag `<TD>` has 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 body** - When on, all properties of the body of the tag selected will be preserved. This is especially useful for JSP/ASP tags where you do not want the embedded Java or VBScript inside the `<% ... %>` to be beautified.

• **Preserve tag position** - When on, the position of the tag within the document is preserved. This is especially useful with JSP/ASP tags where reindenting the tag would interrupt the flow of the script code.

• **Line breaks** - Select the way lines are broken:

  • **Before open tag** - Specify the number of line breaks before the opening tag. For example, if you were to set the number of line breaks before the opening tag to 3 for the `<TD>` tag, and the original content was:

    ```html
    <TR>
    <TD>
    </TD>
    </TR>
    ```

    The resulting content would be:

    ```html
    <TR>
    <TD>
    </TD>
    </TR>
    ```

    Please note that the number of line breaks is not the same as the number of blank lines. If you wanted three blank lines, then you would set the number of line breaks to 4.

  • **After close tag** - Specify the number of line breaks after the closing tag. For example, if you were to set the number of line breaks after the closing tag to 3 for the `<TD>` tag, and the original content was:

    ```html
    <TR>
    <TD>
    </TD>
    ```

    ```html
    <TR>
    <TD>
    ```
• Stand-alone - When on, the selected tag will always have at least one preceding and trailing line break on both its opening and ending tag when beautified. You can specify that there be more than one line break by setting Line breaks for the opening and closing tags.

Attributes/Values Tab

The Attributes/Values tab of the HTML Beautifier is pictured below.

The Attributes/Values tab contains the following settings:

• Attribute case - Specifies how you want attributes cased inside the body of a tag. For example, if you choose UPPER, then <td align="right"> would be beautified to <td ALIGN="right">.
HTML

- **Word value case** - Specifies how you want word values cased after the = of an attribute inside the body of a tag. For example, if you choose **UPPER**, then `<td align="right">` would be beautified to `<td align=RIGHT>`.

- **Hex value case** - Specifies how you want hex values cased after the = of an attribute inside the body of a tag. For example, if you choose **UPPER**, then `<body bgcolor="#ffffff">` would be beautified to `<body bgcolor="#FFFFFF">`.

- **Quote word values** - Specifies whether you want word values enclosed in double quotes after the = of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`. Select **Preserve** if you want word values left alone.

- **Quote number values** - Specifies whether you want number values enclosed in double quotes after the = of an attribute inside the body of a tag. For example, `<td width=590>` would be beautified to `<td width="590">`. Select **Preserve** if you want number values left alone.

- **Quote all values** - When on, all values will be quoted after the = of an attribute inside the body of a tag. For example, `<td align=right>` would be beautified to `<td align="right">`.

**Comments Tab**

The **Comments** tab of the HTML Beautifier is pictured below.

The **Comments** tab contains the following options and settings:

- **Indent stand alone comments** - When on, indicates whether comments which appear on lines by themselves with no content to the left are indented to the current content indent level. The following is an example of a stand-alone comment:

  ```html
  <!-- stand alone comment -->
  ```
• **Indent column 1 comments** - Normally comments that start in column 1 are left alone. Turn this on if you want the indent for these comments to be adjusted as well.

• **Define Comments** - Displays the HTML Comments dialog. This dialog allows you to define what the beautifier recognizes as a comment. The sequence `<!-- -->` is defined as the HTML comment by default. If you delete all comment definitions then all comments will be parsed as content and possibly word-wrapped.

• **Trailing comments** - Specify how trailing comments are treated from the following options:

  • **Specific column** - This text box specifies the column that "trailing comments" should be placed at. By trailing comments, we mean comments which appear at the end of lines which contain tags. An example of a trailing comment is:

    ```html
    <TD> <!-- trailing comment -->
    </TD>
    </TR>
    ```

  • **Original absolute column** - When on, "trailing comments" are placed at the same column as the original source file. By trailing comments, we mean comments which appear at the end of lines which contain tags.

  • **Original relative column** - When on, "trailing comments" are indented by reusing the indent after the last character of the end of the statement or declaration of the original source file. By trailing comments, we mean comments which appear at the end of lines which contain tags.

The following is an example of code before beautifying trailing comments:

```html
<TR>
<TD><four characters><!-- trailing comment -->
</TD>
</TR>
```

The resulting code would be:

```html
<TR>
    <TD><four characters><!-- trailing comment -->
    </TD>
</TR>
```

**Advanced Tab**

The **Advanced** tab of the HTML Beautifier is pictured below.
The following options are available on the Advanced tab:

- **End previous <P> tag when** - Select from the following options:
  - **Hitting another <P> tag** - When on, the beautifier will interpret a <P> tag as a signal of the end of any previous paragraphs.
  - **Hitting a standalone tag** - When on, the beautifier will interpret a standalone tag as a signal of the end of any previous paragraphs. For example, in the following content, the <TABLE> tag (assuming that the <TABLE> tag is a standalone tag) signals the end of the previous <P> paragraph. This has the benefit of cleaning up unwanted (and unexpected) indenting. For example:

    ```html
    <P>
    This is a paragraph of content. The paragraph will be ended by the start of the table below it.
    </P>
    
    <TABLE>
    <TR>
    <TD>a table cell</TD>
    </TR>
    </TABLE>
    ```

- **Remove blank lines** - When on, blank lines are deleted.

- **Beautify JavaScript** - When on, embedded JavaScript is beautified according to your JavaScript beautifier settings.

- **Edit JavaScript Settings** - Click on this button to configure the JavaScript Beautifier settings. The JavaScript Beautifier is the same as the C/C++ Beautifier - see C/C++ Beautifier for more information.
Schemes Tab

To define a new scheme, set the various beautify options and press the **Save Scheme** button. User defined schemes are stored in `uformat.ini`.

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® Core 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 Core: the character or character sequence that you type to trigger translation is actually a special kind of alias, henceforth called a "symbol alias". SlickEdit Core 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 **Window → SlickEdit Preferences**. Expand **Languages** and your language category, then select your language and click `[Language] Formatting Options`. 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 (Window → SlickEdit Preferences → 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.

XML/HTML Formatting
Content in XML and HTML files may be set to automatically wrap and format as you edit. XML/HTML Formatting is essentially comprised of two features: **Content Wrap**, which wraps the content between tags, and **Tag Layout**, which formats tags according to a specified layout. Both can be activated individually for all XML and HTML files that are opened in SlickEdit® Core, or just for the current document.

*Formatting schemes* form the basis for how tags and content are formatted. A formatting scheme contains any number of XML or HTML tags, each of which can be configured individually for indent levels, wrapping, and tag structure. Multiple schemes can be defined, for example, you may want one scheme for HTML files and another for XML files, or perhaps you are required to code certain files to various standards. Schemes can be saved and imported, so they can be shared with your team. Tags for each scheme can be entered manually or you can import tags from the current file.

**Caution**

XML/HTML Formatting does not currently affect XML or HTML Beautifier settings. If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

### Enabling/Disabling XML/HTML Formatting

XML/HTML Formatting is on by default for XML and HTML files that you open in SlickEdit® Core. You can activate and/or deactivate Tag Layout and/or Content Wrap for either file type on a global basis or on a per document basis. Options to turn these features on/off are located on the **Format → XML/HTML Formatting** menu.

![XML/HTML Formatting menu](image)

### Enabling/Disabling Globally

Automatic formatting can be enabled or disabled for every XML and/or HTML file that is created or opened in SlickEdit® Core. These XML- and HTML-specific global options are toggled on/off by placing a check next to **Enable XML Formatting** and/or **Enable HTML Formatting**. To toggle **Tag Layout** and/or **Content Wrap** on or off for either file type, check or uncheck those items. For example, if you want both Tag Layout and Content Wrap enabled for XML files, but you only want Content Wrap enabled for HTML files, place a check next to **Enable XML Formatting** and its sub-menu items **Tag Layout** and **Content Wrap**, then place a check next to **Enable HTML Formatting** and its sub-menu item **Content Wrap**.

The `xml_formatting_toggle` and `html_formatting_toggle` commands can also be used to toggle all
respective formatting features on/off. For use in macros, two more commands are available: `xml_formatting` and `html_formatting`. When you use any of these commands, the command line prompts for yes (Y) or no (N). When these commands are used to enable/disable XML/HTML Formatting, both Tag Layout and Content Wrap are enabled/disabled for both XML and HTML files on a global basis.

You can also turn automatic formatting on and off through the language-specific Formatting Options screen: from the main menu, click **Window → SlickEdit Preferences**, expand **Languages** and your language category, then select your language and click **[Language] Formatting Options**. In the **Auto formatting options** group box, select the default formatting scheme you want to use for this language, then enable/disable the content wrapping or tag layout options. Click the **Configure Schemes** button to open the XML/HTML Formatting Scheme Configuration dialog (see **Working with Schemes**).

**Enabling/Disabling for the Current Document**

Current document options are available so that you can turn off aspects of global formatting for just the current document. For example, if you have both aspects of HTML formatting enabled globally, but you need to edit an old HTML file and do not want automatic tag layout to occur, you can disable HTML Tag Layout for that specific file. The current document settings are remembered each time you open that file.

To change XML or HTML formatting for just the current document, click **Format → XML/HTML Formatting → Current Document Options**, or use the `xml_html_document_options` command. The Current Document Options dialog is displayed.

![Current Document Options dialog]

Select the **Formatting scheme** that you want applied, then check or uncheck the **Auto formatting options** that you want enabled or disabled. Click **Configure Schemes** if you want to modify or create a new scheme to apply to the current document.

Global formatting must be enabled for the current file type in order for these options to be available. For example, see the two screen shots shown previously. The menu screen shot shows that global HTML formatting is enabled for Content Wrap only. This means that tag content for every HTML file that you create or open in SlickEdit® Core will be wrapped, but no Tag Layout settings will be applied. The second screen shot (the Current Document Options dialog) reflects that the global setting for Tag Layout is disabled. Therefore it cannot be enabled for the current document. To enable it for the current document, you would first need to enable the global setting by placing a check next to **Format → XML/HTML Formatting**.
Formatting → Enable HTML Formatting → Tag Layout. Then you can use the Current Document Options dialog to enable it for the current document.

Working with Schemes

A formatting scheme is comprised of any number of individually configurable XML or HTML tags and controls the formatting of your text when XML/HTML Formatting is enabled. You can create different schemes for use with either XML or HTML, and/or different schemes for use with different individual files. For example, you may want one scheme for HTML and a different scheme for XML. Or, you may want one scheme for creating new files and another for editing existing files.

Schemes are stored as XML files in the format SchemeName.xml, and are located in the formatschemes/xwschemes subdirectory of your user configuration directory. Scheme files can be shared or checked into version control to ensure consistency in the formatting of your team’s XML/HTML files.

Use the XML/HTML Formatting Scheme Configuration dialog to work with schemes. To access this dialog, from the main menu, click Format → XML/HTML Formatting → Current Document Options, then click the Configure Schemes button on the Current Document Options dialog. You can also display the dialog by using the xml_html_options command. Available schemes are listed in the Schemes column. The tags that make up each scheme are listed in the Tags column.
Default Schemes

XML/HTML Formatting comes with two default schemes. Each time that you open an HTML file for editing, by default, the html (default html) scheme is used. Each time an XML file is opened, the xml (default xml) scheme is used. You can configure the settings for each default scheme or specify your own default schemes (see Specifying a Different Default Scheme).

The html (default html) scheme is comprised of a (default) tag as well as a list of commonly used HTML tags, that are preconfigured with standard settings. The xml (default xml) scheme is comprised of one (default) tag preconfigured with standard settings.

For any tag that does not appear in the Tags list, the (default) tag settings are used.

Specifying the Scheme to Use
You can specify a scheme to use for just the current document, or a default scheme to use for all HTML and/or XML files that are created or opened in SlickEdit® Core.

To specify the scheme for the current document, click **Format → XML/HTML Formatting → Current Document Options**, and pick the scheme to use from the drop-down list. The scheme you choose is remembered the next time that the document is opened.

**Specifying a Different Default Scheme**

SlickEdit® Core has two predefined default schemes (see [Default Schemes](#)). You can specify your own default scheme for new XML or HTML files by selecting a scheme, then from the right-click context menu, choose **Set as Default XML Scheme** or **Set as Default HTML Scheme**. The name in the **Scheme** list will be appended with the text "(default xml)" or "(default html)". For example, if you have an HTML scheme named readmes and set it as the default, the name in the Schemes list will change to readmes (default html).

**Creating Schemes**

To create your own scheme, right-click in the **Scheme** column of the XML/HTML Formatting dialog, and select **New Scheme**.

Type a name for your scheme, then select an existing scheme on which it should be based. This will "copy" all of the tags and settings from the selected existing scheme to your new scheme. Click **Create** when finished.

**Saving and Deleting Schemes**

In the XML/HTML Formatting Scheme Configuration dialog, modified schemes are denoted by asterisks around the name (for example, "html"). To save a modified scheme, right-click on it and select **Save Scheme Changes**. Alternatively, when you close the XML/HTML Formatting dialog, you are prompted whether to save modified schemes.
To delete a selected scheme, right-click and choose **Delete Scheme**.

**Working with Tags**

As described previously, formatting schemes are comprised of individual tags with associated formatting settings. For example, in HTML, you may want start and end `<div>` tags to be on separate lines above/below the text, while start/end style tags (such as `<b>` and `<i>`) are formatted inline with the text.

In order to configure formatting for individual tags, you must first define a scheme (see [Creating Schemes](#)), or you can use one of the default schemes (see [Default Schemes](#)). The Tags column of the XML/HTML Formatting dialog shows a list of tags associated with the selected scheme.

**Default Tags**

For all schemes, a **(default)** tag is included. It defines the settings that are applied to tags that are not specifically listed in the Tags list. It can also be used as a basis for other tags when you use the **Match tag style of** option on the **General** tab.

The default settings for the **(default)** tag are based on a block-style tag, in that the start and end tags are on separate lines, content and nested tags are indented according to your extension indent style, and content is wrapped to a fixed right margin at column 80. Use the Preview area at the bottom of the dialog to see how tags based on the **(default)** tag will be formatted in your code.

**Base Tags**

Base tags are used to define groups of tags with similar behaviors. By creating a base tag and associating a set of actual tags with that tag, you can configure that set of tags by changing the settings for the base tag. The sample HTML schemes included with SlickEdit® Core include two base tags: **(block)** and **(semi-block)**. When creating your own base tags, be sure to use unique tag names that are not used in formatting to make it easy to spot them.

The **(block)** tag is useful as a base tag for tags like `<div>`, where you want the start and end tags on separate lines from the content and aligned vertically. By default, this style indents nested tags and content according to your extension indent style, and content is wrapped to a fixed right margin at column 80. Because this is a common style in HTML, the **(block)** tag has the same settings as the **(default)** tag.

The **(semi-block)** tag is useful as a base tag for tags such as `<h1>`, where you want the start and end tags on the same line as the content. The default settings for **(semi-block)** are the same as for **(block)**, except for the start and end tags.

Use the Preview area at the bottom of the dialog to see how tags based on these base tags will be formatted in your code. The following are examples:

```html
<div>
  Sample text of a tag set to (block) style.
</div>

<h1>Sample text of a tag set to (semi-block) style. Notice how wrapping occurs within this tag.</h1>
```
Adding and Deleting Tags

To add individual tags to a selected scheme, right-click in the Tags column and select **New Tag**. Type the name of the tag without angle brackets or attributes. Click **OK** and the new tag is now listed in the Tags column.

**Note**

Tag names cannot have spaces. If you create a new tag with spaces, SlickEdit Core converts the spaces to underscores in the dialog.

To add all of the tags from the current file to a selected scheme, right-click in the Tags column and select **Add Tags from Current File**.

To remove a tag from a selected scheme, right-click on the tag and select **Delete Tag**. Deleting SlickEdit® Core (default) tags, or any tag that is based on another, is not recommended. If you attempt to do this, you will be prompted whether to continue.

Formatting Settings

The tabs on the XML/HTML Formatting dialog contain settings that control how your text is formatted when XML/HTML Formatting is enabled. For descriptions of these options, see [Working with Schemes](#).

**Caution**

XML/HTML Formatting does not currently affect XML or HTML Beautifier settings. If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

Outline View for XML

For XML documents, the Outline View 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:

```xml
<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 Outline View 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 Core that help while coding.
Find and Replace

SlickEdit® Core provides several different ways to search and replace:

- For the fastest method of searching and replacing, use Quick Search and Quick Replace (see Quick Search and Replace below).

- To find and replace text "on the fly," or, as you type, use incremental searching (see Incremental Searching below).

- If you are more comfortable with keystrokes, you may prefer command line searching with the find and replace commands (see Find and Replace Commands).

- Use the Find and Replace view if you prefer working within an interface (see Find and Replace View).

- To search for symbols, use the Find Symbol view (see Find Symbol View).

Both the Find and Replace view 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® Core is controlled by the Search Options located in the Options dialog (Window → SlickEdit Preferences → Editing → Search). The options specified here are used each time a search is performed, except for the Find and Replace View, 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 view. Once the value is changed in the view it is remembered and used the next time the Find and Replace view is launched. So changing a value in the Search Options may not have any effect on the Find and Replace view. Changing a setting in the Find and Replace view 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 menu (or by using the quick_search command). The commands find_next (Edit → Find Next or Ctrl+G) and find_prev (Edit → Find Previous 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:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+R</td>
<td>Search in reverse for the next occurrence of the search string.</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Search forward for the next occurrence of the search string.</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Toggle regular expression pattern matching on/off.</td>
</tr>
<tr>
<td>Ctrl+W</td>
<td>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).</td>
</tr>
<tr>
<td>Ctrl+Shift+W</td>
<td>Copy complete word at cursor to search string.</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Toggle case sensitivity. The key bound to the Brief emulation command case_toggle will also toggle the case sensitivity.</td>
</tr>
<tr>
<td>Ctrl+M</td>
<td>Toggle searching within selection.</td>
</tr>
</tbody>
</table>
### Keys

<table>
<thead>
<tr>
<th>Keys</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+O</td>
<td>Toggle incremental search mode.</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote the next character typed.</td>
</tr>
<tr>
<td>Ctrl+S or F5</td>
<td>(Brief emulation) Search forward for the next occurrence of the search string.</td>
</tr>
<tr>
<td>Ctrl+R or Alt+F5</td>
<td>(Brief emulation) Search in reverse for the next occurrence of the search string.</td>
</tr>
<tr>
<td>Ctrl+W</td>
<td>(GNU Emacs emulation) Copy complete word at cursor to search string.</td>
</tr>
<tr>
<td>Ctrl+Shift+W</td>
<td>(GNU Emacs emulation) Toggle word searching on and off.</td>
</tr>
</tbody>
</table>

Incremental searching stops when you press a key that does not insert a character. You can press Esc to terminate an incremental search (only during prompting). Press and hold Ctrl+Alt+Shift to terminate a long search.

You can retrieve your previous search string by invoking the i_search or reverse_i_search command and pressing Ctrl+S or Ctrl+R, respectively, before entering a search string.

### Find and Replace Commands

#### Find and Slash (/) Commands

The command line is available for performing searches. You can use the forward slash (/) or find commands which provide the same functionality as the Find and Replace view. Press Esc to toggle the cursor to the command line.

The syntax of the slash(/) command is:

```
/string[/OptionCharacters]
```

The syntax of the find command is:

```
find /string[/OptionCharacters]
```

*OptionCharacters* is one or more of the following option characters:
<table>
<thead>
<tr>
<th>Option Character(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Exact case.</td>
</tr>
<tr>
<td>I</td>
<td>Ignore case.</td>
</tr>
<tr>
<td>-</td>
<td>Reverse search.</td>
</tr>
<tr>
<td>M</td>
<td>Limit search to selection.</td>
</tr>
<tr>
<td>&lt;</td>
<td>If found, place cursor at beginning of word.</td>
</tr>
<tr>
<td>&gt;</td>
<td>If found, place cursor at end of word.</td>
</tr>
<tr>
<td>R</td>
<td>Interpret search string as a SlickEdit® Core regular expression.</td>
</tr>
<tr>
<td>U</td>
<td>Interpret search string as UNIX regular expression.</td>
</tr>
<tr>
<td>B</td>
<td>Interpret string as a Brief regular expression.</td>
</tr>
<tr>
<td>N</td>
<td>Do not interpret search string as a regular expression.</td>
</tr>
<tr>
<td>P</td>
<td>Wrap to beginning/end when string not found.</td>
</tr>
<tr>
<td>W</td>
<td>Limit search to words. Used to search for variables.</td>
</tr>
<tr>
<td>W=SlickEdit-regular-expression</td>
<td>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).</td>
</tr>
<tr>
<td>W:P</td>
<td>Limit search to word prefix. For example, a search for &quot;pre&quot; matches &quot;pre&quot; and &quot;prefix&quot; but not &quot;supreme&quot; or &quot;supre&quot;.</td>
</tr>
<tr>
<td>W:PS</td>
<td>Limit search to strict word prefix. For example, a search for &quot;pre&quot; matches &quot;prefix&quot; but not &quot;pre,&quot; &quot;supreme&quot; or &quot;supre&quot;.</td>
</tr>
<tr>
<td>W:S</td>
<td>Limit search to word suffix. For example, a search for &quot;fix&quot; matches &quot;fix&quot; and &quot;suffix&quot; but not &quot;fixit&quot;.</td>
</tr>
<tr>
<td>Option Character(s)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>W:SS</strong></td>
<td>Limit search to strict word suffix. For example, a search for “fix” matches “suffix” but not “fix” or “fixit”.</td>
</tr>
<tr>
<td><strong>H</strong></td>
<td>Allow finding search string in hidden lines.</td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td>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 &quot;a&quot;, it will not be found as the second character of a DBCS sequence unless this option is specified.</td>
</tr>
<tr>
<td>, (comma)</td>
<td>Delimiter to separate ambiguous options.</td>
</tr>
<tr>
<td><strong>X CCLetters</strong></td>
<td>Requires the first character of search string NOT be one of the color coding elements specified. For example, <strong>XCS</strong> requires that the first character not be in a comment or string. <em>CCLetters</em> is a string of one or more of the following color coding element letters:</td>
</tr>
<tr>
<td></td>
<td>• <strong>O</strong> - Other</td>
</tr>
<tr>
<td></td>
<td>• <strong>K</strong> - Keyword</td>
</tr>
<tr>
<td></td>
<td>• <strong>N</strong> - Number</td>
</tr>
<tr>
<td></td>
<td>• <strong>S</strong> - String</td>
</tr>
<tr>
<td></td>
<td>• <strong>C</strong> - Comment</td>
</tr>
<tr>
<td></td>
<td>• <strong>P</strong> - Preprocessing</td>
</tr>
<tr>
<td></td>
<td>• <strong>L</strong> - Line number</td>
</tr>
<tr>
<td></td>
<td>• <strong>1</strong> - Symbol 1</td>
</tr>
<tr>
<td></td>
<td>• <strong>2</strong> - Symbol 2</td>
</tr>
<tr>
<td></td>
<td>• <strong>3</strong> - Symbol 3</td>
</tr>
<tr>
<td></td>
<td>• <strong>4</strong> - Symbol 4</td>
</tr>
<tr>
<td></td>
<td>• <strong>F</strong> - Function color</td>
</tr>
<tr>
<td>Option Character(s)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>• V</td>
<td>- No save line</td>
</tr>
<tr>
<td>• A</td>
<td>- Attribute</td>
</tr>
<tr>
<td>C</td>
<td>Requires the first character of search string to be one of the color coding elements specified. See CCLetters above.</td>
</tr>
<tr>
<td>*</td>
<td>Used with the &quot;Search hidden text&quot; (H) or &quot;Highlight matches&quot; (#) options to find all matches and un-hide or highlight them.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Use Wildcard regular expression syntax (*, ?).</td>
</tr>
<tr>
<td>#</td>
<td>Highlight matched patterns with highlight color.</td>
</tr>
<tr>
<td>V</td>
<td>(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.</td>
</tr>
<tr>
<td>$</td>
<td>(Replace commands only) Replaced occurrences are highlighted with modified color.</td>
</tr>
</tbody>
</table>

If you don’t specify options when using the find 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 /string1/string2[/options]
```

or:

```
replace /string1/string2[/options]
```
Find and Replace Commands

The available options are the same as for the `find` and `slash()` commands (see Find and Slash (/) Commands above).

You can perform one of the following actions with the replace command (c) by pressing the corresponding key:

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y or Space</td>
<td>Make change and continue searching.</td>
</tr>
<tr>
<td>N or Backspace</td>
<td>No change and continue searching.</td>
</tr>
<tr>
<td>L or Dot</td>
<td>Make change and stop searching.</td>
</tr>
<tr>
<td>G or !</td>
<td>Make change and change the rest without prompting.</td>
</tr>
<tr>
<td>Q or Esc</td>
<td>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 Window → SlickEdit Preferences → Editing → Search and set the Restore cursor after replace option to True.</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Exit command and restore cursor to its original position.</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Search in reverse for next occurrence of search string.</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Search forward for next occurrence of search string.</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Toggle regular expression pattern matching on/off. The key bound to the Brief emulation command <code>re_toggle</code> will also toggle regular expression pattern matching.</td>
</tr>
<tr>
<td>Ctrl+W</td>
<td>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).</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Toggle case-sensitivity. The key bound to the Brief emulation command <code>case_toggle</code> will also toggle the case-sensitivity.</td>
</tr>
</tbody>
</table>
### Find and Replace Commands

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+M</td>
<td>Toggle searching within selection.</td>
</tr>
<tr>
<td>F1 or ?</td>
<td>Display Help on Find and Replace view.</td>
</tr>
</tbody>
</table>

### Replace Command Search Examples

The table below provides examples of using command line replace.

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c $/$$</td>
<td>Replace occurrences of forward slashes with back slashes.</td>
</tr>
<tr>
<td>c/x/y/m</td>
<td>Replace occurrences of &quot;x&quot; in the selected area with &quot;y&quot; using default search case sensitivity.</td>
</tr>
<tr>
<td>c $x$y$m</td>
<td>Replace occurrences of &quot;x&quot; in the selected area with &quot;y&quot; using default search case sensitivity. The string delimiter &quot;$&quot; has been used requiring a space character after the &quot;c.&quot;</td>
</tr>
<tr>
<td>c/x/y/e*</td>
<td>Replace lowercase occurrences of &quot;x&quot; with &quot;y&quot; without prompting.</td>
</tr>
<tr>
<td>c/i/something__more_meaningful/w</td>
<td>Replace occurrences of the variable &quot;i&quot; with &quot;something__more_meaningful.&quot;</td>
</tr>
<tr>
<td>c/i/j=w=[A-Za-z]</td>
<td>Replace occurrences of the word &quot;i&quot; with &quot;j&quot; and specify valid characters in a word to be alphabetic characters.</td>
</tr>
<tr>
<td>replace/Test/TEMP/v</td>
<td>Replace occurrences of the word &quot;test&quot; with the word &quot;temp&quot;, with the case preserved. For example:</td>
</tr>
<tr>
<td></td>
<td>* Occurrences of &quot;Test&quot; are replaced with &quot;Temp&quot;.</td>
</tr>
<tr>
<td></td>
<td>* Occurrences of &quot;test&quot; are replaced with &quot;temp&quot;.</td>
</tr>
<tr>
<td></td>
<td>* Occurrences of &quot;testT&quot; are replaced with &quot;TEMP&quot; (because a mixed case will retain the actual replacement that you typed).</td>
</tr>
<tr>
<td></td>
<td>* Occurrences of &quot;TEST&quot; are replaced with &quot;TEMP&quot;.</td>
</tr>
</tbody>
</table>
Find and Replace View

You can use the Find and Replace view (Ctrl+F, Edit → Find/Replace) to specify search and replace options and conduct search and replace operations on selections, single files, or multiple files.

Saving Search and Replace Values

When the Find and Replace view 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 view 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 view. 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 view 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 view. Alternatively, you can set default options that are always used instead. To set the default options, right-click on the background of the view and select **Configure Options**. The default search options will always be used when the Find and Replace view is invoked, unless settings are changed on the Find and Replace view. If you change settings on the view and want to use the default options instead, right-click in the view and select **Use Default Options**. See **Search Options** for more information. For information on the individual options on the Find and Replace view, see **Search Dialogs and Views**.

### Search Results Output

You can specify that multi-file search results are displayed in a new editor window or in a new Search Results view.

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 view, 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 view 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 view 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 view 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 view.

Right-click in the Search Results window to access the following options:
Find Symbol View

- **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 View

The Find Symbol view (Search → Find Symbol or gui_push_tag command) is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match.

Searching for a symbol is faster than a normal text search because it is executed against the Context Tagging® database, rather than searching through your source files. Find Symbol also avoids false hits in comments or string literals. Though Syntax-Driven Searching in the regular Search Dialogs and Views provides this same capability, it cannot match the speed of Find Symbol.

See Find Symbol View for information about the options that are available.

Find and Replace with Regular Expressions

SlickEdit® Core supports five types of regular expression syntax that you can use for finding and replacing when regular search and replace operations are too limiting:

- Brief
- Perl
- SlickEdit
- UNIX
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

Cursor on Symbol Shows All Uses in File

SlickEdit® Core 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 Window → SlickEdit Preferences → 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 (Window → SlickEdit Preferences → 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® Core provides several ways to compare and merge files:

- **DIFFzilla®**
- **3-Way Merge**

**DIFFzilla**

DIFFzilla® provides powerful differencing capabilities that let you compare files or directories and view the differences side-by-side. You can make edits, merge changes, and save modified files easily within the results windows. As edits are made, the diff view is updated as you type, so you don't have to re-run the comparison. And, switching from a directory comparison to an individual file difference is as simple as a mouse click.

**Dynamic Difference Editing**

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**

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.

```cpp
Rectangle::Rectangle() {
}

Rectangle::Rectangle()
{
}
```

Using **Source Diff**, DIFFzilla presents the Path 2 file with it's 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.
In this screen shot, you can see a blue 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.

**Using DIFFzilla**

The following sections describe how to use DIFFzilla and the differencing features in SlickEdit Core. 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

**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**.

Alternatively, you can use DIFFzilla to diff files from several Eclipse views, including the Package Explorer and the Project Explorer:

1. From the Eclipse view, select the two files that you want to diff and right-click.

2. From the **Compare With** submenu, click **Each Other (DIFFzilla)**.

**Comparing Two Directories**

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.

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

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**

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**

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.

**3-Way Merge**
The 3-Way Merge editing feature can be used after two people make a local copy of the same source file, and each makes modifications to their local copy. The 3-Way Merge takes both sets of changes and creates a new source file. If there are any differences, a dialog box is displayed that lets you select the changes that you want in the output file. The output file can be viewed side-by-side or interleaved.

Performing a Three-Way Merge

To perform a three-way merge, complete the following steps:

1. From the main menu, 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** button 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:

![3-Way Merge](image)

### 3-Way Merge Settings

For descriptions of the options on the 3-Way Merge Setup dialog, see [Tools, Dialogs and Views](#).
**File History and Backups**

SlickEdit’s Backup History is disabled in SlickEdit Core. Eclipse maintains their own backup history which you can access using the History view (Window → Show View → Other, expand Team and click History).

You can, however, use DIFFzilla® to compare and merge with the History view, instead of using the default Eclipse compare tool.

**Compare Two Backup History Elements Using DIFFzilla®**

- Right-click on a file in an Eclipse explorer view (Package Explorer, Project Explorer, etc.) and select Compare With → Local History. This will activate the History view if necessary.

- Select the two revisions that you want to compare, right-click, and select Compare with Each Other (DIFFzilla).

**Compare a Local Backup History Element with the Current Version Using DIFFzilla®**

- Right-click on a file in an Eclipse explorer view (Package Explorer, Project Explorer, etc.) and select Compare With → Local History. This will activate the History view if necessary.

- Select the revision that you want to compare, right-click, and select Compare Current with Local (DIFFzilla).
Spell Check Operations

You can access spell checking operations from the main menu by clicking **Format → 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® Core creates new files in your configuration directory named userdct1.lst and userdct2.lst. 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 Format → 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 Format → 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® Core 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 **Format → 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Sort in ascending order.</td>
</tr>
<tr>
<td>D</td>
<td>Sort in descending order.</td>
</tr>
<tr>
<td>I</td>
<td>Case insensitive sort (ignore case).</td>
</tr>
<tr>
<td>E</td>
<td>Case sensitive sort (exact case which is the default).</td>
</tr>
<tr>
<td>-N</td>
<td>Numeric sort. C-style floating point numbers with up to 32-digit mantissa are supported.</td>
</tr>
<tr>
<td>-F</td>
<td>File name sort.</td>
</tr>
</tbody>
</table>
FTP support within SlickEdit® Core 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 views or by right-clicking on FTP files after a connection is active.

FTP View

There are two views available for working with FTP: FTP and FTP Client.

- The FTP view can be used to connect to FTP servers and open files. To access this view, from the main menu, click Window → Show View → Other, then expand SlickEdit and select FTP. Right-click on files to display a menu of FTP operations.

- The FTP Client view 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 view, and the FTP server directories and files are on the right. To access this view, from the main menu, click Window → Show View → Other, then expand SlickEdit and select 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 view (Window → Show View → Other, then expand SlickEdit and select FTP) and click the button labeled Start a New Session. The FTP Profile Manager dialog box is displayed, as pictured below.

Display the FTP view 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 views described above, and complete the following steps:

1. Click the **FTP** button ![FTP Button](image) 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 view 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 views, and complete the following steps:

1. Select the connection that you want from the drop-down list at the top of the view.

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 view, 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.

- **Default FTP options** - These are the general default FTP settings. To access these options, from the main menu, click **Window → SlickEdit Preferences**, expand **Network & Internet Options** and select **FTP Default Options**. See **FTP Default Options** for more information.
Regular Expressions

Overview

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® Core supports five types of regular expression syntax:

- Brief
- Perl
- SlickEdit
- UNIX
- Wildcards

SlickEdit Core 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.

See Regular Expression Syntax for charts of the expressions in syntax. 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.
- While regular expressions in SlickEdit Core primarily match the syntax for that language, there are some differences between our implementation and those elsewhere.
Using Regular Expressions in SlickEdit® Core

Specifying the Syntax to Use

All search and replace commands, the Find and Replace view, and incremental search support regular expressions. For search and replace commands and the view, 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 `B` to interpret the search string as Brief regular expression.
  - Use the `L` option to interpret the string as a Perl regular expression.
  - Use `R` to interpret the string as a SlickEdit regular expression.
  - Use the `U` option to interpret the string as a UNIX regular expression.

- **Find and Replace view** - When using the view, 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 Window → SlickEdit Preferences, 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`:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief</td>
<td><code>&lt;\*\*\*\</code></td>
</tr>
<tr>
<td>Perl and UNIX</td>
<td><code>^\*\*\*</code></td>
</tr>
<tr>
<td>SlickEdit</td>
<td><code>^\*\*\*\</code></td>
</tr>
</tbody>
</table>
The following regular expressions, which use the maximal operator, match the string `\path\path2\path3`:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief</td>
<td><code>&lt;\:\:*\:\</code></td>
</tr>
<tr>
<td>Perl and UNIX</td>
<td><code>^\:\:*\</code></td>
</tr>
<tr>
<td>SlickEdit</td>
<td><code>^\:\:*\</code></td>
</tr>
</tbody>
</table>

As a rule of thumb, the following minimal matching operators are generally used after a less-specific regular expression such as `.` in Brief/SlickEdit or `.` in UNIX:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief</td>
<td><code>@</code> and <code>+</code></td>
</tr>
<tr>
<td>Perl and UNIX</td>
<td><code>*?</code> and <code>+?</code></td>
</tr>
<tr>
<td>SlickEdit</td>
<td><code>*</code> and <code>+</code></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Expression</th>
</tr>
</thead>
</table>
| Brief      | Search for: `{[0-9]::+}`  
Replace with: `$\0` |
| Perl and UNIX | Search for: `([0-9]+)`  
Replace with: `$\1` |
| SlickEdit  | Search for: `{[0-9]#}`  
Replace with: `#$0` |

If the minimal matching operator (`+` in Brief/SlickEdit syntax, `+?` in Perl/UNIX) was used in the search string instead of the maximal matching operator (`\:@` in Brief, `+` in Perl/UNIX, `#` in SlickEdit Core), the above search and replace would prefix each digit in the entire file with a `$` character.

**Using 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 are denoted based on the syntax:

- **Brief syntax** - Use {} (curly braces) to denote a tagged expression in the search string. The replace string specifies tagged expressions with a \ (backslash) followed by a tagged expression number 0-9. Count the {} (left braces) in the search string to determine a tagged expression number. The first tagged expression is \0.

- **UNIX syntax** - Use ( ) (parentheses) to denote a tagged expression in the search string. The replace string specifies tagged expressions with a \ (backslash) followed by a tag group number 1-9. Count the ( (left parenthesis) in the search string to determine a tagged expression number. The first tagged expression is \1 and the last is \0.

- **Perl syntax** - Use ( ) (parentheses) to denote a tagged expression in the search string. The replace string specifies tagged expressions with a \ (backslash) or $ (dollar sign) followed by a tag group number 1-9. Count the ( (left parenthesis) in the search string to determine a tagged expression number. The first tagged expression is \1 and the last is \0.

- **SlickEdit® Core syntax** - Use {} (curly braces) to denote a tagged expression in the search string. The replace string specifies tagged expressions with a # (pound sign) followed by a tagged expression number 0-9. Count the {} (left braces) in the search string to determine a tagged expression number. The first tagged expression is #0.

Examples of Tagged Expressions

**Example 1: Replace Occurrences**

The expressions in the table below replace occurrences of “if” and “while” with “xify” and “xwhiley.” Unmatched groups are null. Note that the \1 in Brief syntax, \2 in Perl/UNIX, and #1 in SlickEdit syntax are replaced with null.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief</td>
<td>Search for: {{if}</td>
</tr>
<tr>
<td></td>
<td>Replace with: x\0y\1</td>
</tr>
<tr>
<td>Perl/UNIX</td>
<td>Search for: (if</td>
</tr>
<tr>
<td></td>
<td>Replace with: x\1y\2</td>
</tr>
<tr>
<td>SlickEdit</td>
<td>Search for: {if</td>
</tr>
<tr>
<td></td>
<td>Replace with: x#0y#1</td>
</tr>
</tbody>
</table>

**Example 2: Reverse Text on Lines Containing a Comma**
The expressions in the table below reverse text on lines containing a comma. Lines with "abc,def" will be changed to "def,abc". Notice that the Perl/UNIX regular expression search string uses a `*?` minimal matching operator, so the comma actually matches the first comma in the line and not the last.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief</td>
<td>Search for: ^{<em>},{</em>}$</td>
</tr>
<tr>
<td></td>
<td>Replace with: \1,\0</td>
</tr>
<tr>
<td>Perl/UNIX</td>
<td>Search for: ^([-]?),(*$)</td>
</tr>
<tr>
<td></td>
<td>Replace with: \2,\1</td>
</tr>
<tr>
<td>SlickEdit</td>
<td>Search for: ^{?<em>},{?</em>}$</td>
</tr>
<tr>
<td></td>
<td>Replace with: #1,#0</td>
</tr>
</tbody>
</table>

### Replacing with Regular Expressions

When using regular expressions, some characters have a different meaning when used in the replace string, depending on the syntax:

- **Brief** - The backslash character (\) in the replace string has the same meaning as in the search string except that \c and \:char are not supported.

- **UNIX** - A backslash in the replace string has the same meaning as in the search string except that \c and \:char are not supported.

- **Perl** - A backslash in the replace string has the same meaning as in the search string except that \c and \:char are not supported. A dollar sign ($) must be escaped (\$) when replacing a literal $.

- **SlickEdit® Core** - 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 that \c, :char, and \gd are not supported.

See the [Regular Expression Syntax](#) tables for a list of options for these characters. See [Using 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 Brief, Perl, SlickEdit, and UNIX syntaxes.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\l</td>
<td>Convert next character lowercase.</td>
</tr>
</tbody>
</table>
### Using Regular Expressions in SlickEdit® Core

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\u</td>
<td>Convert next character uppercase.</td>
</tr>
<tr>
<td>\L</td>
<td>Convert all characters lowercase until \E.</td>
</tr>
<tr>
<td>\U</td>
<td>Convert all characters uppercase until \E.</td>
</tr>
<tr>
<td>\Q</td>
<td>Replace all characters literally until \E.</td>
</tr>
<tr>
<td>\E</td>
<td>End all case modification or \Q.</td>
</tr>
</tbody>
</table>

#### Examples of Replacing with Regular Expressions

The table below contains some examples of replace operations using regular expressions.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Expression</th>
</tr>
</thead>
</table>
| Search for occurrences of the string "hat" that occur at the end of a line and replace it with "cat". | **In all syntaxes:**  
Search for: hat$  
Replace with: cat |
| Delete blank lines.                                                        | **Brief:**  
Search for: \n  
Replace with: (leave blank)  
**Perl/SlickEdit/UNIX:**  
Search for: ^\n  
Replace with: (leave blank) |
| Replace occurrences of two consecutive blank lines with one.              | **Brief:**  
Search for: \n\n  
Replace with: \n  
**Perl/SlickEdit/UNIX:**  
Search for: ^\n\n  
Replace with: \n |
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 views in SlickEdit® Core, this view is dockable. Docking options can be accessed by right-clicking on the view's title bar.
Type some samples of the text you are trying to match in the top portion of the view 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: UNIX, SlickEdit® Core, Brief, or Perl. 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 view:
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® Core 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 view of all entries in order to start a new evaluation, click the button at the top of the view 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 view labeled Open a saved expression.

Save the current expression button - To save the current expression, click the diskette button at the top of the view 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 view labeled Save the current expression as.

Regular Expression Syntax

This section provides charts of regular expressions for each supported syntax (Brief, Perl, SlickEdit, UNIX, and Wildcards), including examples.

**Note**

There are some differences between our implementation of these syntaxes and those elsewhere.

**Brief Regular Expressions**

Brief regular expressions are defined in the following table.

<table>
<thead>
<tr>
<th>Brief Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Matches beginning of line.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Matches beginning of line.</td>
</tr>
<tr>
<td>$</td>
<td>Matches end of line.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Matches end of line.</td>
</tr>
<tr>
<td>?</td>
<td>Matches any character except newline.</td>
</tr>
<tr>
<td>*</td>
<td>Minimal match of zero or more of any character</td>
</tr>
<tr>
<td>Brief Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>except newline. This is the same as <code>?@</code>.</td>
</tr>
<tr>
<td>$X+$</td>
<td>Minimal match of one or more occurrences of $X$. See <a href="#">Minimal versus Maximal Matching</a> for more information.</td>
</tr>
<tr>
<td>$X:\ast$</td>
<td>Maximal match of zero or more of any character except newline. This is the same as <code>?:@</code>.</td>
</tr>
<tr>
<td>$X:@$</td>
<td>Maximal match of zero or more occurrences of $X$.</td>
</tr>
<tr>
<td>$X:+$</td>
<td>Maximal match of one or more occurrences of $X$.</td>
</tr>
<tr>
<td>$X:n_1$</td>
<td>Matches exactly $n_1$ occurrences of $X$. Use <code>{}</code> to avoid ambiguous expressions. For example, <code>a:9{1}</code> searches for nine instances of the letter &quot;a&quot; followed by a &quot;1&quot;.</td>
</tr>
<tr>
<td>$X:n_1, n_2$</td>
<td>Maximal match of at least $n_1$ occurrences but not more than $n_2$ occurrences of $X$.</td>
</tr>
<tr>
<td>$X:n_1, n_2?$</td>
<td>Minimal match of at least $n_1$ occurrences but not more than $n_2$ occurrences of $X$.</td>
</tr>
<tr>
<td>$X:n_1, n_2$</td>
<td>Maximal match of at least $n_1$ occurrences but not more than $n_2$ occurrences of $X$.</td>
</tr>
<tr>
<td>$X:n_1?$</td>
<td>Match exactly $n_1$ occurrences of $X$.</td>
</tr>
<tr>
<td>$X:n_1,?$</td>
<td>Minimal match of at least $n_1$ occurrences of $X$.</td>
</tr>
<tr>
<td>$X:n_2?$</td>
<td>Minimal match of at least zero occurrences but not more than $n_2$ occurrences of $X$.</td>
</tr>
<tr>
<td>$X:n_1, n_2?$</td>
<td>Minimal match of at least $n_1$ occurrences but not more than $n_2$ occurrences of $X$.</td>
</tr>
<tr>
<td><code>(X)</code></td>
<td>Matches subexpression $X$ but does not define a tagged expression.</td>
</tr>
<tr>
<td><code>{X}</code></td>
<td>Matches subexpression $X$ and specifies a new tagged expression. See <a href="#">Using Tagged Expressions</a> for more information.</td>
</tr>
</tbody>
</table>
| `{@ d X}` | Matches subexpression $X$ and specifies to use
### Regular Expression Syntax

<table>
<thead>
<tr>
<th>Brief Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagged expression number ( d ) where ( 0 \leq d \leq 9 ). No more tagged expressions are defined by the subexpression syntax ( { X } ) once this subexpression syntax is used. This is the best way to make sure you have enough tagged expressions.</td>
<td></td>
</tr>
<tr>
<td>( X</td>
<td>Y )</td>
</tr>
<tr>
<td>( \neg{X} )</td>
<td>Search fails if expression ( X ) is matched.</td>
</tr>
<tr>
<td>([ \text{char-set} ])</td>
<td>Matches any one of the characters specified by ( \text{char-set} ). A dash (-) character may be used to specify ranges. The expression ([\text{A-Z}] matches any uppercase letter. Backslash () can be used inside the square brackets to define literal characters or define ASCII characters. For example, ( \text{-} ) specifies a literal dash character. The expression ([0-27]) matches ASCII character codes ( 0..27 ). The expression ([]) matches a right bracket. In SlickEdit® Core regular expressions, ([]) matches no characters. In both syntaxes, the expression ([\text{-}]) matches a right bracket.</td>
</tr>
<tr>
<td>([\neg \text{char-set}])</td>
<td>Matches any character not specified by ( \text{char-set} ). A dash (-) character may be used to specify ranges. The expression ([\neg\text{A-Z}]) matches all characters except uppercase letters. The expression ([\text{-}]) matches any character except newline.</td>
</tr>
<tr>
<td>([ \text{char-set1} - \text{ [char-set2] } ])</td>
<td>Character set subtraction. Matches all characters in \text{char-set1} except the characters in \text{char-set2}. For example, ([\text{a-z-[qw]}]) matches all English lowercase letters except &quot;q&quot; and &quot;w&quot;. ([\text{p(L)}-[qw]}]) matches all Unicode lowercase letters except &quot;q&quot; and &quot;w&quot;.</td>
</tr>
<tr>
<td>([ \text{char-set1} &amp; \text{ [char-set2] } ])</td>
<td>Character set intersection. Matches all characters in \text{char-set1} that are also in \text{char-set2}. For example, ([\text{x[0]-x7f}&amp;[\text{p(L)}]}) matches all letters between 0 and 127.</td>
</tr>
<tr>
<td>(\text{x} \ hhhh )</td>
<td>Matches up to 31-bit Unicode hexadecimal character specified by ( hhhh ).</td>
</tr>
<tr>
<td>(\text{p} \ UnicodeCategorySpec )</td>
<td>(Only valid in character set) Matches characters in</td>
</tr>
<tr>
<td>Brief Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>UnicodeCategorySpec. Where UnicodeCategorySpec uses the standard general categories specified by the Unicode consortium. For example, [p{L}] matches all letters. [p{Lu}] matches all uppercase letters. See Unicode Category Specifications for Regular Expressions.</td>
<td></td>
</tr>
<tr>
<td>\P{ UnicodeCategorySpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodeCategorySpec. For example, [\P{L}] matches all characters that are not letters. This is equivalent to [^{p{L}}]. [p{Lu}] matches all characters that are not uppercase letters. See Unicode Category Specifications for Regular Expressions.</td>
</tr>
<tr>
<td>\p{ UnicodeIsBlockSpec }</td>
<td>(Only valid in character set) Matches characters in UnicodeIsBlockSpec. Where UnicodeIsBlockSpec one of the standard character blocks specified by the Unicode consortium. For example, [p{isGreek}] matches Unicode characters in the Greek block. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>\P{ UnicodeIsBlockSpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodeIsBlockSpec. For example, [\P{isGreek}] matches all characters that are not in the Unicode Greek block. This is equivalent to [^{p{isGreek}}]. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>\x hh</td>
<td>Matches hexadecimal character (hh) where (0&lt;=hh&lt;=0xff).</td>
</tr>
<tr>
<td>\d ddd</td>
<td>Matches decimal character (ddd) where (0&lt;=ddd&lt;=255).</td>
</tr>
<tr>
<td>\d</td>
<td>Defines a back reference to tagged expression number (d). For example, (abc)def(0) matches the string abcdefabc. If the tagged expression has not been set, the search fails.</td>
</tr>
<tr>
<td>\c</td>
<td>Specifies cursor position if match is found. If the expression xyz(c) is found, the cursor is placed after z.</td>
</tr>
<tr>
<td>Brief Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>\n</td>
<td>Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file. Use \d10 if you want to match a 10 character.</td>
</tr>
<tr>
<td>\r</td>
<td>Matches carriage return.</td>
</tr>
<tr>
<td>\t</td>
<td>Matches tab character.</td>
</tr>
<tr>
<td>\b</td>
<td>Matches at word boundary. For example, \bre matches all occurrences of &quot;re&quot; that only occur at the beginning of a word. Note that this notation previously matched a backspace character. It can still be used to match a backspace character by using it in a character set (for example, [\b]).</td>
</tr>
<tr>
<td>\B</td>
<td>Matches all except at word boundary. For example, \Bre matches all occurrences of &quot;re&quot; as long as it is not at the start of a new word.</td>
</tr>
<tr>
<td>\Q and \E</td>
<td>\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.</td>
</tr>
<tr>
<td>\f</td>
<td>Matches form feed character.</td>
</tr>
<tr>
<td>\od</td>
<td>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.</td>
</tr>
<tr>
<td>\om</td>
<td>Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end-of-line characters. For example, \om?@ matches the rest of the buffer.</td>
</tr>
<tr>
<td>Brief Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td><code>\ol</code></td>
<td>Turns off multi-line matching (default). You can still use <code>\n</code> to create regular expressions which match one or more lines. However, expressions like <code>\?@</code> will not match multiple lines. This is much safer and usually faster than using the <code>\om</code> option.</td>
</tr>
<tr>
<td><code>\oi</code></td>
<td>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. See also <code>\oc</code>.</td>
</tr>
<tr>
<td><code>\oc</code></td>
<td>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 <code>\oi</code>.</td>
</tr>
<tr>
<td><code>\ char</code></td>
<td>Declares character after slash to be literal. For example, <code>\*</code> represents the asterisk (<code>*</code>) character.</td>
</tr>
<tr>
<td><code>\: char</code></td>
<td>Matches predefined expression corresponding to <code>char</code>. The predefined expressions are:</td>
</tr>
<tr>
<td></td>
<td>- <code>\a [A-Za-z0-9]</code> - Matches an alphanumeric character.</td>
</tr>
<tr>
<td></td>
<td>- <code>\b (\[\t\]#)</code> - Matches blanks.</td>
</tr>
<tr>
<td></td>
<td>- <code>\c [A-Za-z]</code> - Matches an alphabetic character.</td>
</tr>
<tr>
<td></td>
<td>- <code>\d [0-9]</code> - Matches a digit.</td>
</tr>
<tr>
<td></td>
<td>- `\f (([<del>[]</del>/&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>- `\f (([<del>[]</del>/&lt;&gt;</td>
</tr>
<tr>
<td></td>
<td>- <code>\h ([0-9A-Fa-f]#)</code> - Matches a hex number.</td>
</tr>
<tr>
<td></td>
<td>- <code>\i ([0-9]#)</code> - Matches an integer.</td>
</tr>
<tr>
<td></td>
<td>- `\m ([0-9]#([0-9]#([Ee]((+-</td>
</tr>
</tbody>
</table>
Brief Regular Expression

<table>
<thead>
<tr>
<th>Brief Regular Expression Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;defproc</td>
<td>Matches lines that begin with the word defproc.</td>
</tr>
</tbody>
</table>
### Brief Regular Expression Example

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;definit&gt;</code></td>
<td>Matches lines that only contain the word <em>definit</em>.</td>
</tr>
</tbody>
</table>
| `<\*name`          | Matches lines that begin with the string *name*. Notice that the backslash must prefix the special character *.
| `[t ]`             | Matches tab and space characters. |
| `[d9\d32]`         | Matches tab and space characters. |
| `[\x9\x20]`       | Matches tab and space characters. |
| p?t                | Matches any three-letter string starting with the letter p and ending with the letter t. Two possible matches are *pot* and *pat*. |
| s*t                | Matches the letter s followed by any number of characters followed by the nearest letter t. Two possible matches are *seat* and *st*. |
| `{for}|{while}      | Matches the strings *for* or *while*. |
| ^\:p              | Matches lines beginning with a file name. |
| xy+z              | Matches x followed by one or more occurrences of y followed by z. |
| \x0d\x0a\x01\x02 | Matches a sequence of hex binary characters. |
| \d13\d10\d1\d2   | Matches a sequence of decimal binary characters. |

### Perl Regular Expressions

Perl regular expressions are defined in the following table.

<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Matches beginning of line.</td>
</tr>
<tr>
<td>$</td>
<td>Matches end of line.</td>
</tr>
<tr>
<td>.</td>
<td>Matches any character except newline.</td>
</tr>
</tbody>
</table>
### Perl Regular Expression Syntax

<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>X+</td>
<td>Maximal match of one or more occurrences of X. See Minimal versus Maximal Matching.</td>
</tr>
<tr>
<td>X*</td>
<td>Maximal match of zero or more occurrences of X.</td>
</tr>
<tr>
<td>X?</td>
<td>Maximal match of zero or one occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 }</td>
<td>Match exactly (n_1) occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 ,}</td>
<td>Maximal match of at least (n_1) occurrences of X.</td>
</tr>
<tr>
<td>X{, n2 }</td>
<td>Maximal match of at least zero occurrences but not more than (n_2) occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 , n2 }</td>
<td>Maximal match of at least (n_1) occurrences but not more than (n_2) occurrences of X.</td>
</tr>
<tr>
<td>X+?</td>
<td>Minimal match of one or more occurrences of X.</td>
</tr>
<tr>
<td>X*?</td>
<td>Minimal match of zero or more occurrences of X.</td>
</tr>
<tr>
<td>X??</td>
<td>Minimal match of zero or one occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 }?</td>
<td>Matches exactly (n_1) occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 ,}?</td>
<td>Minimal match of at least (n_1) occurrences of X.</td>
</tr>
<tr>
<td>X{, n2 }?</td>
<td>Minimal match of at least zero occurrences but not more than (n_2) occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 , n2 }?</td>
<td>Minimal match of at least (n_1) occurrences but not more than (n_2) occurrences of X.</td>
</tr>
<tr>
<td>(?!X)</td>
<td>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>.</td>
</tr>
<tr>
<td>(?=X)</td>
<td>Assert, positive lookahead. Searches for subexpression X, but X is not returned as part of the match. For example, to match words ending in &quot;ed&quot; while excluding &quot;ed&quot; as part of the match, use <code>\b[a-z]+(?=ed\b)</code>. See also (?!X).</td>
</tr>
<tr>
<td>(?&gt;X)</td>
<td>Prohibit backtracking. This expression is advanced</td>
</tr>
</tbody>
</table>

---

**Regular Expression Syntax**

391
<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>usage. It can be used to prevent the subexpression X from backtracking when using maximal (greedy) matching.</td>
<td></td>
</tr>
<tr>
<td>(?#text)</td>
<td>Comment. No text is matched in this expression; it is used for comment and documentation only.</td>
</tr>
<tr>
<td>(X)</td>
<td>Matches subexpression X and specifies a new tagged expression (see Using Tagged Expressions). No more tagged expressions are defined once an explicit tagged expression number is specified as shown below.</td>
</tr>
<tr>
<td>(? d’X)</td>
<td>Matches subexpression X and specifies to use tagged expression number d where 0&lt;=d&lt;=9. No more tagged expressions are defined by the subexpression syntax (X) once this subexpression syntax is used. This is the best way to make sure you have enough tagged expressions.</td>
</tr>
<tr>
<td>(?:X)</td>
<td>Matches subexpression X but does not define a tagged expression.</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>[ char-set ]</td>
<td>Matches any one of the characters specified by char-set. A dash (-) character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. A backslash () may be used inside the square brackets to define literal characters or define ASCII characters. For example, - specifies a literal dash character. The expression [\d0-\d27] matches ASCII character codes 0..27. The expression [] matches a right bracket. In SlickEdit® Core regular expressions, [] matches no characters. In both syntaxes, the expression [[]] matches a right bracket. The expression [^] matches a caret (^) character but this does not work for SlickEdit regular expressions. In both syntaxes, [^] matches a caret (^) character.</td>
</tr>
<tr>
<td>[^ char-set ]</td>
<td>Matches any character not specified by char-set. A dash (-) character may be used to specify ranges.</td>
</tr>
</tbody>
</table>
| [ char-set1 - [ char-set2 ]] | Character set subtraction. Matches all characters in
### Perl Regular Expression Syntax

<table>
<thead>
<tr>
<th>Perl Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>char-set1 except the characters in char-set2. The expression [^A-Z] matches all characters except uppercase letters. For example, [a-z-[qw]] matches all English lowercase letters except q and w. [\p{L}-[qw]] matches all Unicode lowercase letters except q and w.</td>
<td></td>
</tr>
<tr>
<td>[ char-set1 &amp; [ char-set2 ] ]</td>
<td>Character set intersection. Matches all characters in char-set1 that are also in char-set2. For example, [x(0)-x(7f)&amp;[p(L)]] matches all letters between 0 and 127.</td>
</tr>
<tr>
<td>\x{ hhhh }</td>
<td>Matches up to 31-bit Unicode hexadecimal character specified by hhhh.</td>
</tr>
<tr>
<td>\p{ UnicodeCategorySpec }</td>
<td>(Only valid in character set) Matches characters in UnicodeCategorySpec. Where UnicodeCategorySpec uses the standard general categories specified by the Unicode consortium. For example, \p{L} matches all letters. \p{Lu} matches all uppercase letters. See Unicode Category Specifications for Regular Expressions.</td>
</tr>
<tr>
<td>\P{ UnicodeCategorySpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodeCategorySpec. For example, \P{L} matches all characters that are not letters. This is equivalent to [^p{L}]. \P{Lu} matches all characters that are not uppercase letters. See Unicode Category Specifications for Regular Expressions.</td>
</tr>
<tr>
<td>\p{ UnicodelsBlockSpec }</td>
<td>(Only valid in character set) Matches characters in UnicodelsBlockSpec. Where UnicodelsBlockSpec one of the standard character blocks specified by the Unicode consortium. For example, \p{isGreek} matches Unicode characters in the Greek block. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>\P{ UnicodelsBlockSpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodelsBlockSpec. For example, \P{isGreek} matches all characters that are not in the Unicode Greek block. This is equivalent to [^\p{isGreek}]. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>Perl Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>\x hh</td>
<td>Matches hexadecimal character ( hh ) where ( 0 \leq hh \leq 0xff ).</td>
</tr>
<tr>
<td>#ddd</td>
<td>Matches decimal character where ( 0 \leq ddd \leq 255 ).</td>
</tr>
<tr>
<td>\d</td>
<td>Defines a back reference to tagged expression number ( d ). For example, <code>(abc)def\1</code> matches the string <code>abcdefabc</code>. If the tagged expression has not been set, the search fails.</td>
</tr>
<tr>
<td>\D</td>
<td>Equivalent to <code>[0-9]</code>. Can also be used inside a character class. For example, <code>[A-F\d]</code> is equivalent to <code>[A-F0-9]</code>.</td>
</tr>
<tr>
<td>\W</td>
<td>Equivalent to <code>[a-zA-Z0-9_]</code>. Can also be used inside a character class.</td>
</tr>
<tr>
<td>\W</td>
<td>Equivalent to <code>[^a-zA-Z0-9_]</code>. Can also be used inside a character class.</td>
</tr>
<tr>
<td>\s</td>
<td>Equivalent to <code>[\n\r\f]</code>. Can also be used inside a character class.</td>
</tr>
<tr>
<td>\S</td>
<td>Equivalent to <code>[^\n\r\f]</code>. Can also be used inside a character class.</td>
</tr>
<tr>
<td>\ooo</td>
<td>Octal ASCII value.</td>
</tr>
<tr>
<td>\cx cx</td>
<td>Control character (ASCII values 0-31) <code>'@' &lt;=x'&lt;='</code></td>
</tr>
<tr>
<td>\z z</td>
<td>Specifies cursor position if match is found. If the expression <code>abc\z</code> is found, the cursor is placed after the <code>c</code>. Note that in UNIX, this is the same as <code>\c</code>. However in Perl, <code>\c</code> is used only for control characters.</td>
</tr>
<tr>
<td>\n n</td>
<td>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.</td>
</tr>
<tr>
<td>Perl Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Use \d10 if you want to match an ASCII 10 character.</td>
<td></td>
</tr>
<tr>
<td>\r</td>
<td>Matches carriage return (ASCII 13). What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file.</td>
</tr>
<tr>
<td>\t</td>
<td>Matches tab character.</td>
</tr>
<tr>
<td>\b</td>
<td>Matches at word boundary. For example, \bre matches all occurrences of &quot;re&quot; that only occur at the beginning of a word.</td>
</tr>
<tr>
<td>\B</td>
<td>Matches all except at word boundary. For example, \B re matches all occurrences of &quot;re&quot; as long as it is not at the start of a new word.</td>
</tr>
<tr>
<td>\Q and \E</td>
<td>\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.</td>
</tr>
<tr>
<td>\f</td>
<td>Matches form feed character.</td>
</tr>
<tr>
<td>\od</td>
<td>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.</td>
</tr>
<tr>
<td>\om</td>
<td>Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end-of-line characters. For example, \om.+ matches the rest of the buffer.</td>
</tr>
<tr>
<td>\ol</td>
<td>Turns off multi-line matching (default). You can still use \n to create regular expressions which match one or more lines. However, expressions like .+ will not match multiple lines. This is much safer and</td>
</tr>
<tr>
<td>Perl Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>\oi</td>
<td>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. See also \oc.</td>
</tr>
<tr>
<td>\oc</td>
<td>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 the Perl syntax ?-i. See also \oi.</td>
</tr>
<tr>
<td>\char</td>
<td>Declares character after slash to be literal. For example, * represents the star character.</td>
</tr>
</tbody>
</table>
| \: char                 | Matches predefined expression corresponding to char. The pre-defined expressions are:  
  - \:c [A-Za-z] - Matches an alphabetic character.  
  - \:b (?:[ \t]+) - Matches blanks.  
  - \:d [0-9] - Matches a digit.  
  - \:f (?:[^\[][^\t \n<>|\=-\+\,\s\"\']+) - 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.  
Perl Regular Expression Definition

- \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.

**Warning**

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.

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.

The precedence of operators, from highest to lowest, is as follows:

- \+, \*, \?, \{|\}, \+?, \*?, \??, \{\}? (These operators have the same precedence.)
- concatenation
- |
<table>
<thead>
<tr>
<th>Perl Regular Expression Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^defproc</code></td>
<td>Matches lines that begin with the word <code>defproc</code>.</td>
</tr>
<tr>
<td><code>^definit$</code></td>
<td>Matches lines that only contain the word <code>definit</code>.</td>
</tr>
<tr>
<td><code>^\*name</code></td>
<td>Matches lines that begin with the string <code>\*name</code>. Notice that the backslash must prefix the special character <code>\</code>.</td>
</tr>
<tr>
<td><code>[t ]</code></td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td><code>[d9\d32]</code></td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td><code>[x9\x20]</code></td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td><code>p.t</code></td>
<td>Matches any three-letter string starting with the letter <code>p</code> and ending with the letter <code>t</code>. Two possible matches are <code>pot</code> and <code>pat</code>.</td>
</tr>
<tr>
<td><code>s.*?t</code></td>
<td>Matches the letter <code>s</code> followed by any number of characters followed by the nearest letter <code>t</code>. Two possible matches are <code>seat</code> and <code>st</code>.</td>
</tr>
<tr>
<td>`for</td>
<td>while`</td>
</tr>
<tr>
<td><code>^:\p</code></td>
<td>Matches lines beginning with a file name.</td>
</tr>
<tr>
<td><code>xy+z</code></td>
<td>Matches <code>x</code> followed by one or more occurrences of <code>y</code> followed by <code>z</code>.</td>
</tr>
<tr>
<td><code>[a-z-[qw]]</code></td>
<td>Character set subtraction. Matches all English lowercase letters except <code>q</code> and <code>w</code>.</td>
</tr>
<tr>
<td><code>[^p(isGreek)&amp;[^p(L)]]</code></td>
<td>Character set intersection. Matches all Unicode letters in the Greek block.</td>
</tr>
<tr>
<td><code>\x{6587}</code></td>
<td>Matches Unicode character with hexadecimal value <code>6587</code>. Character set intersection. Matches all Unicode letters in the Greek block.</td>
</tr>
<tr>
<td><code>[^p(L)-[qw]]</code></td>
<td>Matches all Unicode letters except <code>q</code> and <code>w</code>.</td>
</tr>
<tr>
<td><code>[^p(L)]</code></td>
<td>Matches all Unicode letters.</td>
</tr>
<tr>
<td><code>[^p(Lul)]</code></td>
<td>Matches all Unicode uppercase and lowercase</td>
</tr>
</tbody>
</table>
### Perl Regular Expression Example

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\P{L}]</td>
<td>Matches all Unicode characters that are not letters.</td>
</tr>
<tr>
<td>[\p{isGreek}]</td>
<td>Matches all Unicode characters in the Greek block.</td>
</tr>
<tr>
<td>\x0d\x0a\x01\x02</td>
<td>Matches a sequence of hex binary characters.</td>
</tr>
<tr>
<td>\d13\d10\d1\d2</td>
<td>Matches a sequence of decimal binary characters.</td>
</tr>
</tbody>
</table>

### SlickEdit Regular Expressions

SlickEdit regular expressions are defined in the following table.

<table>
<thead>
<tr>
<th>SlickEdit Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Matches beginning of line.</td>
</tr>
<tr>
<td>$</td>
<td>Matches end of line.</td>
</tr>
<tr>
<td>?</td>
<td>Matches any character except newline.</td>
</tr>
<tr>
<td>X+</td>
<td>Minimal match of one or more occurrences of X. See Minimal versus Maximal Matching for more information.</td>
</tr>
<tr>
<td>X#</td>
<td>Maximal match of one or more occurrences of X.</td>
</tr>
<tr>
<td>X*</td>
<td>Minimal match of zero or more occurrences of X.</td>
</tr>
<tr>
<td>X@</td>
<td>Maximal match of zero or more occurrences of X.</td>
</tr>
<tr>
<td>X: n1</td>
<td>Matches exactly n1 occurrences of X. Use () to avoid ambiguous expressions. For example a:9()1 searches for nine instance of the letter a followed by a 1.</td>
</tr>
<tr>
<td>X: n1,</td>
<td>Maximal match of at least n1 occurrences of X.</td>
</tr>
<tr>
<td>X: n1, n2</td>
<td>Maximal match of at least n1 occurrences but not more than n2 occurrences of X.</td>
</tr>
</tbody>
</table>
### SlickEdit Regular Expression Syntax

<table>
<thead>
<tr>
<th>SlickEdit Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>X:* (n_1)</td>
<td>Minimal match of at least (n_1) occurrences of (X).</td>
</tr>
<tr>
<td>X:* (n_1, n_2)</td>
<td>Minimal match of at least (n_1) occurrences but not more than (n_2) occurrences of (X).</td>
</tr>
<tr>
<td>(~X)</td>
<td>Search fails if expression (X) is matched. The expression (^~(\text{if})) matches the beginning of all lines that do not start with (\text{if}).</td>
</tr>
<tr>
<td>((X))</td>
<td>Matches subexpression (X).</td>
</tr>
<tr>
<td>{X}</td>
<td>Matches subexpression (X) and specifies a new tagged expression. See Using Tagged Expressions for more information.</td>
</tr>
<tr>
<td>{# d X}</td>
<td>Matches subexpression (X) and specifies to use tagged expression number (d) where (0\leq d\leq 9).</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>[ char-set ]</td>
<td>Matches any one of the characters specified by (char-set). A dash (-) character may be used to specify ranges. The expression ([A-Z]) matches any uppercase letter. Backslash () may be used inside the square brackets to define literal characters or define ASCII characters. For example, ([-) specifies a literal dash character. The expression ([0-27]) matches ASCII character codes (0..27). The expression ([]) matches no characters. In UNIX regular expressions, ([]) matches a right bracket. In both syntaxes, the expression ([^]) matches a caret (^) character in both syntaxes.</td>
</tr>
<tr>
<td>[^ char-set ]</td>
<td>Matches any character not specified by (char-set). A dash (-) character may be used to specify ranges. The expression [^A-Z] matches all characters except uppercase letters.</td>
</tr>
<tr>
<td>[^ char-set ]</td>
<td>Same as [^char-set] above.</td>
</tr>
</tbody>
</table>
| \^[ char-set \]             | Character set subtraction. Matches all characters in \(char-set1\) except the characters in \(char-set2\). For example, \([a-z-[qw]]\) matches all English lowercase
<table>
<thead>
<tr>
<th>SlickEdit Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ char-set1 &amp; [ char-set2 ] ]</td>
<td>Character set intersection. Matches all characters in char-set1 that are also in char-set2. For example, [x(0)-x(7f)&amp;[p(L)]] matches all letters between 0 and 127.</td>
</tr>
<tr>
<td>\x{ hhhh }</td>
<td>Matches up to 31-bit Unicode hexadecimal character specified by hhhh.</td>
</tr>
<tr>
<td>\p{ UnicodeCategorySpec }</td>
<td>(Only valid in character set) Matches characters in UnicodeCategorySpec. Where UnicodeCategorySpec uses the standard general categories specified by the Unicode consortium. For example, [p(L)] matches all letters. [p(Lu)] matches all uppercase letters. See Unicode Category Specifications for Regular Expressions.</td>
</tr>
<tr>
<td>\P{ UnicodeCategorySpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodeCategorySpec. For example, [P(L)] matches all characters that are not letters. This is equivalent to [^p(L)]. [P(Lu)] matches all characters that are not uppercase letters. See Unicode Category Specifications for Regular Expressions.</td>
</tr>
<tr>
<td>\p{ UnicodesBlockSpec }</td>
<td>(Only valid in character set) Matches characters in UnicodesBlockSpec. Where UnicodesBlockSpec one of the standard character blocks specified by the Unicode consortium. For example, [p{isGreek}] matches Unicode characters in the Greek block. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>\P{ UnicodesBlockSpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodesBlockSpec. For example, [P{isGreek}] matches all characters that are not in the Unicode Greek block. This is equivalent to [^p{isGreek}]. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>\x hh</td>
<td>Matches hexadecimal character hh where 0&lt;=hh&lt;=0xff.</td>
</tr>
</tbody>
</table>
## Regular Expression Syntax

<table>
<thead>
<tr>
<th>SlickEdit Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ddd</td>
<td>Matches decimal character \ddd where 0&lt;=\ddd&lt;=255.</td>
</tr>
<tr>
<td>\g d</td>
<td>Defines a back reference to tagged expression number d. For example, {abc}def\g0 matches the string abcdefabc. If the tagged expression has not been set, the search fails.</td>
</tr>
<tr>
<td>\c</td>
<td>Specifies cursor position if match is found. If the expression xyz\c is found, the cursor is placed after the z.</td>
</tr>
<tr>
<td>\n</td>
<td>Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user-defined ASCII file. Use \d10 if you want to match an ASCII 10 character.</td>
</tr>
<tr>
<td>\r</td>
<td>Matches carriage return.</td>
</tr>
<tr>
<td>\t</td>
<td>Matches tab character.</td>
</tr>
<tr>
<td>\b</td>
<td>Matches at word boundary. For example, \bre matches all occurrences of &quot;re&quot; that only occur at the beginning of a word. Note that this notation previously matched a backspace character. It can still be used to match a backspace character by using it in a character set (for example, [\b]).</td>
</tr>
<tr>
<td>\B</td>
<td>Matches all except at word boundary. For example, \Bre matches all occurrences of &quot;re&quot; as long as it is not at the start of a new word.</td>
</tr>
<tr>
<td>\Q and \E</td>
<td>\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.</td>
</tr>
<tr>
<td>\f</td>
<td>Matches form feed character.</td>
</tr>
<tr>
<td>SlickEdit Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>\od</td>
<td>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.</td>
</tr>
<tr>
<td>\om</td>
<td>Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end-of-line characters. For example, \om?# matches the rest of the buffer. <strong>NOTE</strong>: Test the regular expression on a very small file before using it on a large file. This option may cause the editor to use a lot of memory.</td>
</tr>
<tr>
<td>\ol</td>
<td>Turns off multi-line matching (default). You can still use \n 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.</td>
</tr>
<tr>
<td>\oi</td>
<td>Ignore case. Turns off case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside all ( ) and { } groups, after which case matching is restored to the previous case match setting. See also \oc.</td>
</tr>
<tr>
<td>\oc</td>
<td>Case-sensitive match. Turns on case-sensitive matching in the pattern, overriding the global case setting. This modifier is localized inside all ( ) and { } groups, after which case matching is restored to the previous case match setting. See also \oi.</td>
</tr>
<tr>
<td>\char</td>
<td>Declares character after slash to be literal. For example, : represents the colon character.</td>
</tr>
<tr>
<td>: char</td>
<td>Matches predefined expression corresponding to char. The predefined expressions are:</td>
</tr>
<tr>
<td></td>
<td>• :a [A-Za-z0-9] - Matches an alphanumeric character.</td>
</tr>
<tr>
<td></td>
<td>• :b ([ \t#]) - Matches blanks - note that :b is not like the Perl/.NET\s.</td>
</tr>
<tr>
<td>SlickEdit Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>:f ([<del>[[</del>] ][~][[]])</td>
<td>Windows: Matches a file name part.</td>
</tr>
<tr>
<td>:f ([<del>[[</del>] ][~][[]])</td>
<td>UNIX: Matches a file name part.</td>
</tr>
<tr>
<td>:h ([0-9A-Fa-f][#]#)</td>
<td>Matches a hex number.</td>
</tr>
<tr>
<td>:i ([0-9][#]#)</td>
<td>Matches an integer.</td>
</tr>
<tr>
<td>:n ((([0-9][#][0-9][#])[Ee](+</td>
<td>-)[0-9][#][#])[#])</td>
</tr>
<tr>
<td>:p (([A-Za-z]:[[]):f([[]))@:f</td>
<td>Windows: Matches a path.</td>
</tr>
<tr>
<td>:p (([/]):f([/)@:f)</td>
<td>UNIX: Matches a path.</td>
</tr>
<tr>
<td>:q ('&quot;[~][&quot; ]@&quot;</td>
<td>Matches a quoted string.</td>
</tr>
<tr>
<td>:v ([A-Za-z_$][A-Za-z0-9_$]@)</td>
<td>Matches a C variable.</td>
</tr>
<tr>
<td>:w ([A-Za-z][#])</td>
<td>Matches a word.</td>
</tr>
</tbody>
</table>

**Warning**

:f and :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.

**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
SlickEdit Regular Expression Definition

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.

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.

<table>
<thead>
<tr>
<th>SlickEdit Regular Expression Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^defproc</td>
<td>Matches lines that begin with the word defproc.</td>
</tr>
<tr>
<td>^definit$</td>
<td>Matches lines that only contain the word definit.</td>
</tr>
<tr>
<td>^:name</td>
<td>Matches lines that begin with the string :name. Notice that the backslash must prefix the colon character (:).</td>
</tr>
<tr>
<td>[\t ]</td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td>[\9\32]</td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td>[\x9\x20]</td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td>p?t</td>
<td>Matches any three-letter string starting with the letter p and ending with the letter t. Two possible matches are pot and pat.</td>
</tr>
<tr>
<td>s?t</td>
<td>Matches the letter s followed by any number of characters followed by the nearest letter t. Two possible matches are seat and st.</td>
</tr>
<tr>
<td>for</td>
<td>while</td>
</tr>
</tbody>
</table>
**SlickEdit Regular Expression Example**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^:p</td>
<td>Matches lines beginning with a file name.</td>
</tr>
<tr>
<td>xy+z</td>
<td>Matches x followed by one or more occurrences of y followed by z.</td>
</tr>
<tr>
<td>\x0d\x0a\x01\x02</td>
<td>Matches a sequence of hex binary characters.</td>
</tr>
<tr>
<td>\13\10\1\2</td>
<td>Matches a sequence of decimal binary characters.</td>
</tr>
</tbody>
</table>

**UNIX Regular Expressions**

UNIX regular expressions are defined in the following table.

<table>
<thead>
<tr>
<th>UNIX Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Matches beginning of line.</td>
</tr>
<tr>
<td>$</td>
<td>Matches end of line.</td>
</tr>
<tr>
<td>.</td>
<td>Matches any character except newline.</td>
</tr>
<tr>
<td>X+</td>
<td>Maximal match of one or more occurrences of X. See Minimal versus Maximal Matching.</td>
</tr>
<tr>
<td>X*</td>
<td>Maximal match of zero or more occurrences of X.</td>
</tr>
<tr>
<td>X?</td>
<td>Maximal match of zero or one occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 }</td>
<td>Match exactly n1 occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 ,}</td>
<td>Maximal match of at least n1 occurrences of X.</td>
</tr>
<tr>
<td>X{ n2 }</td>
<td>Maximal match of at least zero occurrences but not more than n2 occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 , n2 }</td>
<td>Maximal match of at least n1 occurrences but not more than n2 occurrences of X.</td>
</tr>
<tr>
<td>X+?</td>
<td>Minimal match of one or more occurrences of X.</td>
</tr>
<tr>
<td>X*?</td>
<td>Minimal match of zero or more occurrences of X.</td>
</tr>
<tr>
<td>UNIX Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X??</td>
<td>Minimal match of zero or one occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 }?</td>
<td>Matches exactly n1 occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 ,}?</td>
<td>Minimal match of at least n1 occurrences of X.</td>
</tr>
<tr>
<td>X{, n2 }?</td>
<td>Minimal match of at least zero occurrences but not more than n2 occurrences of X.</td>
</tr>
<tr>
<td>X{ n1 , n2 }?</td>
<td>Minimal match of at least n1 occurrences but not more than n2 occurrences of X.</td>
</tr>
<tr>
<td>(!X)</td>
<td>Search fails if expression X is matched. The expression ^(!if) matches the beginning of all lines that do not start with if.</td>
</tr>
<tr>
<td>(=X)</td>
<td>Assert, positive lookahead. Searches for subexpression X, but X is not returned as part of the match. For example, to match words ending in &quot;ed&quot; while excluding &quot;ed&quot; as part of the match, use \b[a-z]+(?=ed\b). See also (!X).</td>
</tr>
<tr>
<td>(&gt;X)</td>
<td>Prohibit backtracking. This expression is advanced usage. It can be used to prevent the subexpression X from backtracking when using maximal (greedy) matching.</td>
</tr>
<tr>
<td>(#text)</td>
<td>Comment. No text is matched in this expression; it is used for comment and documentation only.</td>
</tr>
<tr>
<td>(X)</td>
<td>Matches subexpression X and specifies a new tagged expression (see Using Tagged Expressions). No more tagged expressions are defined once an explicit tagged expression number is specified as shown below.</td>
</tr>
<tr>
<td>(? dX)</td>
<td>Matches subexpression X and specifies to use tagged expression number d where 0&lt;=d&lt;=9. No more tagged expressions are defined by the subexpression syntax (X) once this subexpression syntax is used. This is the best way to make sure you have enough tagged expressions.</td>
</tr>
<tr>
<td>(:X)</td>
<td>Matches subexpression X but does not define a</td>
</tr>
<tr>
<td>UNIX Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>tagged expression.</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>[ char-set ]</td>
<td>Matches any one of the characters specified by char-set. A dash (-) character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. A backslash () may be used inside the square brackets to define literal characters or define ASCII characters. For example, - specifies a literal dash character. The expression [\d0-\d27] matches ASCII character codes 0..27. The expression [] matches a right bracket. In SlickEdit® Core regular expressions, [] matches no characters. In both syntaxes, the expression [^] matches a caret (^) character but this does not work for SlickEdit regular expressions. In both syntaxes, [^] matches a caret (^) character.</td>
</tr>
<tr>
<td>[^ char-set ]</td>
<td>Matches any character not specified by char-set. A dash (-) character may be used to specify ranges.</td>
</tr>
<tr>
<td>[ char-set1 - [ char-set2 ]]</td>
<td>Character set subtraction. Matches all characters in char-set1 except the characters in char-set2. The expression [^A-Z] matches all characters except uppercase letters. For example, [a-z-[qw]] matches all English lowercase letters except q and w. [\p{L}-[qw]] matches all Unicode lowercase letters except q and w.</td>
</tr>
<tr>
<td>[ char-set1 &amp; [ char-set2 ]</td>
<td>Character set intersection. Matches all characters in char-set1 that are also in char-set2. For example, [x(0)-x(7f)][\p{L}]] matches all letters between 0 and 127.</td>
</tr>
<tr>
<td>\x{ hhhh }</td>
<td>Matches up to 31-bit Unicode hexadecimal character specified by hhhh.</td>
</tr>
<tr>
<td>\p{ UnicodeCategorySpec }</td>
<td>(Only valid in character set) Matches characters in UnicodeCategorySpec. Where UnicodeCategorySpec uses the standard general categories specified by the Unicode consortium. For example, \p{Lu} matches all uppercase letters. See Unicode</td>
</tr>
<tr>
<td>UNIX Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>\P{ UnicodeCategorySpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodeCategorySpec. For example, [\P{L}] matches all characters that are not letters. This is equivalent to [^\p{L}]. [\P{Lu}] matches all characters that are not uppercase letters. See Unicode Category Specifications for Regular Expressions.</td>
</tr>
<tr>
<td>\p{ UnicodelsBlockSpec }</td>
<td>(Only valid in character set) Matches characters in UnicodelsBlockSpec. Where UnicodelsBlockSpec one of the standard character blocks specified by the Unicode consortium. For example, [\p{isGreek}] matches Unicode characters in the Greek block. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>\P{ UnicodelsBlockSpec }</td>
<td>(Only valid in character set) Matches characters not in UnicodelsBlockSpec. For example, [\P{isGreek}] matches all characters that are not in the Unicode Greek block. This is equivalent to [^\p{isGreek}]. See Unicode Character Blocks for Regular Expressions.</td>
</tr>
<tr>
<td>\x hh</td>
<td>Matches hexadecimal character hh where 0&lt;=hh&lt;=0xff.</td>
</tr>
<tr>
<td>\d ddd</td>
<td>Matches decimal character ddd where 0&lt;=ddd&lt;=255.</td>
</tr>
<tr>
<td>\d</td>
<td>Defines a back reference to tagged expression number d. For example, (abc)def\1 matches the string abcdedefabc. If the tagged expression has not been set, the search fails.</td>
</tr>
<tr>
<td>\c</td>
<td>Specifies cursor position if match is found. If the expression xyz\c is found the cursor is placed after the z.</td>
</tr>
<tr>
<td>\n</td>
<td>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.</td>
</tr>
</tbody>
</table>
### UNIX Regular Expression Syntax

<table>
<thead>
<tr>
<th>UNIX Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use \d10 if you want to match an ASCII 10 character.</td>
<td></td>
</tr>
<tr>
<td>\r</td>
<td>Matches carriage return (ASCII 13). What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file.</td>
</tr>
<tr>
<td>\t</td>
<td>Matches tab character.</td>
</tr>
<tr>
<td>\b</td>
<td>Matches at word boundary. For example, \bre matches all occurrences of &quot;re&quot; that only occur at the beginning of a word.</td>
</tr>
<tr>
<td>\B</td>
<td>Matches all except at word boundary. For example, \Bre matches all occurrences of &quot;re&quot; as long as it is not at the start of a new word.</td>
</tr>
<tr>
<td>\Q and \E</td>
<td>\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.</td>
</tr>
<tr>
<td>\f</td>
<td>Matches form feed character.</td>
</tr>
<tr>
<td>\od</td>
<td>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.</td>
</tr>
<tr>
<td>\om</td>
<td>Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end-of-line characters. For example, \om.+ matches the rest of the buffer.</td>
</tr>
<tr>
<td>\ol</td>
<td>Turns off multi-line matching (default). You can still use \in to create regular expressions which match one or more lines. However, expressions like .+ will not match multiple lines. This is much safer and</td>
</tr>
<tr>
<td>UNIX Regular Expression</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>\oi</td>
<td>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. See also \oc.</td>
</tr>
<tr>
<td>\oc</td>
<td>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 \oi.</td>
</tr>
<tr>
<td>\char</td>
<td>Declares character after slash to be literal. For example, * represents the star character.</td>
</tr>
<tr>
<td>: char</td>
<td>Matches predefined expression corresponding to char. The pre-defined expressions are:</td>
</tr>
<tr>
<td></td>
<td>• \a [A-Za-z0-9] - Matches an alphanumeric character.</td>
</tr>
<tr>
<td></td>
<td>• \c [A-Za-z] - Matches an alphabetic character.</td>
</tr>
<tr>
<td></td>
<td>• \b (?:\t+) - Matches blanks.</td>
</tr>
<tr>
<td></td>
<td>• \d [0-9] - Matches a digit.</td>
</tr>
<tr>
<td></td>
<td>• \f (?:[^\t]+) - Windows: Matches a file name part.</td>
</tr>
<tr>
<td></td>
<td>• \f (?:[^\t]+) - UNIX: Matches a file name part.</td>
</tr>
<tr>
<td></td>
<td>• \h (?:[0-9A-Fa-f]+) - Matches a hex number.</td>
</tr>
<tr>
<td></td>
<td>• \i (?:[0-9]+) - Matches an integer.</td>
</tr>
<tr>
<td></td>
<td>• \p (?:[A-Za-z]:</td>
</tr>
<tr>
<td></td>
<td>• \p (?::/)?:f(?:/)*:f - UNIX: Matches a path.</td>
</tr>
</tbody>
</table>
UNIX Regular Expression

<table>
<thead>
<tr>
<th>UNIX Regular Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>\q (?:&quot;[^&quot;]*&quot;</td>
<td>'[^']*')</td>
</tr>
<tr>
<td>\v (?:[A-Za-z_]$)[A-Za-z0-9_]* )</td>
<td>- Matches a C variable.</td>
</tr>
<tr>
<td>\w (?:[A-Za-z]+)</td>
<td>- Matches a word.</td>
</tr>
</tbody>
</table>

**Warning**

\f and \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.

**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.

The precedence of operators, from highest to lowest, is as follows:

- +, *, ?, {}, +?, *?, ??, {}? (These operators have the same precedence.)
- concatenation
- |

**UNIX Regular Expression Examples**

The table below shows examples of UNIX regular expressions.
<table>
<thead>
<tr>
<th>UNIX Regular Expression Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^defproc</td>
<td>Matches lines that begin with the word <code>defproc</code>.</td>
</tr>
<tr>
<td>^definit$</td>
<td>Matches lines that only contain the word <code>definit</code>.</td>
</tr>
<tr>
<td>^*name</td>
<td>Matches lines that begin with the string <code>*name</code>. Notice that the backslash must prefix the special character <code>*</code>.</td>
</tr>
<tr>
<td>[\t ]</td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td>[\d9\d32]</td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td>[\x9\x20]</td>
<td>Matches tab and space characters.</td>
</tr>
<tr>
<td>p.t</td>
<td>Matches any three-letter string starting with the letter <code>p</code> and ending with the letter <code>t</code>. Two possible matches are <code>pot</code> and <code>pat</code>.</td>
</tr>
<tr>
<td>s.*?t</td>
<td>Matches the letter <code>s</code> followed by any number of characters followed by the nearest letter <code>t</code>. Two possible matches are <code>seat</code> and <code>st</code>.</td>
</tr>
<tr>
<td>for</td>
<td>while</td>
</tr>
<tr>
<td>^:p</td>
<td>Matches lines beginning with a file name.</td>
</tr>
<tr>
<td>xy+z</td>
<td>Matches <code>x</code> followed by one or more occurrences of <code>y</code> followed by <code>z</code>.</td>
</tr>
<tr>
<td>[a-z-[qw]]</td>
<td>Character set subtraction. Matches all English lowercase letters except <code>q</code> and <code>w</code>.</td>
</tr>
<tr>
<td>[\p{isGreek}&amp;[\p{L}]]</td>
<td>Character set intersection. Matches all Unicode letters in the Greek block.</td>
</tr>
<tr>
<td>\x{6587}</td>
<td>Matches Unicode character with hexadecimal value <code>6587</code>. Character set intersection. Matches all Unicode letters in the Greek block.</td>
</tr>
<tr>
<td>[\p[L]-[qw]]</td>
<td>Matches all Unicode letters except <code>q</code> and <code>w</code>.</td>
</tr>
<tr>
<td>[\p{L}]</td>
<td>Matches all Unicode letters.</td>
</tr>
<tr>
<td>[\p{Lul}]</td>
<td>Matches all Unicode uppercase and lowercase</td>
</tr>
<tr>
<td>UNIX Regular Expression Example</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>[\P{L}]</td>
<td>Matches all Unicode characters that are not letters.</td>
</tr>
<tr>
<td>[\p{isGreek}]</td>
<td>Matches all Unicode characters in the Greek block.</td>
</tr>
<tr>
<td>\x0d\x0a\x01\x02</td>
<td>Matches a sequence of hex binary characters.</td>
</tr>
<tr>
<td>\d13\d10\d1\d2</td>
<td>Matches a sequence of decimal binary characters.</td>
</tr>
</tbody>
</table>

**Wildcard Expressions**

SlickEdit® Core 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.

**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).

<table>
<thead>
<tr>
<th>Subcategory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lu</td>
<td>Letter, Uppercase</td>
</tr>
<tr>
<td>Li</td>
<td>Letter, Lowercase</td>
</tr>
<tr>
<td>Lt</td>
<td>Letter, Titlecase</td>
</tr>
<tr>
<td>Lo</td>
<td>Letter, Other</td>
</tr>
<tr>
<td>Mn</td>
<td>Mark, Non-Spacing</td>
</tr>
<tr>
<td>Mc</td>
<td>Mark, Spacing Combining</td>
</tr>
<tr>
<td>Me</td>
<td>Mark, Enclosing</td>
</tr>
<tr>
<td>Nd</td>
<td>Number, Decimal Digit</td>
</tr>
<tr>
<td>Ni</td>
<td>Number, Letter</td>
</tr>
<tr>
<td>No</td>
<td>Number, Other</td>
</tr>
<tr>
<td>Pc</td>
<td>Punctuation, Connector</td>
</tr>
<tr>
<td>Pd</td>
<td>Punctuation, Dash</td>
</tr>
<tr>
<td>Ps</td>
<td>Punctuation, Open</td>
</tr>
<tr>
<td>Pe</td>
<td>Punctuation, Close</td>
</tr>
<tr>
<td>Pi</td>
<td>Punctuation, Initial quote (may behave like Ps or Pe depending on usage)</td>
</tr>
<tr>
<td>Pf</td>
<td>Punctuation, Final quote (may behave like Ps or Pe depending on usage)</td>
</tr>
<tr>
<td>Po</td>
<td>Punctuation, Other</td>
</tr>
</tbody>
</table>
Unicode Categories and Character Blocks

<table>
<thead>
<tr>
<th>Subcategory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sm</td>
<td>Symbol, Math</td>
</tr>
<tr>
<td>Sc</td>
<td>Symbol, Currency</td>
</tr>
<tr>
<td>Sk</td>
<td>Symbol, Modifier</td>
</tr>
<tr>
<td>So</td>
<td>Symbol, Other</td>
</tr>
<tr>
<td>Zs</td>
<td>Separator, Space</td>
</tr>
<tr>
<td>Zl</td>
<td>Separator, Line</td>
</tr>
<tr>
<td>Zp</td>
<td>Separator, Paragraph</td>
</tr>
<tr>
<td>Cc</td>
<td>Other, Control</td>
</tr>
<tr>
<td>Cf</td>
<td>Other, Format</td>
</tr>
<tr>
<td>Cs</td>
<td>Other, Surrogate</td>
</tr>
<tr>
<td>Co</td>
<td>Other, Private Use</td>
</tr>
<tr>
<td>Cn</td>
<td>Other, Not Assigned (no characters in the file have this property)</td>
</tr>
</tbody>
</table>

Unicode Character Blocks for Regular Expressions

The Unicode consortium standard regular expression block categories are supported. The syntax for specifying a character block is:

\p{Is BlockName}

The above syntax matches the characters in the block specified. The following syntax matches all characters not in the block specified:

\P{Is BlockName}

The \p and \P notations may only be used inside a character set specification. For example, \p{isBasicLatin} matches all characters in the Greek block. \P{isBasicLatin} matches all characters that are not in the Greek block.

The following table lists the non-standard valid character block names. These character tables were generated from XML standards found at the World Wide Web Consortium Web site [http://www.w3c.org].
## Unicode Categories and Character Blocks

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLNameStartChar</td>
<td>All characters that are valid for the start of an XML tag name.</td>
</tr>
<tr>
<td>XMLNameChar</td>
<td>All characters that are valid in an XML tag name.</td>
</tr>
</tbody>
</table>

The following table lists the valid character block names. These character tables were generated using the `blocks.txt` file found on the Unicode Consortium Web site ([http://unicode.org](http://unicode.org)).

<table>
<thead>
<tr>
<th>Range</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000..007F</td>
<td>BasicLatin</td>
</tr>
<tr>
<td>0080..00FF</td>
<td>Latin-1Supplement</td>
</tr>
<tr>
<td>0100..017F</td>
<td>LatinExtended-A</td>
</tr>
<tr>
<td>0180..024F</td>
<td>LatinExtended-B</td>
</tr>
<tr>
<td>0250..02AF</td>
<td>IPAExtensions</td>
</tr>
<tr>
<td>02B0..02FF</td>
<td>SpacingModifierLetters</td>
</tr>
<tr>
<td>0300..036F</td>
<td>CombiningDiacriticalMarks</td>
</tr>
<tr>
<td>0370..03FF</td>
<td>Greek</td>
</tr>
<tr>
<td>0400..04FF</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>0530..058F</td>
<td>Armenian</td>
</tr>
<tr>
<td>0590..05FF</td>
<td>Hebrew</td>
</tr>
<tr>
<td>0600..06FF</td>
<td>Arabic</td>
</tr>
<tr>
<td>0700..074F</td>
<td>Syriac</td>
</tr>
<tr>
<td>0780..07BF</td>
<td>Thaana</td>
</tr>
<tr>
<td>0900..097F</td>
<td>Devanagari</td>
</tr>
<tr>
<td>0980..09FF</td>
<td>Bengali</td>
</tr>
<tr>
<td>Range</td>
<td>Block Name</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>0A00..0A7F</td>
<td>Gurmukhi</td>
</tr>
<tr>
<td>0A80..0AFF</td>
<td>Gujarati</td>
</tr>
<tr>
<td>0B00..0B7F</td>
<td>Oriya</td>
</tr>
<tr>
<td>0B80..0BFF</td>
<td>Tamil</td>
</tr>
<tr>
<td>0C00..0C7F</td>
<td>Telugu</td>
</tr>
<tr>
<td>0C80..0CFF</td>
<td>Kannada</td>
</tr>
<tr>
<td>0D00..0D7F</td>
<td>Malayalam</td>
</tr>
<tr>
<td>0D80..0DFF</td>
<td>Sinhala</td>
</tr>
<tr>
<td>0E00..0E7F</td>
<td>Thai</td>
</tr>
<tr>
<td>0E80..0EFF</td>
<td>Lao</td>
</tr>
<tr>
<td>0F00..0FFF</td>
<td>Tibetan</td>
</tr>
<tr>
<td>1000..109F</td>
<td>Myanmar</td>
</tr>
<tr>
<td>10A0..10FF</td>
<td>Georgian</td>
</tr>
<tr>
<td>1100..11FF</td>
<td>HanguJamo</td>
</tr>
<tr>
<td>1200..137F</td>
<td>Ethiopic</td>
</tr>
<tr>
<td>13A0..13FF</td>
<td>Cherokee</td>
</tr>
<tr>
<td>1400..167F</td>
<td>UnifiedCanadianAboriginalSyllabics</td>
</tr>
<tr>
<td>1680..169F</td>
<td>Ogham</td>
</tr>
<tr>
<td>16A0..16FF</td>
<td>Runic</td>
</tr>
<tr>
<td>1780..17FF</td>
<td>Khmer</td>
</tr>
<tr>
<td>1800..18AF</td>
<td>Mongolian</td>
</tr>
<tr>
<td>1E00..1EFF</td>
<td>LatinExtendedAdditional</td>
</tr>
<tr>
<td>Range</td>
<td>Block Name</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>1F00..1FFF</td>
<td>GreekExtended</td>
</tr>
<tr>
<td>2000..206F</td>
<td>GeneralPunctuation</td>
</tr>
<tr>
<td>2070..209F</td>
<td>SuperscriptsandSubscripts</td>
</tr>
<tr>
<td>20A0..20CF</td>
<td>CurrencySymbols</td>
</tr>
<tr>
<td>20D0..20FF</td>
<td>CombiningMarksforSymbols</td>
</tr>
<tr>
<td>2100..214F</td>
<td>LetterlikeSymbols</td>
</tr>
<tr>
<td>2150..218F</td>
<td>NumberForms</td>
</tr>
<tr>
<td>2190..21FF</td>
<td>Arrows</td>
</tr>
<tr>
<td>2200..22FF</td>
<td>MathematicalOperators</td>
</tr>
<tr>
<td>2300..23FF</td>
<td>MiscellaneousTechnical</td>
</tr>
<tr>
<td>2400..243F</td>
<td>ControlPictures</td>
</tr>
<tr>
<td>2440..245F</td>
<td>OpticalCharacterRecognition</td>
</tr>
<tr>
<td>2460..24FF</td>
<td>EnclosedAlphanumerics</td>
</tr>
<tr>
<td>2500..257F</td>
<td>BoxDrawing</td>
</tr>
<tr>
<td>2580..259F</td>
<td>BlockElements</td>
</tr>
<tr>
<td>25A0..25FF</td>
<td>GeometricShapes</td>
</tr>
<tr>
<td>2600..26FF</td>
<td>MiscellaneousSymbols</td>
</tr>
<tr>
<td>2700..27BF</td>
<td>Dingbats</td>
</tr>
<tr>
<td>2800..28FF</td>
<td>BraillePatterns</td>
</tr>
<tr>
<td>2E80..2EFF</td>
<td>CJKRadicalsSupplement</td>
</tr>
<tr>
<td>2F00..2FDF</td>
<td>KangxiRadicals</td>
</tr>
<tr>
<td>2FF0..2FFF</td>
<td>IdeographicDescriptionCharacters</td>
</tr>
</tbody>
</table>
## Unicode Categories and Character Blocks

<table>
<thead>
<tr>
<th>Range</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000..303F</td>
<td>CJKSymbolsandPunctuation</td>
</tr>
<tr>
<td>3040..309F</td>
<td>Hiragana</td>
</tr>
<tr>
<td>30A0..30FF</td>
<td>Katakana</td>
</tr>
<tr>
<td>3100..312F</td>
<td>Bopomofo</td>
</tr>
<tr>
<td>3130..318F</td>
<td>HangulCompatibilityJamo</td>
</tr>
<tr>
<td>3190..319F</td>
<td>Kanbun</td>
</tr>
<tr>
<td>31A0..31BF</td>
<td>BopomofoExtended</td>
</tr>
<tr>
<td>3200..32FF</td>
<td>EnclosedCJKLettersandMonths</td>
</tr>
<tr>
<td>3300..33FF</td>
<td>CJKCompatibility</td>
</tr>
<tr>
<td>3400..4DB5</td>
<td>CJKUnifiedIdeographsExtensionA</td>
</tr>
<tr>
<td>4E00..9FFF</td>
<td>CJKUnifiedIdeographs</td>
</tr>
<tr>
<td>A000..A48F</td>
<td>YiSyllables</td>
</tr>
<tr>
<td>A490..A4CF</td>
<td>YiRadicals</td>
</tr>
<tr>
<td>AC00..D7A3</td>
<td>HangulSyllables</td>
</tr>
<tr>
<td>D800..DB7F</td>
<td>HighSurrogates</td>
</tr>
<tr>
<td>DB80..DBFF</td>
<td>HighPrivateUseSurrogates</td>
</tr>
<tr>
<td>DC00..DFFF</td>
<td>LowSurrogates</td>
</tr>
<tr>
<td>E000..F8FF</td>
<td>PrivateUse</td>
</tr>
<tr>
<td>F900..FAFF</td>
<td>CJKCompatibilityIdeographs</td>
</tr>
<tr>
<td>FB00..FB4F</td>
<td>AlphabeticPresentationForms</td>
</tr>
<tr>
<td>FB50..FDFF</td>
<td>ArabicPresentationForms-A</td>
</tr>
<tr>
<td>FE20..FE2F</td>
<td>CombiningHalfMarks</td>
</tr>
<tr>
<td>Range</td>
<td>Block Name</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>FE30..FE4F</td>
<td>CJKCompatibilityForms</td>
</tr>
<tr>
<td>FE50..FE6F</td>
<td>SmallFormVariants</td>
</tr>
<tr>
<td>FE70..FEFE</td>
<td>ArabicPresentationForms-B</td>
</tr>
<tr>
<td>FEFF..FEFF</td>
<td>Specials</td>
</tr>
<tr>
<td>FF00..FFEF</td>
<td>HalfwidthandFullwidthForms</td>
</tr>
<tr>
<td>FFF0..FFFF</td>
<td>Specials</td>
</tr>
<tr>
<td>10300..1032F</td>
<td>OldItalic</td>
</tr>
<tr>
<td>10330..1034F</td>
<td>Gothic</td>
</tr>
<tr>
<td>10400..1044F</td>
<td>Deseret</td>
</tr>
<tr>
<td>1D000..1D0FF</td>
<td>ByzantineMusicalSymbols</td>
</tr>
<tr>
<td>1D100..1D1FF</td>
<td>MusicalSymbols</td>
</tr>
<tr>
<td>1D400..1D7FF</td>
<td>MathematicalAlphanumericSymbols</td>
</tr>
<tr>
<td>20000..2A6D6</td>
<td>CJKUnifiedIdeographsExtensionB</td>
</tr>
<tr>
<td>2F800..2FA1F</td>
<td>CJKCompatibilityIdeographsSupplement</td>
</tr>
<tr>
<td>E0000..E007F</td>
<td>Tags</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;</code>, `</td>
<td>`</td>
</tr>
<tr>
<td><code>^</code></td>
<td>xor</td>
</tr>
<tr>
<td><code>+</code>, blank(s), <code>-</code></td>
<td>addition, implied addition, subtraction</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>math 2.5*2</td>
<td>Multiplies 2.5 times 2</td>
</tr>
<tr>
<td>math 5/2</td>
<td>Divides 5 by 2</td>
</tr>
<tr>
<td>mathx 255</td>
<td>Converts 255 to hexadecimal</td>
</tr>
<tr>
<td>math xFF</td>
<td>Converts hexadecimal FF to decimal</td>
</tr>
<tr>
<td>math o77</td>
<td>Converts octal 77 to decimal</td>
</tr>
<tr>
<td>matho 255</td>
<td>Converts 255 to octal</td>
</tr>
<tr>
<td>math 077+0xff+10</td>
<td>Adds octal 77, hex FF, and 10</td>
</tr>
</tbody>
</table>

### 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® Core provides a way to display the operating system's (OS) file manager/browser. For example, Windows Explorer is displayed on Windows, Finder on Mac OS X, Konquerer on Linux KDE desktop, etc.

To display the OS file browser, 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.
Macros and Macro Programming

This chapter describes how to use macros with SlickEdit Core/

There are two types of macros in SlickEdit® Core:

• **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 the SlickEdit Core editor and can even add 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® Core 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 Core.

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 Core 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.
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 Core 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 Window → SlickEdit Preferences → Keyboard → 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 Core 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 Core 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**
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® Core 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
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® Core.

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 Window → SlickEdit Preferences → 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

The Slick-C® macro programming language is behind most of the actions performed in SlickEdit® Core. 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 views.

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 Core are located in the \[EclipseInstallDir]/plugins/com.slickedit.windows.libs_[VERSION]/slickedit/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® Core command line. The Open dialog is displayed, prompting you for the file to load.

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 Core (standard SlickEdit Core modules). Unloading a standard module could cause the editor to behave unpredictably.
- Standard modules are identified by file name. If you load a module with a name that matches a standard SlickEdit Core module, it will replace the standard module, which (like unloading a standard 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® Core 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 Core for the first time. See Creating Tag Files for Compiler-Specific Libraries for more information.

**State File**

SlickEdit® Core 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 is located in your User Configuration Directory. On Windows, the state file is named vslick.sta, and on UNIX, vslick.stu. The file extensions were made different to avoid any confusion between the two, since a Windows state file is not compatible with UNIX, and vice versa.

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, views, 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.

For information about writing Slick-C, see the Slick-C Macro Programming Guide, which is included in the SlickEdit® Core Help system and also available as a stand-alone PDF in the docs installation subdirectory.
This chapter contains a comprehensive description of the menus and dialogs available in SlickEdit Core. For a general overview, see User Interface.
Menu Overview

SlickEdit Core adds a number of new menus to the Eclipse main menu. Additionally, some existing Eclipse menus have new items added to them.

The following menus are added by SlickEdit Core:

- Format
- Display
- Macro
- Tools

The following standard Eclipse menus have items added to them by SlickEdit Core:

- File
- Edit
- Navigate
- Search
- Window
- Help

The following sections document the menus and menu entries added by SlickEdit Core. Standard Eclipse menus and menu entries are not listed or described.
This section describes items on the **File** menu and associated dialogs and views.

## File Menu

The table below describes the items added to the standard Eclipse File menu by SlickEdit core.

<table>
<thead>
<tr>
<th>File Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Item from SlickEdit Template</td>
<td>Displays the Add New Item dialog, which allows you to create a new file from a template. See Code Templates.</td>
<td><code>add_item</code></td>
</tr>
<tr>
<td>Print</td>
<td>Displays the Print dialog (or Text Mode Print dialog for Mac/Linux/UNIX), which contains options to print the current file or selection and provides setup options. See Print Dialog for Windows and Print Dialog for Mac, Linux, and UNIX.</td>
<td><code>gui_print</code></td>
</tr>
<tr>
<td>SlickEdit Template Manager</td>
<td>Create, edit, and delete your templates. See Code Templates.</td>
<td><code>template_manager</code></td>
</tr>
</tbody>
</table>

## File Dialogs and Views

This section describes the dialogs that are associated with **File** menu items.

### Print Dialog for Windows

On Windows, this is the default dialog used to configure print options and send text or files to a printer. It is displayed when you click **File** → **Print** or use the `gui_print` command. Linux has a different print dialog (see Print Dialog for Mac, Linux, and UNIX for its description. See Printing for more information about printing from within SlickEdit Core).
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 Fonts 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).

- **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**
- **Schemes Tab**

**General Tab**

This tab on the Print Dialog for Windows 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 coding** - When this option is selected, language-specific color coding is printed using the same font attributes (bold, italic, underline).

• **Print color** - When this option is selected, language-specific color coding is printed using the same colors.

• **Print hex** - When this option is selected, printed output is displayed as if it were the hex display mode.

• **Two up** - When this option is selected, it specifies two columns where one page is printed in the left column and the next page is printed in the right column. This is useful when printing in landscape mode, especially when you are using a small font.

• **Orientation** - Specifies whether text is printed top to bottom (portrait) or left to right (landscape).

• **Space between** - This text specifies the width between columns, in inches. This text box is unavailable unless the Two Up check box is marked.

• **Number lines every** - When the value is not zero, lines at intervals of this value are printed with line numbers.

• **Number of copies** - This value specifies the number of copies to print.

**Header/Footer Tab**

This tab on the Print Dialog for Windows 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 for Windows* 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.

### Schemes Tab

This tab on the [Print Dialog for Windows](#) lets you to save the current printer settings as a scheme. See [Print Schemes](#) for more information.

### Print Dialog for Linux

On Linux, the Text Mode Print dialog is the default dialog used to configure print options and send text or files to a printer. It is displayed when you click **File → Print** or use the `gui_print` command. Windows uses a different print dialog. See [Print Dialog for Windows](#) for its description. See [Printing](#) for more information about printing from within SlickEdit®.
The Text Mode Print dialog contains the following general settings:

- **OK** - Sends the selection or active buffer to the print device.

- **Save Settings** - Saves the settings that are configured in the Text Mode Print dialog. The next time a print command or this dialog is invoked, the previously saved settings are restored.

- **Files** - Expands the Text Mode Print dialog to allow you to pick another file or multiple files for printing (as opposed to the active file).

The remaining options are categorized into the following tabs:

- **General Tab**
- **Header/Footer Tab**
- **Advanced Tab**

**General Tab**

This tab on the Print Dialog for Mac, Linux, and UNIX contains general print settings.
• **Selection Only** - When this option is selected, only the selection is printed. To print the selection immediately using the default configuration, use the `print_selection` command.

• **Visible lines only** - When this option is selected, only visible lines are printed. This option allows you to print selective display.

• **Print hex** - When this option is selected, printed output is displayed as if it were the hex display mode.

• **Form feed after last page** - Determines whether the built-in print text formatting prints a form feed after the last page.

• **Use PostScript (Enscript)** - When this option is selected, the enscript program is used to print a source file. This allows for print formatting such as fonts, page layout, etc. To specify the options for enscript, see [Advanced Tab](#). Activating this option will make the following settings available:

  • **Print color coding** - When this option is selected, language-specific color coding is printed using the same font attributes (bold, italic, underline).

  • **Print color** - When this option is selected, language-specific color coding is printed using the same colors.

  • **Two Up** - When this option is selected, it specifies two columns where one page is printed in the left column and the next page is printed in the right column.

• **Number Lines Every** - When the value is not zero, lines at intervals of this value are printed with line numbers.

• **Lines per page** - Maximum number of lines that can be printed. This includes header, footer, and blank lines.
• **Columns per line** - Maximum number of columns that can be printed on a line.

**Header/Footer Tab**

This tab on the *Print Dialog for Mac, Linux, and UNIX* is the same as the **Header/Footer Tab** on the *Print Dialog for Windows*.

**Advanced Tab**

This tab on the *Print Dialog for Mac, Linux, and UNIX* is used to configure advanced print options.

![Advanced Tab](image)

• **Print command** - Specifies the program that will be executed to print the formatted text. If no print command is specified, the text is written to the Print Device specified. The default print command is `lp -c -o nobanner %f`. The `%f` specifies the file to be printed. If the print program is not found, the command `lpr %f` is used instead.

• **PostScript Options** - To use PostScript options, you must first select the option **Use PostScript (Enscript)** on the **General** print settings tab. The default options are `--media=Letter --no-header --quiet`. For a list of all available enscript options, see the enscript man page.

• **Print filter** - Specifies an external program to format text before it is printed and overrides the built-in print text formatting. The program specified here must accept input from standard in and writes its output to standard out. A typical print filter command is `pr -1 -w76 -o0 -i50`.

• **Print device** - Specifies a printer device file name in which to write the formatted text. This text box is ignored if a **Print Command** has been specified. The default print device is `/dev/lp0`. 
## Edit Menu

The table below describes items on the **Edit** menu.

<table>
<thead>
<tr>
<th>Edit Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Undoes the last edit operation.</td>
<td>undo</td>
</tr>
<tr>
<td>Redo</td>
<td>Undoes an undo operation.</td>
<td>redo</td>
</tr>
<tr>
<td>Multi-file Undo</td>
<td>Undoes the last multi-file operation (i.e.: Find and Replace in all files).</td>
<td>mfundo</td>
</tr>
<tr>
<td>Multi-file Redo</td>
<td>Undoes an multi-file undo operation.</td>
<td>mfredo</td>
</tr>
<tr>
<td>Cut</td>
<td>Deletes the selected text and copies it to the clipboard.</td>
<td>cut</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the selected text to the clipboard.</td>
<td>copy_to_clipboard</td>
</tr>
<tr>
<td>Paste</td>
<td>Inserts the clipboard into the current file.</td>
<td>paste</td>
</tr>
<tr>
<td>List Clipboards</td>
<td>Displays the Clipboards view, which allows you to view and insert the selected clipboard. See Clipboards and also Edit Dialogs and Views.</td>
<td>list_clipboards</td>
</tr>
<tr>
<td>Copy Word</td>
<td>Copies the current word to the clipboard.</td>
<td>copy_word</td>
</tr>
<tr>
<td>Append to Clipboard</td>
<td>Appends the selected text to the clipboard.</td>
<td>append_to_clipboard</td>
</tr>
<tr>
<td>Append Cut</td>
<td>Deletes the selected text and appends it to the clipboard.</td>
<td>append_cut</td>
</tr>
<tr>
<td>Insert Literal</td>
<td>Displays the Insert Literal dialog, which allows you to insert a</td>
<td>insert_literal</td>
</tr>
</tbody>
</table>
### Edit Menu

<table>
<thead>
<tr>
<th>Edit Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>specified character code. See <a href="#">Inserting Literal Characters</a>.</td>
<td></td>
</tr>
<tr>
<td>Select</td>
<td>Displays menu for selecting and deselecting text. See <a href="#">Edit Select Menu</a>.</td>
<td>N/A</td>
</tr>
<tr>
<td>Delete</td>
<td>Displays menu for deleting text. See <a href="#">Edit Delete Menu</a>.</td>
<td>N/A</td>
</tr>
<tr>
<td>Complete Previous</td>
<td>Retrieves previous word or variable matching word prefix at cursor.</td>
<td>complete_prev</td>
</tr>
<tr>
<td>Word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete Next Word</td>
<td>Retrieves next word or variable matching word prefix at cursor.</td>
<td>complete_next</td>
</tr>
<tr>
<td>Fill</td>
<td>Displays the Fill Selection dialog, which lets you fill the selected text with a character you choose.</td>
<td>gui_fill_selection</td>
</tr>
<tr>
<td>Indent</td>
<td>Indents the selected text based on the tabs or indent for each level.</td>
<td>indent_selection</td>
</tr>
<tr>
<td>Unindent</td>
<td>Unindents the selected text based on the tabs or indent for each level.</td>
<td>unindent_selection</td>
</tr>
<tr>
<td>Other</td>
<td>Displays menu containing more edit-related commands. See <a href="#">Edit Other Menu</a>.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Edit Select Menu

The **Edit → Select** menu contains selection operations. See [Selections](#) for more information.

<table>
<thead>
<tr>
<th>Edit Select Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char</td>
<td>Starts or ends a character/stream selection.</td>
<td>select_char</td>
</tr>
<tr>
<td>Block</td>
<td>Starts or ends a block/column</td>
<td>select_block</td>
</tr>
</tbody>
</table>
## Edit Menu

<table>
<thead>
<tr>
<th>Edit Select Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>Starts or ends a line selection.</td>
<td>select_line</td>
</tr>
<tr>
<td>Word</td>
<td>Selects the word under cursor.</td>
<td>select_whole_word</td>
</tr>
<tr>
<td>Code Block</td>
<td>Selects text in current code block (if/loop/switch block etc.).</td>
<td>select_code_block</td>
</tr>
<tr>
<td>Procedure</td>
<td>Selects text in current function including function heading.</td>
<td>select_proc</td>
</tr>
<tr>
<td>Deselect</td>
<td>Unhighlights selected text.</td>
<td>deselect</td>
</tr>
<tr>
<td>All</td>
<td>Selects all text in current buffer.</td>
<td>select_all</td>
</tr>
</tbody>
</table>

## Edit Delete Menu

The **Edit → Delete** menu contains text deletion operations. See [Cutting and Deleting Text](#) for more information.

<table>
<thead>
<tr>
<th>Edit Delete Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>Deletes text from the cursor to the end of the current word and copies it to the clipboard.</td>
<td>cut_word</td>
</tr>
<tr>
<td>Line</td>
<td>Deletes the current line and copies it to the clipboard.</td>
<td>cut_line</td>
</tr>
<tr>
<td>To End of Line</td>
<td>Deletes text from the cursor to the end of the line and copies it to the clipboard.</td>
<td>cut_end_line</td>
</tr>
<tr>
<td>Selection</td>
<td>Deletes the selected text.</td>
<td>delete_selection</td>
</tr>
<tr>
<td>All</td>
<td>Delete all text in current buffer.</td>
<td>delete_all</td>
</tr>
</tbody>
</table>

## Edit Other Menu

The **Edit → Other** menu contains miscellaneous editing operations.
<table>
<thead>
<tr>
<th>Edit Other Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowcase</td>
<td>Translates the characters in the selected text to lowercase.</td>
<td>lowcase_selection</td>
</tr>
<tr>
<td>Upcase</td>
<td>Translates the characters in the selected text to uppercase.</td>
<td>upcase_selection</td>
</tr>
<tr>
<td>Capitalize</td>
<td>Capitalizes the first character of the current word.</td>
<td>cap_selection</td>
</tr>
<tr>
<td>Shift Left</td>
<td>Deletes the first column of text in each line of the selected text.</td>
<td>shift_selection_left</td>
</tr>
<tr>
<td>Shift Right</td>
<td>Inserts a space at the first column of each line of the selected text.</td>
<td>shift_selection_right</td>
</tr>
<tr>
<td>Overlay Block</td>
<td>Overwrites selected block/column of text at the cursor.</td>
<td>overlay_block_selection</td>
</tr>
<tr>
<td>Adjust Block</td>
<td>Overlays the selected text at the cursor and fills the original selected text with spaces.</td>
<td>adjust_block_selection</td>
</tr>
<tr>
<td>Enumerate</td>
<td>Displays the Enumerate dialog, which allows you to add incrementing numbers to a selection. See Enumerate Dialog.</td>
<td>guiEnumerate</td>
</tr>
<tr>
<td>Filter Selection</td>
<td>Displays a Command dialog, which allows you to filter selected text through an external command. See Filter Selection: Command Dialog.</td>
<td>filter_selection</td>
</tr>
<tr>
<td>Copy UCN As Unicode</td>
<td>Copies selected UCN to the clipboard as Unicode. See Using Unicode.</td>
<td>copy_ucn_as_unicode</td>
</tr>
<tr>
<td>Copy Unicode As</td>
<td>Displays menu containing commands for copying Unicode as UCN. See Copy Unicode As Menu.</td>
<td>N/A</td>
</tr>
<tr>
<td>Tabs to Spaces</td>
<td>Converts tabs to appropriate number of spaces. If there is no</td>
<td>convert_tabs2spaces</td>
</tr>
<tr>
<td>Edit Other Menu Item</td>
<td>Description</td>
<td>Command</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>selection the entire buffer is converted.</td>
<td></td>
<td>convert_spaces2tabs</td>
</tr>
</tbody>
</table>

| 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 |

| 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.

<table>
<thead>
<tr>
<th>Copy Unicode As Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ (UTF-16 \xHHHH)</td>
<td>Copies Unicode characters in selection as C++ UTF-16 \xHHHH notation.</td>
<td>copy_unicode_as_c</td>
</tr>
</tbody>
</table>

| 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 |

| UCN (UTF-32 \uHHHH and \UHHHHHHHHH) | Copies Unicode characters in selection as UCN UTF-32 \uHHHH and \UHHHHHHHHH notation. | copy_unicode_as_ucn |

<p>| SGML/XML hexadecimal (UTF-32 &amp;xHHHH;) | Copies Unicode characters in selection as SGML/XML hexadecimal UTF-32 &amp;xHHHH; notation. | copy_unicode_as_xml |</p>
<table>
<thead>
<tr>
<th>Copy Unicode As Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGML/XML decimal (UTF-32 &amp;xDDDD;)</td>
<td>Copies Unicode characters in selection as SGML/XML decimal UTF-32 &amp;xDDDD; notation.</td>
<td>copy_unicode_as_xmldec</td>
</tr>
</tbody>
</table>

---

**Edit Dialogs and Views**

This section describes the dialogs and views 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 view 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.

![Select Text to Paste](#)

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 (Window → SlickEdit Preferences → 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 `guiEnumerate` 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.

• **Pad to number of digits** - Specifies the digit width for each number. Number is padded to at least this number of digits by adding leading zeros.

• **Output** - Both the **Hexadecimal** and **Hexadecimal flags** options specify hexadecimal syntax output based on the buffers extension. We determine the hexadecimal syntax based on the color coding which supports $0xhhhh$ (C syntax), $\&Hdddd$ (Basic), $hhhhH$ (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.
# Format Menu

This section describes items on the **Format** menu and associated dialogs and views.

## Format Menu

The **Format** menu contains items pertaining to editor windows and the current document. The table below lists a summary of these items.

<table>
<thead>
<tr>
<th>Document Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Beautify</strong></td>
<td></td>
<td><strong>gui_beautify</strong></td>
</tr>
<tr>
<td><strong>Format Paragraph</strong></td>
<td>Reflows the text in the current paragraph according to the margins.</td>
<td><strong>reflow_paragraph</strong></td>
</tr>
<tr>
<td><strong>Format Selection</strong></td>
<td>Reflows the selected text according to the margins.</td>
<td><strong>reflow_selection</strong></td>
</tr>
<tr>
<td><strong>Format Columns</strong></td>
<td>Format columns according to words.</td>
<td><strong>format_columns</strong></td>
</tr>
<tr>
<td><strong>Add Selected Expr</strong></td>
<td>Adds the result of evaluating each line in a selected area of text.</td>
<td><strong>add</strong></td>
</tr>
<tr>
<td><strong>Quick Refactoring</strong></td>
<td>Displays Quick Refactoring menu. See <a href="#">Quick Refactoring Menu</a>.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Imports</strong></td>
<td>Displays Imports refactoring menu. See <a href="#">Imports Menu</a>.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Sort</strong></td>
<td>Sorts current buffer or selected text. See <a href="#">Sorting Text</a>.</td>
<td><strong>gui_sort</strong></td>
</tr>
<tr>
<td><strong>Spell Check</strong></td>
<td>Displays menu of spell checking commands. See <a href="#">Spell Check Menu</a>.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Edit Javadoc Comment</strong></td>
<td>Edits Javadoc comments for the current source file.</td>
<td><strong>javadoc_editor</strong></td>
</tr>
<tr>
<td><strong>Comment Block</strong></td>
<td>Converts selected text into block comment using box comment setup characters. See</td>
<td><strong>box</strong></td>
</tr>
<tr>
<td>Document Menu Item</td>
<td>Description</td>
<td>Command</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Comment Lines</td>
<td>Converts selected lines into line comments using the line comment setup. See Commenting.</td>
<td>comment</td>
</tr>
<tr>
<td>Uncomment Lines</td>
<td>Uncomment any commented lines and ignores any that isn't commented. See Commenting.</td>
<td>comment_erase</td>
</tr>
<tr>
<td>Reflow Comment</td>
<td>Reflow and reformats the current block comment. See Reflow Comment Dialog.</td>
<td>gui_reflow_comment</td>
</tr>
<tr>
<td>Comment Setup</td>
<td>Displays the language-specific Comment options screen, which contains settings for box and line comments. See Language-Specific Comment Options.</td>
<td>comment_setup</td>
</tr>
<tr>
<td>Comment Wrap</td>
<td>Toggles comment wrap on/off.</td>
<td>comment_wrap_toggle</td>
</tr>
<tr>
<td>XML/HTML Formatting</td>
<td>Displays the XML/HTML Formatting menu. See Document Menu.</td>
<td>N/A</td>
</tr>
<tr>
<td>Select Mode</td>
<td>List all modes and lets you select one. See Language Editing Mode.</td>
<td>select_mode</td>
</tr>
<tr>
<td>Language Options</td>
<td>Displays the Options dialog open to the language-specific General options screen for the language in the current buffer. See Language Options.</td>
<td>setupext</td>
</tr>
<tr>
<td>Tabs</td>
<td>Sets tab stops.</td>
<td>gui_tabs</td>
</tr>
<tr>
<td>Margins</td>
<td>Sets margins.</td>
<td>guiMargins</td>
</tr>
<tr>
<td>Indent with Tabs</td>
<td>Toggles indenting with tabs on/off. See Syntax Indent.</td>
<td>indent_with_tabs_toggle</td>
</tr>
<tr>
<td>Word Wrap</td>
<td>Toggles word wrap on/off.</td>
<td>word_wrap_toggle</td>
</tr>
<tr>
<td>Justify</td>
<td>Sets/displays paragraph</td>
<td>gui_justify</td>
</tr>
</tbody>
</table>
Format Menu

<table>
<thead>
<tr>
<th>Document Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>justification style.</td>
<td></td>
</tr>
<tr>
<td>Read Only Mode</td>
<td>Toggles read-only mode on/off.</td>
<td>read_only_mode_toggle.</td>
</tr>
<tr>
<td>Adaptive Formatting</td>
<td>Toggles Adaptive Formatting on/off. See Adaptive Formatting.</td>
<td>adaptive_format_toggle</td>
</tr>
</tbody>
</table>

Quick Refactoring Menu

The **Format → 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.

<table>
<thead>
<tr>
<th>Quick Refactoring Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename</td>
<td>Rename symbol.</td>
<td>refactor_quick_rename</td>
</tr>
<tr>
<td>Extract Method</td>
<td>Extract the selected code block into a new function.</td>
<td>refactor_quick_extract_method</td>
</tr>
<tr>
<td>Modify Parameter List</td>
<td>Modify the parameter list of a method.</td>
<td>refactor_quick_modify_params</td>
</tr>
<tr>
<td>Encapsulate Field</td>
<td>Encapsulate field using Context Tagging®.</td>
<td>refactor_quick_encapsulate_field</td>
</tr>
<tr>
<td>Replace Literal with Constant</td>
<td>Replace literal value with a declared constant.</td>
<td>refactor_quick_replace_literal</td>
</tr>
</tbody>
</table>

Imports Menu

The **Format → Imports** menu contains options for organizing Java imports. For more information, see Organize Imports.

<table>
<thead>
<tr>
<th>Imports Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organize Imports</td>
<td>Organize import statements in a Java file.</td>
<td>jrefactor_organize_imports</td>
</tr>
<tr>
<td>Add Import</td>
<td>Add import statement for symbol under cursor.</td>
<td>jrefactor_add_import</td>
</tr>
</tbody>
</table>
## Imports Menu Item

<table>
<thead>
<tr>
<th>Imports Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td>Displays the Options dialog open to Organize Java Imports node. See <a href="#">Organize Imports</a>.</td>
<td>refactor_organize_imports_options</td>
</tr>
</tbody>
</table>

## Spell Check Menu

The **Format → 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.

<table>
<thead>
<tr>
<th>Spell Check Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check from Cursor</td>
<td>Spell check starting from cursor.</td>
<td>spell_check</td>
</tr>
<tr>
<td>Check Comments and Strings</td>
<td>Spell check comments and strings starting from cursor.</td>
<td>spell_check_source</td>
</tr>
<tr>
<td>Check Selection</td>
<td>Spell check words in selection.</td>
<td>spell_check_selection</td>
</tr>
<tr>
<td>Check Word at Cursor</td>
<td>Spell check word at cursor.</td>
<td>spell_check_word</td>
</tr>
<tr>
<td>Check Files</td>
<td>Spell check multiple source files.</td>
<td>spell_check_files</td>
</tr>
<tr>
<td>Spell Options</td>
<td>Display/modify spell checker options.</td>
<td>spell_options</td>
</tr>
</tbody>
</table>

## XML/HTML Formatting Menu

The **Format → XML/HTML Formatting** menu is used to activate these features in SlickEdit Core. You can enable and/or disable [Tag Layout](#) and/or [Content Wrap](#) for either file type on a global basis or on a per document basis. See [XML/HTML Formatting](#) for more information.

<table>
<thead>
<tr>
<th>XML/HTML Formatting Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Document Options</td>
<td>Displays the Current Document Options Dialog.</td>
<td>xml_html_document_options</td>
</tr>
<tr>
<td>Enable XML Formatting</td>
<td>Toggles XML Formatting on/off.</td>
<td>xml_formatting_toggle</td>
</tr>
<tr>
<td>(XML) Content Wrap</td>
<td>Toggles XML Content Wrap on/off.</td>
<td>xml_contentwrap_toggle</td>
</tr>
</tbody>
</table>
Format Dialogs and Tool Windows

This section describes the dialogs and views that are associated with the **Format** menu items.

**Reflow Comment Dialog**

The Reflow Comment dialog (**Format** → **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 (Format → 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.

**Current Document Options Dialog**

The Current Document Options dialog allows you to enable/disable aspects of XML or HTML Formatting for just the current document. It can be displayed by clicking Format → XML/HTML Formatting →
Current Document Options, or by using the `xml_html_document_options` command.

The dialog contains the following:

- **Formatting scheme** - This drop-down specifies the formatting scheme applied to this document. Choose from the list of available schemes.

- **Auto formatting options** - These are the aspects of XML/HTML Formatting that can be enabled/disabled for the current document.

- **Configure Schemes button** - Displays the XML/HTML Formatting Scheme Configuration Dialog, which allows you to modify or create a new scheme to apply to the current document.

For more detailed information, see **Enabling/Disabling for the Current Document**.

**XML/HTML Formatting Scheme Configuration Dialog**

This dialog is used to configure the way XML and HTML code is automatically formatted as you edit. Note that XML/HTML Formatting must be enabled in order for these settings to work. See **XML/HTML Formatting** for more information.

To access this dialog, from the main menu, click **Format → XML/HTML Formatting → Current Document Options**, then click the **Configure Schemes** button on the Current Document Options dialog. Or, use the `xml_html_options` command to display the dialog.
The Scheme column shows a list of available schemes. The Tags column shows a list of tags associated with the selected scheme. Right-click in this column for a menu of operations such as add, save, and more. See Working with Schemes and Working with Tags for more details.

A Preview at the bottom of the dialog shows how the selected tag will be formatted in your code. Click the XML Options or HTML Options button to jump to the language-specific Formatting Options page for each language (this is the same as Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Formatting).

The following sections describe the tabs and how each setting works. Before configuring settings, be sure the scheme and tag(s) that you want to affect are selected.

Caution
XML/HTML Formatting does not currently affect XML or HTML Beautifier settings. If you run the beautifier on documents that have been automatically formatted through XML/HTML Formatting, you may find unexpected results.

General Settings

The General tab of the XML/HTML Formatting dialog is shown below.

It contains the following general settings:

- **End tags settings** - These options control how the end tags for the selected tag are formatted:
  - **Has end tag** - This option tells SlickEdit Core whether or not the specified tag is intended to be closed with an end tag. This information is used for SlickEdit Core to know when to start or stop calculating information based on the tag. For example, the `<div>` tag in HTML has a start and end tag, while the `<br>` tag does not have an end tag.
  - **Insert end tags on '>'** - When selected, SlickEdit Core automatically inserts the end tag after you type the closing brace (>) of the start tag. For example, when you type `<div>`, `</div>` is automatically inserted. The placement of the inserted tag depends on other settings you have specified, such as whether or not the end tag should be on a separate line (specified on the Tag Layout tab).

- **Match tag style of** - Select this option if you want the selected tag's settings to match the style of
another tag, then pick the tag to use from the drop-down list. This can be a time-saver when adding a batch of new tags that should all have the same settings.

- **Scheme uses case-sensitive tag search** - When this option is selected, tags in the Tags column are displayed in the list exactly as you have typed them, with the case preserved. When you type the tags in the editor, the case must match exactly or the tag will not be recognized (and the (default) tag settings will be applied). This option is appropriate for XML. HTML is not case-sensitive.

### Content Wrap Settings

The **Content Wrap** tab of the XML/HTML Formatting dialog contains options for specifying how wrapping should occur for the selected tag's content.

There are three main (mutually-exclusive) options:

- **Wrap tag content** - When selected, content between tags is wrapped according to the settings specified in the **Tag content width** group box. See **Tag Content Width Settings** below for details.

- **Treat as content** - When selected, the tag as well as its content is wrapped according to the parent tag content. The tag is treated as "inline" with the surrounding text. This could be useful for style tags such as `<b>` or `<i>` that you want to appear "inline" with the content. For example:

  ```html
  <div>
    This is a sample paragraph with the <b>bold tags</b> being treated as
  </div>
  ```
content, "inline" with the rest of the text.
</div>

- **Preserve content** - When selected, content between start and end tags is not wrapped, but the tags are laid out properly with the parent tag. This could be useful for tags such as `<pre>`, where the content needs to be rendered exactly as it appears in the code.

**Note**

- When **Treat as content** is selected, the wrapping options in the **Tag content width** group box, as well as options on the **Tag Layout** tab, are unavailable.
- When **Preserve content** is selected, the wrapping options in the **Tag content width** group box are unavailable.

**Tag Content Width Settings**

When **Wrap tag content** is selected, content between tags is wrapped according to the settings specified in this group box. The options **Fixed width**, **Automatic width**, and **Fixed right margin** are mutually exclusive.

- **Fixed width** - When selected, tag content is formatted to the specified width. The original left margin of the content is maintained, and the right margin is adjusted to meet the target width.

  If **Maximum right column** is used, lines will be wrapped when they reach the specified column, even if they have not reached the specified fixed width. This is useful if coding standards mandate that text should not exceed a specified column.

- **Automatic width** - When selected, the width of the longest multi-line paragraph in the tag’s content is used as the width. This is useful for preserving the formatting of existing content.

  If **Maximum right column** is used, lines will be wrapped when they reach the specified column, even if they have not reached the specified automatic width. This is useful if coding standards mandate that text should not exceed a specified column.

- **Fixed right margin** - When selected, lines will break before the specified number of columns in the **Right column** field has been reached.

**Tip**

If coding standards mandate that text should not exceed a specified column, you can still use Fixed or Automatic width settings. Select and set the **Maximum right column**, and lines will be wrapped when they reach the specified column, even if they have not reached the specified fixed or automatic width.
• **Parent tag right margin** - When selected, tag content is wrapped at the right margin to the width of the parent tag.

• **Include tags in width calculation** - When selected, the start and end tag characters are counted in addition to the content. The number of characters and spaces (including attributes) within tags are calculated, and the specified width is adjusted accordingly. This is useful for producing uniform blocks of text.

• **Preserve width of existing content** - When selected, SlickEdit Core preserves the width of the existing content while editing. The width is determined by the length of the longest multi-line paragraph. If the width of the existing content cannot be determined, the formatting option specified (Fixed, Automatic, or Fixed right) will be used instead.

**Tag Layout Settings**

The **Tag Layout** tab of the XML/HTML Formatting dialog contains options to control the location of the start tag, end tag, and the content between them.

- **Start tag on separate line** - When selected, the start tag occurs on a separate line, and the cursor will be placed on the line below for you to type the content. Note the cursor location in the examples below.

- **End tag on separate line** - When selected, the end tag placed on a separate line below the content.
Example of both settings checked:

```html
<div>
  |
</div>
```

Example of both settings unchecked:

```html
<div>|</div>
```

Example of start checked and end unchecked:

```html
<div>
  |
</div>
```

**Note**

Dynamic Surround is triggered when you type a tag that has both options **Start and End tag on separate line** checked. Note that Dynamic Surround cannot wrap content and only indents to match your indent style setting (Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Indent). See Syntax Indent for more information about indent styles.

- **Content indent** - These mutually exclusive options control how content between tags is indented:
  - **Match extension indent style** - When selected, tag content is indented according to your indent style setting (Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Indent). See Syntax Indent for more information about indent styles.
  - **Indent** - When selected, indenting for the tag occurs at the column number specified in the spin box. When **Indent after start tag '>'** is selected, indenting is relative to the close bracket. Otherwise indenting is relative to the open bracket.
  - **Nested tag indent** - Indenting is activated after the end tag is typed (or automatically inserted if **Insert end tags on '>'** is checked on the General tab). These mutually exclusive settings apply to all tags in the selected scheme:
    - **Match extension indent style** - When selected, SlickEdit Core indents the selected tag according to your indent style setting (Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Indent). See Syntax Indent for more information about indent styles.
    - **Match content indent** - When selected, SlickEdit Core indents the selected tag to the same level as its parent content.
    - **Insert line breaks** - The values for **Before start tag** and **After end tag** specify the number of line breaks that are inserted before and after start and end tags. Note that in order to insert one or more
blank lines, the values should be set to 2 or higher.
Display

This section describes items related to viewing and displaying within the editor. For more information, see Viewing and Displaying.

Display Menu

The Display menu contains options that pertain to viewing and displaying special characters, code, and comments.

<table>
<thead>
<tr>
<th>Display Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>Toggles hex/ASCII display. See Hex Mode Editing.</td>
<td>hex</td>
</tr>
<tr>
<td>Line Hex</td>
<td>Toggles line hex/ASCII display. See Hex Mode Editing.</td>
<td>linehex</td>
</tr>
<tr>
<td>Special Chars</td>
<td>Toggles viewing of tabs, spaces, and new line character(s) on/off. See Viewing Special Characters.</td>
<td>view_specialchars_toggle</td>
</tr>
<tr>
<td>New Line Chars</td>
<td>Toggles viewing of new line character(s) on/off. See Viewing Special Characters.</td>
<td>view_nlchars_toggle</td>
</tr>
<tr>
<td>Tab Chars</td>
<td>Toggles viewing of tab character(s) on/off. See Viewing Special Characters.</td>
<td>view_tabs_toggle</td>
</tr>
<tr>
<td>Spaces</td>
<td>Toggles viewing of space character(s) on/off. See Viewing Special Characters.</td>
<td>view_spaces_toggle</td>
</tr>
<tr>
<td>Line Numbers</td>
<td>Toggles the display of line numbers on/off for the current document. See Viewing Line Numbers.</td>
<td>view_line_numbers_toggle</td>
</tr>
<tr>
<td>Soft Wrap</td>
<td>Toggles wrapping of long lines to window width.</td>
<td>softwrap_toggle</td>
</tr>
<tr>
<td>Symbol Coloring</td>
<td>Displays Symbol Coloring menu. See Symbol Coloring Menu.</td>
<td>N/A</td>
</tr>
<tr>
<td>Display Menu Item</td>
<td>Description</td>
<td>Command</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Language View Options</td>
<td>Displays View options for the language in the current buffer. This is the same as if you had selected Window → SlickEdit Preferences → Languages → Application Languages → C/ C++ → View from the main menu. See Language-Specific View Options.</td>
<td>setupext -view</td>
</tr>
<tr>
<td>Overview Ruler</td>
<td>Toggles the display of the vertical bar on the right of the editor pane, which is used to display the locations of errors, bookmarks, etc.</td>
<td>N/A</td>
</tr>
<tr>
<td>Fullscreen</td>
<td>Toggles full screen editing mode.</td>
<td>fullscreen</td>
</tr>
<tr>
<td>Selective Display</td>
<td>Displays the Selective Display dialog, which allows you to hide lines and create an outline. See Selective Display.</td>
<td>selective_display</td>
</tr>
<tr>
<td>Hide All Comments</td>
<td>Hides all lines that only contain a comment.</td>
<td>hide_all_comments</td>
</tr>
<tr>
<td>Hide Code Block</td>
<td>Hides lines inside current code block. See Expanding/Collapsing Code Blocks.</td>
<td>hide_code_block</td>
</tr>
<tr>
<td>Hide Selection</td>
<td>Hides selected lines.</td>
<td>hide_selection</td>
</tr>
<tr>
<td>Hide #region Blocks</td>
<td>Hides .NET #region blocks.</td>
<td>hide_dotnet_regions</td>
</tr>
<tr>
<td>Function Headings</td>
<td>Collapses all function code blocks in the current file. See Selective Display.</td>
<td>show_procs</td>
</tr>
<tr>
<td>Expand/Collapse Block</td>
<td>Toggles between hiding and showing the code block under the cursor. See Expanding/Collapsing Code Blocks.</td>
<td>plusminus</td>
</tr>
<tr>
<td>Copy Visible</td>
<td>Copies text not hidden by</td>
<td>copy_selective_display</td>
</tr>
</tbody>
</table>
Display Dialogs and Views

Selective Display Dialog

Symbol Coloring Menu
The table, below, describes each item on the Display → Symbol Coloring menu and its corresponding command. For more information see Symbol Coloring.

<table>
<thead>
<tr>
<th>Symbol Coloring Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customize</td>
<td>Opens the Symbol Coloring preferences screen.</td>
<td>config Symbol Coloring</td>
</tr>
<tr>
<td>Enable Symbol Coloring</td>
<td>Turns Symbol Coloring on/off for this file.</td>
<td>symbol_coloring_toggle</td>
</tr>
<tr>
<td>Highlight Unidentified Symbols</td>
<td>Turns on/off highlighting of symbols that are unidentified by the Symbol Coloring engine.</td>
<td>symbol_coloring_errors_toggle</td>
</tr>
<tr>
<td>All symbols - Default</td>
<td>Selects the default, All symbols scheme.</td>
<td>symbol_coloring_set_scheme</td>
</tr>
<tr>
<td>Global Variables</td>
<td>Selects the Global Variables scheme.</td>
<td>symbol_coloring_set_scheme</td>
</tr>
<tr>
<td>Protected and Private</td>
<td>Selects the Protected and Private scheme.</td>
<td>symbol_coloring_set_scheme</td>
</tr>
<tr>
<td>Unidentified Symbols Only</td>
<td>Selects the Unidentified Symbols Only scheme.</td>
<td>symbol_coloring_set_scheme</td>
</tr>
</tbody>
</table>

Display Dialogs and Views

Selective Display Dialog
The Selective Display dialog (Display → 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. Each region contains settings that are specific to that region. The dialog also contains static options for expanding. See Selective Display for more information about working with this feature.

Search Text

Select Search text to specify a search string and display lines containing the search string specified or lines not containing the search string specified. Click the right-pointing arrow button to the right of the field to display a menu containing specific search syntax options such as Character in Range, Beginning of Line, and Decimal Digit. The following settings are available:

- **Match case** - When checked, a case sensitive search is performed.

- **Match whole word** - When checked, a word search is performed. Before a search is considered successful, the characters to the left and right of the occurrence of the search string found are checked to be non-word characters. The default word characters are [A-Za-z0-9_] and may be changed by using the Word chars field on the language-specific General options screen (see Language-Specific Display Dialogs and Views).
General Options)

- **Regular expression** - When checked, a regular expression search is performed. See Find and Replace with Regular Expressions for more information.

- **Reset selective display** - When checked, all lines are made visible and **Plus** and/or **Minus** bitmaps are removed before a search is performed.

- **Hide matched lines** - When checked, lines containing the search pattern are hidden.

### Function Headers

Select **Function headers** to display only function headings and optional function heading comments. The following settings affect how comments before function headings are handled:

- **Show comments** - When checked, comments above function headings are displayed as if they were part of the function heading.

- **Collapse comments** - When checked, comments above function headings are visible but multi-line comments will require that you expand them to see all comments.

When both options are off, comments will not be visible at all, making it difficult to copy or move functions and comments.

### Preprocessor Directives

Select **Preprocessor directives** to display a source file as if it were preprocessed according to the define values you specify. If you do not remember your defines, use the **Scan for Defines** button. The following settings are available:

- **Defines** - Specifies defines and optional values used when you select the **Preprocessor Directives** option on the Selective Display dialog box. The syntax is:

  
  \[name1 \[=value1\] name2 \[=value2\]\]

  
  For example:

  
  `WIN32S VERSION=4`

- **Warning if Not Defined** - If on when you preprocess your source, a message box is displayed for each define found in an expression which does not have a value.

- **Scan for Defines** - Searches for define variables in the current source file and lets you specify values. Resulting values are placed in the **Defines** combo box.

### Multi-Level

Select **Multi-level** to set multiple levels of selective display based on braces or indent. The following settings are available:
• **Braces** - When on, multiple levels of selective display are set to correspond to curly brace nesting levels.

• **Indentation** - When on, multiple levels of selective display are set to correspond to indentation levels.

• **Limit levels** - When too many nested levels of selective display get confusing, place a limit on the maximum number of nested levels. Nesting deeper than this specified level is ignored.

**Paragraphs**

Select **Paragraphs** to display the first line of each paragraph. A paragraph is defined by a group of lines followed by one or more blank lines.

**Hide Selection**

Select **Hide selection** to hide the lines in the current selection.

**Expansion Options**

The following expansion options can be applied for each region:

• **Expand sub-levels** - When on, expanding hidden lines expands all nested hidden lines.

• **Collapse sub-levels** - When on, expanding hidden lines collapses all nesting hidden lines.

• **Remember sub-levels** - When on, expanding hidden lines displays nested hidden lines the way they were last displayed.
Navigate

This section documents the menus and dialogs associated with the Navigate menu.

Navigate Menu

The standard Eclipse Navigate menu has the following items added by SlickEdit Core.

<table>
<thead>
<tr>
<th>Navigate Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Go to Line</strong></td>
<td>Places the cursor on a line you specify. See Navigating to a Specific Line.</td>
<td>gui_goto_line</td>
</tr>
<tr>
<td><strong>Go to Offset</strong></td>
<td>Places the cursor on a byte/character offset in the current file.</td>
<td>gui_seek</td>
</tr>
<tr>
<td><strong>Go to Matching Parenthesis</strong></td>
<td>Finds the matching parenthesis or begin/end structure pair. See Begin/End Structure Matching.</td>
<td>find_matching_paren</td>
</tr>
<tr>
<td><strong>Go to Definition</strong></td>
<td>Pushes a bookmark at the cursor and navigates to the definition of the current symbol. See Symbol Navigation.</td>
<td>gui_push_tag</td>
</tr>
<tr>
<td><strong>Go to Reference</strong></td>
<td>Searches for references to the symbol under the cursor. See Symbol Navigation.</td>
<td>push_ref</td>
</tr>
</tbody>
</table>

Navigate Dialogs and Views

This section describes the dialogs that are associated with entries on the Navigate menu.

Select Symbol Dialog

The Select Symbol dialog is automatically displayed when you use the Go to Definition feature (by pressing Ctrl+Dot, clicking Navigate → Go to Definition from the main menu, or using the push_tag command), when multiple code locations match the selected symbol. 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 Core 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 Window → SlickEdit Preferences → 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, and this dialog does not appear again unless there is more than one unique definition or declaration.

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.

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 view. 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.
Search Menu

The Search menu contains items pertaining to search and replace, navigation, and bookmarks.

<table>
<thead>
<tr>
<th>Search Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push Bookmark</td>
<td>Pushes a bookmark at the cursor. See Bookmarks.</td>
<td>push_bookmark</td>
</tr>
<tr>
<td>Pop Bookmark</td>
<td>Pops the last bookmark. See Bookmarks.</td>
<td>pop_bookmark</td>
</tr>
<tr>
<td>Bookmark Stack</td>
<td>Displays the Bookmark Stack tool window. See Bookmark Stack Dialog.</td>
<td>pop_bookmark</td>
</tr>
<tr>
<td>Set Bookmarks</td>
<td>Sets a persistent bookmark on the current line. See Bookmarks.</td>
<td>set_bookmark</td>
</tr>
<tr>
<td>Go to Bookmark</td>
<td>Displays the Go to Bookmark dialog from which you can select a bookmark to navigate to. See Navigating Named Bookmarks.</td>
<td>goto_bookmark</td>
</tr>
<tr>
<td>Toggle Bookmark</td>
<td>Toggles setting a bookmark on the current line. See Bookmarks.</td>
<td>toggle_bookmark</td>
</tr>
</tbody>
</table>

Search Dialogs and Views

This section describes the dialogs and views that are associated with searching and replacing. Note that default search options are also available (Window → SlickEdit Preferences → Editing → Search). See Search Options for a description of these settings.

Find and Replace View

This view is displayed when you click one of the find or replace items on the Search menu, or when you click Window → Show View → Other, then expand SlickEdit and select SlickEdit Search. See Find and Replace for information about searching and replacing in SlickEdit® Core.
The Find and Replace view contains a right-click context menu and four tabs: the Find Tab, the Find in Files Tab, the Replace Tab, and the Replace in Files Tab.

**Find and Replace View: Context Menu**

Right-click in the background of the Find and Replace view to access the following items:

- **Saved Search Expressions** - See Saving Search and Replace Values.

- **Configure Options** - Displays the Search Options screen, from which you can set the default search options that the view 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 view. See Default Search Options.

- **Clear All Options** - Clears all options that are selected in the Find and Replace view.

- **Set Current Options as Default** - If selected, the options that are selected on the view replace the settings in the Search Options.

- **Hide/Show Tabs** - Toggles the display of the tabs on the Find and Replace view.
• **Clear Highlights** - Removes all highlighting from text that was highlighted during a search or replace operation.

**Find Tab**

This tab on the **Find and Replace** view provides fields and options for searching and finding text.

![Find and Replace](image)

- **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 view. 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 **Window → SlickEdit Preferences → 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 (UNIX)**
  - **Regular expression (Brief)**
  - **Regular expression (SlickEdit® Core)**
  - **Regular expression (Perl)**
  - **Wildcards (\*,?)**

See [Using Regular Expressions in SlickEdit®](#) for more information.

- **Color** - Displays the Color Coding Search Preferences dialog. This dialog lets you pick various syntactic elements to filter a search. These are the same elements used by the Color Coding engine. Using these filters helps to reduce the number of false positives you find in a search. Each check box has three states:
  - **Neutral (the default)** - All check boxes start in the neutral state. These elements will be used in a search until 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. `<img src=..>.

- **Wrap at beginning/end** - If selected, the search will always be performed on the entire buffer, starting from the cursor.
- **Place cursor at end** - If selected, the cursor is placed at the end of the occurrence found.

- **Search backward** - Select this option to have the search performed from the end to the beginning.

- **Search hidden text** - Select this option to search for text hidden by Selective Display. Matches found that were set to be hidden by Selective Display will be revealed. To set Selective Display options, from the main menu click **Display → 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 **Window → SlickEdit Preferences → 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 view.

- **Bookmark matches** - Select this option to bookmark lines with matching patterns and display the Bookmarks view when a match is bookmarked.

- **Incremental search** - Select this option to search incrementally on patterns being typed into the **Search for** field, showing the location of the match at the cursor. See **Incremental Searching** for more information on this method of searching.

- **Find button** - Click this button when you have entered all desired search options and are ready to initiate a search. If no matches are found, the **Search for** field will turn red, and the text **String not found** will be displayed in the status area of the editor.

### Find in Files Tab

This tab on the **Find and Replace View** provides the same functionality as the **Find Tab**, with the added ability to conduct multi-file searches. Additional options are described below.
- **Look in** - This field allows you to specify a range for your search to the current selection, current procedure, current buffer or all buffers.
Click the right-pointing arrow button to the right of the **Look in** field to display a menu containing more specific range options such as **Directory**, **Project**, and **All Buffers**. From this sub-menu, you may also select **Append** and choose an item for which to have the search results appended.

- **File types** - Specifies one or more file types (extensions) to search for. Type in this field or use the drop-down list to select the extensions desired. When a file title is specified in the **Look in** field, the file types wildcards are ignored.

- **Exclude** - Paths, files, or file types can be excluded from a multi-file search by specifying them with wildcards or full path names. To specify multiple patterns, separate them with semicolons (colons on UNIX). No files are searched in a path that is excluded, including any files in sub-directories beneath. See examples of exclude patterns, below.

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>math</em>.cpp</td>
<td>Exclude any .cpp with &quot;math&quot; in the file name.</td>
</tr>
<tr>
<td>readme.txt</td>
<td>Exclude all files named readme.txt.</td>
</tr>
<tr>
<td>*.a</td>
<td>Exclude any file with extension .a.</td>
</tr>
<tr>
<td><em>.png;</em>.ico;*.jpg</td>
<td>Exclude any file with extension .png, .ico, or .jpg.</td>
</tr>
<tr>
<td>*/CVS\</td>
<td>Exclude any files in paths named &quot;CVS&quot;.</td>
</tr>
<tr>
<td>/home/build/debug</td>
<td>Exclude all files in this path name.</td>
</tr>
<tr>
<td><em>demo</em></td>
<td>Exclude any file or path with &quot;demo&quot; in the name.</td>
</tr>
</tbody>
</table>

- **Look in subfolders** - Select this option to expand the search to sub-directories of the folder specified in the **Look in** field.

- **Results options** - Click this button to expand or contract the **Results** options section of the view. 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 view with each new search. See **Search Results Output** for more information.

Right-click in the Search Results window to access the following options:

- **Quick Search** - Finds the next occurrence of the text selected.
• **Filter Search Results** - Select this option to display the Filter Search Results dialog. From here, if a match is found, you can choose to keep or delete lines with additional searches, match case, limit to current default regular expression syntax and/or remove matches found on the same line number in the same file (this can also be accomplished by selecting **List matching lines** only from the **Find in Files** tab).

• **Open as Editor window** - Opens current search results in a new editor window.

• **Go to Line** - Goes to the file/line number of the current line in the Search Results window.

• **Bookmark Line** - Places a bookmark at the line in the file where the result was found.

• **Clear Window** - Clears all results in the current Search Results window.

• **Align Columns** - Aligns the line numbers and column numbers for all search results.

• **Collapse All** - Collapses all Selective Display levels. See **Selective Display** for more information.

• **Expand All** - Expands all Selective Display levels. See **Selective Display** for more information.

• **Output to editor window** - If selected, search results are sent to an editor window.

• **Append to output** - Select this option to append search results to the search results window that is in focus.

• **List 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 filenames only** - If selected, only file names and not occurrences are listed in the search output.

• **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® Core 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 View** 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.

- **Replace with** - Enter the text or regular expression for which to replace the item that is searched. You can retrieve previous replacement text or regular expressions by clicking the drop-down list button. Click the right-pointing arrow button to the right of the **Replace with** field to display a menu containing tagged expressions. See [Using Tagged 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 View provides the same functionality as the Replace Tab, with the added ability to conduct multi-file replacements. It contains one additional option, described below.
• **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.

**Find Symbol View**

The Find Symbol view is used to locate symbols in your code. It allows you to search for symbols by name using either a regular expression, substring, or fast prefix match. The view is displayed when you click `Search → Find Symbol` or when you use the `gui_push_tag` command.

See [Find Symbol View](#) under the [Symbol Browsing](#) topic for more information.

- **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® Core displays a progress bar at the top of this view while a search is in progress.

Incremental matches are displayed with each character you type, and the first element in the list is selected. Press Tab to put focus into the list of matches. Press Enter to navigate to the first match. Press Down to select the next match. Press Escape to stop the search.

• Symbol List - The list of search results are refreshed as you type the search string. They include the symbol name, the file that contains it, and the line number. You can sort by any of the three columns.

The selected match is highlighted and is displayed in the Preview view. Single-click or use the arrow keys to select a match. Double-click or press Enter to navigate to that match.

• Look in - Use this control to specify the scope of the symbol search. The options are:

  • <Use Context Tagging®> - This is the default setting. It uses Context Tagging to intelligently determine which tag files to search.

  • <Current File> - Select this setting to only search the tags in the current file, including local variables in the current function scope.

  • <Current Project> - Select this setting to only search in files that are in the current project.

  • <Current Workspace> - Select this setting to only search in files that are in the current workspace.

  • <Language Tag Files> - Select this setting to search all language-specific tag files for the indicated extension. This may also include your workspace tag file.

  • Specific tag files - Select one of the specific tag files listed to limit search to that file.

  • <All Tag Files> - Select this setting to search all tag files for all languages.

• Search Options - The search options can be expanded or collapsed to save space.

  • Match case - When selected, SlickEdit Core uses a case-sensitive search to find symbol matches. When this option is not selected, SlickEdit Core uses a case-insensitive search. When this option is in the neutral (mixed) state, SlickEdit Core 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 substring (slower) - When selected, SlickEdit Core searches for the specified string within the available symbols. For example, finding all symbols containing the word "order," not just those that begin with "order." Selecting this option causes the search to execute more slowly.

  • Use pattern (slower) - When selected, SlickEdit Core interprets the search string as a regular expression or wildcard expression. This can result in slower search times, since SlickEdit Core must test every symbol in the tag file against the regular expression.

  • Filters - Use filters to restrict the search to certain types of symbols. The filters are the same the ones available on the Definitions view. See Outline View for more information.
Search Dialogs and Views

• **Buttons** - The following buttons are located at the bottom of the view:
  
  • **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 (Window → SlickEdit Preferences → 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 View and, optionally, navigates to the first reference. Click Window → SlickEdit Preferences → Editing → Context Tagging and check the option Go to Reference only lists references if you just want to build the list of references. See Code Navigation for more information.
  
  • **Show in symbol browser** - Displays the selected symbol in the Symbols View. 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.

**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 → Bookmark Stack, or use the bookmark_stack command.

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® Core 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.

**Select Symbol Dialog**

The Select Symbol dialog is automatically displayed when you use the Go to Definition feature (by pressing Ctrl+Dot, clicking Navigate → Go to Definition from the main menu, or using the push_tag command), when multiple code locations match the selected symbol. See Symbol Navigation for more information about this feature.

![Select Symbol Dialog](image)

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 Core 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 Window → SlickEdit Preferences → 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, and this dialog does not appear again unless there is more than one unique definition or declaration.

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.
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 view. 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.
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.

<table>
<thead>
<tr>
<th>Macro Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Module</td>
<td>Loads a macro source module.</td>
<td>gui_load</td>
</tr>
<tr>
<td>Unload Module</td>
<td>Unloads a Slick-C macro file from the state file.</td>
<td>gui_unload</td>
</tr>
<tr>
<td>List User-Loaded Macros</td>
<td>Lists user-loaded Slick-C modules. See Macro Dialogs and Views.</td>
<td>gui_list_macfiles</td>
</tr>
<tr>
<td>Record Macro</td>
<td>Starts recording a Slick-C language macro.</td>
<td>record_macro_toggle</td>
</tr>
<tr>
<td>Stop Recording Macro</td>
<td>Stops recording a Slick-C language macro.</td>
<td>record_macro_toggle</td>
</tr>
<tr>
<td>Execute last-macro</td>
<td>Runs last recorded macro.</td>
<td>record_macro_end_execute</td>
</tr>
<tr>
<td>Save last-macro</td>
<td>Saves the last recorded macro under a name you specify. See Save Macro Dialog.</td>
<td>gui_save_macro</td>
</tr>
<tr>
<td>List Macros</td>
<td>Lists saved, recorded macros. See List Macros Dialog.</td>
<td>list_macros</td>
</tr>
<tr>
<td>Set Macro Variable</td>
<td>Allows you to set global macro variables. See Set Variable Dialog and Variable Editor Dialog.</td>
<td>gui_set_var</td>
</tr>
<tr>
<td>Start Slick-C Debugger</td>
<td>Activates the Slick-C debugger window. See &quot;Slick-C Debugger&quot; in the Help → Index.</td>
<td>slickc_debug_start</td>
</tr>
<tr>
<td>Go to Slick-C Definition</td>
<td>Opens a macro source file and</td>
<td>gui_find_proc</td>
</tr>
<tr>
<td>Macro Menu Item</td>
<td>Description</td>
<td>Command</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Macro Menu Item</td>
<td>Description</td>
<td>Command</td>
</tr>
<tr>
<td>Places your cursor on the</td>
<td>definition of a macro symbol.</td>
<td></td>
</tr>
<tr>
<td>Find Slick-C Error</td>
<td>Places your cursor on the macro source line which caused the last</td>
<td>find_error</td>
</tr>
<tr>
<td></td>
<td>interpreter run-time error.</td>
<td></td>
</tr>
<tr>
<td>New Form</td>
<td>Opens a new form for editing with the Dialog editor.</td>
<td>new_form</td>
</tr>
<tr>
<td>Open Form</td>
<td>Opens an existing or new form for editing with the Dialog editor.</td>
<td>open_form</td>
</tr>
<tr>
<td>Selected Form</td>
<td>Displays edited form window currently selected.</td>
<td>show_selected</td>
</tr>
<tr>
<td>Load and Run Form</td>
<td>Loads form, loads Slick-C code, and runs the currently selected/edited form.</td>
<td>run_selected</td>
</tr>
<tr>
<td>Grid</td>
<td>Sets form grid settings. This affects the distance displayed between the</td>
<td>gui_grid</td>
</tr>
<tr>
<td></td>
<td>dots on a form that is being edited. See Grid Settings Dialog.</td>
<td></td>
</tr>
<tr>
<td>Menus</td>
<td>Lists all menus and allows you to edit, create, delete, or show menus.</td>
<td>open_menu</td>
</tr>
<tr>
<td></td>
<td>Provides access to the Menu Editor dialog box. See Menu Editor Dialog.</td>
<td></td>
</tr>
<tr>
<td>Insert Form or Menu Source</td>
<td>Inserts source code into current file for a form or menu you specify.</td>
<td>insert_object</td>
</tr>
</tbody>
</table>

**Macro Dialogs and Views**

**User-Loaded Modules Dialog**

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® Core command line.

![User-Loaded Modules](image)

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 Core.

When you add a module to the list, SlickEdit Core prompts to load the module in the editor. When you delete a module from the list, SlickEdit Core 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
• **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** 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® Core command line.

![List Macros Dialog](image)

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).

![Set Variable Dialog](image)

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** - Display first level of variable with nothing expanded.
* **Update** - Sets the contents of the variable to what is currently displayed in the Variable Editor.
• **Refresh** - Cancels changes and displays current value of variable which is not necessarily the same as when this dialog box was originally displayed.

• **Squish** - Deletes array items which have the value `_notinit`.

**Grid Settings Dialog**

The Grid Settings dialog (Macro → Grid or gui_grid command) is used to set the width and height of grid dots displayed on forms when you use the Dialog Editor. These settings affect the distance between the dots on a form that is being edited.

The width and height parameters are in twips (1440 twips equal one inch on the display).

**Menu Editor Dialog**

The Menu Editor dialog, shown below, contains options for editing menus. To access this dialog, 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® Core 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.
This section describes items on the **Tools** menu and associated dialogs and views.

## Tools Menu

The table below describes each item on the **Tools** menu and its corresponding command.

<table>
<thead>
<tr>
<th>Tools Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quick Start Configuration</strong></td>
<td>Displays the Quick Start Configuration Wizard. See <a href="#">Quick Start Configuration Wizard</a>.</td>
<td>quick_start</td>
</tr>
<tr>
<td><strong>Regex Evaluator</strong></td>
<td>Shows the Regex Evaluator view. See <a href="#">The Regex Evaluator</a>.</td>
<td>activate_regex_evaluator</td>
</tr>
<tr>
<td><strong>OS Shell</strong></td>
<td>Runs operating system command shell (DOS).</td>
<td>dos</td>
</tr>
<tr>
<td><strong>OS File Browser</strong></td>
<td>Runs operating system file system browser. See <a href="#">OS File Browser</a>.</td>
<td>explore or finder</td>
</tr>
<tr>
<td><strong>Calculator</strong></td>
<td>Displays the Calculator, which allows you to evaluate mathematical expressions. See <a href="#">Using the Calculator and Math Commands</a>.</td>
<td>calculator</td>
</tr>
<tr>
<td><strong>ASCII Table</strong></td>
<td>Opens ASCII table file.</td>
<td>ascii_table</td>
</tr>
<tr>
<td><strong>File Merge</strong></td>
<td>Displays the 3-Way Merge Setup dialog, which provides settings for merging two sets of changes made to a file. See <a href="#">3-Way Merge</a> and <a href="#">Tools Dialogs and Views</a>.</td>
<td>merge</td>
</tr>
<tr>
<td><strong>File Difference</strong></td>
<td>Displays the DIFFzilla® dialog, which allows you to view and edit differences between files. See <a href="#">DIFFzilla®</a> and <a href="#">DIFFzilla® Dialog</a>.</td>
<td>diff</td>
</tr>
<tr>
<td><strong>Tag Files</strong></td>
<td>Displays a dialog which allows you to build tag files for use by the</td>
<td>gui_make_tags</td>
</tr>
</tbody>
</table>
## Tools Menu

<table>
<thead>
<tr>
<th>Tools Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Symbols view and other Context Tagging® features. See <a href="https://www.example.com">Creating Language-Specific Tag Files</a> and <a href="https://www.example.com">Context Tagging - Tag Files Dialog</a>.</td>
<td></td>
</tr>
<tr>
<td>Retag Workspace</td>
<td>Retags the files in your workspace. See <a href="https://www.example.com">Creating Tag Files for Workspace Files</a>.</td>
<td>build_workspace_tagfiles</td>
</tr>
</tbody>
</table>

## Beautify Menu

The Format → **Beautify** menu items are summarized in the table below. For more information about working with beautifiers, see [Beautifying Code](https://www.example.com).

<table>
<thead>
<tr>
<th>Beautify Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beautify</td>
<td>Beautifies the current buffer with the current beautifier settings.</td>
<td>beautify</td>
</tr>
<tr>
<td>Beautify With</td>
<td>(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.</td>
<td>N/A</td>
</tr>
<tr>
<td>Edit Current Profile</td>
<td>(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.</td>
<td>beautifier_edit_current_profile</td>
</tr>
<tr>
<td>Options</td>
<td>Brings up the Beautifier options for the language in the current buffer.</td>
<td>beautifier_options</td>
</tr>
</tbody>
</table>
Tools Dialogs and Views

This section describes the dialogs and views that are associated with the Tools menu items.

3-Way Merge Dialog

The 3-Way Merge dialog (Tools > File Merge), shown below, is used for merging file differences.

![3-Way Merge Dialog Setup](image)

The Ellipses buttons to the right of the text boxes are used to select files. The B buttons to the right of the text boxes are used to select from the open buffers.

The list below describes the remaining fields and settings:
• **Base file** - Specifies the file/buffer name of the original source file before any changes are made.

• **Revision 1** and **2** - Specifies the file/buffer names of the modified versions of the base file.

• **Output file** - Specifies the output file name.

• **Merge style** - The following merge styles are available:
  
  • **Auto merge** - If selected, if a change does not cause a conflict, the change is automatically applied to the output file and no indication is made that the change was already applied.
  
  • **Show changes** - If selected, if a change does not cause a conflict, the change is automatically applied to the output file and the change IS indicated, so that using the **Next Conflict** button will show you the change.

• **Output style** - **Output style** has no effect if there are no conflicts. The following output styles are available:

  • **Interactive** - Provides a friendly side-by-side dialog box which lets you pick the change you want in the output file. It also lets you edit.

  • **Interleaved buffer** - Creates an editor buffer which you must edit to resolve conflicts.

• **Use smart merge** - If selected, the number of conflicts found is reduced.

• **Ignore spaces** - If selected, leading and trailing spaces are ignored. The side-by-side output allows you to easily select the change that you want.

**DIFFzilla**

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.

Diff Types

The following Diff Types are available:

• **Compare Two Files** - Compares two files and shows the differences between them. When this option is selected, after you click OK on this dialog to start the comparison, the interactive Diff dialog is displayed, allowing you to preview the differences one-by-one before committing.

  If the option on the Options tab, **Instead of an interactive dialog, output one buffer with the differences labeled**, is checked, a buffer with the differences between the two files marked up will be displayed instead.

• **Compare Two Folders** - Compares two directories or directory trees, and shows which files do not match. Select **Recurse into subdirectories** to search subdirectories recursively. Click **Previous diff** to load a diff state file (.dif), restoring the saved state of a multi-file diff session.

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. To configure that option, see [Expert mode setup dialog](#).

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** - 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.

• **Use file on disk** - Select this option to diff the file on disk and not the copy in memory.

**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 Code Diff** - turns Source Diff on or off. When Source Diff is off, the icon contains a small, red 'x'. Source Diff is on by default.

**Filespecs**

For multi-file diff, you can set specific file types to compare or to exclude:

- **Filespecs** - Enter a space-delimited list of wildcard file specifications to difference. For example, enter 
  "*.* .c .cpp .* .h" to difference all files with .c, .cpp, and .h extensions.

- **Exclude filespecs** - Enter a space-delimited list of wildcard file specifications to be excluded from the differencing. For example, enter junk* test* to exclude all files with names beginning with the words "junk" or "test". To exclude a specific directory, provide the full path. 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.

**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**
The file compare options, shown above, are described as follows:

- **Expand tabs into spaces before comparing** - When selected, tabs are expanded to the appropriate number of spaces before lines from each file that is compared.

- **Ignore leading spaces before text on each line** - When selected, differences in leading spaces of lines are ignored.

- **Ignore trailing spaces after text on each line** - When selected, differences in trailing spaces at the end of lines are ignored.

- **Ignore all spaces in file** - When selected, differences in spacing between characters in lines are ignored.

- **Compare files case insensitive** - When selected, differences in character casing are ignored.

- **Do not compare newline characters** - When selected, differences in end-of-line characters are ignored. This is useful when comparing UNIX-formatted files with DOS-formatted files.

- **Skip comments at the beginning of the file** - When selected, leading comments are ignored. This is useful if you are using a version control system that automatically inserts comment file headers.

- **Instead of an interactive dialog, output one buffer with the differences labeled** - When selected, a new buffer is created that contains color-coded difference output. You can edit the output buffer. When this option is not selected, the Diff dialog box opens displaying the two files side-by-side and the differences are color-coded.
• **Source Diff** - Determines whether Code 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.

• **Date and Size Optimization** - 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**

![DIFFzilla® dialog setup options](image)

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.
Tools Dialogs and Views

• **Automatic directory mapping** - When selected, the Path 2 text box is automatically updated when you type a directory in the Path 1 text box.

• **During multi-file diff, automatically close after last difference** - When selected, clicking Next Diff on the Diff dialog box when there are no more differences, triggers the Close button on that dialog box.

• **When closing multi-file diff, do not prompt to save results** - suppresses the prompt to save results after a multi-file diff.

• **Put buttons at top of diff dialog** - When selected, the buttons that control operations such as Next Diff, Prev Diff, and Block, are displayed at the top of the Diff dialog box.

• **Launch multi-file diffs in a separate process** - When selected, source trees are diffed in a separate process so you can continue working.

• **Expert mode setup dialog** - When this is on, Compare Two Files or Compare Two Folders will be selected automatically based on whether the paths contain folder names or file names. This was the how SlickEdit 2008 and previous versions worked.

• **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.
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** - 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** - click this button to move the current block of differences from one side to the other.
- **Line** - use this button to move the contents of the current line from one side to the other.
- **Merge All** - merges all changes from the selected side to the other.
- **Save File** - saves the file, including all changes made during the diff session.
- **Read 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.

• **Find** - performs a search in the two files.

• **Undo** - 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** - 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 Window → SlickEdit Preferences → 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**

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
Context Tagging - Tag Files Dialog

The Context Tagging® - Tag Files dialog, shown below, is used to manage all your tag files. For more information on tagging 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.

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:

- **Done** - Saves tag file settings and closes the dialog box.

- **Add Files** - Displays the Add Source Files dialog box, from which you can add a set of files to the currently selected tag file. This button will be unavailable for read-only tag files and auto-updated tag files. If you add files to your workspace tag file, you will be prompted if you want to also add the files to your project.

- **Add Tree** - Displays the Add Tree dialog box, from which you can recursively add a directory of files to the currently selected tag file. This button will be unavailable for read-only tag files and auto-updated tag files. If you add files to your workspace tag file, you will be prompted if you want to also add the files
• **Remove Tag File** - Deletes the currently selected tag file. You will be prompted whether or not to delete the tag file from the list, and then whether or not to permanently delete the tag file from disk. Note that some language-specific tag files are automatically generated, and thus will be automatically regenerated if you delete them.

• **Remove Src. File** - Removes the selected files from the currently selected tag file. If no files are selected, you will be prompted whether or not to remove all source files from the tag file. If you remove files from your workspace tag file, you will be prompted if you want to also remove the files from your project.

• **Options** - Displays the Context Tagging® Options screen for you to configure Context Tagging® options. See [Context Tagging® Options](#) for more information.

• **Up** - Moves the selected tag file higher in the search order. This primarily applies to language-specific tag files (see [Creating Language-Specific Tag Files](#)).

• **Down** - Moves the selected tag file lower in the search order. This primarily applies to language-specific tag files (see [Creating Language-Specific Tag Files](#)).

• **Add Tag File** - Displays the Add Tag File dialog box, which allows you to choose from a list of languages the source type for which to insert the tag file. To automatically create tag files for C++, Java, and .NET, you can instead use the Create Tag Files for Compiler Libraries dialog (see [Creating Tag Files for Compiler-Specific Libraries](#)).

• **Retag Src. Files** - Updates the Context Tagging information for the selected files in the currently selected tag file. If no files are selected, you will be prompted whether or not to retag all source files.

• **Rebuild Tag File** - Displays the Rebuild Tag File dialog box containing options for rebuilding the selected file. See [Rebuilding Tag Files](#).

• **Auto Tag** - Displays the [Create Tag Files for 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](#)).
Preferences

Preferences Dialog

The Preferences dialog is used to configure SlickEdit® Core. To display it, from the main menu, click Window → SlickEdit Preferences, or use the config command on the SlickEdit® Core command line.

The following sections describe how to use the Preferences dialog. For descriptions of individual preferences, skip to the index located at Option Categories.

Using the Preferences Dialog

The Preferences 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 preferences. 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 preferences. The layout of the option panel can be either a property sheet in tabular format (like Window → SlickEdit Preferences → Appearance → General), or a form with radio buttons, check boxes, etc. (like Window → SlickEdit Preferences → Appearance → Colors).
Tip

The Preferences 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:

- 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 preferences 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 preferences with switches** - Click anywhere on the switch to toggle it off and on.

- **For preferences 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 preferences 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 preferences** - Double-click on the color block or press **F4**, and the color picker is displayed.

- **For file and directory path preferences** - 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® Core 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 preferences 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 Preferences 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 Preferences dialog is closed, the view is saved and restored the next time the dialog is opened.

Navigating to Previously Viewed Panels

Some preferences link to other option panels, like preferences 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 preferences page as a favorite for quicker access. To add the current page to your favorites list, click the Add [PreferencesPage] 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 preferences 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 preferences 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 preferences in forms embedded in the Preferences 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 Preferences Dialog**

If you prefer to keep your hands on the keyboard, invoke the Preferences dialog with the `config` command and use keyboard shortcuts to navigate and change preferences.

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 **Window → SlickEdit Preferences → 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 Preferences dialog, or **Esc** to prompt for changes before closing the dialog.

The table below describes all of the available shortcuts.

<table>
<thead>
<tr>
<th><strong>Summary</strong></th>
<th><strong>Keyboard Shortcut(s)</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation</td>
<td><strong>Alt+ [ letter ] or Alt+ [ number ]</strong></td>
<td>Performs the following actions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If neither form nor property sheet is visible, jumps to the first Alt-prefixed shortcut on the Preferences dialog proper (for example, <strong>Alt+C</strong> corresponds to the <strong>Clear</strong> button next to the <strong>Search</strong> field). Note that for most operating systems, Alt-prefixed shortcuts correspond to the underlined letters in labels on forms and dialogs.</td>
</tr>
<tr>
<td>Summary</td>
<td>Keyboard Shortcut(s)</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If a form is visible, jumps to the first Alt-prefixed shortcut on the form (for example, in the <strong>Appearance Colors</strong> form, <strong>Alt+E</strong> corresponds to the <strong>Screen element</strong> list).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If a property sheet is visible, jumps to the first option starting with that letter or number (for example, on the <strong>Window → SlickEdit Preferences → Appearance → General</strong> property sheet, <strong>Alt+T</strong> selects the first option that starts with a &quot;T&quot;, which is <strong>Top of file line</strong>).</td>
</tr>
<tr>
<td>Navigation</td>
<td>Tab</td>
<td>Moves the focus to each area/button on the Preferences dialog. In a form, <strong>Tab</strong> moves between each area/button on the form until reaching the last element and placing focus back on the Preferences dialog proper. In a property sheet, <strong>Tab</strong> jumps back to the dialog proper.</td>
</tr>
<tr>
<td>Navigation</td>
<td>Up and Down</td>
<td>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.</td>
</tr>
<tr>
<td>Navigation (dialog to panel)</td>
<td>Shift+Tab</td>
<td>Jumps from the <strong>OK</strong> button on the Preferences dialog to the currently visible option panel.</td>
</tr>
<tr>
<td>Navigation (tree to panel)</td>
<td>Alt+O</td>
<td>Shifts focus from the tree to the currently visible option panel. Subsequently, if the panel is a property sheet, use the <strong>Up</strong> and <strong>Down</strong> keys to navigate through the preferences. If the panel is a form, use <strong>Tab</strong> to navigate through</td>
</tr>
<tr>
<td>Summary</td>
<td>Keyboard Shortcut(s)</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Navigation (between panels)</td>
<td>Alt+Left and Alt+Right</td>
<td>Navigates backward and forward, respectively, between previously viewed option panels. Same as using the Back and Forward buttons on the Preferences dialog.</td>
</tr>
<tr>
<td>Navigation or exit</td>
<td>Enter</td>
<td>Saves changes and closes the Preferences dialog, or, if inside a text box or combo box, shifts focus back to the option panel. Same as pressing OK on the Preferences dialog.</td>
</tr>
<tr>
<td>Navigation or exit</td>
<td>Esc</td>
<td>Prompts to save any changes and closes the Preferences dialog, or, inside a text box or combo box, shifts focus back to the option panel. Same as pressing Cancel on the Preferences dialog.</td>
</tr>
<tr>
<td>Expands/collapses tree nodes and enables property sheet controls</td>
<td>Right and Left</td>
<td>Expands/collapses nodes in the tree. In property sheets, enables numeric text boxes and drops down combo boxes (but not up).</td>
</tr>
<tr>
<td>Enables property sheet controls</td>
<td>F4</td>
<td>Toggles combo boxes up and down, enables numeric text boxes, and displays color and directory pickers.</td>
</tr>
<tr>
<td>Enables property sheet controls</td>
<td>Alt+Up and Alt+Down</td>
<td>Toggles combo boxes on property sheets up and down, respectively.</td>
</tr>
<tr>
<td>Enables property sheet controls</td>
<td>Space</td>
<td>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).</td>
</tr>
<tr>
<td>Scrolling</td>
<td>Ctrl+Down and Ctrl+Up</td>
<td>Scrolls the option tree and long</td>
</tr>
</tbody>
</table>
### Preferences Dialog

<table>
<thead>
<tr>
<th>Summary</th>
<th>Keyboard Shortcut(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property sheets one line at a time.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scrolling</td>
<td>PageUp and PageDown</td>
<td>Scrolls the option tree and long property sheets one page at a time.</td>
</tr>
<tr>
<td>Clears a directory/file name property</td>
<td>Delete</td>
<td>Clears a directory or file name field, resetting the default value.</td>
</tr>
<tr>
<td>Option search</td>
<td>Alt+C</td>
<td>Clears the Search box on the Preferences dialog. Same as using the Clear button. See Option Search for more information.</td>
</tr>
<tr>
<td>Option favorites</td>
<td>Alt+F</td>
<td>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 Preferences dialog. See Option Favorites for more information.</td>
</tr>
<tr>
<td>Option favorites</td>
<td>Alt+S</td>
<td>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.</td>
</tr>
<tr>
<td>Save changes</td>
<td>Alt+A</td>
<td>Saves all option changes yet leaves the Preferences dialog displayed. Same as using the Apply button on the Preferences dialog.</td>
</tr>
<tr>
<td>Help</td>
<td>Alt+H</td>
<td>Displays the Help topic for the Preferences dialog.</td>
</tr>
</tbody>
</table>

### Option Categories

The tree in the Preferences dialog (Window → SlickEdit Preferences) contains the following category nodes, which are described in subsequent sections:
Appearance Preferences

Appearance preferences (Window → SlickEdit Preferences → Appearance) allow you to customize the look of SlickEdit® Core. You can change fonts and colors, specify which toolbars and views to display, enable display of special characters, and more. Appearance option categories are:

- General Appearance Options
- Color Options
- Symbol Coloring Options
- Font Options
- Special Character Options

General Appearance Preferences

General appearance preferences are shown below (Window → SlickEdit Preferences → Appearance → General).
The preferences are described as follows:

- **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.

- **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.

• **Vertical line column** - Specifies the column in which the editor is to display a vertical line. When the value is set to **0**, the vertical line is not displayed. 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.

• **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® Core command line. See [Command Line Completion](#) for more information.

• **Scroll style** preferences 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** preferences 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.

**Color Preferences**

Color preferences are shown below (Window → **SlickEdit Preferences** → **Appearance** → **Colors**). These preferences let you specify colors for screen elements in SlickEdit® Core and create and manage color schemes. See [Colors](#) for more information.
The preferences are described as follows:

- **Scheme** - Specify the color scheme to use from the drop-down list. Several predefined schemes are available or you can define your own. See Using Color Schemes for more information.

- **Save Scheme As...** - Allows you to save the current color settings as a new color scheme with a name you specify.

- **Reset Scheme** - Resets a SlickEdit Core installed scheme back to its default configuration.

- **Delete Scheme** - Deletes the selected color scheme. Only available for user color schemes.

- **Rename Scheme...** - Allows you to rename the selected color scheme. Only available for user color schemes.

- **Associated symbol coloring scheme** - Each color scheme is associated with a corresponding symbol coloring scheme. This is necessary in order to insure that the symbol colors are compatible with the base colors in the color scheme. When you switch color schemes, you also switch symbol coloring schemes.

Press the **Symbol Coloring...** button to jump to the Symbol Coloring preferences dialog to take a closer look at the available schemes. For more information about symbol coloring, see Symbol Coloring.

- **Reset Colors** - Restores all colors to the values they were when the Color Preferences dialog was
invoked. All color changes to the selected scheme are saved when you exit the dialog or switch schemes.

- **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.

- **Window Text** - This is the color of other text which is not a specific syntax element.

- **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® Core 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** - Click the color square to change colors for the selected element. The Color Picker dialog is displayed, allowing you to pick a color from the palette or set your own custom color using RGB values.

- **Background color** - Click the color square to change colors for the selected element. The Color Picker dialog is displayed, allowing you to pick a color from the palette or set your own custom color using RGB values.

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 lexer 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.

Select the Inherit option to specify that the background color and embedded code color should be inherited from the basic Window Text color. This feature can be used to keep the background colors synchronized among editor color elements.

- **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 and Message 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, bolded, italicized, or underlined. 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 scheme 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.

**Note**

Due to limitations in the Sample Code display, when you switch between color schemes with different background colors, the new color scheme will be automatically applied, as if you had pressed the **Apply** button.

**Color Elements**

Each of the following color elements which can be configured from within SlickEdit® Core.

- **General** - These color preferences 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 preferences 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.

- **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.

- **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 Symbol** - This color is used to highlight identifiers which match one of the user-defined symbols defined in the Color Coding for the current language.

- **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.

- **Selections** - These color preferences 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 opposed 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 it's 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 Display → 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. Unrecognized attributes are colored using the Window Text color. Attribute values are colored using the String 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 Keyword color.

• **XHTML Element in XSL** - This color is used for an XHTML element in an XSL style sheet.

• **Modifications** - These color 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.

  • **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.

  • **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 Display → 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 preferences. 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 column](#) for more information.

• **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® Core window.

  • **Modified file** - This color is used in the **File Tabs** tool window and the **Files** view to indicate that the current file has unsaved modifications.

  • **Modified variable** - This color is used in the debugger view and the to indicate that the 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® Core window.

### Symbol Coloring Preferences

Symbol Coloring preferences are shown below (Window → SlickEdit Preferences → Appearance → Symbol Coloring). These preferences let you specify colors for symbols identified using Context Tagging® and create and manage symbol coloring schemes. See [Symbol Coloring Schemes](#) for more information.
Symbol Coloring settings

The preferences are described as follows:

- **Scheme** - Specify the symbol coloring scheme to use from the drop-down list. Several predefined schemes are available or you can define your own. See Selecting a Symbol Coloring Scheme for more information.

- **Save As...** - Allows you to save the current symbol coloring scheme as a new scheme with a name you specify.

- **Reset Scheme** - Allows you to reset a modified system scheme back to its original configuration.

- **Delete Scheme** - Deletes the selected symbol coloring scheme. Only available for user-created schemes.

- **Rename...** - Allows you to rename the selected symbol coloring scheme. Only available for user-created schemes.

These options let you specify colors for specific symbol types and create and manage symbol coloring schemes. Select the (None) scheme to disable symbol coloring. The “All symbols - Light background” scheme works best with color schemes with white backgrounds. Likewise, “All symbols - Dark background” is optimized for black backgrounds.
**Appearance Preferences**

- **Compatible with** and **Edit Compatibility**... - Each symbol coloring scheme can, optionally, be associated with a set of color schemes that it is compatible with. This is necessary in order to insure that the symbol colors are compatible with the base colors in the color scheme. If a scheme does not list any specific compatible color schemes, it will be regarded as compatible with all color schemes. For example, the **Protected and Private** scheme only changes font attributes and is therefore compatible with any color scheme.

Press the **Edit Compatibility**... button to select which color schemes which are compatible with the currently selected symbol coloring scheme.

Generally, compatibility hinges on the amount of contrast between the selection of foreground colors in the symbol coloring scheme and the selection of background color in the base color scheme. This is why the standard schemes are classified, generally, as **Light** and **Dark** background.

- **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 scheme that matches all of the requirements. See **Color Rules** for detailed descriptions of each of the standard symbol coloring rules shipped with SlickEdit® Core.

- **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, it's 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 schemes 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 it's color information from another symbol coloring rule within the currently selected symbol coloring scheme. It can also inherit color information from certain colors from the basic color scheme, listed below.

  - **--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 preferences 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 Symbol--** - This color is used to highlight identifiers which match one of the user-defined symbols 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.

- **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 it's background color instead of setting it's 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 Preferences 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.
• **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.

• **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 column** - This type is used for the column name of a database table. It only applies to SQL dialects.

• **Database cursor** - This type is used for a database cursor type. It only applies to SQL dialects.

• **Database index** - This type is used for a database index name. It only applies to SQL dialects.

• **Database table** - This type is used for a database table 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 view** - This type is used for a database view name. It only applies to SQL dialects.

• **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 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 COBOL file descriptor declaration.

• **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.

- **Package import or using statement** - This type is used for an import or using statement. Note that 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 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.

**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`.

```cpp
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.

- **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 echos 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 - Default** symbol coloring scheme are described below. These same rules are used in most all the standard symbol coloring schemes, just with different colors, or
a subset of the rules.

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.

• **Local variable** - This rule matches local variables within a function, excluding static local variables. In most schemes, variable types are colored green, mirroring the color used for variables in the Defs view.

• **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 and template argument names. Like local variables, they are colored green, matching the color used for variables in the Defs view.

• **Public member variable** - This rule matches public member variables. In most schemes, variable types are colored green, mirroring the color used for variables in the Defs view.

• **Package member variable** - This rule matches package scope member variables. In most schemes, variable types are colored green, mirroring the color used for variables in the Defs view.

• **Protected member variable** - This rule matches protected member variables. In most schemes, variable types are colored green, mirroring the color used for variables in the Defs view. 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 schemes, variable types are colored green, mirroring the color used for variables in the Defs view. 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 schemes, variable types are colored green, mirroring the color used for variables in the Defs view. 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 schemes, variable types are colored green, mirroring the color used for variables in the Defs view. 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 schemes, variable types are colored green, mirroring the color used for variables in the Defs view. 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 schemes, variable types are colored green, mirroring the color used for variables in the Defs view. 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 schemes, variable types are colored green, mirroring the color used for variables in the Defs view.

• **Static global variable** - This rule matches static global variables, which are visible only within the current module. In most schemes, variable types are colored green, mirroring the color used for variables in the Defs view. 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 schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view.

- **Static global function** - This rule matches static global functions. In most schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view. 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 view.

- **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 view.

- **Public member function** - This rule matches public class member functions. In most schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view.

- **Package member function** - This rule matches package scope class member functions. In most schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view.

- **Protected member function** - This rule matches protected scope class member functions. In most schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view. 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 schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view. 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 schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view. 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 schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view. 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 schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view. 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 schemes, functions are colored magenta, mirroring the color used for functions, procedures, and prototypes in the Defs view. 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 schemes, properties are colored cyan, mirroring the color used for properties in the Defs view.

• **Package class property** - This rule matches package scope class property names. In most schemes, properties are colored cyan, mirroring the color used for properties in the Defs view.

• **Protected class property** - This rule matches protected class property names. In most schemes, properties are colored cyan, mirroring the color used for properties in the Defs view. 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 schemes, properties are colored cyan, mirroring the color used for properties in the Defs view. 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 view.

• **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 view.

• **Abstract class** - This rule matches an abstract class name. Class names are colored blue, mirroring the color used for class constructors in the Defs view.

• **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 view.

• **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 view.

• **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 scheme.

• **Package or namespace** - This rule matches a package or namespace name. In most schemes, they are colored red, mirroring the color used for package, namespace, and programs in the Defs view.

• **Symbolic constant** - This rule matches symbol constants used to give names to constant literal values. In most schemes, they are colored gray, mirroring the color used for constants in the Defs view.

• **Enumerated type or constant** - This rule matches enumerated types and constants defined in enumerated types. In most schemes, they are colored cyan, mirroring the color used for enums in the Defs view.
- **Statement label** - This rule matches statement labels. They are colored blue, matching the color used for labels in the Defs view.

- **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 Preferences

You can specify which fonts are used by screen elements using the preferences shown below (Window → SlickEdit Preferences → Appearance → Fonts). See Fonts for more information about changing fonts and a list of recommendations.

The preferences are described as follows:

- **Element** - This drop-down contains the screen elements for which fonts can be changed. When an element is selected, the font type and size will automatically adjust to the current settings for that element, and a preview of the font will be displayed in the Sample area. Select from the following elements:
Appearance Preferences

- **SBCS/DBCS Source Windows** - Editor windows that are displaying non-Unicode content (for example, plain text).

- **Hex Source Windows** - Editor windows that are being viewed in Hex mode (Display → Hex).

- **Unicode Source Windows** - Editor windows that are displaying Unicode content (for example, XML).

- **File Manager Windows** - Controls the display of the SlickEdit® Core File Manager (File → File Manager).

- **Diff Editor Source Windows** - The editor windows used by DIFFzilla®.

- **Parameter Info** - Controls the fonts used to display pop-ups with information about symbols and parameters.

- **Parameter Info Fixed** - Used when SlickEdit® Core needs to display a fixed-width font for parameter info, such as when displaying example code.

- **Selection List** - The font used for selection lists, like the buffer list (Format → List Buffers).

- **Dialog** - Controls the font used in SlickEdit® Core dialogs and views.

- **HTML Proportional** - The default font used by HTML controls for proportional fonts. In particular, this affects the Version Control History dialog, the About SlickEdit Core dialog, and the Cool Features dialog.

- **HTML Fixed** - The default font used by HTML controls for fixed-space fonts.

- **Font and Size** - The Font and Size fields allow you to make typeface and point size changes to the selected screen element. The fonts that are listed are the fonts that are installed on your computer.

- **Style** - Styles, such as bold and italic, can be set to affect the selected font.

- **Sample area** - This area provides a preview of the selected font, size, and style.

- **Fixed Fonts Only** - Select this option to display only fixed fonts in the Font field. By default, this option is not selected.

- **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.

- **Script (Windows only)** - Choose Default unless you are editing files that have characters not in the active code pages. Choose Western to use the typical English characters.

Special Character Preferences
Special character preferences are shown below (Window → SlickEdit Preferences → Appearance → Special Characters). These preferences 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.

Note

The viewing of special characters is only available for ASCII files.

Keyboard and Mouse Preferences

Keyboard and Mouse preferences (Window → SlickEdit Preferences → Keyboard) pertain to use of the keyboard and mouse, and include preferences 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 Preferences**

Emulation preferences (Window → SlickEdit Preferences → Keyboard → Emulation) are shown below. Use these preferences 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 Preferences tree) or by using the prompt that is displayed when you switch emulations. See Emulations for more information about these settings.

---

**Key Binding Preferences**

Key binding preferences are shown below (Window → SlickEdit Preferences → Keyboard → Key Bindings). From here, you can view, create, and manage key binding associations for SlickEdit® Core commands and user-recorded macros. You can also import, export, or save a chart of your key bindings. See Managing Bindings for more information.
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® Core 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® Core 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.

![Bind Key Dialog](image)

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+RButtonDownDn**, 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 Preferences**

Redefine Common Key preferences (Window → SlickEdit Preferences → Keyboard → Redefine Common Keys) allow you to change the behavior of certain common keys. See also Redefining Common Keys for more information.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace in Replace mode</td>
<td>Remove previous character only</td>
</tr>
<tr>
<td>Backspace over tab</td>
<td>Remove tab</td>
</tr>
<tr>
<td><strong>Redefinable keys</strong></td>
<td></td>
</tr>
<tr>
<td>Backspace key</td>
<td>Cursor wraps to previous line (Default)</td>
</tr>
<tr>
<td>Delete key</td>
<td>Next line always joins (Default)</td>
</tr>
<tr>
<td>End key</td>
<td>Moves cursor to end of line (Default)</td>
</tr>
<tr>
<td>Enter key</td>
<td>Splits current line at cursor (Default)</td>
</tr>
<tr>
<td>Home key</td>
<td>Toggles cursor between first non-blank character and column one (Default)</td>
</tr>
</tbody>
</table>

The preferences are described as follows:

- **Backspace in Replace mode** - Specifies the behavior of the Backspace key when the Start mode is set to Replace (Window → SlickEdit Preferences → 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** preferences 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 your vertical line column and the last non-blank column.

  - **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** (Window → SlickEdit Preferences → 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.

**Advanced Keyboard and Mouse Preferences**

Advanced keyboard preferences are shown below (Window → SlickEdit Preferences → Keyboard → Advanced).

The preferences are described as follows:

• **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-Mac OS X 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 Mac OS X 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® Core 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 (Display → 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 preferences are specific to the Vim emulation and are only available after you have selected it. See Emulation Options.

- **Enter command 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 command mode.

- **Change cursor shape between modes** - when set to On, the cursor will change shape when switching between insert mode and command 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 command mode** - when set to On, the editor will switch to command mode any time you
switch to a different buffer.

**ISPF Options**

The following preferences are specific to the ISPF emulation and are only available after you have selected it. See [Emulation Options](#).

- **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 Preferences**

Editing preferences ([Window → SlickEdit Preferences → Editing](#)) directly impact your SlickEdit® Core coding experience. By customizing these preferences so that SlickEdit® Core works the way you prefer
and to which you are accustomed, you can greatly improve your coding speed and efficiency. Editing preferences include default search/replace values, selection styles, specifying the size of new editor windows, and more.

Editing option categories are:

- General Editing Options
- Cursor Movement
- Context Tagging Options
- Selection Options
- Search Options
- Auto-Close
- Global Alias Options

**General Editing Preferences**

General editing preferences are shown below \(\text{Window} \rightarrow \text{SlickEdit Preferences} \rightarrow \text{Appearance} \rightarrow \text{General}\).
The preferences are described as follows:

- **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® Core 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.

• **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® Core displays a notification. If you attempt to save a read-only file, SlickEdit® Core 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.

• **Parenthesis matching** - See [Begin/End Structure Matching](#) for more information about parenthesis matching. The following preferences available:

  • **Parenthesis matching style** - When **Highlight** is selected, after typing a closing paren, SlickEdit® Core temporarily block-selects the text within the paren pair. When **Cursor to Begin Paren** is selected, after typing a closing paren, SlickEdit® Core temporarily places the cursor on the matching begin paren. When **None** is selected, SlickEdit® Core just inserts the closing paren.

  • **Highlight matching blocks** - When set to **On**, the corresponding parenthesis, brace, bracket, or begin/end word pair under the cursor is automatically highlighted.

**Tip**

To customize the highlight color, go to **Window → SlickEdit Preferences → 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 preferences apply when working with large files. Use these settings to improve performance when working with such files.

  • **Automatically set buffer cache size** - When this value is turned on, the buffer cache size is automatically reset according to the current file size.

  • **Set buffer cache size when file larger than (KB)** - This setting is only available when **Automatically set buffer cache size** is turned on. This setting determines how large a file must be before the buffer cache size is automatically set.
• **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 for all files to improve performance.

## Cursor Movement

Cursor Movement preferences are shown below (Window → SlickEdit Preferences → Editing → Cursor Movement). These preferences control the movement of the cursor within editor windows.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cursor right/left wraps to next/previous line</td>
<td>Off</td>
</tr>
<tr>
<td>Cursor up/down places cursor in virtual space</td>
<td>True</td>
</tr>
<tr>
<td>Click past end of line</td>
<td>On</td>
</tr>
<tr>
<td>Cursor up/down within soft wrapped lines</td>
<td>On</td>
</tr>
<tr>
<td>Cursor left/right in leading spaces</td>
<td>Move cursor by one physical character</td>
</tr>
<tr>
<td>Line wrap</td>
<td>Wrap when left margin is reached</td>
</tr>
<tr>
<td>Jump over tab characters</td>
<td>On</td>
</tr>
<tr>
<td>Subword Navigation</td>
<td>Off</td>
</tr>
<tr>
<td>Undo affects cursor movement</td>
<td>Off</td>
</tr>
</tbody>
</table>

The following preferences 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 On, 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 Off, `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 (Display → 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 (Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → Word Wrap), wrapping occurs when the left margin is reached regardless of this setting.

• **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 Window → SlickEdit Preferences → 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.

**Context Tagging® Preferences**

Context Tagging preferences are shown below (Window → SlickEdit Preferences → Editing → Context Tagging). These preferences 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.
<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background tagging of open files</strong></td>
<td></td>
</tr>
<tr>
<td>Tag file on save</td>
<td>ON</td>
</tr>
<tr>
<td>Background tagging of open files</td>
<td>ON</td>
</tr>
<tr>
<td>Use background thread when possible</td>
<td>ON</td>
</tr>
<tr>
<td>Start after seconds idle</td>
<td>1</td>
</tr>
<tr>
<td>Max size of files to tag (MB)</td>
<td>256</td>
</tr>
<tr>
<td>Max number of tags per file</td>
<td>122072</td>
</tr>
<tr>
<td><strong>Background updating of tag files</strong></td>
<td></td>
</tr>
<tr>
<td>Update workspace tag file on open</td>
<td>ON</td>
</tr>
<tr>
<td>Update workspace tag file on activate</td>
<td>ON</td>
</tr>
<tr>
<td>Background tagging of other files</td>
<td>OFF</td>
</tr>
<tr>
<td>Workspace tag file only</td>
<td>ON</td>
</tr>
<tr>
<td>Start after minutes idle</td>
<td>0</td>
</tr>
<tr>
<td>Minutes before restarting</td>
<td>10</td>
</tr>
<tr>
<td><strong>Background tagging threads</strong></td>
<td></td>
</tr>
<tr>
<td>Number of tagging threads to start</td>
<td>2</td>
</tr>
<tr>
<td>Create dedicated tagging thread for reading files</td>
<td>OFF</td>
</tr>
<tr>
<td>Create dedicated tagging thread for writing to the</td>
<td>OFF</td>
</tr>
<tr>
<td>tag database</td>
<td>OFF</td>
</tr>
<tr>
<td>Use background thread to build workspace tag file</td>
<td>OFF</td>
</tr>
<tr>
<td>Use background thread to build language support tag</td>
<td>OFF</td>
</tr>
<tr>
<td>file when possible</td>
<td>OFF</td>
</tr>
<tr>
<td>Report background tagging progress on status bar</td>
<td>OFF</td>
</tr>
<tr>
<td>Process background tagging jobs after (ms) idle</td>
<td>590</td>
</tr>
<tr>
<td>Background tagging timeout (ms)</td>
<td>290</td>
</tr>
<tr>
<td>Maximum number of active tagging jobs</td>
<td>580</td>
</tr>
<tr>
<td>Maximum write locking when updating database (uses</td>
<td>OFF</td>
</tr>
<tr>
<td>more memory)</td>
<td></td>
</tr>
<tr>
<td><strong>Tagging Tool Windows</strong></td>
<td></td>
</tr>
<tr>
<td>Update tool windows after (ms)</td>
<td>590</td>
</tr>
<tr>
<td>Additional time to wait before updating tool windows</td>
<td>250</td>
</tr>
<tr>
<td>Preview window timeout (ms)</td>
<td>1800</td>
</tr>
<tr>
<td>Show preview of symbols in tagging tool windows on</td>
<td>OFF</td>
</tr>
<tr>
<td>mouse-over</td>
<td></td>
</tr>
<tr>
<td><strong>References</strong></td>
<td></td>
</tr>
<tr>
<td>Build workspace tag file with references</td>
<td>ON</td>
</tr>
<tr>
<td>Find references incrementally (faster)</td>
<td>OFF</td>
</tr>
<tr>
<td>Update references and call time on single click</td>
<td>OFF</td>
</tr>
<tr>
<td>&quot;Go to reference&quot; only lists references</td>
<td>OFF</td>
</tr>
<tr>
<td>&quot;Search for word matches if symbol is not found&quot;</td>
<td>OFF</td>
</tr>
<tr>
<td>Highlight references in editor</td>
<td>OFF</td>
</tr>
<tr>
<td>Allow mixed language references</td>
<td>OFF</td>
</tr>
<tr>
<td><strong>Maximums (time for performance)</strong></td>
<td></td>
</tr>
<tr>
<td>Maximum functions found by parameterize help</td>
<td>100</td>
</tr>
<tr>
<td>Maximum class/struct members shown</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum response time for list members (ms)</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum candidates for list parameters</td>
<td>200</td>
</tr>
<tr>
<td>Maximum response time for list parameters (ms)</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum tags found in symbol search</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum items found in references search</td>
<td>1024</td>
</tr>
<tr>
<td>Tag file cache size (KB)</td>
<td>65536</td>
</tr>
<tr>
<td>Tag file cache maximum (KB)</td>
<td>524288</td>
</tr>
<tr>
<td>Update after (ms) idle</td>
<td>0</td>
</tr>
<tr>
<td>Maximum response time (ms) for highlighting matching symbols</td>
<td>1800</td>
</tr>
<tr>
<td><strong>Symbol coloring performance</strong></td>
<td></td>
</tr>
<tr>
<td>Update after (ms) idle</td>
<td>590</td>
</tr>
<tr>
<td>Timeout after (ms)</td>
<td>1800</td>
</tr>
<tr>
<td>Number of lines to color above and below the current page</td>
<td>180</td>
</tr>
<tr>
<td>Number of off page lines to color per page (chunk size)</td>
<td>28</td>
</tr>
<tr>
<td><strong>Auto Complete performance tuning</strong></td>
<td></td>
</tr>
<tr>
<td>Maximum symbols</td>
<td>190</td>
</tr>
<tr>
<td>Maximum function prototypes</td>
<td>29</td>
</tr>
<tr>
<td>Maximum word completions</td>
<td>190</td>
</tr>
<tr>
<td>Deadline after (ms) idie</td>
<td>290</td>
</tr>
<tr>
<td>Update after (ms) idie</td>
<td>59</td>
</tr>
<tr>
<td>Timeout after (ms) when automatic</td>
<td>590</td>
</tr>
<tr>
<td>Timeout after (ms) on demand</td>
<td>15000</td>
</tr>
</tbody>
</table>
The following preferences are available:

- **Background tagging of open files:**
  - **Tag file on save** - When set to On, files are retagged when you save a modified file.
  - **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.
  - **Max size of files to tag** - Specifies the maximum size, in kilobytes, a file is allowed to have in order to be tagged.
  - **Max 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 views.

- **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.
  - **Update workspace tag file on activate** - When set to On, background tagging will update the workspace tag file on a thread when SlickEdit® Core loses focus or gains 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® Core 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.
Editing Preferences

- **Create dedicated tagging thread for reading files from disk** - When set to **On**, an additional background tagging thread is created which is dedicated to reading files from disk. This improves performance by reducing disk contention between tagging threads and by pipelining reading and parsing operations. You have to restart the editor for a change to this setting to take effect.

- **Create dedicated tagging thread for writing to the tag database** - When set to **On**, an additional background tagging thread is created which is dedicated to writing tagging information to the tag database. This improves performance by reducing disk contention between tagging threads and by pipelining parsing operations and tag database updates. You have to restart the editor for a change to this setting to take effect.

- **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.

- **Minimize write locking when updating database (uses more memory)** - When set to **On**, specifies that background tagging should minimize the amount of time the database is locked for writing. It does this by caching modified blocks in memory until background tagging is done updating all the records for a file. There is no set limit on how much more memory this option uses, but in general, it will require an additional 10-50M while background tagging is running. You have to restart the editor for a change to this setting to take effect.

- **Tagging Views**:

  - **Update tool windows after (ms) idle** - 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 tagging 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 tagging tool windows after (ms)** - Specifies the amount of time to delay, in milliseconds, when hovering over symbols in tagging-related tool windows or in the file tabs before showing the item under the mouse in the preview window.

• **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 incrementally (faster)** - When set to On, reference queries are faster because analysis stops when a file is found containing a valid reference. However, you may see files which do not have any references to the symbol you are looking for listed in the References view. When set to Off, all files with potential references are searched and analyzed so that the files which do not contain any references are removed.
  - **Update references and call tree on single click** - When set to On, references in the References view are updated when you click on a new symbol in the Classes, Defs, or Symbols view.
  - **"Go to references" only lists references** - When set to On, Go to Reference searches for references but does not jump immediately to the first reference. When set to Off, Go to 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 (Edit → 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.
  - **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.

• **Maximums (tune for performance)** - You can tune Context Tagging performance and accuracy by adjusting these values. Higher values will find more tags but increase search time. Lower values improve performance but may cause tags to be omitted.

• **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 View.

• **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® Core 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 View and select Find Tag). This setting also controls the number of duplicate tags that are tried when SlickEdit® Core 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.

- **Tag file cache size** - Specifies the cache size, in kilobytes, 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 Window → SlickEdit Preferences → Application Options → Virtual Memory.

- **Tag file cache maximum** - Specifies the maximum cache size, in kilobytes, 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 Core is started.

  Note that this is the same as the **Tag file cache maximum** option under Window → SlickEdit Preferences → Application Options → Virtual Memory.

- **Update after (ms) idle (0 implies no delay)** - This option controls the idle time in milliseconds before the List Members feature displays a list.

- **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.

- **Symbol coloring performance** - Use these preferences to fine tune symbol coloring performance.
• **Update 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.

• **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.

  • **Display after (ms) idle** - The number of milliseconds the editor must be idle before auto-completion suggestions will be displayed. This setting affects all extensions.

  • **Update after (ms) idle** - The number of milliseconds the editor must be idle before auto-completion suggestions will be refreshed. This setting affects all extensions.

  • **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.

**Workspace Tagging Excludes**

You can exclude certain paths from your workspace tags by using the Workspace Tagging Excludes preferences. The dialog is shown below and found at **Window → SlickEdit Preferences → Editing → Workspace Tagging Excludes**.

Use the following controls to manage your excluded directories:
• **Add Full Path...** - Browse to a directory on the file system, and add that directory to the list of workspace tagging exclusions.

• **Add Path Component...** - Specify a partial path component to add to the list of workspace tagging exclusions. For example, specifying `build` will result in any source file under a directory named "build" being excluded from workspace tagging.

• **Delete** - Remove an item from the list.

• **Up/Down** - Move items in the list up or down, affecting the order in which exclusions are matched against source files when tagging is performed. Items at the top of the list are matched first.

### Selection Preferences

Selection preferences are shown below (**Window → SlickEdit Preferences → Editing → Selections**). See also [Selections](#) for more information.

The following preferences are available:

• **Styles** - Choose the selection style you wish to use from the following preferences:
  
  • **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** - When this style is selected, selected text is deleted before a paste or character is inserted unless the selection is locked. Pressing the **Backspace** or **Delete** keys deletes the selection unless the selection is locked. Advanced selections (those selections not started with the mouse or **Shift+<arrow keys>**) are extended as the cursor moves. Locking a selection requires one of the emulation commands **select_line**, **select_block**, or **select_char**. To access these commands from **Edit** pull-down menu, select this option in any emulation.
• **SlickEdit default** - When this style is selected, SlickEdit® Core uses the default styles that are enabled when the product is installed.

• **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.

• **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.

• **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 Preferences

Search preferences are shown below ([Window](#) SlickEdit Preferences → Editing → Search). This option screen can also be displayed from the Find and Replace view ([Edit](#) Find/Replace or Ctrl+F), or right-click in the background and select **Configure Preferences**.

These are the default search preferences that control the behavior of Find and Replace operations in the following instances:
• The very first time the Find and Replace view is displayed. After that, dialog history takes over, unless:

  • You set the default search option Initialize with default preferences setting to On, then the preferences are reset to default every time the view is invoked, or

  • You use the right-click context menu in the Find and Replace view to select Use Default Preferences, which resets to the default state.

• The default preferences are always applied when using:

  • Quick Search and Quick Replace.

  • Incremental Search.

  • Command-line searches (find and /) if you don't specify preferences explicitly.

  • Selective Display commands when searching by text.

For more information, see Find and Replace.
The following preferences are available:

- **Default search preferences** - The following default search preferences apply to all command line searches, quick searches and incremental searches, and to the Find and Replace view when the option Initialize with default preferences 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 to complete a search.

- **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 preferences, from the main menu click **Display → Selective Display**. See **Selective Display** for more information.

- **View preferences** - Select from the following:
  - **Close after find/replace** - When set to On, the Find and Replace view is closed after finding text in the buffer.
  - **Initialize with default preferences** - When set to On, preferences on the Find and Replace view are reset to the original default values each time the window is launched.

- **Search string initialization** - The following preferences 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 view when the window is activated. When **History retrieval** is selected, the Find and Replace view 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 view 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 view when the window is activated. If a selection doesn't exist, the value specified by the **Initialize search string** option is used.

- **Additional preferences** - Select from the following:
  - **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 (**Window → SlickEdit Preferences → 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.
Auto-Close

Auto-Close automatically inserts matching closing punctuation when opening punctuation is entered. Auto-Close preferences are shown below (Window → SlickEdit Preferences → Editing → Auto-Close).

The following Auto-Close preferences 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 Preferences 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.

Global Alias Preferences

Global alias preferences are shown below (Window → SlickEdit Preferences → 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 preferences page uses the same form that is used to create and manage Language-Specific Aliases. See Global Aliases for more information about this feature.

Language Preferences

SlickEdit® Core provides many preferences 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® Core 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 Preferences dialog (Window → SlickEdit Preferences →
Languages → [Language Category] → [Language]) to control the behavior of SlickEdit Core 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 preferences for the current buffer is to use the Format →
[Language] Options menu item. This will open the Preferences dialog to the General language-
specific option screen for that language.

The preferences available for each language are further categorized by type. In general, the option
categories are the same for each language (General, Indent, View, etc.), although if a particular language
does not support a particular set of preferences, that preferences category is not included. Conversely,
some languages include additional preferences that are not available in other languages. While most of
the preferences are common among all languages, the default settings for these preferences vary.

Many of the language preferences can be set for all languages as well, using Window → SlickEdit
Preferences → Languages → All Languages. This avoids having to repetitively set preferences for
things like viewing line numbers. The All Languages preferences are arranged in the same hierarchy and
screens used for the individual language preferences. 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 the Extensionless File Manager 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 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 Manager

The Language Manager provides an alphabetical listing of all languages known to SlickEdit Core. 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.
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.

Extensionless File Manager

Some files do not have extensions. The Extensionless File Manager allows you to specify the language for these files. You can specify the language for a specific file or a group of files that match a pattern.
Files

The top part of the **Extensionless File Manager** screen allows you to define the language for specific files. The following buttons are provided to manage the list:

- **Edit** - click this button to edit the language for the specified file.
- **Add** - click this button to add a new file to the list.
- **Delete** - click this button to remove the selected file from the list.

Patterns

The bottom part of the **Extensionless File Manager** screen allows you to define the language for files that match the specified patterns. The following buttons are provided to manage the list:

- **Edit** - click this button to edit the selected pattern.
- **Add filename** - click this button to add a pattern that matches a filename. You will be prompted for the filename and the language it should be associated with.
- **Add path** - click this button to add a pattern that matches a path. You will be prompted for the path and the language it should be associated with.
- **Add pattern** - click this button to add a pattern on your own. You will be prompted for the pattern and the language it should be associated with. You can use this button to add patterns for filenames or paths, but it is easier to enter them using the buttons described, above.
• **Delete** - click this button to remove the selected pattern from the list.

• **Move up** - click this button to move the selected pattern higher in the list.

• **Move down** - click this button to 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 Preferences (Window → SlickEdit Preferences → Languages → All Languages). By expanding this category, you can see the same options nodes found underneath the individual languages.

**Setting All Languages Preferences**

To set an all language preferences, 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 preferences.

**Initial Settings**

When you first view an preferences node for an individual language, the preferences 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.

• **Text boxes** - Text boxes will be left blank to indicate that all languages do not share the same value.
• **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.

![](image)

• **Property Sheet Preferences** - 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*.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Options</td>
<td>Languages Differ</td>
</tr>
</tbody>
</table>

### Language-Specific General Preferences

This option screen shows the mode name and associated file extensions for the selected language, and provides other general preferences. The settings on this page depend on the selected language. As an example, the C/C++ General preferences are shown below (Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → General).
The preferences 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 Format → 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.

- **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.

- **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).

- **Context menus** - These preferences 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.

- **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.

- **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.

- **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 unindents** - 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.

- **Beautify syntax expansions** - 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** - 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 on { } ; etc...** - 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.

- **When tab key reindents the line** - These preferences specify that the **Tab** key be used to beautify or reindent the current line. Select from the following settings:
• **Never** - When this option is selected, pressing **Tab** will never reindent the line. It will indent to the next tab stop.

• **Always** - Pressing the **Tab** key in any column will reindent the current line.

• **In leading blanks** - Pressing the **Tab** key will reindent the line if the cursor is positioned within the leading white space of the line. Otherwise it will indent to the next tab stop. This option is further controlled by the **Strict** check box.

• **Strict** - Strict only applies to the **In leading blanks** option. When this option is selected, it reindents the line only if the cursor position is before the intended indent location; otherwise, it will insert an additional tab stop. When this option is cleared, it reindents the line when the cursor is located on the leading whitespace, regardless of whether the column is before or after the intended indent location.

• **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® Core 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** preferences on the **Tokens tab** of the Color Coding preferences page (*Window* → **SlickEdit Preferences** → **Languages** → [Language Category] → [Language] → **Color Coding**).

• **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.

**Language-Specific View Preferences**

These preferences control the display of special characters, line numbers, and more. C/C++ View preferences are shown below (*Window* → **SlickEdit Preferences** → **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, and Newline characters as well as all of the other special characters listed on the Window → SlickEdit Preferences → Appearance → Special Characters option screen.

Alternately, select the individual preferences 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 Display → 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 Window → SlickEdit Preferences → Appearance → Special Characters node in the Preferences 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 Core 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 Display → Line Numbers (or the view_line_numbers_toggle command). See Viewing Line Numbers for more information.

**Symbol Coloring**

Use this to enable Symbol Coloring for the selected language.

**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.

**Hex View**

This option is used to enable Hex mode editing (choose Hex or Line hex). You can also enable Hex mode on a per-document basis with the menu items Display → Hex and Display → Line Hex (or use the commands hex and linehex). See Hex Mode Editing for more information.

**Language-Specific Formatting Preferences**

These preferences let you configure the way SlickEdit® Core 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 preferences that are available depend on the selected language.

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® Core reformats your code when you select Format → Beautify. The Formatting Beautifier options are shown below. For more information, see Beautifying Code.

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 preferences:
Language Preferences

- **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.

Other languages have more advanced preferences. For more information, see the following section for your language (or the one that most closely relates to your language):

- Java Formatting Options Interface
- XML Formatting Options
- HTML Formatting Options
- XML/HTML Formatting (the feature)
- Ada Formatting Options
- COBOL Formatting Options
- Pascal Formatting Options
- PL/I Formatting Options

**Language-Specific Adaptive Formatting Preferences**

Adaptive Formatting scans a file for the formatting styles in use and automatically matches those settings for the current editing session. The preferences on this screen are used to enable/disable Adaptive Formatting and configure the styles that SlickEdit® Core should recognize for the language. The C/C++ Adaptive Formatting preferences are shown below (Window → SlickEdit Preferences → Languages →

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 Core should scan. The individual style settings that appear on the Preferences screen will vary depending on the language. See Adaptive Formatting for more information.

**Language-Specific Comment Preferences**

Comment preferences let you control how block and line comments are created.

To comment out selected lines, select text in the editor and then click **Format → Comment Block** or **Format → 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 **Window → SlickEdit Preferences → 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++ preferences for comments are shown below (Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → Comments).
Comment block

These settings are used when you comment out a selected block of text (Format → Comment Block or box command). SlickEdit® Core 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 Core 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:

```c
if (!enabled) {
    tabState = TIS_DISABLED;
}
```

From the main menu, click Format → Comment Block, and the selection is commented out as follows:

```c
/*
    if (!enabled) {
        tabState = TIS_DISABLED;
    }
*/
```
Select from the following comment block preferences:

- **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:
  ```c
  /*
   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:
  ```c
  /*
   if (!enabled) {
     tabState = TIS_DISABLED;
   }*/
  ```

**Comment line**

These settings are used when you comment out selected lines ([Format → 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® Core to create a line comment. The placement of the Left characters can be controlled through the Location preferences 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 [Format → Comment Line](#) comments out the example code as follows:

  ```c
  // if (!enabled) { //
  //     tabState = TIS_DISABLED; //
  // } //
  ```

- **Location** - Mutually exclusive location preferences 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:

```c
//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.

### Block Comments

**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.

### Doc comments

Select from the following preferences:

• **Automatically expand doc comments** - When this option is selected, SlickEdit® Core 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.

• **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 editing

The following preferences control comment editing behaviors. These preferences 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:

```c
// The quick brown fox [CURSOR_HERE]jumped over the lazy dog.
```
Pressing **Enter** will result in:

```
// The quick brown fox
// [CURSOR_HERE]jumped over the lazy dog.
```

- **Extend line comments** - If selected, when you press **Enter** at the end of a line containing a line comment, and there is also an aligned line comment on the line before or after the current line, a new line comment will automatically be started on the new line. For example:

```
// The quick brown fox
// jumped over the lazy dog.[CURSOR_HERE]
```

Pressing **Enter** will result in:

```
// The quick brown fox
// jumped over the lazy dog.
// [CURSOR_HERE]
```

- **Join comments when joining lines** - If selected, when you press **Delete** at the end of a line containing a line comment to join the current line with the next line, and the next line is also a line comment, the line comment characters will automatically be deleted. For example:

```
// The quick brown fox [CURSOR_HERE]
// jumped over the lazy dog.
```

Pressing **Delete** will result in:

```
// The quick brown fox[CURSOR_HERE] jumped over the lazy dog.
```

**String editing**

If **Split strings on Enter** is selected, when you press **Enter** to split a line when the cursor is inside of a string, the closing and opening quotes and, if necessary, operators, will automatically be inserted, and the string will be aligned with the original string. For example:

```java
String x = "The quick brown fox [CURSOR_HERE]jumped over the lazy dog.";
```

Pressing **Enter** will result in:

```java
String x = "The quick brown fox " +
        "[CURSOR_HERE]jumped over the lazy dog.";
```

**Language-Specific Comment Wrap Preferences**

Comment Wrap preferences 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 preferences are shown below (Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → Comment Wrap).

The preferences 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 preferences 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® Core 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 column**.

### Language-Specific Word Wrap Preferences

These language-specific preferences 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 preferences are shown below (Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → Word Wrap).
The preferences are described as follows:

- **Margins** - Sets the left, right, and new paragraph margins. Specify the column number at which each margin should begin. The margin indicator will only appear if the **Word wrap** option is selected, which activates word wrapping.

- **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 preferences 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.
  - **Word wrap** - This option activates/deactivates Word Wrap. When selected, the editor keeps the cursor within the margins when entering text, moving the cursor, and deleting characters. 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 preferences 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 Preferences

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 preferences 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 preferences 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 **Surround With** check box is used to specify that you are working with aliases for Surround With. Surround With is a feature that lets you surround existing code with text or predefined structures. When this option is selected, the alias list changes to show only Surround With aliases. When this option is not selected, the alias list shows only normal aliases. 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 bottom of the Aliases preferences 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 preferences 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 Preferences**

These preferences 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 preferences are shown below (Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → Auto-Complete).
The preferences are described as follows:

- **Enable auto-completion** - If selected, activates the Auto-Complete feature. See [Auto-Complete](#) for more information.

- **Symbols** - If selected, symbols will be displayed as completion preferences if the word prefix at the cursor matches one or more symbols using a strict, context-sensitive and language-specific tag search.

- **Locals** - If selected, local variables and parameters will be displayed as completion choices. This functions identically to the **Symbols** setting, except that the results are limited strictly to locals. **Locals** can be enabled even if **Symbols** is disabled for performance.

- **Current class** - 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 the current class. **Current class** can be enabled even if **Symbols** is disabled.

- **Current file** - - 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. **Current file** can be enabled even if **Symbols** is disabled for performance.

- **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.

- **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® Core. 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** - If selected, turns Auto-Complete on in the Build view 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 completions.

- **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.

- **Show parameters** - 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** - If selected, for symbol completions, this will show the symbol declaration as a comment to the right of the symbol completion.

- **Show comments** - If selected, for symbol completions, 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.

- **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.

- **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 on this tab. In order for
Dynamic Surround to work, the option Syntax Expansion must also be selected (see below). See
Dynamic Surround for more information on how to use this feature.

- **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.

- **List symbols** - The following preferences 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® Core to display a list of the members for the corresponding type. To access this
feature on demand, press Alt+Dot. If you use this feature on demand, and you are not in a member
expression, this feature will display a list of 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.

  - **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.

  - **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.

  - **List include files after typing #include** - 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.

**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.

- **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 to perform symbol completion.

- **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.

- **Use strict case-sensitivity rules** - If checked, matches symbols with the same case as what was entered for case-sensitive languages. Otherwise, it will first search for exact-case matches, then case-insensitive symbol matches.

- **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.

For example, if List Members is active and the current line is as follows:
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>
```

**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 preferences are shown below (Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → Auto-Close).
The following preferences 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.
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.

- **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 Context Tagging® Preferences

These preferences let you configure language-specific settings for Context Tagging (see [Context Tagging Features](#)). Note that global Context Tagging preferences are located at **Window → SlickEdit Preferences → 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 preferences are shown below (**Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → Context Tagging**).
Parameter Information

The following preferences 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).

**Go to Definition**

These preferences control the behavior when you navigate from a symbol to its definition. You can do this by selecting **Navigate → Go to Definition**, selecting **Go to Definition** from the context menu, pressing `Ctrl+Dot` in CUA emulation, or by executing the `push_tag` command from the SlickEdit Core 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. In any case, if you use `Ctrl+Dot` to jump to a symbol, you can cycle through the alternate symbols by pressing `Ctrl+Dot` repeatedly. You can step backwards through the list of matches by pressing `Ctrl+Comma`. However, once you reach the first match, `Ctrl+Comma` will then pop you back to your original location, where you were before you pressed `Ctrl+Dot`.

Independent of the settings for these preferences, in the following circumstances, SlickEdit® Core 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.

• **Ignore forward class declarations** - When this option is enabled, Go to Definition filters out forward class declarations, and only shows the actual class definitions and 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 Core attempts to filter out functions with the same name but with different function signatures. As stated, this can slow down the matching.

**Highlighting and Completion**
• **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.

• **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. Advanced configuration preferences are available. See [Cursor on Symbol Shows All Uses in File](#) for more information.

• **Use strict case sensitivity when completing identifiers** - When selected, factors upper/lowercase letters into the matching criteria.

### Language-Specific Color Coding Preferences

Color Coding is a feature that displays various portions of code in color for identification purposes. The syntactic elements colored by Color Coding are determined by the settings on this preferences screen. The colors used for Color Coding are set through . See also [Colors, Color Coding, and Symbol Colors](#) for more information.

The settings on the Color Coding preferences page depend on the selected language. The C/C++ Color Coding preferences are used in the screen shots for this section.

The Color Coding preferences page contains the following general preferences:

• **Lexer name** - Select the language lexer to use from the **Lexer name** drop-down list. This sets the active lexer for that language.

• **New** - Click this button, located next to **Lexer name**, to prompt for a lexer name to start a new language-specific color coding definition (see [Creating Color Coding for a New Language](#)).

• **Delete** - Click this button, located next to **Lexer name**, to remove a lexer name from the list. You can only delete user-created lexers.

• **Import** - Click this button to import lexer definitions from a VLX file.

• **Colors** - Click this button to jump to the **Window → SlickEdit Preferences → Appearance → Colors** option screen where you can specify the colors used. Click the **Back** button on the Preferences dialog to return to the Color Coding Language tab. See [Setting Colors for Screen Elements](#) for more information.

Other preferences are categorized into the following tabs:

• **Color Coding Tokens Tab**

• **Color Coding Numbers Tab**

• **Color Coding Strings Tab**

• **Color Coding Language Tab**
The Tokens tab provides the capability to specify unique tokens to help you when working with your code.

The following preferences are available:

- **Token type** - Select from the following token types:
  - **Keywords** - When this option is selected, the list box displays the words that have keyword color.
• **CS keywords** - When this option is selected, the list box to the right displays case-sensitive words that have keyword color. These words are always case-sensitive even if the **Case Sensitive** check box is not selected.

• **Preprocessor** - When this option is selected, the list box to the right displays preprocessor keywords in preprocessor color. All preprocessor keywords must start with the same character.

• **Punctuation, Lib Symbols, Operators, User Defined** - When one of these preferences is selected, the list boxes to the right display the words associated with each.

• **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.

• **New** - Click this button on the **Tokens** tab to add one or more words. Separate each word with a space.

• **Delete** - Deletes selected items in a list box.

• **Get** - Click this button to add words by selecting the file that contains the keywords that you want to add.

**Color Coding Numbers Tab**

The Numbers tab provides preferences for color coding numerical values when working with SlickEdit® Core.
This tab contains the following:

- **Hex numbers**
  - 0x#### (C-Style) - When this option is selected, text such as 0x123ABC is color coded in number color.
  - ####h (Intel assembler) - When this option is selected, text such as 123ABCh is color coded in number color.
  - $#### (Motorola) - When this option is selected, text such as $123ABC is color coded in number color.
• **&H#### (Basic)** - When this option is selected, text such as `&H123ABC` is color coded in number color.

• **"####"X (Rexx)** - When this option is selected, strings such as `"123ABC"X` are color coded in number color.

• **Z"####" (Fortran)** - When this option is selected, strings such as `Z"123ABC"` are color coded in number color.

• **No Hex (COBOL)** - When this option is selected, text such as `123ABC` is not color coded in number color. By default (for most languages set in **Language** tab) `123ABC` is color coded in number color.

• **Octal and binary numbers**

  • **####O (Intel assembler)** - When this option is selected, octal numbers such as `17723O` are color coded in number color.

  • **####Q (Intel assembler)** - When this option is selected, octal numbers such as `17723Q` are color coded in number color.

  • **####B (Intel assembler)** - When this option is selected, octal numbers such as `17723B` are color coded in number color.

  • **&O#### (Basic)** - When this option is selected, octal numbers such as `&017723` are color coded in number color.

  • **O#### (C)** - When this option is selected, octal numbers with a leading zero, such as `017723` are color coded in number color.

  • **####B (Binary numbers)** - When this option is selected, binary numbers such as `110100B` are color coded in number color.

• **Floating point numbers**

  • **#base#number#exponent float (Ada)** - When this option is selected, text such as `#23#56#67` is color coded in number color.

  • **Floating point with "E" exponent** - When this option is selected, text such as `123.4E24` is color coded in number color.

  • **Floating point with "D" exponent** - When this option is selected, text such as `123.4D24` is color coded in number color.

  • **No exponents (COBOL)** - When this option is selected, it specifies that the language does not allow floating point exponents of any kind.

• **Do not color code numbers (HTML)** - When this option is selected, text such as `123.4E24` and `123ABC` is not color coded in number color. By default (for most languages set in the **Language** tab), `123.4E24` and `123ABC` is color coded in number color.

• **Allow underscores in integers (Ada)** - When this option is selected, text such as `12_34` is color coded in number color.
**Color Coding Strings Tab**

The Strings tab contains preferences for color coding strings.

This tab contains the following:

- **Double quoted strings**
  - **Two consecutive quotes represent one (REXX)** - """" for REXX represents a string of length one which is a double quote character.
• \" represents a double quote (C) - "\"" for C represents a string of length one which is a double quote character.

• **Double quoted strings are always 1 char long** - When this option is selected, this means that a double quote character is followed by an additional character and then the terminating double quote character. There is never more than one character between the start and end double quote.

• **Trailing backslash continues string across lines (C)** - When this option is selected, it indicates that searching for the terminating quote continues to the next line if the lines end with a backslash character.

• **Search for end quote across multiple lines** - When this option is selected, it indicates that the string does not have to be terminated on the same line as the start quote character.

• **Delay color coding until end quote** - When this option is selected, a string is not color coded unless an end quote is seen on the same line. This does not support multi-line strings.

• **Single quotes**

  • **Two consecutive quotes represent one (Pascal)** - "" (four consecutive single quote characters) for Pascal represents a string of length one which is a single quote character.

  • \' represents a single quote (C) - "" represents a string of length one which is a single quote character.

  • **Single quoted strings are always 1 char long** - When this option is selected, a single quote character is followed by an additional character and then the terminating single quote character. There is never more than one character between the start and end single quote.

  • **Trailing backslashes continues string across lines (C)** - When this option is selected, it indicates that searching for the terminating quote continues to the next line if the lines end with a backslash character.

  • **Search for end quote across multiple lines** - When this option is selected, it indicates that the string does not have to be terminated on the same line as the start quote character.

  • **Delay color coding until end quote** - When this option is selected, a string is not color coded unless an end quote is seen on the same line. This does not support multi-line strings.

• **Backquoted strings (Perl, Bourne shell, C shell)** - When this option is selected, backquoted shell expansion strings such as ~ls -d /home/~ are color coded in string color.

**Color Coding Language Tab**

The Language tab is used to set more language-specific color coding preferences.
This tab contains the following:

- **Language specific** - To avoid requiring complicated BNF for defining color-coding, some hardware language-specific adjustments have been added. You may be able to use one of these language-specific settings for another language, but there's no guarantee it will work.

- **Color Code Line Numbers (Basic/COBOL)** - When this option is selected, indicates that leading line

   □ Color code line numbers (Basic/COBOL)
   □ Backslash escapes next character (Bourne Shell)
   □ Here document (UNIX Shells/Perl)
   □ Color identifiers followed by '(' as a function
   □ Special coloring for 'package' and 'import' statements (Java)
   □ Preprocessing keywords can appear anywhere
   □ Identifiers may start with a number (COBOL)
   □ @"###" Unicode strings (C#)
   □ Treat everything after 0x1a as comments (end of file)
   □ Color inactive code regions (C/C++ Preprocessing, eg. #if 0)
numbers should be color-coded in line number color.

- **Backslash escapes next character (Bourne Shell)** - Backslash escapes the character that follows. This is useful for UNIX shell scripts which use " to indicate that the double quote is not the start of a string.

- **Here Document (UNIX Shells/Perl)** - Activates support for HERE documents. Note that if you prefix your terminator with one of our lexer names, you will get embedded language color-coding. Example of a HERE document in Perl, where `HTMLEOF` is used as the terminator to get HTML embedded language color-coding:

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

Unknown languages are color-coded in string color. Embedded language colors are user-definable.

- **Color identifiers followed by '(' as a function** - For language such as C++, Java, and Slick-C®, an identifier followed by a parenthesis always indicates a function.

- **Special coloring for 'package' and 'import' statements (Java)** - When this option is selected, the Java syntax package and import statements are supported. This option is forced on for the lexer name Java. You must add the `package` and/or `import` keywords to your keyword list in order for this option to have any effect.

- **Preprocessing keywords can appear anywhere** - When this option is selected, preprocessing keywords are color-coded even if they are not only preceded by white space.

- **Identifiers may start with a number (COBOL)** - When this option is selected, identifiers may start with one or more decimal digits. By default, leading decimal digits indicate a number.

- **@"###" Unicode strings (C#)** - When this option is selected, text in the form of `@"any text"` is coded as a string.

- **Treat everything after 0x1a as comments (end of file)** - Historically, DOS used 0x1a to mark the end of the file. When checked, SlickEdit Core 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.

**Color Coding Comments Tab**

The Comments tab is used to set comment preferences for Color Coding.
This tab contains the following:

- **New Line comment** - Click this button to define new single-line comments.
- **New Multi-line comment** - Click this button to define new multi-line comments.
- **Delete** - Click this button to remove the selected comment definition from the list.
• **MultiLine comment** - The following preferences apply to multi-line comments:
  
  • **Start delimiter** - Delimiter which starts the multi-line comment. Currently, the first character of this string cannot be a valid identifier character.
  
  • **End delimiter** - Delimiter which ends the multi-line comment. Currently, the first character of this string cannot be a valid identifier character.
  
  • **Nesting allowed** - When this option is selected, this multi-line comment may have this multi-line comment inside it.
  
  • **Only if first non-blank character in line** - Indicates the start delimiter must be the first non-blank character in the line in order to be considered the start of a comment. This check box is available only when the **Only when start delimiter is in column** text box is completed.
  
  • **Check for start delimiter first** - When this option is selected, the lexer checks for the start delimiter before looking for other items. When this option is specified, the start delimiter is limited to one character in length.
  
  • **End delimiter must be the last character on the line** - When this option is selected, the end delimiter text must occur at the end of a line to terminate the comment.
  
  • **Only when start delimiter is in column** - Indicates that the start delimiter text starts a comment only when found in the column specified.
  
  • **Color as** - Specifies color used for this comment. This color is not used when the start delimiter is immediately followed by one of the Comment Keywords. When the start delimiter is immediately followed by one of the Comment Keywords, keyword color is used.

• **Line Comment** - The following preferences apply to single-line comments:
  
  • **After column** - Colors all characters after the specified column as a line comment.
  
  • **Use delimiter** - Uses the specified delimiter to recognize the following characters as a line comment.
  
  • **Only if first non-blank character in line** - If checked, a line comment cannot be preceded by any non-blank characters.
  
  • **Check columns first** - When checked, the lexer will look for the delimiter in the specified columns before doing other checks. This option clears any check marks in **Only if first non-blank character in line** and **Only if preceded by blank**.
  
  • **End column is end of line** - Put a check in this box if you are using **Delimiter column** to specify a column after which the delimiter is recognized as beginning a line comment. Then you only need to specify a start column using the fields for **Delimiter column**.
  
  • **Only if preceded by blank** - If checked, the delimiter must be preceded by a blank (space) to be recognized as starting a line comment.
  
  • **Used for API documentation** - If checked, the comment is colored using the **Documentation Comment** color rather than the **Line Comment** color.
• **Delimiter column** - Specifies the columns in which the selected delimiter is recognized as beginning a line comment. If the delimiter begins in any other column, the text is colored as regular text. Specify a begin and end column to set a range of columns. Specify only a start column if line comments must begin in a specific column. If **End column is end of line** is checked, you need not specify an end column and the delimiter will be recognized in the start column and any column thereafter.

### 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 preferences:

• **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 Preferences

These preferences are used to specify load and save preferences for files on a language-specific basis. As an example, the C/C++ file preferences are shown below (Window → SlickEdit Preferences → Languages → Application Languages → C/C++ → File Options).

The preferences 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 preferences, as well as all language-specific preferences.

• **Expand tabs to spaces** - When set to Default, SlickEdit® Core uses the setting for the global file load option, Expand tabs to spaces (Window → SlickEdit Preferences → File Options → Load). When set to On, SlickEdit Core always loads files with tabs expanded to spaces. When set to Off, tabs are always left unexpanded.

### Save Options:

• **Strip trailing spaces** - Specifies if and when to remove trailing spaces from the ends of lines. When set to Default, SlickEdit Core uses the setting for the global file save option, Strip trailing spaces (Window → SlickEdit Preferences → 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 designate a file type for the line breaks. For example, if you are working in Windows and using CVS, using UNIX line breaks will make using CVS easier. Therefore, set the file format to **UNIX**.

**Language-Specific Compiler Properties**

These settings are used to configure your compiler so that SlickEdit® Core can correctly perform full preprocessing, parsing, symbol analysis, and cross-referencing. The fields and preferences on this page depend on the selected language.

The following fields and preferences 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.

**File Preferences**

These global file preferences (**Window** → **SlickEdit Preferences** → **File Options**) are used to make settings regarding file operations such as loading and saving.

File option categories are:

• **Load File Options**

• **Save File Options**

• **AutoSave File Options**
• **Files of Type Filter Options**

**Load File Preferences**

Load file preferences are shown below (*Window → SlickEdit Preferences → File Options → Load*).
The preferences are described as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast line count on partial load</td>
<td>OFF</td>
</tr>
<tr>
<td>Show EOF character</td>
<td>ON</td>
</tr>
<tr>
<td>Expand tabs to spaces</td>
<td>OFF</td>
</tr>
<tr>
<td>File locking</td>
<td>OFF</td>
</tr>
<tr>
<td>Reinsert after current</td>
<td>ON</td>
</tr>
<tr>
<td>Wrap line length</td>
<td>4000</td>
</tr>
<tr>
<td>Use undo</td>
<td>ON</td>
</tr>
<tr>
<td>Max undo steps</td>
<td>32000</td>
</tr>
<tr>
<td>Save/restore file position</td>
<td>ON</td>
</tr>
<tr>
<td>Max files</td>
<td>1000</td>
</tr>
<tr>
<td>Encoding</td>
<td>Auto Unicode</td>
</tr>
</tbody>
</table>

**Load entire file**

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load entire file</td>
<td>ON</td>
</tr>
<tr>
<td>Load partially for large files</td>
<td>ON</td>
</tr>
<tr>
<td>Load partially when files are larger than (KB)</td>
<td>8000</td>
</tr>
<tr>
<td>Count number of lines</td>
<td>OFF</td>
</tr>
<tr>
<td>Truncate file at EOF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

**Auto reload**

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto reload</td>
<td>ON</td>
</tr>
<tr>
<td>Suppress prompt unless modified</td>
<td>OFF</td>
</tr>
<tr>
<td>Compare file contents before auto reload</td>
<td>ON</td>
</tr>
<tr>
<td>Size limit for comparing contents (KB)</td>
<td>2000</td>
</tr>
<tr>
<td>Auto read only</td>
<td>ON</td>
</tr>
<tr>
<td>Fast auto read only</td>
<td>ON</td>
</tr>
<tr>
<td>Reload on switch buffer</td>
<td>ON</td>
</tr>
<tr>
<td>Auto reload current file only</td>
<td>OFF</td>
</tr>
<tr>
<td>Auto reload all files if current file changed</td>
<td>OFF</td>
</tr>
<tr>
<td>Auto reload timeout (ms)</td>
<td>5000</td>
</tr>
<tr>
<td>Show auto reload timeout notifications</td>
<td>ON</td>
</tr>
</tbody>
</table>
• **Fast line count on partial load** - When set to On, SlickEdit® Core 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 **Window → SlickEdit Preferences → 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 Core 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.

• **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.

• **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 Core 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 preferences 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 preferences are available:
  - **Auto reload** - When set to On, SlickEdit Core 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 Core.
  - **Compare file contents before auto reload** - When set to On, if auto reload detects that the current file has been changed on disk, SlickEdit Core 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 Core 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 Core 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 Core 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 Core.

- **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 Core 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 Preferences

Save file preferences are shown below (Window → SlickEdit Preferences → File Options → Save).

The preferences are described as follows:
• **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](#)).

• **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](#).

## AutoSave File Preferences

AutoSave file preferences are shown below (Window → SlickEdit Preferences → File Options → AutoSave).

![AutoSave Preferences](image.png)

The preferences 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 when AutoSave runs subsequently. AutoSave files are deleted when you manually save the file, close the file, or exit SlickEdit Core normally.

  AutoSave does not save buffers that are not named. In addition, AutoRestore does not restore files that do not exist on the disk drive of your system. Save your file at least one time to ensure that the file has a file name and exists on the disk drive.

  **Save after period of inactivity** - 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 Core on AutoSave** - When set to **On**, the SlickEdit® Core 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 preferences for AutoSave temporary files. Select from:
  
  • **Save to different directory** - This option places AutoSave files in the directory specified by the **AutoSave directory** field. Use this option to clean up or find all of the AutoSave files if an abnormal editor exit occurs. Note that if you use this option, when editing two files with the same name but in different directories, one AutoSave temporary file is overwritten by the other.

  • **Same name, different extension** - This option places the AutoSave file 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.prg~**. If you are editing two files in the same directory which differ only by the third character, one AutoSave temporary file will be overwritten by the other.

  • **Same name** - This option automatically saves the modified files and no AutoSave temporary files are created.

• **AutoSave directory** - Specifies a different directory to use for AutoSave temporary files when **Save to different directory** is selected in the **Save to** preferences. If this field is blank, `<configuration_directory>\autosave` is used. To find the location of your configuration directory, see **Help → About SlickEdit Core**. Press **Delete** to clear this field, specifying the default.

• **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 Preferences

The Files of Type Filter preferences are shown below (**Window → SlickEdit Preferences → File Options → Files of Type Filters**). They are used to specify the list file filters for the Open 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 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 preferences 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 (*.*)

Application Preferences

Application preferences (Window → SlickEdit Preferences → Application Options) pertain to the
SlickEdit® Core application. You can specify what parts of SlickEdit Core should be restored on startup, the amount of virtual memory to use, and more.

Application option categories are:

- **Virtual Memory Options**

**Virtual Memory Preferences**

Virtual memory preferences are shown below (Window → SlickEdit Preferences → Application Options → Virtual Memory).

**Note**

You must restart SlickEdit® Core for these settings to take effect.

The preferences 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 (KB)** - Specifies the maximum amount of memory, in kilobytes, used to store text buffer data. A value that is less than zero specifies all available memory.

**Caution**

If the operating system starts the swapping process before the cache is full, performance might be degraded. The cache size must be smaller than the amount of actual memory available.

- **Tag file cache size (KB)** - Specifies the cache size, in kilobytes, 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 Window → SlickEdit Preferences → Editing → Context Tagging. For more information about tagging, see Building and Managing Tag Files.

- Tag file cache maximum - Specifies the maximum cache size, in kilobytes, 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 Core is started.

  Note that this is the same as the Tag file cache maximum option under Window → SlickEdit Preferences → Editing → Context Tagging. For more information about tagging, see Building and Managing Tag Files.

<table>
<thead>
<tr>
<th>Tagging Performance</th>
<th>Recommended Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>8 MB</td>
</tr>
<tr>
<td>Default</td>
<td>64 MB</td>
</tr>
<tr>
<td>Ideal</td>
<td>Sum of tag file sizes</td>
</tr>
<tr>
<td>Maximum</td>
<td>25% of physical system memory</td>
</tr>
</tbody>
</table>

**Notification Options**

The Notifications system informs you when SlickEdit Core performs automatic actions. These actions are divided into two groups: background processes and feature notifications. 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.
The Notifications preferences 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 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 feature 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 preferences screen contains the following settings:

- Turn all notifications to - choose a notification level for all features.
Set notification level by feature - allows you to set the notification level differently for each feature. 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.

- Feature - select the feature 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 Core 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 preferences screen for the selected feature or view help about this feature.

Product Improvement Program Options

You can opt in or out of the SlickEdit Core Product Improvement Program using the preferences found at (Window → SlickEdit Preferences → Application Options → Product Improvement Program). These preferences are shown below.
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 Preferences**

Network and Internet preferences (Window → SlickEdit Preferences → Network & Internet Options) are used to configure the IP setting, FTP connection profiles and preferences, 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 (Window → SlickEdit Preferences → Network & Internet Options → Network Settings). These preferences are used to set the Internet Protocol (IP) version.
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 and SFTP. The preferences 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 **IPv6 only** to force IPv6 address connections.

### FTP Default Preferences

Default FTP preferences are shown below (Window → SlickEdit Preferences → Network & Internet Options → FTP Default Options). This Preferences screen is also displayed when you click File → FTP → Default Options or when you click the Default Preferences button on the FTP Profile Manager. Preferences 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 preferences 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**
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 Preferences 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 Preferences 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® Core.

### FTP Default Preferences SSH/SFTP Tab

The SSH/SFTP tab contains the following preferences:

- **SSH executable** - The location of the SSH client program that is used to establish the secure connection with the SSH server.

  SFTP support requires the OpenSSH client program to operate. Windows users can obtain the SSH client by downloading and installing the Cygwin package ([www.cygwin.com](http://www.cygwin.com)) and making sure to choose the `openssh` package during install.

- **Subsystem/Service name** - The name of the SFTP service being run by the SSH server. Defaults to `sftp`.

### FTP Default Preferences Debug Tab

This tab is used by SlickEdit® Core Product Support to debug customer FTP/SFTP issues.

### URL Mapping Preferences

URL Mapping preferences are shown below (Window → SlickEdit Preferences → Network & Internet)
Options → URL Mappings). These preferences 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 Preferences

URI Scheme preferences are shown below (Window → SlickEdit Preferences → Network & Internet Options → URI Schemes). These preferences let you specify URL types that SlickEdit Core 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 Preferences**

Proxy preferences are shown below (`Window → SlickEdit Preferences → Network & Internet Options → Proxy Settings`). These preferences allow you to configure a proxy server to use when SlickEdit® Core needs to use an Internet connection. Internet Explorer settings are used by default.
The following preferences 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 preferences and fields on the page are inapplicable and therefore dimmed.

- **Use proxy server** - If selected, the remaining preferences 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 Preferences**
Web browser setup preferences are shown below (Window → SlickEdit Preferences → Network & Internet Options → Web Browser Setup). Use these preferences to specify the browser to use when SlickEdit® Core needs to launch one. Selecting a preferred browser automatically sets the defaults for the other preferences on this form.

The following preferences are available:

- **Browsers** - 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 Mac OS X platforms - You need to specify which Web browser you are using. In addition, you need to give the full path to the program executable.

- **Program** - Indicates the program to run. You may specify a %F in this text box or any of the other text boxes on this dialog box to have the HTML file name inserted into the command that is executed.

- **DDE** - The Application, Topic, and Item text boxes specify DDE XTYP_REQUEST parameters and are used only if the Use DDE option is selected.
Tool Preferences

Preferences for tools (Window → SlickEdit Preferences → Tools) pertain to tools such as Spell Check, and utilities supported by SlickEdit® Core.

Tools option categories are:

- Spell Check Options

Spell Check Preferences

Spell Check preferences are shown below (Window → SlickEdit Preferences → Tools → Spell Check). These settings control the behavior of Spell Check in the editor (Format → Spell Check). You can also access these preferences from the main menu item Format → Spell Check → Spell Options, or by using the spell_preferences command.

The Spell Check Preferences screen contains the following:

- **Common list** - Specifies the default dictionary list. Dictionary lists are text files that have the extension .lst. Words that are frequently misspelled are spelled correctly in these files for matching during spell checking. Each line of a dictionary list can contain only one word. The default dictionary list is named scommon.lst and is located in the SlickEdit® Core installation directory.

- **User lists** - You can have up to two custom dictionaries. When a word is not found during a spell check and you add the word to **User list 1 or 2**, the word is added to the associated file specified in these fields. By default, the first time words are added, SlickEdit Core will create files named userdct1.lst and userdct2.lst in your configuration directory, so these file names are filled in for you. If you choose to create your own files, be sure to place them in the default configuration directory and add the file names here.

- **Ignore all UPPERCASE words** - When set to On, all words in uppercase are ignored during a spell check operation. 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. This applies to all spell check operations.
Preferences History

The Preferences History node in the Preferences dialog (Window → SlickEdit Preferences → Preferences History) is used to see changed preferences. From the drop-down, select Anytime to see all preferences that have been changed from the default values since the editor was installed, or, choose to see only those preferences that were changed today, yesterday, or within the last week or month. Only the most recent date is shown for preferences that have been changed more than once.

Note

Preferences History only shows changes that were made through the Preferences dialog. Changes made by setting configuration variables, in macro code, or in other dialogs are not viewable in the Preferences History.

The results are displayed in the Preferences dialog in the results window, as shown in the following screen.

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 preferences changed on forms embedded in the Preferences dialog, the
results show only the name of the form. For example, if you made a change to a color under Window →
SlickEdit Preferences → Appearance → Colors on the Preferences dialog, the Preferences History
results show “Colors” as the name of the option and “Appearance” as the path.

Export/Import Preferences

To export preferences, select Window → SlickEdit Preferences → Export/Import Preferences. You
can export all your preferences at once or you can export a designated group of them. To export all
preferences click the Export All Preferences button. To export a particular set of preferences click the
Setup Export Groups button. To import already exported preferences, click Import Preferences. See
below for details of each operation.

Note

Moving preferences to a machine with a different operating system is allowed, but not supported.
The same applies to exporting preferences from a one version of SlickEdit Core and then
imported into another. While these operations may work for some preferences, 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.

Import Options

You can import the settings of other users using the Import Options feature. You
can also restore settings 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.
Export All Preferences

To Export all of your preferences, click the Export All Preferences 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.

Setup Export Groups

Export Groups allow you to export a specific set of preferences. 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.
Add or remove preferences 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 preferences 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 Preferences**

You can import preferences by clicking the Import Preferences button on Window → SlickEdit Preferences → Export/Import Preferences. Then navigate to the location of the export package file.
Once the export package is read, a tree of the preferences within the package is displayed. You can choose which preferences to import. Once you have made your selections, click **Import**. The preferences 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.

### Additional Preferences

Some settings are not configurable through the preferences dialog but are still available for export/import. These preferences can be found under the **Additional Preferences** node in the Setup Export Groups dialog. They are also included automatically if you export all preferences.

The options available for export/import under **Additional Preferences** are:

- **Menu Customizations** - Exports any changes made to menus by adding, removing, or modifying individual menu items.

- **Toolbar and View 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](#).

• **XML/HTML Formatting** - Exports the schemes used to automatically format content in XML and HTML files. To learn more about this feature, see [XML/HTML Formatting](#).

• **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.

### Configuration Backup

Whenever you import any preferences, certain configuration files are backed up so that you can restore your application should the imported preferences cause any problems. To restore your configuration after an preferences 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, vusrdefs.e.bak, and vusrobjs.e.bak (UNIX: vslick.stu.bak, vunxdefs.e.bak, vunxBobj.e.bak).

3. Remove the existing configuration files, named vslick.sta, vusrdefs.e, and vusrobjs.e (UNIX: vslick.stu, vunxdefs.e, vunxBobj.e). Also remove the loaded macro files vusrdefs.ex and vusrobjs.ex (UNIX: vunxdefs.ex, vsrdobjsex).

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 preferences import.
Window

SlickEdit Core adds a single entry to the standard Eclipse Windows menu, **SlickEdit Preferences**, which is used to access preferences that apply to the features contributed by SlickEdit Core. For more information on setting preferences, see *Chapter 4, User Preferences*. 
Help

This section describes items on the Help menu and associated dialogs and views. For information about how to obtain product support, see Product Support.

Help Menu

SlickEdit Core adds the following items to the standard Eclipse Help menu:

<table>
<thead>
<tr>
<th>Help Menu Item</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SlickEdit Cool Features</td>
<td>Displays the Cool Features dialog, which shows SlickEdit® Core feature tips. See Help Dialogs and Views.</td>
<td>cool_features</td>
</tr>
<tr>
<td>What Is Key</td>
<td>Used to discover the 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.</td>
<td>what_is</td>
</tr>
<tr>
<td>Where Is Command</td>
<td>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.</td>
<td>where_is</td>
</tr>
<tr>
<td>SlickEdit License Manager</td>
<td>Displays the SlickEdit License Manager for managing licenses. See Licensing.</td>
<td>lmw 1</td>
</tr>
<tr>
<td>About SlickEdit Core</td>
<td>Displays a property sheet containing information about your product, such as serial and version numbers, as well as release notes and the license agreement.</td>
<td>version</td>
</tr>
<tr>
<td>SlickEdit Product Support</td>
<td>Used to invoke a Web browser which opens to a form on the SlickEdit Web site that you can use to contact Product Support.</td>
<td>do_webmail_support</td>
</tr>
</tbody>
</table>
### Help Dialogs and Views

This section describes the dialogs and views 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 → SlickEdit Cool Features, or use the `cool_features` command on the SlickEdit® Core command line.

![Cool Features Dialog](image)

**Help Menu Item** | **Description** | **Command**
--- | --- | ---
See [Product Support](#) |
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.
Appendix

This chapter contains reference information about encodings, emulations, and configuring SlickEdit Core.
Encoding

Encodings are used to convert a file to either SBCS/DBCS for the active code page or Unicode (more specifically UTF-8) data. By default, XML and Unicode files with signatures (UTF-8, UTF-16 and UTF-32) files are automatically loaded as Unicode UTF-8 data, while other more common program source files like .c, .java, and .cs source files are loaded as SBCS/DBCS active code page data.

All file data can be configured to Unicode UTF-8 data, but this would cause some problems. Loading files containing SBCS/DBCS data would take significantly longer, slowing down parsing by Context Tagging® and any other multi-file operations. In addition, Unicode editors cannot support all the features supported by SBCS/DBCS editors due to font limitations. For more information, see Unicode Limitations.

To provide better support for editing Unicode and non-Unicode files, two modes of editing exist: Unicode and SBCS/DBCS mode. Files that contain Unicode, XML, or code page data not compatible with the active code page should be opened as Unicode files.

The following are non-Unicode encodings and put the editor in SBCS/DBCS editing mode: Default, Text, SBCS/DBCS mode, Binary, SBCS/DBCS mode, and EBCDIC, SBCS/DBCS mode. In addition, the Auto Unicode, Auto Unicode2, Auto EBCDIC and Unicode, and Auto EBCDIC and Unicode2 encodings put the editor into SBCS/DBCS editing mode when the file is determined not to be Unicode. All other encodings put the editor in Unicode mode and require that the file data be converted to UTF-8.

There are many encodings available, including:

- **Auto XML** - This encoding specifies that the file encoding be determined based on XML standards and that the file be loaded as Unicode data. The encoding is determined based on the encoding specified by the ?xml tag. If the encoding is not specified by the ?xml, the file data is assumed to be UTF-8 data which is consistent with XML standards. We applied some modifications to the standard XML encoding determination to allow for some user error. If the file has a standard Unicode signature, the Unicode signature is assumed to be correct and the encoding defined by the ?xml tag is ignored.

- **Auto Unicode** - When this encoding is chosen and the file has a standard Unicode signature, the file is loaded as Unicode data. Otherwise the file is loaded as SBCS/DBCS data.

- **Auto Unicode2** - When this encoding is chosen and the file has a standard Unicode signature or looks like a Unicode file, the file is loaded as Unicode data. Otherwise the file is loaded as SBCS/DBCS data. This option is NOT fool-proof and may give incorrect results.

- **Auto EBCDIC** - When this encoding is chosen and the file looks like an EBCDIC file, the file is loaded as Unicode data. Otherwise, the file is loaded as SBCS/DBCS data. This option is NOT fool-proof and may give incorrect results. The option does attempt to support binary EBCDIC files.

- **Auto EBCDIC and Unicode2** - This encoding is a combination of the Auto EBCDIC and Auto Unicode2 encodings described above.

Using Unicode

To use encodings in SlickEdit® Core, Unicode support is required (OEMs typically turn this feature off).
Unicode is supported for the following list of features:

- All Context Tagging® features.
- Color Coding.
- Level 1 regular expressions as defined by the Unicode consortium.
- Multi-file search and replace.
- Support for many encodings including UTF-8, UTF-16, UTF-32, and many code pages. Automatic encoding recognition for XML files. Configure encoding recognition per extension or globally. Optionally store signatures and specify little endian or big endian. Use the Save As or Write Selection dialog to convert data to a particular file encoding.
- Support for converting Unicode to UNC data and visa versa. Supported UCN formats include \xHHHH, \x(HHHH), \uHHHH, &xHHHH; and &xDDDD; This is useful for specifying Unicode character strings in SBCS/DBCS active code page source files. See Converting Unicode to UCN.
- Multiple clipboards.
- Sorting.
- 3-Way Merge.
- Support for composite and surrogate characters.
- Support for storing up to 31-bit Unicode characters.
- SmartPaste®.
- Syntax Expansion and Syntax Indenting.
- Code beautifiers.
- Support for almost all of SlickEdit Core's SBCS/DBCS active code page features.

**Unicode File Recognition**

By default, XML and Unicode files with signatures (UTF-8, UTF-16 and UTF-32) files are automatically loaded as Unicode. If you have Unicode files that are not XML and do not have signatures, configure default options to get the best recognition possible. This is important because some features such as drag/drop files and DIFFzilla® do not prompt you for the file encoding.

Each extension may have its own encoding specification. If the extension-specific encoding is set to Default, then the global setting defined at Window → SlickEdit Preferences → 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® Core 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:

- **Bold and italics color-coding is not supported. Support for this will be added in a future version.**

- **Tab character operations are not fully supported. Tab display, the Expand tabs to spaces save option (Window → SlickEdit Preferences → File Options → Save), and save with tabs (save +t) only work correctly if all the characters are below 128. The Expand tabs to spaces load option (Window → SlickEdit Preferences → File Options → Load) is ignored.**

- **Column selections do not fully support Unicode. If all the characters are below 128 and the font is fixed then it works. Support for this will be added in a future version.**

- **Word Wrap does not fully support Unicode. If all the characters are below 128 and the font is fixed, then it works. Support for this will be added in a future version.**

- **The Unicode line end character 0x2048 is not supported.**

- **Hex editing is not supported. The current character (Composite character) is displayed on the status line. Also, use the Open dialog with the Binary, SBCS/DBCS mode encoding to view a Unicode file in hexadecimal.**

- **Casing features (uppercase, lowercase, ignore case) do not support surrogates. Windows is relied upon for casing support, and Windows casing features do not support surrogates. See Surrogate Support.**

- **Current Line Highlight (Window → SlickEdit Preferences → Appearance → General) does not support rulers.**

- **Vertical line column (Window → SlickEdit Preferences → Appearance → General) is not supported.**

- **Truncation line length is not supported.**

- **Record width on the File Open dialog is not supported.**

- **DDE is not supported. Unicode DDE does not work with Internet Explorer or Netscape®. You can view files with Unicode data in Internet Explorer; however, this feature will fail if the file name contains characters not in the active code page.**

- **Version control supports files containing Unicode data but does not support file names that contain characters not in the active code page.**

- **Special character display is not supported for Unicode buffers.**

- **The grew program does not support Unicode and can only be used on SBCS/DBCS active code page text.**

- **If you load the same source file in Unicode and SBCS/DBCS mode, the Context Tagging® database will have incorrect seek positions. It is important to use the default load options and to always load source files in the same encoding so that the Context Tagging seek positions match the editor seek positions.**

- **The install (setup.exe), unionist (uninstall.exe), and update (update.exe) programs are not**
Unicode applications so the installation directory must contain characters in the active code page.

• When using Cursor Style: Use vertical cursor (Window → SlickEdit Preferences → Appearance → General), cursor shape does not change when toggling Insert/Replace mode for Unicode files.

**Unicode Implementation**

Native Unicode and SBCS/DBCS editing modes are supported. When you edit a SBCS/DBCS (active code page) file such as a .c, .h, or .java file, the data is loaded as SBCS/DBCS data and is not converted to Unicode. When you edit a Unicode file, such as an XML file, the data is converted to UTF-8 that is one of the standard formats for supporting Unicode files. There are several advantages to this implementation:

• Since almost all source files for programming are stored as SBCS/DBCS, loading these files is significantly faster. This is very important to our customers who expect superior performance from SlickEdit® Core.

• Unicode editing modes cannot support all the features you were used to when editing SBCS/DBCS files (see Unicode Limitations).

• Macros can be written once to support both editing modes. This was very important to us because we wanted to reduce development time.

• Since Unicode is stored as UTF-8, only one set of binaries is required. Most products that support SBCS/DBCS and Unicode (UTF-16), use preprocessing. This requires two sets of binaries.
Environment Variables

Below is a list of environment variables that can be used within SlickEdit® Core. Configuration environment variables are set in the operating system or in vslick.ini file. For more information see Setting Environment Variables in vslick.ini

You can also use the set command from the SlickEdit Core 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 VSECLIPSECONFIG environment variable in vslick.ini. VSECLIPSECONFIG determines where the editor looks for vslick.ini. When the editor starts up, it sets the value of environment variables specified in vslick.ini. For more information, see Setting Environment Variables in vslick.ini.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSLICKRESTORE</td>
<td>Directory to store Auto Restore files.</td>
</tr>
<tr>
<td>VSECLIPSECONFIG</td>
<td>Directory where user's local configuration files are stored. Used in multi-user environments. Defaults to:</td>
</tr>
<tr>
<td></td>
<td>• (Windows) .../My Documents/My SlickEdit Core Config/[version]/</td>
</tr>
<tr>
<td></td>
<td>• (Linux, UNIX) $HOME/.secore/[version]/</td>
</tr>
<tr>
<td>Note that VSLICKCONFIG and VSLICKCLASSICCONFIG are no longer supported. VSLICKCONFIG has been replaced with a new variable, VSECLIPSECONFIG.</td>
<td></td>
</tr>
<tr>
<td>VSLICK</td>
<td>Specifies additional command line arguments to editor as if you were typing them in when invoking the editor.</td>
</tr>
<tr>
<td>VSLICKPATH</td>
<td>One or more directories separated with a semicolon (;) (or a colon [:] on UNIX) where batch macros or executable files are searched.</td>
</tr>
<tr>
<td>VSLICKMACROS</td>
<td>One or more directories separated with a semicolon</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>VSLICKBIN</td>
<td>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.</td>
</tr>
<tr>
<td>VSLICKBITMAPS</td>
<td>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.</td>
</tr>
<tr>
<td>VSLICKMISC</td>
<td>One or more directories separated with a semicolon (;) (or a colon [:] on UNIX) that contain miscellaneous files including *.vlx, *.slk, *.api, *.idx, vslick.sta (UNIX: vslick.stu), *.hlp, scommon.lst, main.dct, *.pif, *.ini (except for vslick.ini), and *.lst. VSLICKPATH must also contain the directories listed here.</td>
</tr>
<tr>
<td>VSLICKALIAS</td>
<td>One or more file names separated with a semicolon (;) (or a colon [:] on UNIX) that contain alias definitions.</td>
</tr>
<tr>
<td>VSLICKTAGS</td>
<td>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 vslick.ini.</td>
</tr>
<tr>
<td>VSLICKBACKUP</td>
<td>Directory to place backup files. Affects +D (default) and -D backup configurations only.</td>
</tr>
<tr>
<td>VSLICKSAVE</td>
<td>Allows save options to be specified per drive.</td>
</tr>
<tr>
<td>VSLICKLOAD</td>
<td>Allows load options to be specified per drive.</td>
</tr>
</tbody>
</table>
| VSLICK_XIM           | Set this to VSLICK_XIM=scim to activate the Smart Common Input Method (SCIM) editor. XMODIFIERS (XMODIFIERS=@scim) is set by your operating system and you must enable SCIM support for this option to have any effect. (This may
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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`
will create xterm windows with a width of 80 characters and a height of 40 characters. |
| VSUSER               | The License Manager handles system crashes better if each user sets the `VSUSER` environment variable to a unique name. |
| VST                  | Specifies additional command line arguments to the macro compiler as if you typed them in when invoking the compiler. |
| VSLICKXNOBLINK       | Suppresses the blinking cursor. |
| VSLICKXNOPLUSNEWMSG  | Suppresses a message when starting a second instance of SlickEdit Core. |

### Setting Environment Variables in vslick.ini

Along with whatever facilities are provided by your operating system to set environment variables, you can set configuration environment variables in the file `vslick.ini`. This file is located in the following default directory based on your platform (if it does not exist, it can be created manually):

- **Windows**: `.../My Documents/My SlickEdit Core Config/[version]/`
- **Linux and UNIX**: `$HOME/.secore/[version]/`

Below is text from a sample `vslick.ini` file with an environment section.

```ini
[Environment]
VSLICKPATH=c:\vslick\win;c:\vslick\macros;c:\vslick\bitmaps;c:\vmacros
VSLICKALIAS=c:\vmacros\alias.slk
VSLICKINCLUDE=c:\vslick\macros;c:\vmacros
VSLICKLOAD=a: +l b: +l
```

667
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 Core 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® Core 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® Core 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.

Alternatively, you can use the Symbols View 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 Core command line. This is an abbreviation of the `find_proc` command, which will trigger tagging if it hasn't already been done.

Setting/Changing Configuration Variables

There are two ways to set/change these macro variables:

- From the SlickEdit® Core 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 Core 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.

<table>
<thead>
<tr>
<th>Configuration Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def_alias_case</code></td>
<td>Controls whether alias identifier matching is case-sensitive. Set to <code>i</code> to make alias matching case-insensitive (default). Set to <code>e</code> to turn on case-sensitivity.</td>
</tr>
<tr>
<td>Configuration Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>def_auto_linecomment</td>
<td>Change to 0 to turn off automatic line comment insertion.</td>
</tr>
<tr>
<td>def_binary_ext</td>
<td>This variable is used by the editflst.e macro for the Brief emulation. When the editflst.e macro is loaded, the space-delimited extensions listed by this variable are filtered out by the edit command's completion. The default is .ex .obj .exe .lib.</td>
</tr>
<tr>
<td>def_ctags_flags</td>
<td>This variable is a safeguard against parsing past the end of a proc when the braces mismatch. To have SlickEdit® Core recognize the second dd, set the value of this variable to 10.</td>
</tr>
<tr>
<td>def_debug_logging</td>
<td>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.</td>
</tr>
<tr>
<td>def_deselect_copy</td>
<td>Set to 1 in Brief emulation to deselect after a copy.</td>
</tr>
<tr>
<td>def_do_block_mode_key</td>
<td>Set this variable’s value equal to 0 to stop SlickEdit Core from inserting characters on every line of a block selection.</td>
</tr>
<tr>
<td>def_error_re2</td>
<td>Edit this variable to change from the SlickEdit Core regular expression used for compile/build errors.</td>
</tr>
<tr>
<td>def_fast_auto_readonly</td>
<td>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.</td>
</tr>
<tr>
<td>def_from_cursor</td>
<td>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.</td>
</tr>
<tr>
<td>def_linewrap</td>
<td>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</td>
</tr>
<tr>
<td>Configuration Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>whitespace only on the line below it (spaces or tabs), the whitespace is removed entirely acting as a line delete.</td>
<td>def linux1_shell</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>This variable is set for performance reasons. You can increase the amount of information displayed in the Output view during a multi-file search by changing this to your desired setting.</td>
<td>def max mffind output</td>
</tr>
<tr>
<td>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.</td>
<td>def modal paste</td>
</tr>
<tr>
<td>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.</td>
<td>def plusminus blocks</td>
</tr>
<tr>
<td>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 save xyz, and an xyz file already exists, and xyz is not the name of the current buffer, you are prompted by default whether you wish to overwrite the file.</td>
<td>def preplace</td>
</tr>
<tr>
<td>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?</td>
<td>def rwprompt</td>
</tr>
<tr>
<td>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.</td>
<td>def save macro</td>
</tr>
<tr>
<td>Set this value to 1 for Shift+Up or Shift+Down to select the current line.</td>
<td>def shift updown line select</td>
</tr>
<tr>
<td>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</td>
<td>def switchbuf cd</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Variable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>def_top_bottom_push_bookmark</td>
<td>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.</td>
</tr>
<tr>
<td>def_undo_with_cursor</td>
<td>Set this value to 1 to enable the undo of each cursor movement.</td>
</tr>
<tr>
<td>def_update_context_max_file_size</td>
<td>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).</td>
</tr>
<tr>
<td>def_vc_advanced_options</td>
<td>Set to this variable to 0 to remove advanced options that decrease performance when using ClearCase version control.</td>
</tr>
<tr>
<td>def_vtg_tornado</td>
<td>Set this variable value to 0 to prevent Context Tagging of Tornado files.</td>
</tr>
<tr>
<td>def_xml_no_schema_list</td>
<td>To prevent SlickEdit Core 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 Core from attempting to connect to the Internet for these extensions.</td>
</tr>
</tbody>
</table>
Configuration Directories and Files

User Configuration Directory

Your SlickEdit® Core configuration directory contains configuration files representing the changes you have made through setting editor options, and it preserves the state of SlickEdit Core by using the state file, Table of User Configuration Files.

You should make periodic backups of your SlickEdit Core 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 Core 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.

Directory Location

By default, the configuration directory is located in My Documents/My SlickEdit Core Config/<Version> on Windows, and $HOME/.secore/<Version> on Linux. You can view the path to the config directory by clicking Help → About SlickEdit Core.

Changing the Configuration Directory

If you would like to use a different directory for your config files, you can pass the –vsconfig argument to Eclipse. This works the same as passing any argument to eclipse.exe (or eclipse on Linux). For example: C:\Path_To_Eclipse\eclipse.exe –vsconfig=C:\My_Config.

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 Eclipse with SlickEdit® Core 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 but NOT the directory itself.

4. Start Eclipse with SlickEdit Core.

After completing these steps, SlickEdit Core 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 Core for the first time.

Note
If, instead of deleting the contents of the directory, you delete the directory itself, SlickEdit Core 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.

<table>
<thead>
<tr>
<th>User Config File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*als</td>
<td>A text file that contains user-defined language-specific aliases.</td>
</tr>
<tr>
<td>alias.slk</td>
<td>A text file that contains user-defined global aliases (directory aliases).</td>
</tr>
<tr>
<td>ftp.ini</td>
<td>A text file that contains user-defined FTP configurations.</td>
</tr>
<tr>
<td>(UNIX: uftp.ini)</td>
<td></td>
</tr>
<tr>
<td>project.vpe</td>
<td>A text file that contains user-defined language-specific projects.</td>
</tr>
<tr>
<td>(UNIX: uproject.vpe)</td>
<td></td>
</tr>
<tr>
<td>ubox.ini</td>
<td>A text file that contains user-defined box and line comment styles.</td>
</tr>
<tr>
<td>uformat.ini</td>
<td>A text file that contains user-defined beautifier schemes.</td>
</tr>
<tr>
<td>uprint.ini</td>
<td>A text file that contains user-defined printing schemes.</td>
</tr>
<tr>
<td>uscheme.ini</td>
<td>A text file that contains user-defined color schemes.</td>
</tr>
<tr>
<td>user.vlx</td>
<td>A text file that contains color coding changes (keywords, etc.). This file is updated when you close the Color Coding dialog box.</td>
</tr>
<tr>
<td>usercpp.h</td>
<td>A text file that contains defines (default preprocessing) for Context Tagging® of C++ and C code.</td>
</tr>
<tr>
<td>usercvc.slk</td>
<td>A text file that contains user-defined version control systems.</td>
</tr>
<tr>
<td>usrprjtemplates.vpt</td>
<td>A text file that contains user-defined project</td>
</tr>
<tr>
<td>User Config File</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>vrestore.slk</td>
<td>A text file that contains auto-restore information, such as buffer information and command line history. The workspace files also contain auto-restore information, but only for the files/windows previously open.</td>
</tr>
<tr>
<td>vslick.ini</td>
<td>A text file that contains a few miscellaneous options. The user-configured backup directory is stored here. In addition, some customizable environment variables for path searching for macros, bitmaps, and binary files are stored here as well.</td>
</tr>
<tr>
<td>vslick.sta</td>
<td>A binary file that contains dialog boxes, menus, macro pcode, key bindings, and all other configuration data not stored in one of the other configuration files. Both user and system configuration information is stored here.</td>
</tr>
<tr>
<td>vusrdefs.e</td>
<td>A Slick-C® text file that contains the emulation setting, key bindings, color settings, language setup information, and some other miscellaneous options.</td>
</tr>
<tr>
<td>vusroobjs.e</td>
<td>A text file that contains user-defined dialog boxes and menus in Slick-C syntax.</td>
</tr>
<tr>
<td>vusr*.e</td>
<td>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.</td>
</tr>
</tbody>
</table>

**System Configuration Files**

System configuration files are located in the SlickEdit® Core 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 Core.

**Table of System Configuration Files**
The table below provides a list of the system configuration files.

<table>
<thead>
<tr>
<th>System Config File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias.slk</td>
<td>A text file that contains default global aliases (for example, directory aliases).</td>
</tr>
<tr>
<td>box.ini</td>
<td>A text file that contains default box and line comment styles. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.</td>
</tr>
<tr>
<td>format.ini</td>
<td>A text file that contains default beautifier schemes.</td>
</tr>
<tr>
<td>print.ini</td>
<td>A text file that contains default printing schemes. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.</td>
</tr>
<tr>
<td>prjtemplates.vpt</td>
<td>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.</td>
</tr>
<tr>
<td>syscpp.h</td>
<td>A text file that contains system-defined default preprocessing for Context Tagging® of C++ and C code.</td>
</tr>
<tr>
<td>(UNIX: usyscpp.h)</td>
<td></td>
</tr>
<tr>
<td>vcsystem.slk</td>
<td>A text file that contains default version control systems. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.</td>
</tr>
<tr>
<td>(UNIX: uvcsys.slk)</td>
<td></td>
</tr>
<tr>
<td>vslick.ini</td>
<td>A text file that contains a few miscellaneous options. Some customizable environment variables for path searching for macros, bitmaps, and binary files are stored here as well. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.</td>
</tr>
<tr>
<td>vslick.sta</td>
<td>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.</td>
</tr>
<tr>
<td>(UNIX: vslick.stu)</td>
<td></td>
</tr>
<tr>
<td>vslick.vlx</td>
<td>A text file that contains default color coding lexer definitions.</td>
</tr>
<tr>
<td>vsscheme.ini</td>
<td>A text file that contains default color schemes. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.</td>
</tr>
</tbody>
</table>
## File Search Order

<table>
<thead>
<tr>
<th>System Config File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>preserved when a new editor is installed.</td>
</tr>
</tbody>
</table>
File Search Order

Search Order for Configuration Files

Several files are automatically searched for, either immediately when the editor is invoked or during the course of operation. The search order for configuration files such as vslick.ini, vslick.sta, and vrestore.slk is:

1. Configuration directory.
2. Current directory.
3. Paths specified in VSLICKPATH environment variable.
4. Paths specified in PATH environment variable.

Search Order for Executable Files

The search order for executable files, batch macro programs, and miscellaneous files is:

2. Configuration directory.
3. Paths specified in VSLICKPATH environment variable.
4. Paths specified in PATH environment variable.
For more basic information about using Color Coding, see Colors, Color Coding, and Symbol Colors.

To modify the color coding for VLX files, use one of the following methods:

- Use the language-specific Color Coding options screen (see Language-Specific Color Coding Options).
- Modify the vslick.vlx file.
- Create a new VLX file.

The vslick.vlx file defines language-specific coloring support for the following languages:

- Ada
- Assembler
- AWK
- C
- C++
- CFScript
- CICS
- COBOL
- dBASE
- Delphi
- Fortran
- HTML
- Java
- Modula-2
- Pascal
- Perl
- Python
- REXX
- Slick-C®
Modifying the VLX File to Change a Color Definition

To modify an existing language-specific coloring definition, complete the following steps:

1. Open vslick.vlx for editing.

2. Search for one of the section names: CPP, Java, Delphi, Pascal, AWK, REXX, Perl, HTML, Modula-2, AWK, COBOL, Python, CICS, Fortran, Visual Basic .NET, Ada, or Slick-C®.

3. Modify the definition. See below for information on the syntax of definitions.

4. Invoke the `cload` command from the command line. If the current buffer has a `.vlx` extension, it will be loaded. Otherwise you will be prompted to specify a file name. Specify `vslick.vlx` including path as the file name.

Creating a Lexer Name and a New VLX File

To create a new lexer name (and thus a new section in the VLX file), first complete all of the preceding steps under Modifying the VLX File to Change a Color Definition, then complete the steps below.

1. From the main menu, click Window → SlickEdit Preferences → Languages, expand your language category and language, then select General.

2. If this lexer definition is for a new extension, create the extension with the New button. Otherwise, choose the appropriate extension.

3. Set the Lexer Name for the new lexer definition you created.

4. Turn on the Language Specific check box.

5. Click Update to commit the changes.

Files with a `.vlx` extension are text files that have a syntax similar to a `.ini` file. If the first non-blank character in a line is a semicolon, the line is considered a comment. Each definition of a language starts with a section name (the lexer name) enclosed in square brackets. Within each section are statements that look like `name=value`.

The table below shows the statements that can be used.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`case-sensitive=[Y</td>
<td>N]`</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>idchars=start_id_chars after_id_chars</td>
<td>Defines the characters that are the start of a valid identifier and additional valid characters that may follow. This statement must be the first or second statement within the section. You may use a 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.</td>
</tr>
<tr>
<td>styles= style</td>
<td>Defines zero or more styles. See Table of style Values below for a list of available styles.</td>
</tr>
</tbody>
</table>
| mlcomment= start_symbol end_symbol [nesting] [followedby idchars ] [ colorname ] | Defines a multi-line comment. *start_symbol* and *end_symbol* define strings which start and end the comment. Specify *nesting* if the lexer should look for another occurrence of *start_symbol* when looking for the end comment symbol. The *followedby*idchars is used to require certain characters to follow *start_symbol*. You can use a dash (-) character to specify a range, such as A-Z, which specifies uppercase letters. To specify a dash (-) or backslash (\) character as a valid word character, place a backslash before the character. *followedby*is ignored when the html style is specified. Currently, *start_symbol* and *end_symbol* may not be valid identifiers. No more than four multi-line comments may be defined. *colorname* can be used to indicate that a different color such as keyword color be used instead of comment color when a match is found. *colorname* may be *keyw...
### Statement | Description
--- | ---
| | coding of symbols in the line. When the `checkfirst` option is specified, `start_symbol` is limited to one character in length. `leading` specifies that `symbol` is considered a line comment only if it appears as the first non-blank character. Space or tab characters are considered blanks. Currently, `end_symbol` may not be a valid identifier. `lastchar` specifies that `end_symbol` must appear as the last character on a line to terminate the comment. No more than two multi-line comments may be defined.

| `mlckeywords=[ keyword ] [ keyword ] ...` | Defines keywords for the last `mlcomment` statement. When one of these keywords follows the `start_symbol` defined for the last `mlcomment` statement, the keyword color is used to color the comment instead of comment color. Keywords do not have to be valid identifiers. This statement is useful for tag languages like HTML. See the HTML definition in the file `vslick.vlx` for an example.

| `keywordattrs=[ mlckeyword ] [ attribute ] [ attribute ] ...` | Defines attributes for the `mlckeyword` specified which belongs to the last `mlcomment` statement. Currently this statement only supports HTML syntax attributes and requires that the HTML style be specified. For example, `keywordattrs=SCRIPT LANGUAGE SRC`.

| `linecomment=[ symbol ] [ col | col + | start_col - end_col ] [checkfirst|leading]` | Defines a line comment. `symbol` defines the character(s) which start this line comment. If no column limits are specified, the remainder of the line is considered a comment regardless of where `symbol` appears. A plus sign (+) after a column specifies an unlimited `end_col`. `checkfirst` specifies that the lexer should check if the line is a comment before determining the color coding of symbols in the line. When the `checkfirst` option is specified, `symbol` is limited to one character in length. If `symbol` is not specified, all characters will be ignored at or after the column specified (ex. `linecomment=73+`). This is useful for Fortran which requires that all characters at or after column 73 be ignored. `leading` specifies that `symbol` is considered a line comment only if it appears as the first non-blank character. Space or tab characters are considered blanks.
### Statement

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>keywords= [ keyword ] [ keyword ] ...</code></td>
<td>Defines words that should be displayed in keyword color. Keywords do not have to be valid identifiers.</td>
</tr>
<tr>
<td><code>cskeywords= [ keyword ] [ keyword ] ...</code></td>
<td>(Case-sensitive keywords) Defines words that should be displayed in keyword color only if found in the case specified. This statement should only be used for languages such as HTML which are case insensitive except for a few words. For other languages, use the <code>case-sensitive</code> and <code>keywords</code> statements. Keywords do not have to be valid identifiers.</td>
</tr>
<tr>
<td><code>ppkeywords= [ keyword ] [ keyword ] ...</code></td>
<td>Defines words that should be displayed in preprocessor color. The first character of a preprocessor keyword must not be a valid identifier. Preprocessing keywords must appear as the first non-blank symbol in the line.</td>
</tr>
<tr>
<td><code>symbol1= [ keyword ] [ keyword ] ...</code></td>
<td>Defines words that should be displayed in <code>symbol1</code> color. Keywords do not have to be valid identifiers.</td>
</tr>
<tr>
<td><code>symbol2= [ keyword ] [ keyword ] ...</code></td>
<td>Defines words that should be displayed in <code>symbol2</code> color. Keywords do not have to be valid identifiers.</td>
</tr>
<tr>
<td><code>symbol3= [ keyword ] [ keyword ] ...</code></td>
<td>Defines words that should be displayed in <code>symbol3</code> color. Keywords do not have to be valid identifiers.</td>
</tr>
<tr>
<td><code>symbol4= [ keyword ] [ keyword ] ...</code></td>
<td>Defines words that should be displayed in <code>symbol4</code> color. Keywords do not have to be valid identifiers.</td>
</tr>
</tbody>
</table>

### Table of Style Values

The table below describes the `style` values that can be used:

<table>
<thead>
<tr>
<th>Value of style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linenum</td>
<td>Line numbers may be found as the first non-blank symbol of a line like BASIC.</td>
</tr>
<tr>
<td>dqbackslash</td>
<td>Color double-quoted strings. Characters following a backslash in a double-quoted string are included in the string (like C).</td>
</tr>
<tr>
<td>Value of style</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dqbackslashml</td>
<td>Color double-quoted strings. If a double-quoted string ends in a backslash, it continues the string to the next line (like C).</td>
</tr>
<tr>
<td>dqmultiline</td>
<td>Color double-quoted strings. String may span multiple lines.</td>
</tr>
<tr>
<td>dqdoubles</td>
<td>Color double-quoted strings. Two double quotes represent one double quote.</td>
</tr>
<tr>
<td>dqterminate</td>
<td>Do not color-code a double-quoted string until the string is terminated. This style does not support dqmultiline or dqbackslashml.</td>
</tr>
<tr>
<td>dqlen1</td>
<td>Color double-quoted strings. Double-quoted strings contain exactly one character.</td>
</tr>
<tr>
<td>sqbackslash</td>
<td>Color single-quoted strings. Characters following a backslash in a single-quoted string are included in the string (like C).</td>
</tr>
<tr>
<td>sqbackslashml</td>
<td>Color single-quoted strings. If a double-quoted string ends in a backslash, it continues the string to the next line.</td>
</tr>
<tr>
<td>sqmultiline</td>
<td>Color single-quoted strings. String may span multiple lines.</td>
</tr>
<tr>
<td>sqdoubles</td>
<td>Color single-quoted strings. Two consecutive single quotes represent one single quote (like Pascal).</td>
</tr>
<tr>
<td>sqterminate</td>
<td>Do not color-code a single-quoted string until the string is terminated. This style does not support sqmultiline or sqbackslashml.</td>
</tr>
<tr>
<td>sqlen1</td>
<td>Single-quoted strings contain exactly one character (like Ada).</td>
</tr>
<tr>
<td>amphhex</td>
<td>Hexadecimal numbers are of the form &amp;Hdddd (like BASIC).</td>
</tr>
<tr>
<td>ampoocct</td>
<td>Octal numbers are of the form &amp;Oddd (like BASIC).</td>
</tr>
</tbody>
</table>
# Value of style

<table>
<thead>
<tr>
<th>Value of style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hexh</td>
<td>Hexadecimal numbers are of the form <code>dddhH</code> (like Intel Assembler).</td>
</tr>
<tr>
<td>octo</td>
<td>Octal numbers are of the form <code>dddoO</code> (like Intel Assembler).</td>
</tr>
<tr>
<td>octq</td>
<td>Octal numbers are of the form <code>dddoQ</code> (like Intel Assembler).</td>
</tr>
<tr>
<td>poundbase</td>
<td>Based numbers are of the form <code>#base#number#exponent</code> (like Ada).</td>
</tr>
<tr>
<td>underlineint</td>
<td>Numbers may have underlines between the numbers (like Ada).</td>
</tr>
<tr>
<td>xhex</td>
<td>Hexadecimal numbers are of the form <code>0xhhhh</code> (like C).</td>
</tr>
<tr>
<td>nonumbers</td>
<td>Do not color-code numbers. This style is useful for tag languages like HTML. Using this style with other number color-coding styles will produce unpredictable results.</td>
</tr>
<tr>
<td>rexxhex</td>
<td>Hexadecimal strings are followed by an upper or lowercase letter <code>X</code>. For example, <code>'414141X</code> or <code>414141X</code> are REXX-style hexadecimal strings that are both equivalent to the string <code>AAA</code>.</td>
</tr>
<tr>
<td>packageimport</td>
<td>Language has Java syntax package and import statement where non-quoted file name follows <code>package</code> and <code>import</code> keyword.</td>
</tr>
<tr>
<td>idparenfunction</td>
<td>An identifier followed by an open parenthesis indicates a function (like C++ and Java).</td>
</tr>
<tr>
<td>html</td>
<td>Enables HTML syntax embedded languages and attribute coloring.</td>
</tr>
<tr>
<td>backslashescapechars</td>
<td>Backslash escapes the character that follows.</td>
</tr>
<tr>
<td>heredocument</td>
<td>Enables support for Heredocuments. Note that if you prefix the terminator with one of the lexer names, you will get embedded language color-coding.</td>
</tr>
<tr>
<td>Value of style</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>perl</td>
<td>Adds support for Perl <strong>format</strong> statement and some other Perl-specific changes.</td>
</tr>
<tr>
<td>tcl</td>
<td>Special support for TCL language color-coding.</td>
</tr>
<tr>
<td>bquote</td>
<td>Perl- and Linux Shell-style backquote (subshell).</td>
</tr>
<tr>
<td>model204</td>
<td>Special support for Model 204 language.</td>
</tr>
<tr>
<td>cics</td>
<td>Special support for CICS embedded in COBOL.</td>
</tr>
<tr>
<td>python</td>
<td>Special support for Python.</td>
</tr>
</tbody>
</table>
Editing the Key Binding Source

If you are creating a new emulation or if you change many key bindings, you might want to edit your key binding source instead of using the Key Binding Options screen. To create a Slick-C® batch macro containing your current key bindings, enter the command `list_source` on the command line. One of the files generated by this command is `vusrdefs.e`. It is placed in your configuration directory. If you open this file (`Ctrl+O`), the first part of the source code is your key binding, which looks like the following:

```plaintext
defeventtab default_keys
def 'A-a'-'A-z'=
def 'A-F6'=
def 'F10'=
def 'C-A' = select all
def 'C-B' = select_block
def 'C-C' = copy_to_clipboard
def 'C-D' = gui_cd
```

The `default_keys` are the key bindings that are active in Fundamental mode. The other event tables defined by the `defeventtab` primitive are mode event tables containing key bindings which override the Fundamental mode key bindings. Make changes to this buffer by adding or modifying the `def keyname=` command lines and then save the buffer by pressing `Ctrl+S`. The valid key names are listed in the Help system under Event Names. You can also list the key names of the keys through the Help by invoking the command `help Event Names`. To run this batch program, type the name `vusrdefs` without the extension on the command line. The path is not necessary if it is included in your VSLICKPATH or PATH environment variable.
Using the ISPF and XEDIT Emulations

This 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 Core must be in ISPF emulation mode. Then, from the main menu, click Window → SlickEdit Preferences, 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISPF Line Labels</td>
<td>Define a label.</td>
</tr>
<tr>
<td>ISPF Line Command BNDS</td>
<td>Insert a column boundary ruler line.</td>
</tr>
<tr>
<td>ISPF Line Command COLS</td>
<td>Insert a column ruler line.</td>
</tr>
<tr>
<td>ISPF Line Command First</td>
<td>Expose one or more lines at the beginning of a</td>
</tr>
</tbody>
</table>
### ISPF Primary Commands

<table>
<thead>
<tr>
<th>ISPF Line Command Last</th>
<th>block of excluded lines.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISPF Line Command Show</td>
<td>Expose one or more lines having the leftmost indentation level in a block of excluded lines.</td>
</tr>
<tr>
<td>ISPF Line Command TABS</td>
<td>Displays the tab definition line.</td>
</tr>
<tr>
<td>ISPF Line Command Exclude</td>
<td>Specifies one or more lines to be hidden (excluded).</td>
</tr>
<tr>
<td>ISPF Line Command Select</td>
<td>Select a block of lines.</td>
</tr>
</tbody>
</table>

- **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 options screen (Window → SlickEdit Preferences → Languages → [Language Category] → [Language] → General). 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 Language-Specific General Options 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ispf_autosave</td>
<td>Turn on or off prompting to save changes.</td>
</tr>
<tr>
<td>ispf_bounds</td>
<td>Set or reset the left and right edit boundaries.</td>
</tr>
<tr>
<td>ispf_bnds</td>
<td>Set or reset the left and right edit boundaries</td>
</tr>
<tr>
<td>ispf_browse</td>
<td>Browse a data set or member.</td>
</tr>
<tr>
<td>ispf_cancel</td>
<td>Closes the current file or PDS member without saving changes.</td>
</tr>
<tr>
<td>ispf_caps</td>
<td>Turn on or off automatic capitalization mode.</td>
</tr>
<tr>
<td>ispf_change</td>
<td>Replace one string with another within the current buffer.</td>
</tr>
<tr>
<td>ispf_chg</td>
<td>Replace one string with another within the current buffer.</td>
</tr>
<tr>
<td>ispf_compare</td>
<td>Compare the file you are editing with another file.</td>
</tr>
<tr>
<td>ispfs</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>ispfs_copy</td>
<td>Insert the contents of a file. This command requires a full path, and will not work with only a PDS member name specified.</td>
</tr>
<tr>
<td>ispfs_create</td>
<td>Create a new file or PDS member containing the contents of the buffer.</td>
</tr>
<tr>
<td>ispfs_cut</td>
<td>Cut lines out of the current buffer and place them in the clipboard.</td>
</tr>
<tr>
<td>ispfs_delete</td>
<td>Delete lines in the given line range, or the entire buffer.</td>
</tr>
<tr>
<td>ispfs_edit</td>
<td>This command is identical to the built-in edit command.</td>
</tr>
<tr>
<td>ispfs_end</td>
<td>Close the current file.</td>
</tr>
<tr>
<td>ispfs_exclude</td>
<td>Hide (exclude) lines that match the given search string.</td>
</tr>
<tr>
<td>ispfs_find</td>
<td>Find occurrences of the given search string in the current buffer.</td>
</tr>
<tr>
<td>ispfs_flip</td>
<td>Reverse the exclude status of lines.</td>
</tr>
<tr>
<td>ispfs_hex</td>
<td>Toggle display of the document in Hexadecimal mode.</td>
</tr>
<tr>
<td>ispfs_hilite</td>
<td>Specify the use of color-coding in the editor.</td>
</tr>
<tr>
<td>ispfs_locate</td>
<td>Find lines with a specific line prefix.</td>
</tr>
<tr>
<td>ispfs_move</td>
<td>Move the contents of a file or PDS member into the buffer.</td>
</tr>
<tr>
<td>ispfs_nonumber</td>
<td>Turn off numbering mode.</td>
</tr>
<tr>
<td>ispfs_number</td>
<td>Controls line numbering mode. Unlike ISPF, this command does affect how lines are inserted.</td>
</tr>
<tr>
<td>ispfs_paste</td>
<td>Copy lines from the clipboard to the buffer.</td>
</tr>
<tr>
<td>ispfs_preserve</td>
<td>Controls saving of trailing blanks.</td>
</tr>
<tr>
<td>ispfs_rchange</td>
<td>Repeat the change requested by the most recent</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ispf_renumber</td>
<td>Immediately update the line numbers in a file.</td>
</tr>
<tr>
<td>ispf_replace</td>
<td>Save the contents of the current buffer to an existing file.</td>
</tr>
<tr>
<td>ispf_reset</td>
<td>Reset the contents of the line prefix area.</td>
</tr>
<tr>
<td>ispf_return</td>
<td>Close the current file.</td>
</tr>
<tr>
<td>ispf_rfind</td>
<td>Repeat the last find operation requested.</td>
</tr>
<tr>
<td>ispf_save</td>
<td>This command is identical to the built-in save command.</td>
</tr>
<tr>
<td>ispf_sort</td>
<td>Sort lines of data in a specified order.</td>
</tr>
<tr>
<td>ispf_submit</td>
<td>Submit the contents of the current buffer for batch processing.</td>
</tr>
<tr>
<td>ispf_swap</td>
<td>Switch to the next buffer.</td>
</tr>
<tr>
<td>ispf_tabs</td>
<td>Define logical tab positions.</td>
</tr>
<tr>
<td>ispf_unnumber</td>
<td>Blank out the line numbers in a file.</td>
</tr>
<tr>
<td>ispf_undo</td>
<td>This command is identical to the undo command.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ISPF Line Labels</td>
<td>Define a label.</td>
</tr>
<tr>
<td>ISPF Line Command Shift</td>
<td>Shift data left or right.</td>
</tr>
<tr>
<td>ISPF Line Command A</td>
<td>Identify a line after which lines are to be inserted.</td>
</tr>
<tr>
<td>ISPF Line Command B</td>
<td>Identify a line before which lines are to be inserted.</td>
</tr>
<tr>
<td>ISPF Line Command BNDS</td>
<td>Insert a column boundary ruler line.</td>
</tr>
<tr>
<td>ISPF Line Command Copy S</td>
<td>Specify lines to be copied to another location.</td>
</tr>
<tr>
<td>ISPF Line Command COL</td>
<td>Insert a column ruler line.</td>
</tr>
<tr>
<td>ISPF Line Command Delete</td>
<td>Delete one or more lines.</td>
</tr>
<tr>
<td>ISPF Line Command First</td>
<td>Expose one or more lines at the beginning of a block of excluded lines.</td>
</tr>
<tr>
<td>ISPF Line Command I</td>
<td>Insert one or more blank data entry lines.</td>
</tr>
<tr>
<td>ISPF Line Command Lowercase</td>
<td>Convert all uppercase letter alphabetic characters in one or more lines to lowercase.</td>
</tr>
<tr>
<td>ISPF Line Command Last</td>
<td>Expose one or more lines at the beginning of a block of excluded lines.</td>
</tr>
<tr>
<td>ISPF Line Command Move</td>
<td>Specify lines to be moved to another location.</td>
</tr>
<tr>
<td>ISPF Line Command MASK</td>
<td>Display the contents of the mask used with the insert (I) and text entry (TE) line commands.</td>
</tr>
<tr>
<td>ISPF Line Command Make Data</td>
<td>Convert one or more no-save lines to data so that they may be saved when the buffer is saved.</td>
</tr>
<tr>
<td>ISPF Line Command Overlay</td>
<td>Identify one or more lines over which the copy or move block is to be overlaid.</td>
</tr>
<tr>
<td>ISPF Line Command Repeat</td>
<td>Specify lines to be repeated immediately following this line or block.</td>
</tr>
<tr>
<td>ISPF Line Command Show</td>
<td>Expose one or more lines having the left-most indentation level in a block of excluded lines.</td>
</tr>
</tbody>
</table>
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).
- .zcsp - 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
ISP Printer Line Commands

- \( ( [ n ] \) - 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 \( (, <, \text{ or } > \) 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

ISP Printer 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

ISP Printer 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
ISPF Line Commands

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[n] 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 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 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 Line Commands

ispf_caps, ISPF Line Command Lowercase, lowcase, upcase

ISPF Exclude Lines X and XX for blocks

Usage

• X [ n ] - Exclude n lines starting with the line with the command.
• XX - Exclude a block of lines, must match another XX.

Remarks

Specifies one or more lines to be hidden (excluded). There are two forms to this command. The first form (X [ n ]) specifies that the line and the subsequent n-1 lines are to be excluded. The second (block) form (XX) is placed on the first and last lines of the block to be excluded.

See Also

ispf_exclude, ispf_reset, ISPF Line Command First, ISPF Line Command Last, ISPF Line Command Show

ISPF Select Lines Z and ZZ for blocks

Usage

• Z [ n ] - Select n lines starting with the line with the command.
• ZZ - Select a block of lines, must match another ZZ.

Remarks

Select a block of lines. There are two forms to this command. The first form (Z [ n ]) specifies that the line and the subsequent n-1 lines are to be selected. The second (block) form (ZZ) is placed on the first and last lines of the block to be selected.

See Also

ispf_cut, ispf_paste

XEDIT Line Commands

The following XEDIT line commands are supported and override the like-named ISPF commands when there is a conflict. XEDIT commands can be enabled using the ISPF Options dialog box (Window → SlickEdit Preferences → ISPF Options).

<table>
<thead>
<tr>
<th>XEDIT</th>
<th>ISPF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>R</td>
<td>Repeat the marked line.</td>
</tr>
<tr>
<td>F</td>
<td>A</td>
<td>Paste text following line.</td>
</tr>
</tbody>
</table>
ISPF Line Commands

<table>
<thead>
<tr>
<th>XEDIT</th>
<th>ISPF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>I</td>
<td>Add (insert) line(s).</td>
</tr>
<tr>
<td>P</td>
<td>B</td>
<td>Paste text before line.</td>
</tr>
<tr>
<td>L</td>
<td>LC</td>
<td>Make line lowercase.</td>
</tr>
<tr>
<td>LL</td>
<td>LCC</td>
<td>Make block lowercase.</td>
</tr>
<tr>
<td>U</td>
<td>UC</td>
<td>Make line uppercase.</td>
</tr>
<tr>
<td>UU</td>
<td>UCC</td>
<td>Make block uppercase.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Unsupported Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autolist</td>
<td>Control the automatic printing of data to the ISPF list data set.</td>
</tr>
<tr>
<td>builtin</td>
<td>Process a built-in command, even if overloaded by a macro.</td>
</tr>
<tr>
<td>define</td>
<td>Define a name as an alias or macro.</td>
</tr>
<tr>
<td>imacro</td>
<td>Save the name of an initial macro in the edit profile.</td>
</tr>
<tr>
<td>level</td>
<td>Set the modification level number in PDF library statistics.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>model</td>
<td>Copy a model into the buffer or defines a model class.</td>
</tr>
<tr>
<td>notes</td>
<td>Control whether the MODEL command display notes or not.</td>
</tr>
<tr>
<td>nulls</td>
<td>Control null spaces.</td>
</tr>
<tr>
<td>pack</td>
<td>Control whether data is to be stored compressed or not.</td>
</tr>
<tr>
<td>profile</td>
<td>Display edit profile.</td>
</tr>
<tr>
<td>recovery</td>
<td>Specify edit recovery options.</td>
</tr>
<tr>
<td>rmacro</td>
<td>Save a recovery macro in the edit profile.</td>
</tr>
<tr>
<td>setundo</td>
<td>Control the UNDO mode.</td>
</tr>
<tr>
<td>stats</td>
<td>Generate library statistics.</td>
</tr>
<tr>
<td>version</td>
<td>Set the version number in the PDF library statistics.</td>
</tr>
<tr>
<td>view</td>
<td>Save as browse command but prompts on save.</td>
</tr>
</tbody>
</table>

The following commands are supported in ISPF emulation mode.

<table>
<thead>
<tr>
<th>Supported Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ispf_bottom</td>
<td>Move cursor to the end of the buffer.</td>
</tr>
<tr>
<td>ispf_down</td>
<td>Move cursor to next page of text.</td>
</tr>
<tr>
<td>ispf_enter</td>
<td>Handle the Enter key or Right Control key in ISPF emulation.</td>
</tr>
<tr>
<td>ispf_home</td>
<td>Place the focus on the command line in ISPF emulation.</td>
</tr>
<tr>
<td>ispf_retrieve</td>
<td>Does command line retrieval, getting the next command line from the list.</td>
</tr>
<tr>
<td>ispf_retrieve_back</td>
<td>Identical to the ispf_retrieve back command.</td>
</tr>
</tbody>
</table>
### Menu Editing

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ispf_top</code></td>
<td>Move cursor up to the top of the buffer.</td>
</tr>
<tr>
<td><code>ispf_up</code></td>
<td>Move cursor up to the previous page of text.</td>
</tr>
<tr>
<td><code>ispf_do_lc</code></td>
<td>Immediately process all commands found in the line prefix area.</td>
</tr>
</tbody>
</table>
Menu Editing

For information about accessing menus in SlickEdit Core and associated options, see Menus.

Creating and Editing Menus

Menus in SlickEdit Core 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 Core.

Warning

SlickEdit Core 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 Core. 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 Core, 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.

Note

The menu editor can only be used to edit SlickEdit Core menus, like the context menu in the editor window. Some of the menus visible in the Menu Editor are not used by SlickEdit Core. They appear in the list due to the shared heritage with our standalone editor, SlickEdit.

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
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 **Window → SlickEdit Preferences → 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 Core 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

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Home</td>
<td>Begin line</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Back indent text to previous tab stop</td>
</tr>
<tr>
<td>Ctrl+J</td>
<td>Go to line</td>
</tr>
</tbody>
</table>

### CUA Inserting Text
## CUA Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Enter</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character typed</td>
</tr>
</tbody>
</table>

## CUA Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
</tbody>
</table>

## CUA Searching

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F</td>
<td>Find</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Replace</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Find next occurrence</td>
</tr>
<tr>
<td>Ctrl+Shift+G</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>Ctrl+I</td>
<td>Incremental search</td>
</tr>
<tr>
<td>Ctrl+Shift+I</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>
## CUA Selection

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Select all</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Select block/column</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Select line</td>
</tr>
<tr>
<td>F8</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Deselect</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Alt+=</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

## CUA Clipboard
### CUA Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+C, Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Ctrl+V, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+X, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
</tbody>
</table>

### CUA Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
</tbody>
</table>
### CUA Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Ctrl+Shift+X</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>Select word</td>
</tr>
<tr>
<td>Esc</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### CUA Files and Buffers

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2, Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>F7, Ctrl+O</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
</tbody>
</table>
### CUA Keys

<table>
<thead>
<tr>
<th>Shift+F6</th>
<th>Resync after compare mismatch</th>
</tr>
</thead>
</table>

#### CUA Windowing

<table>
<thead>
<tr>
<th>Ctrl+H</th>
<th>Split window horizontally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Tab, Ctrl+F6</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab, Ctrl+Shift+F6</td>
<td>Previous window</td>
</tr>
<tr>
<td>Ctrl+Shift+Z</td>
<td>Zoom window toggle</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Close window</td>
</tr>
<tr>
<td>Alt+F1</td>
<td>API Apprentice on current word</td>
</tr>
<tr>
<td>Alt+F2</td>
<td>Move window edge</td>
</tr>
<tr>
<td>Alt+F3</td>
<td>Create window edge</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Move</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Size</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Minimize</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

#### CUA Compiling and Programming Support

<table>
<thead>
<tr>
<th>Alt+Dot</th>
<th>List symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
</tbody>
</table>

714
### CUA Keys

<table>
<thead>
<tr>
<th>Keyboard Shortcuts</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Push a bookmark and go to the first reference to the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Find next reference</td>
</tr>
<tr>
<td>Ctrl+Shift+G</td>
<td>Find previous reference</td>
</tr>
<tr>
<td>Ctrl+M</td>
<td>Build project</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>F4, Ctrl+Shift+Down</td>
<td>Next error</td>
</tr>
<tr>
<td>Shift+F4, Ctrl+Shift+Up</td>
<td>Previous error</td>
</tr>
<tr>
<td>Ctrl+Shift+S</td>
<td>Set next error</td>
</tr>
<tr>
<td>Ctrl+Shift+E</td>
<td>List errors</td>
</tr>
<tr>
<td>Shift+F10</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Alt+1</td>
<td>Cursor to error/include file</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+M</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
<tr>
<td>Alt+F7</td>
<td>Project properties</td>
</tr>
</tbody>
</table>

### CUA Debugging

<table>
<thead>
<tr>
<th>Keyboard Shortcuts</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Start/continue debugging</td>
</tr>
</tbody>
</table>
### CUA Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### CUA Macros

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F11</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Terminate recording &amp; run last recorded macro</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with <code>get_event()</code></td>
</tr>
<tr>
<td>Key Combination</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
</tbody>
</table>

### CUA Miscellaneous

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>F10</td>
<td>Menu bar</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>SDK Help on word at cursor</td>
</tr>
<tr>
<td>Alt+F4</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Esc</td>
<td>Cancel or command line toggle</td>
</tr>
<tr>
<td>Ctrl+Z, Alt+Backspace</td>
<td>undo</td>
</tr>
<tr>
<td>Shift+F9</td>
<td>Undo with cursor motion grouping</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Redo</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+J</td>
<td>Go to bookmark</td>
</tr>
<tr>
<td>Ctrl+1..Ctrl+0</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Ctrl+Shift+N</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Ctrl+]</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
</tbody>
</table>
### Visual C++ Keys

#### Visual C++ Cursor Movement

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Home</td>
<td>Begin line</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
</tbody>
</table>
### Visual C++ Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Back indent text to previous tab stop</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Go to line, offset, bookmark, error, definition, or reference</td>
</tr>
<tr>
<td>Ctrl+J</td>
<td>Previous preprocessing condition</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Next preprocessing condition</td>
</tr>
</tbody>
</table>

### Visual C++ Inserting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Enter</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character typed</td>
</tr>
</tbody>
</table>

### Visual C++ Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Cut line</td>
</tr>
<tr>
<td>Alt+Shift+L</td>
<td>Cut sentence</td>
</tr>
</tbody>
</table>

### Visual C++ Searching

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F</td>
<td>Find</td>
</tr>
</tbody>
</table>
### Visual C++ Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+H</td>
<td>Replace</td>
</tr>
<tr>
<td>F3</td>
<td>Find next occurrence</td>
</tr>
<tr>
<td>Shift+F3</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>Ctrl+I</td>
<td>Incremental search</td>
</tr>
<tr>
<td>Ctrl+Shift+I</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>

### Visual C++ Selection

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Select all</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Select block/column</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Select line</td>
</tr>
<tr>
<td>F8</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Ctrl+Shift+J</td>
<td>Select previous preprocessing condition</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Select next preprocessing condition</td>
</tr>
<tr>
<td>Alt+U</td>
<td>Deselect</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Alt+=</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
</tbody>
</table>
### Visual C++ Keys

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### Visual C++ Clipboard

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+C, Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Ctrl+V, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+X, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Alt+Shift+L</td>
<td>Cut sentence</td>
</tr>
</tbody>
</table>

### Visual C++ Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
</tbody>
</table>
## Visual C++ Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Delete line</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Alt+N</td>
<td>Insert buffer name</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

## Visual C++ Files and Buffers
### Visual C++ Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2, Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+O</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Resync after compare mismatch</td>
</tr>
</tbody>
</table>

### Visual C++ Windowing

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Tab, Ctrl+F6</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab</td>
<td>Previous window</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Close window</td>
</tr>
</tbody>
</table>

### Visual C++ Compiling and Programming Support

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Alt+T, Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Shift+Space, Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>F12, Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Shift+F12, Ctrl+/</td>
<td>Push a bookmark and go to the first reference to the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
</tbody>
</table>
### Visual C++ Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3</td>
<td>Find next reference</td>
</tr>
<tr>
<td>Shift+F3</td>
<td>Find previous reference</td>
</tr>
<tr>
<td>F4, Ctrl+Shift+Down</td>
<td>Next error</td>
</tr>
<tr>
<td>Shift+F4, Ctrl+Shift+Up</td>
<td>Previous error</td>
</tr>
<tr>
<td>F7</td>
<td>Build project</td>
</tr>
<tr>
<td>Shift+F10</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+G</td>
<td>Cursor to error/include file</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+M</td>
<td>Start concurrent process</td>
</tr>
</tbody>
</table>

### Visual C++ Debugging

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Start/continue debugging</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
</tbody>
</table>
### Visual C++ Keys

<table>
<thead>
<tr>
<th>Keyboard Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### Visual C++ Macros

<table>
<thead>
<tr>
<th>Keyboard Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+R</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Terminate recording &amp; run last recorded macro</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Make and load current buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Visual C++ Miscellaneous

<table>
<thead>
<tr>
<th>Keyboard Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+R</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Menu bar</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>SDK Help on word at cursor</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete next word/variable</td>
</tr>
</tbody>
</table>
### SlickEdit® Core Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esc</td>
<td>Cancel or command line toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+R</td>
<td>Undo</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Redo</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Alt+Shift+F2</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Activates search history</td>
</tr>
<tr>
<td>Alt+F5</td>
<td>Restore MDI window</td>
</tr>
<tr>
<td>Alt+F7</td>
<td>Project properties</td>
</tr>
<tr>
<td>Alt+F8</td>
<td>Beautify selection</td>
</tr>
<tr>
<td>Alt+F10</td>
<td>Maximize MDI window</td>
</tr>
<tr>
<td>Ctrl+Up</td>
<td>Scroll up</td>
</tr>
<tr>
<td>Ctrl+Down</td>
<td>Scroll down</td>
</tr>
<tr>
<td>Ctrl+Shift+T</td>
<td>Transpose words</td>
</tr>
<tr>
<td>Alt+Shift+T</td>
<td>Transpose lines</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Cursor Movement

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow, Ctrl+J</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow, Ctrl+L</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow, Ctrl+I</td>
<td>Cursor up</td>
</tr>
<tr>
<td><strong>SlickEdit® Core Keys</strong></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>Down arrow, Ctrl+K</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+Home, Ctrl+X Ctrl+U</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+End, Ctrl+X Ctrl+J</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Home, Ctrl+U</td>
<td>Begin line</td>
</tr>
<tr>
<td>End, Ctrl+O</td>
<td>End line</td>
</tr>
<tr>
<td>PgUp, Ctrl+P</td>
<td>Page up</td>
</tr>
<tr>
<td>PgDn, Ctrl+N</td>
<td>Page down</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Tab</td>
<td>Next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Previous tab stop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SlickEdit® Core Inserting Text</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins, Ctrl+X Ctrl+O</td>
</tr>
<tr>
<td>Enter</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
</tr>
<tr>
<td>Shift+Enter</td>
</tr>
<tr>
<td>Alt+N</td>
</tr>
<tr>
<td>Shift+Space</td>
</tr>
<tr>
<td>Ctrl+X Tab</td>
</tr>
</tbody>
</table>
### SlickEdit® Core Keys

#### Ctrl+Q
Quote next character typed

#### Alt+S
Split line at cursor

### SlickEdit® Core Deleting Text

<table>
<thead>
<tr>
<th>Ctrl+D, Ctrl+D</th>
<th>Delete char under cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Alt+W</td>
<td>Cut word</td>
</tr>
<tr>
<td>Alt+J</td>
<td>Join line to cursor</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Searching

<table>
<thead>
<tr>
<th>Ctrl+F</th>
<th>Find next occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+F</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Incremental search</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Z</td>
<td>Resume search and replace(Supports command line replace command only)</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+R</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search and replace</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Selection

<table>
<thead>
<tr>
<th>Alt+B</th>
<th>Select block/column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Alt+Z</td>
<td>Select character/stream</td>
</tr>
</tbody>
</table>

728
<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+U</td>
<td>Deselect</td>
</tr>
<tr>
<td>Alt+C</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Alt+K</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Alt+M</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Alt+F</td>
<td>Fill selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Ctrl+F3</td>
<td>Uppercase selection</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Lowercase selection</td>
</tr>
<tr>
<td>Alt+A</td>
<td>Move/overlay block</td>
</tr>
<tr>
<td>Alt+O</td>
<td>Overlay block selection</td>
</tr>
<tr>
<td>Alt+E</td>
<td>Go to end of selection</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Go to beginning of selection</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+P</td>
<td>Reflow selection</td>
</tr>
<tr>
<td>Alt+=</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>
SlickEdit® Core Clipboard

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+V, Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+W</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Y</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Ctrl+Y, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Alt+W</td>
<td>Cut word</td>
</tr>
<tr>
<td>Alt+K, Shift+Del</td>
<td>Cut selection</td>
</tr>
</tbody>
</table>

SlickEdit® Core 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](#)): |

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+A..Alt+Z</td>
<td>Taken over by dialog manager for selecting controls</td>
</tr>
</tbody>
</table>

SlickEdit® Core Command Line Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esc, Ctrl+A</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
</tbody>
</table>
### SlickEdit® Core Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+F2</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Alt+W</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Alt+V</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+W</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Y</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Alt+N</td>
<td>Insert buffer name</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Expand alias at cursor. Use alias command to define aliases.</td>
</tr>
<tr>
<td>Up arrow, Ctrl+I</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Down arrow, Ctrl+K</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Files and Buffers
### SlickEdit® Core Keys

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2, Ctrl+X Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>F3, Ctrl+X 'K'</td>
<td>Quit current buffer</td>
</tr>
<tr>
<td>F8, Ctrl+B</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Ctrl+F8, Ctrl+V</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>F4</td>
<td>Save and quit current buffer</td>
</tr>
<tr>
<td>F7</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>Ctrl+X 'B'</td>
<td>Find buffer</td>
</tr>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Resync after compare mismatch</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Windowing

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X '2'</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Ctrl+W, Ctrl+Tab</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab</td>
<td>Previous window</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>Zoom window toggle</td>
</tr>
<tr>
<td>Ctrl+X '1'</td>
<td>One window</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Compiling and Programming Support

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
</tbody>
</table>
### SlickEdit® Core Keys

<table>
<thead>
<tr>
<th>shortcut</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+H, Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Push a bookmark and go to the first reference to the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+H, Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+F</td>
<td>Find next reference</td>
</tr>
<tr>
<td>Ctrl+Shift+F</td>
<td>Find previous reference</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+X 'M'</td>
<td>Build project</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Alt+F10, Ctrl+X Ctrl+N</td>
<td>Next error</td>
</tr>
<tr>
<td>Ctrl+X 'N'</td>
<td>Set next error</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Alt+1</td>
<td>Cursor to error/include file</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+L</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Stop concurrent process</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+M</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Debugging

<table>
<thead>
<tr>
<th>shortcut</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
</tbody>
</table>
### SlickEdit® Core Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Macros

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+R</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Terminate recording &amp; run last recorded macro</td>
</tr>
<tr>
<td>Ctrl+Shift+T,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+L</td>
<td>Make and load current buffer</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
</tbody>
</table>
### SlickEdit® Core Keys

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X 'E'</td>
<td>Run last recorded macro</td>
</tr>
<tr>
<td>Ctrl+X '('</td>
<td>Start recording macro</td>
</tr>
<tr>
<td>Ctrl+X ')'</td>
<td>End recording macro</td>
</tr>
</tbody>
</table>

### SlickEdit® Core Miscellaneous

<table>
<thead>
<tr>
<th>Function</th>
<th>Keyboard Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help menu</td>
<td>F1</td>
</tr>
<tr>
<td>Configuration menu</td>
<td>F5</td>
</tr>
<tr>
<td>Menu bar</td>
<td>F10</td>
</tr>
<tr>
<td>Safe exit</td>
<td>Alt+X, Ctrl+X Ctrl+C</td>
</tr>
<tr>
<td>Complete previous word/variable</td>
<td>Ctrl+Shift+Comma</td>
</tr>
<tr>
<td>Complete next word/variable</td>
<td>Ctrl+Shift+Dot</td>
</tr>
<tr>
<td>Complete more</td>
<td>Ctrl+Shift+Space</td>
</tr>
<tr>
<td>Cancel</td>
<td>Esc, Ctrl+G</td>
</tr>
<tr>
<td>Undo</td>
<td>F9</td>
</tr>
<tr>
<td>Undo with cursor motion grouping</td>
<td>Ctrl+F9</td>
</tr>
<tr>
<td>Redo</td>
<td>Shift+F9, Ctrl+X R</td>
</tr>
<tr>
<td>Hex display toggle</td>
<td>Ctrl+Shift+H</td>
</tr>
<tr>
<td>Upcase word</td>
<td>Ctrl+F1</td>
</tr>
<tr>
<td>Lowcase word</td>
<td>Ctrl+F2</td>
</tr>
<tr>
<td>Center line</td>
<td>Shift+F5</td>
</tr>
<tr>
<td>Scroll up</td>
<td>Shift+F1</td>
</tr>
<tr>
<td>Scroll down</td>
<td>Shift+F2</td>
</tr>
<tr>
<td>Scroll left</td>
<td>Shift+F3</td>
</tr>
</tbody>
</table>
## Brief Keys

### Brief Cursor Movement

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+PgUp, Home(3x)</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+PgDn, End(3x)</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Home</td>
<td>Begin line</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
<tr>
<td>Alt+G</td>
<td>Go to line</td>
</tr>
<tr>
<td>Alt+J</td>
<td>Go to bookmark</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
</tbody>
</table>

### Brief Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+F4</td>
<td>Scroll right</td>
</tr>
<tr>
<td>Alt+S</td>
<td>Split line</td>
</tr>
<tr>
<td>Alt+J</td>
<td>Join line</td>
</tr>
<tr>
<td>Alt+P</td>
<td>Reflow paragraph</td>
</tr>
<tr>
<td>Ctrl+Shift+N</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Alt+T</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Alt+R</td>
<td>Fundamental mode for next key press</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+E</td>
<td>OS Shell</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+D</td>
<td>Alias change directory</td>
</tr>
</tbody>
</table>
# Brief Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+Home, Home(2x)</td>
<td>Top of window</td>
</tr>
<tr>
<td>Ctrl+End, End(2x)</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Tab</td>
<td>Insert tab or next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Previous tab stop</td>
</tr>
<tr>
<td>Shift+Home</td>
<td>Left side of window</td>
</tr>
<tr>
<td>Shift+End</td>
<td>Right side of window</td>
</tr>
</tbody>
</table>

## Brief Inserting Text

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+I</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Enter</td>
<td>Maybe split insert line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>No split insert line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Alt+Q</td>
<td>Quote next character typed</td>
</tr>
<tr>
<td>Ctrl+A</td>
<td>Insert buffer name</td>
</tr>
</tbody>
</table>

## Brief Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del</td>
<td>Delete char or selection</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Brief Keys</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>PadMinus</td>
<td>Cut line or selection</td>
</tr>
<tr>
<td>Alt+K</td>
<td>Delete to end line</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Delete word</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Delete line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Delete previous word</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Brief Searching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+S, F5</td>
</tr>
<tr>
<td>Shift+F5</td>
</tr>
<tr>
<td>Alt+F5, Ctrl+Shift+F5</td>
</tr>
<tr>
<td>Ctrl+F5</td>
</tr>
<tr>
<td>Ctrl+F6</td>
</tr>
<tr>
<td>Alt+T, F6</td>
</tr>
<tr>
<td>Shift+F6</td>
</tr>
<tr>
<td>Alt+F6, Ctrl+Shift+F6</td>
</tr>
<tr>
<td>Ctrl+S</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Brief Clipboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+E</td>
</tr>
<tr>
<td>Ctrl+L</td>
</tr>
<tr>
<td>Ins, Ctrl+Y, Shift+Ins</td>
</tr>
<tr>
<td>PadPlus, Ctrl+Ins</td>
</tr>
</tbody>
</table>
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):

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+A..Alt+Z</td>
<td>Taken over by dialog manager for selecting controls</td>
</tr>
</tbody>
</table>

**Brief Command Line Keys**

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F10, Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Alt+I</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Alt+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ins, Ctrl+Y</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Paste recent clipboard</td>
</tr>
<tr>
<td>Ctrl+A</td>
<td>Insert buffer name</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Expand alias at cursor. Use alias command to define aliases.</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>Upcase word</td>
</tr>
</tbody>
</table>
### Brief Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F2</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Capitalize word</td>
</tr>
<tr>
<td>Alt+K</td>
<td>Delete to end line</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Delete word</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Delete line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Delete prev word</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### Brief Files and Buffers

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+W</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+Minus</td>
<td>Quit current buffer</td>
</tr>
<tr>
<td>Alt+E</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Alt+X</td>
<td>Safe exit w/write all option</td>
</tr>
<tr>
<td>Alt+O</td>
<td>Change buffer name</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>File compare</td>
</tr>
<tr>
<td>Shift+F10</td>
<td>Resync after compare mismatch</td>
</tr>
<tr>
<td>Alt+B</td>
<td>List buffers</td>
</tr>
</tbody>
</table>
### Brief Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+N</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Alt+Minus</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Save all buffers and exit</td>
</tr>
<tr>
<td>Alt+R</td>
<td>Read file</td>
</tr>
</tbody>
</table>

### Brief Windowing

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Change window</td>
</tr>
<tr>
<td>F2</td>
<td>Move window edge</td>
</tr>
<tr>
<td>F3</td>
<td>Create window edge</td>
</tr>
<tr>
<td>F4</td>
<td>Delete window edge</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>Zoom window toggle</td>
</tr>
<tr>
<td>Ctrl+J</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Ctrl+W, Ctrl+Tab</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab</td>
<td>Previous window</td>
</tr>
<tr>
<td>Shift+Left</td>
<td>Switch to left window</td>
</tr>
<tr>
<td>Shift+Right</td>
<td>Switch to right window</td>
</tr>
<tr>
<td>Shift+Up</td>
<td>Switch to window above</td>
</tr>
<tr>
<td>Shift+Down</td>
<td>Switch to window below</td>
</tr>
</tbody>
</table>

### Brief Compiling and Programming Support

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
</tbody>
</table>
### Brief Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Push a bookmark and go to the first reference to the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Alt+F9, Ctrl+Shift+F9</td>
<td>Build project</td>
</tr>
<tr>
<td>Alt+F10, Ctrl+Shift+F10</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Next error</td>
</tr>
<tr>
<td>F9</td>
<td>Make and load macro</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>List errors</td>
</tr>
<tr>
<td>Ctrl+O</td>
<td>Stop concurrent process</td>
</tr>
<tr>
<td>Ctrl+I</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>List buffer procedures</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
</tbody>
</table>

### Brief Debugging

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F9</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
</tbody>
</table>
### Brief Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### Brief Selection

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+C</td>
<td>Select block/column</td>
</tr>
<tr>
<td>Alt+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Alt+M</td>
<td>Inclusive char selection</td>
</tr>
<tr>
<td>Alt+A</td>
<td>Non-inclusive char selection</td>
</tr>
<tr>
<td>Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>PadMinus</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Go to beginning of selection</td>
</tr>
<tr>
<td>Ctrl+F3</td>
<td>Upcase selection</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Lowcase selection</td>
</tr>
<tr>
<td>Alt+=</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Alt+F</td>
<td>Fill selection</td>
</tr>
<tr>
<td><strong>Brief Keys</strong></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Brief Macros</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>F7</td>
</tr>
<tr>
<td>Shift+F7</td>
</tr>
<tr>
<td>F9</td>
</tr>
<tr>
<td>F8</td>
</tr>
<tr>
<td>Alt+F7, Ctrl+Shift+F7</td>
</tr>
<tr>
<td>Alt+F8, Ctrl+Shift+F8</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
</tr>
<tr>
<td>Ctrl+Break</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Brief Miscellaneous</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>744</td>
</tr>
<tr>
<td>Shortcuts</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>Alt</td>
</tr>
<tr>
<td>Alt+H</td>
</tr>
<tr>
<td>Ctrl+F</td>
</tr>
<tr>
<td>Alt+X</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
</tr>
<tr>
<td>Ctrl+X</td>
</tr>
<tr>
<td>Esc</td>
</tr>
<tr>
<td>Alt+H</td>
</tr>
<tr>
<td>Ctrl+F</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
</tr>
<tr>
<td>Ctrl+T</td>
</tr>
<tr>
<td>Ctrl+B</td>
</tr>
<tr>
<td>Alt+U, PadStar</td>
</tr>
<tr>
<td>Ctrl+F9</td>
</tr>
<tr>
<td>Ctrl+U</td>
</tr>
<tr>
<td>Shift+F1</td>
</tr>
<tr>
<td>Ctrl+D, Shift+F2</td>
</tr>
<tr>
<td>Shift+F3</td>
</tr>
<tr>
<td>Shift+F4</td>
</tr>
<tr>
<td>Ctrl+F1</td>
</tr>
<tr>
<td>Ctrl+F2</td>
</tr>
</tbody>
</table>
### Epsilon Keys

#### Epsilon Cursor Movement

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A, Alt+Left</td>
<td>Begin line</td>
</tr>
<tr>
<td>Ctrl+E, Alt+Right</td>
<td>End of line</td>
</tr>
<tr>
<td>Left arrow, Ctrl+B</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow, Ctrl+F</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Ctrl+N, Down arrow</td>
<td>Next line</td>
</tr>
<tr>
<td>Ctrl+P, Up arrow</td>
<td>Previous line</td>
</tr>
<tr>
<td>Ctrl+V, PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>Alt+V, PgUp</td>
<td>Page up</td>
</tr>
</tbody>
</table>
### Epsilon Keys

<table>
<thead>
<tr>
<th>Keyboard Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Home, Alt+&lt;</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+End, Alt+&gt;</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Ctrl+Left, Alt+B</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right, Alt+F</td>
<td>Next word</td>
</tr>
<tr>
<td>Home</td>
<td>Beginning of window</td>
</tr>
<tr>
<td>End</td>
<td>End of window</td>
</tr>
<tr>
<td>Ctrl+Alt+B</td>
<td>Previous level</td>
</tr>
<tr>
<td>Ctrl+Alt+F</td>
<td>Next level</td>
</tr>
<tr>
<td>Alt+)</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Alt+[, Alt+Up</td>
<td>Previous paragraph</td>
</tr>
<tr>
<td>Alt+], Alt+Down</td>
<td>Next paragraph</td>
</tr>
<tr>
<td>Alt+M</td>
<td>First non blank</td>
</tr>
<tr>
<td>Alt+A, Ctrl+Up</td>
<td>Previous sentence</td>
</tr>
<tr>
<td>Alt+E, Ctrl+Down</td>
<td>Next sentence</td>
</tr>
<tr>
<td>Ctrl+X, ‘G’</td>
<td>Go to line</td>
</tr>
</tbody>
</table>

### Epsilon Inserting Text

<table>
<thead>
<tr>
<th>Keyboard Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Tab, Ctrl+I</td>
<td>Indent to previous line or insert tab character</td>
</tr>
<tr>
<td>Enter, Ctrl+M, Ctrl+J</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
</tbody>
</table>
Epsilon Keys

<table>
<thead>
<tr>
<th>Shift+Space</th>
<th>Insert a space (no syntax expansion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+O</td>
<td>Split line</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Insert literal</td>
</tr>
</tbody>
</table>

Epsilon Deleting Text

<table>
<thead>
<tr>
<th>Alt+\</th>
<th>Delete horizontal space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X Ctrl+O</td>
<td>Delete blank lines</td>
</tr>
<tr>
<td>Alt+Del</td>
<td>Cut previous level</td>
</tr>
<tr>
<td>Ctrl+Alt+K</td>
<td>Cut level</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Cut word</td>
</tr>
<tr>
<td>Alt+K</td>
<td>Cut sentence</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Del, Ctrl+D</td>
<td>Delete character under cursor</td>
</tr>
<tr>
<td>Backspace, Ctrl+H</td>
<td>Delete character before cursor</td>
</tr>
</tbody>
</table>

Epsilon Searching

<table>
<thead>
<tr>
<th>Ctrl+S</th>
<th>Incremental search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+R</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+S</td>
<td>Regular expression search</td>
</tr>
<tr>
<td>Ctrl+Alt+R</td>
<td>Reverse regular expression search</td>
</tr>
<tr>
<td>Alt+%</td>
<td>Search and replace with prompting</td>
</tr>
</tbody>
</table>
### Epsilon Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+&amp;</td>
<td>Search and replace without prompting</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>

### Epsilon Selection

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+2</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Alt+@</td>
<td>Select block/column</td>
</tr>
<tr>
<td>Alt+H</td>
<td>Select paragraph</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+X</td>
<td>Exchange point and start selection</td>
</tr>
<tr>
<td>Ctrl+X ‘W’</td>
<td>Write selection to file</td>
</tr>
<tr>
<td>Ctrl+Alt+\</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Alt+I</td>
<td>Tabify selection</td>
</tr>
<tr>
<td>Ctrl+X Alt+I</td>
<td>Untabify selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift block selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift block selection right</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### Epsilon Clipboard
### Epsilon Keys

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+W, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Y, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Select clipboard to insert</td>
</tr>
<tr>
<td>Alt+W, Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+A</td>
<td>Copy word</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Cut word</td>
</tr>
<tr>
<td>Alt+K</td>
<td>Cut sentence</td>
</tr>
<tr>
<td>Alt+Del</td>
<td>Cut previous level</td>
</tr>
<tr>
<td>Ctrl+Alt+K</td>
<td>Cut level</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Ctrl+Alt+W</td>
<td>Append next clipboard</td>
</tr>
</tbody>
</table>

### Epsilon Files and Buffers

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+X 'K'</td>
<td>Quit buffer</td>
</tr>
<tr>
<td>Ctrl+X 'B'</td>
<td>Select buffer</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+F</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+W</td>
<td>Save and rename buffer</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Write buffer to file</td>
</tr>
<tr>
<td>Ctrl+F2, Ctrl+X 'C'</td>
<td>Compare windows</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+B, Ctrl+X Alt+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+V</td>
<td>Replace buffer with a file on disk</td>
</tr>
</tbody>
</table>
### Epsilon Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X 'I'</td>
<td>Insert file</td>
</tr>
<tr>
<td>Ctrl+X 'W'</td>
<td>Write selection to file</td>
</tr>
<tr>
<td>Ctrl+X 'D'</td>
<td>Directory edit mode</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Resync after compare mismatch</td>
</tr>
</tbody>
</table>

### Epsilon Windowing

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X '2'</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Ctrl+X '1'</td>
<td>One window</td>
</tr>
<tr>
<td>Ctrl+X 'O', Alt+PgDn</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+X 'P', Alt+PgUp</td>
<td>Prev window</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+D</td>
<td>Delete window</td>
</tr>
<tr>
<td>Home</td>
<td>Cursor to beginning of window</td>
</tr>
<tr>
<td>End</td>
<td>Cursor to end of window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Shrink window</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Expand window</td>
</tr>
</tbody>
</table>

### Epsilon Compiling and Programming Support

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
</tbody>
</table>
## Epsilon Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+X 'M'</td>
<td>Build project</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+N, Alt+F10</td>
<td>Next error</td>
</tr>
<tr>
<td>Alt+F6</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Stop concurrent process</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+M</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+X Comma, Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+H, Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+X Dot</td>
<td>Go to definition</td>
</tr>
<tr>
<td>Ctrl+X Alt+Comma</td>
<td>Context Tagging® - Tag Files dialog box</td>
</tr>
<tr>
<td>F3</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
</tbody>
</table>

## Epsilon Debugging

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Start/continue debugging</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
</tbody>
</table>

752
## Epsilon Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

## Epsilon Macros

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X 'E'</td>
<td>Run last recorded macro</td>
</tr>
<tr>
<td>F3</td>
<td>Make and load current buffer</td>
</tr>
<tr>
<td>Ctrl+X Alt+N</td>
<td>Save last recorded macro</td>
</tr>
<tr>
<td>Ctrl+X '('</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+X ')'</td>
<td>End macro recording</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 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):

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+A..Alt+Z</td>
<td>Taken over by dialog manager for selecting controls</td>
</tr>
</tbody>
</table>

### Epsilon Command Line Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2, Alt+X</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Delete character</td>
</tr>
<tr>
<td>Ctrl+H, Backspace</td>
<td>Delete previous character</td>
</tr>
<tr>
<td>Tab, Ctrl+I</td>
<td>Insert tab character</td>
</tr>
<tr>
<td>Ctrl+J, Enter, Ctrl+M</td>
<td>ENTER argument</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert toggle</td>
</tr>
<tr>
<td>Alt+\</td>
<td>Delete horizontal space</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Expand alias at cursor. Use alias command to define aliases.</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Cut word</td>
</tr>
</tbody>
</table>
## Epsilon Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+W, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Y, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Paste recent clipboard</td>
</tr>
<tr>
<td>Ctrl+Alt+W</td>
<td>Append next clipboard</td>
</tr>
<tr>
<td>Alt+C</td>
<td>Capitalize word</td>
</tr>
<tr>
<td>Alt+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Alt+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Transpose characters</td>
</tr>
<tr>
<td>Alt+T</td>
<td>Transpose words</td>
</tr>
<tr>
<td>Ctrl+N, Down</td>
<td>Retrieve previous argument (cmdline only)</td>
</tr>
<tr>
<td>Ctrl+P, Up</td>
<td>Retrieve next argument (cmdline only)</td>
</tr>
<tr>
<td>F2, Alt+X</td>
<td>Complete and enter argument (cmdline only)</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Ctrl+H, Backspace</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Tab, Ctrl+I</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+J, Enter, Ctrl+M</td>
<td>Select word</td>
</tr>
<tr>
<td>Space</td>
<td>Select line</td>
</tr>
</tbody>
</table>

## Epsilon Miscellaneous

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1, Alt+?, Ctrl+-</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Alt+P</td>
<td>Configuration menu</td>
</tr>
<tr>
<td>F10</td>
<td>Main menu for mode</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+C</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Cancel</td>
</tr>
<tr>
<td>F9, Ctrl+X 'U'</td>
<td>Undo</td>
</tr>
<tr>
<td>Shift+F9, Ctrl+X 'R'</td>
<td>Redo</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Undo w/cursor grouping</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Transpose characters</td>
</tr>
<tr>
<td>Alt+T</td>
<td>Transpose words</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+T</td>
<td>Transpose lines</td>
</tr>
<tr>
<td>Alt+U</td>
<td>Uppercase word</td>
</tr>
<tr>
<td>Alt+L</td>
<td>Lowercase word</td>
</tr>
<tr>
<td>Alt+C</td>
<td>Capitalize word</td>
</tr>
<tr>
<td>Ctrl+X 'F'</td>
<td>Set margins</td>
</tr>
<tr>
<td>Alt+Q</td>
<td>Reflow paragraph</td>
</tr>
<tr>
<td>Alt+S</td>
<td>Center line within margins</td>
</tr>
<tr>
<td>Alt+)</td>
<td>Find matching start paren</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Center line within window</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Ctrl+X /</td>
<td>Alias change directory</td>
</tr>
<tr>
<td>F8, Ctrl+F8</td>
<td>Set macro variable</td>
</tr>
<tr>
<td>F4</td>
<td>Bind to key</td>
</tr>
</tbody>
</table>
### Epsilon Argument/Repeating a Key

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+U</td>
<td>Select number of times to invoke a command. Sets argument-count.</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cuts argument-count complete lines</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+O</td>
<td>Change number of blank lines at or before cursor to argument-count</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+I, Ctrl+X Tab</td>
<td>Indent or unindent selected lines by argument-count characters</td>
</tr>
<tr>
<td>Alt+0..Alt+9</td>
<td>Select argument-count 0..9</td>
</tr>
</tbody>
</table>

### Vim Keys

#### Vim Cursor Movement

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow, Ctrl+J</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow, Ctrl+L</td>
<td>Cursor right</td>
</tr>
</tbody>
</table>
## Vim Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up arrow, Ctrl+I</td>
<td>Cursor up</td>
</tr>
<tr>
<td>Down arrow, Ctrl+K</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+Home, Ctrl+X Ctrl+U</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+End, Ctrl+X Ctrl+J</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Home</td>
<td>Begin line</td>
</tr>
<tr>
<td>End, Ctrl+O</td>
<td>End line</td>
</tr>
<tr>
<td>PgUp, Ctrl+B</td>
<td>Page up</td>
</tr>
<tr>
<td>PgDn, Ctrl+F</td>
<td>Page down</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Indent to previous tab stop</td>
</tr>
</tbody>
</table>

## Vim Cursor Movement - Command Mode Only

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gj</td>
<td>Down screen line (different when lines wrap)</td>
</tr>
<tr>
<td>gk</td>
<td>Up screen line (different when lines wrap)</td>
</tr>
<tr>
<td>-</td>
<td>Begin previous line</td>
</tr>
<tr>
<td>^</td>
<td>Begin text</td>
</tr>
<tr>
<td>0</td>
<td>Begin line</td>
</tr>
<tr>
<td>$</td>
<td>End line</td>
</tr>
<tr>
<td>G, gg</td>
<td>Go to line</td>
</tr>
<tr>
<td>Key</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><code>l</code></td>
<td>Go to column</td>
</tr>
<tr>
<td><code>w</code></td>
<td>Next word</td>
</tr>
<tr>
<td><code>W</code></td>
<td>Next non-white space word</td>
</tr>
<tr>
<td><code>b</code></td>
<td>Previous word</td>
</tr>
<tr>
<td><code>B</code></td>
<td>Previous non-white space word</td>
</tr>
<tr>
<td><code>ge</code></td>
<td>Backward to end of word</td>
</tr>
<tr>
<td><code>gE</code></td>
<td>Backward to end of non-white space word</td>
</tr>
<tr>
<td><code>e</code></td>
<td>End of word</td>
</tr>
<tr>
<td><code>E</code></td>
<td>End of non-white space word</td>
</tr>
<tr>
<td><code>( </code></td>
<td>Previous sentence</td>
</tr>
<tr>
<td><code>)</code></td>
<td>Next sentence</td>
</tr>
<tr>
<td><code>{ </code></td>
<td>Previous paragraph</td>
</tr>
<tr>
<td><code>)</code></td>
<td>Next paragraph</td>
</tr>
<tr>
<td><code>[[</code></td>
<td>Previous section</td>
</tr>
<tr>
<td><code>]]</code></td>
<td>Next section</td>
</tr>
<tr>
<td>`%</td>
<td>Find matching paren</td>
</tr>
<tr>
<td><code>N %</code></td>
<td>Move N % down a buffer</td>
</tr>
<tr>
<td><code>H</code></td>
<td>Move to upper-left corner of window</td>
</tr>
<tr>
<td><code>M</code></td>
<td>Move to middle of window</td>
</tr>
<tr>
<td><code>L</code></td>
<td>Move to lower-left corner of window</td>
</tr>
<tr>
<td><code>' </code></td>
<td>Jump to bookmarked line</td>
</tr>
<tr>
<td><code>~</code></td>
<td>Jump to bookmarked column</td>
</tr>
<tr>
<td><code>gm</code></td>
<td>Move to middle of window on the current line</td>
</tr>
</tbody>
</table>
Vim Keys

Vim Cursor Movement - Visual Mode Only

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aw</td>
<td>Select a word</td>
</tr>
<tr>
<td>iw</td>
<td>Select inner word</td>
</tr>
<tr>
<td>aW</td>
<td>Select</td>
</tr>
<tr>
<td>iW</td>
<td>Select inner</td>
</tr>
<tr>
<td>as</td>
<td>Select a sentence</td>
</tr>
<tr>
<td>is</td>
<td>Select inner sentence</td>
</tr>
<tr>
<td>ap</td>
<td>Select a paragraph</td>
</tr>
<tr>
<td>ip</td>
<td>Select inner paragraph</td>
</tr>
<tr>
<td>ab</td>
<td>Select a block</td>
</tr>
<tr>
<td>ib</td>
<td>Select inner block</td>
</tr>
<tr>
<td>aB</td>
<td>Select a Block</td>
</tr>
<tr>
<td>iB</td>
<td>Select inner Block</td>
</tr>
</tbody>
</table>

Vim Inserting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+V</td>
<td>Quote next character typed</td>
</tr>
<tr>
<td>Ctrl+X Tab</td>
<td>Move text tab</td>
</tr>
<tr>
<td>Alt+N</td>
<td>Insert buffer name</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
</tbody>
</table>

Vim Inserting Text - Command Mode Only

760
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Insert text</td>
</tr>
<tr>
<td>I</td>
<td>Insert text at beginning of line</td>
</tr>
<tr>
<td>a</td>
<td>Append text</td>
</tr>
<tr>
<td>A</td>
<td>Append text after end of line</td>
</tr>
<tr>
<td>o</td>
<td>Insert text below current line</td>
</tr>
<tr>
<td>O</td>
<td>Insert text above current line</td>
</tr>
</tbody>
</table>

### Vim Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del</td>
<td>Delete character under cursor</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Alt+J</td>
<td>Join line to cursor</td>
</tr>
</tbody>
</table>

### Vim Deleting Text - Command Mode Only

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Delete character under cursor</td>
</tr>
<tr>
<td>(visual) x</td>
<td>Delete selection</td>
</tr>
<tr>
<td>X</td>
<td>Delete character before cursor</td>
</tr>
<tr>
<td>(visual) d</td>
<td>Delete selection</td>
</tr>
<tr>
<td>d</td>
<td>Delete text</td>
</tr>
<tr>
<td>D</td>
<td>Delete to end of line</td>
</tr>
</tbody>
</table>

### Vim Searching

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+R</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Incremental search</td>
</tr>
</tbody>
</table>
### Vim Keys

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X Ctrl+Z</td>
<td>Resume search &amp; replace (Command line only)</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+R</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>

### Vim Searching - Command Mode Only

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Search forward (accommodates multipliers)</td>
</tr>
<tr>
<td>?</td>
<td>Search backward (accommodates multipliers)</td>
</tr>
<tr>
<td>n</td>
<td>Forward repeat last search</td>
</tr>
<tr>
<td>N</td>
<td>Backward repeat last search</td>
</tr>
<tr>
<td>f</td>
<td>Forward character search</td>
</tr>
<tr>
<td>F</td>
<td>Backward character search</td>
</tr>
<tr>
<td>t</td>
<td>Move cursor up to character</td>
</tr>
<tr>
<td>T</td>
<td>Move cursor backward after character</td>
</tr>
<tr>
<td>;</td>
<td>Repeat character search</td>
</tr>
<tr>
<td>,</td>
<td>Reverse repeat character search</td>
</tr>
<tr>
<td>m</td>
<td>Set bookmark</td>
</tr>
</tbody>
</table>

### Vim Selection

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Z</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Alt+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift block selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift block selection right</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
</tbody>
</table>
### Vim Keys

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### Vim Selection - Command Mode Only

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Character visual mode</td>
</tr>
<tr>
<td>V</td>
<td>Line visual mode</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Block visual mode</td>
</tr>
<tr>
<td>o</td>
<td>Move cursor to beginning (or end) of selection</td>
</tr>
</tbody>
</table>

### Vim Clipboard

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+W</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Y</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Ctrl+Y, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Alt+K, Shift+Del</td>
<td>Cut selection</td>
</tr>
</tbody>
</table>

### Vim Clipboard - Command Mode Only
### Vim Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Paste text after cursor</td>
</tr>
<tr>
<td>]p</td>
<td>Paste text after cursor (adjust indent)</td>
</tr>
<tr>
<td>gp</td>
<td>Paste text after cursor (leave cursor after new text)</td>
</tr>
<tr>
<td>(visual) p</td>
<td>Paste clipboard contents over selection</td>
</tr>
<tr>
<td>P</td>
<td>Paste text before cursor</td>
</tr>
<tr>
<td>[p</td>
<td>Paste text before cursor (adjust indent)</td>
</tr>
<tr>
<td>gP</td>
<td>Paste text before cursor (leave cursor after new text)</td>
</tr>
<tr>
<td>y</td>
<td>Copy text to clipboard</td>
</tr>
<tr>
<td>(visual) y</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Y</td>
<td>Copy line to clipboard</td>
</tr>
</tbody>
</table>

### 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**):

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+A..Alt+Z</td>
<td>Taken over by dialog manager for selecting controls</td>
</tr>
</tbody>
</table>

### Vim Command Line and Text Box Editing - Command Mode Only

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
</tbody>
</table>

764
## Vim Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+F2</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+W</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Y</td>
<td>Paste recent clipboard</td>
</tr>
<tr>
<td>Alt+N</td>
<td>Insert buffer name</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Expand alias at cursor. Use the <code>alias</code> command to define aliases or the Ex command <code>:abbr</code>.</td>
</tr>
<tr>
<td>Up arrow, Ctrl+I</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Down arrow, Ctrl+K</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### Vim Files and Buffers

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2, Ctrl+X Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>F3, Ctrl+X K</td>
<td>Quit current buffer</td>
</tr>
</tbody>
</table>
### Vim Keys

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F8</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>F4</td>
<td>Save and quit current buffer</td>
</tr>
<tr>
<td>F7, Ctrl+X Ctrl+F</td>
<td>Open a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>Ctrl+X B</td>
<td>Find buffer</td>
</tr>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Resync after compare mismatch</td>
</tr>
</tbody>
</table>

### Vim Files and Buffers - Command Mode Only

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:w</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>:q</td>
<td>Quit current buffer</td>
</tr>
<tr>
<td>:wq</td>
<td>Save and quit current buffer</td>
</tr>
<tr>
<td>:q!</td>
<td>Quit current buffer without saving</td>
</tr>
<tr>
<td>:wa[ll]</td>
<td>Write all buffers</td>
</tr>
<tr>
<td>:qa[ll]</td>
<td>Quit all buffers</td>
</tr>
<tr>
<td>:wqa[ll]</td>
<td>Write and quit all buffers</td>
</tr>
<tr>
<td>:qa[ll]!</td>
<td>Quit all buffers without saving</td>
</tr>
<tr>
<td>:xa[ll]!</td>
<td>Write all changed buffers and exit</td>
</tr>
</tbody>
</table>

### Vim Windowing

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X '2'</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Next window</td>
</tr>
</tbody>
</table>
### Vim Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Z</td>
<td>Zoom window toggle</td>
</tr>
<tr>
<td>Ctrl+X '1'</td>
<td>One window</td>
</tr>
</tbody>
</table>

### Vim Windowing - Command Mode Only

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:split</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>:vsplit</td>
<td>Split window vertically</td>
</tr>
<tr>
<td>Ctrl+w ]</td>
<td>Split window and jump to symbol under cursor</td>
</tr>
<tr>
<td>Ctrl+w f</td>
<td>Split window and edit file name under the cursor</td>
</tr>
<tr>
<td>Ctrl+w n</td>
<td>Split window with empty new window</td>
</tr>
<tr>
<td>Ctrl+w o</td>
<td>Make current window the only visible window</td>
</tr>
<tr>
<td>Ctrl+w j</td>
<td>Move cursor to window below</td>
</tr>
<tr>
<td>Ctrl+w k</td>
<td>Move cursor to window above</td>
</tr>
<tr>
<td>Ctrl+w Ctrl+w</td>
<td>Move cursor to window below (wrap)</td>
</tr>
<tr>
<td>Ctrl+w W</td>
<td>Move cursor to window above (wrap)</td>
</tr>
<tr>
<td>Ctrl+w t</td>
<td>Move cursor to top window</td>
</tr>
<tr>
<td>Ctrl+w b</td>
<td>Move cursor to bottom window</td>
</tr>
</tbody>
</table>

### Vim Compiling and Programming Support

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
</tbody>
</table>
### Vim Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+], Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Push a bookmark and go to the first reference to the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+], Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+X M, Ctrl+F5</td>
<td>Build project</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+N, Alt+F10</td>
<td>Next error</td>
</tr>
<tr>
<td>Ctrl+X N</td>
<td>Set next error</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Alt+1</td>
<td>Cursor to error</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+L</td>
<td>Make load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Stop concurrent process</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+M</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
</tbody>
</table>

### Vim Debugging

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
</tbody>
</table>
### Vim Keys

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### Vim Macros

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F11</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+L</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>If editing dialog box or macro, load and run dialog box/macro.</td>
</tr>
<tr>
<td>Ctrl+X (</td>
<td>Start recording macro</td>
</tr>
<tr>
<td>Ctrl+X )</td>
<td>End recording macro</td>
</tr>
</tbody>
</table>

### Vim Macros - Command Mode Only

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot</td>
<td>Repeat last insert or delete</td>
</tr>
</tbody>
</table>
# Vim Keys

## Vim Miscellaneous

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+X, Ctrl+X Ctrl+C</td>
<td>Safe exit</td>
</tr>
<tr>
<td>F10</td>
<td>Menu bar</td>
</tr>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>F5</td>
<td>Configuration menu</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Esc, Ctrl+Q</td>
<td>Cancel</td>
</tr>
<tr>
<td>F9</td>
<td>Undo</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Undo with cursor motion grouping</td>
</tr>
<tr>
<td>Shift+F9, Ctrl+X R</td>
<td>Redo</td>
</tr>
<tr>
<td>Shift+F1</td>
<td>Scroll up</td>
</tr>
<tr>
<td>Shift+F2</td>
<td>Scroll down</td>
</tr>
<tr>
<td>Shift+F3</td>
<td>Scroll left</td>
</tr>
<tr>
<td>Shift+F4</td>
<td>Scroll right</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Center line</td>
</tr>
<tr>
<td>Alt+J</td>
<td>Join line</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+F2</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Alt+P</td>
<td>Reflow paragraph</td>
</tr>
<tr>
<td>Alt+R</td>
<td>Fundamental mode for next key press</td>
</tr>
</tbody>
</table>

770
### Vim Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+T</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+D</td>
<td>Alias change directory</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+E</td>
<td>Shell to DOS</td>
</tr>
</tbody>
</table>

### Vim Miscellaneous - Command Mode Only

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>Full undo</td>
</tr>
<tr>
<td>(visual) u</td>
<td>Change selected text to lowercase</td>
</tr>
<tr>
<td>U</td>
<td>Undo with cursor motion grouping</td>
</tr>
<tr>
<td>(visual) U</td>
<td>Change selected text to uppercase</td>
</tr>
<tr>
<td>~</td>
<td>Toggle the case of current character</td>
</tr>
<tr>
<td>(visual) ~</td>
<td>Toggle the case of the selected text</td>
</tr>
<tr>
<td>c</td>
<td>Change text</td>
</tr>
<tr>
<td>C</td>
<td>Change to end of line</td>
</tr>
<tr>
<td>(visual) c, C</td>
<td>Change text in selection</td>
</tr>
<tr>
<td>r</td>
<td>Overstrike character(s)</td>
</tr>
<tr>
<td>(visual) r</td>
<td>Replace text in selection</td>
</tr>
<tr>
<td>R</td>
<td>Overstrike text</td>
</tr>
<tr>
<td>s</td>
<td>Substitute character(s)</td>
</tr>
<tr>
<td>S</td>
<td>Substitute line</td>
</tr>
<tr>
<td>&lt;</td>
<td>Shift text left</td>
</tr>
<tr>
<td>&gt;</td>
<td>Shift text right</td>
</tr>
<tr>
<td>J</td>
<td>Join line (insert spaces)</td>
</tr>
</tbody>
</table>
### GNU Emacs Keys

#### GNU Emacs Cursor Movement

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+G</td>
<td>Go to line</td>
</tr>
<tr>
<td>Ctrl+A</td>
<td>Begin line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>End of line</td>
</tr>
<tr>
<td>Left arrow, Ctrl+B</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow, Ctrl+F</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Ctrl+N, Down arrow</td>
<td>Next line</td>
</tr>
<tr>
<td>Ctrl+P, Up arrow</td>
<td>Previous line</td>
</tr>
<tr>
<td>Ctrl+V, PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>Alt+V, PgUp</td>
<td>Page up</td>
</tr>
<tr>
<td>Home, Alt+&lt;</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>End, Alt+&gt;</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Alt+Left, Ctrl+Left, Alt+B</td>
<td>Previous word</td>
</tr>
<tr>
<td>Alt+Right, Ctrl+Right, Alt+F</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+Alt+B</td>
<td>Previous level</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Ctrl+Alt+F</td>
<td>Next level</td>
</tr>
<tr>
<td>Alt+}</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Alt+[, Alt+Up</td>
<td>Previous paragraph</td>
</tr>
<tr>
<td>Alt+], Alt+Down</td>
<td>Next paragraph</td>
</tr>
<tr>
<td>Alt+M</td>
<td>First non blank</td>
</tr>
<tr>
<td>Alt+A, Ctrl+Up</td>
<td>Previous sentence</td>
</tr>
<tr>
<td>Alt+E, Ctrl+Down</td>
<td>Next sentence</td>
</tr>
<tr>
<td>Ctrl+X, ‘G’</td>
<td>Go to line</td>
</tr>
<tr>
<td>Alt+Home</td>
<td>Top of file in next window</td>
</tr>
<tr>
<td>Alt+End</td>
<td>End of file in next window</td>
</tr>
<tr>
<td>Alt+PgDn</td>
<td>Page down next window</td>
</tr>
<tr>
<td>Alt+PgUp</td>
<td>Page up next window</td>
</tr>
</tbody>
</table>

### GNU Emacs Inserting Text

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Tab, Ctrl+I</td>
<td>Indent to previous line or insert tab character</td>
</tr>
<tr>
<td>Enter, Ctrl+M, Ctrl+J</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ctrl+O</td>
<td>Split line</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Insert literal</td>
</tr>
</tbody>
</table>
### GNU Emacs Keys

#### GNU Emacs Deleting Text

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+\</td>
<td>Delete horizontal space</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+O</td>
<td>Delete blank lines</td>
</tr>
<tr>
<td>Alt+Del</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Ctrl+Alt+K</td>
<td>Cut level</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Cut word</td>
</tr>
<tr>
<td>Alt+K</td>
<td>Cut sentence</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Del, Ctrl+D</td>
<td>Delete character under cursor</td>
</tr>
<tr>
<td>Backspace, Ctrl+H</td>
<td>Delete character before cursor</td>
</tr>
</tbody>
</table>

#### GNU Emacs Searching

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+S</td>
<td>Incremental search</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+S</td>
<td>Regular expression search</td>
</tr>
<tr>
<td>Ctrl+Alt+R</td>
<td>Reverse regular expression search</td>
</tr>
<tr>
<td>Alt+%</td>
<td>Search and replace with prompting</td>
</tr>
<tr>
<td>Alt+&amp;</td>
<td>Search and replace without prompting</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>

#### GNU Emacs Selection
## GNU Emacs Keys

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+2</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Alt+@</td>
<td>Select next word</td>
</tr>
<tr>
<td>Alt+H</td>
<td>Select paragraph</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+X</td>
<td>Exchange point and start selection</td>
</tr>
<tr>
<td>Ctrl+X 'W'</td>
<td>Write selection to file</td>
</tr>
<tr>
<td>Ctrl+Alt+\</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Alt+I</td>
<td>Tabify selection</td>
</tr>
<tr>
<td>Ctrl+X Alt+I</td>
<td>Untabify selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift block selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift block selection right</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

## GNU Emacs Clipboard

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+W, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Y, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Select clipboard to insert</td>
</tr>
</tbody>
</table>
### GNU Emacs Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+W, Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+A</td>
<td>Copy word</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Cut word</td>
</tr>
<tr>
<td>Alt+K</td>
<td>Cut sentence</td>
</tr>
<tr>
<td>Alt+Del</td>
<td>Delete previous word</td>
</tr>
<tr>
<td>Ctrl+Alt+K</td>
<td>Cut level</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Ctrl+Alt+W</td>
<td>Append next clipboard</td>
</tr>
</tbody>
</table>

### GNU Emacs Files and Buffers

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+X 'K'</td>
<td>Quit buffer</td>
</tr>
<tr>
<td>Ctrl+X 'B'</td>
<td>Select buffer</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+F</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+W</td>
<td>Save and rename buffer</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Write buffer to file</td>
</tr>
<tr>
<td>Ctrl+F2, Ctrl+X 'C'</td>
<td>Compare windows</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+B, Ctrl+X Alt+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+V</td>
<td>Replace buffer with a file on disk</td>
</tr>
<tr>
<td>Ctrl+X 'I'</td>
<td>Insert file</td>
</tr>
<tr>
<td>Ctrl+X 'W'</td>
<td>Write selection to file</td>
</tr>
<tr>
<td>Ctrl+X 'D'</td>
<td>Directory edit mode</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Function</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Resync after compare mismatch</td>
</tr>
</tbody>
</table>

**GNU Emacs Windowing**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X '2'</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Ctrl+X '1'</td>
<td>One window</td>
</tr>
<tr>
<td>Ctrl+X 'O'</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+X 'P'</td>
<td>Prev window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Shrink window</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Expand window</td>
</tr>
</tbody>
</table>

**GNU Emacs Compiling and Programming Support**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+X 'M'</td>
<td>Build project</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+N, Alt+F10</td>
<td>Next error</td>
</tr>
<tr>
<td>Alt+F6</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Ctrl+C C</td>
<td>Stop concurrent process</td>
</tr>
</tbody>
</table>
## GNU Emacs Keys

<table>
<thead>
<tr>
<th>Keyboard Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X Ctrl+M</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+X Comma, Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+H, Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+X Dot</td>
<td>Go to definition</td>
</tr>
<tr>
<td>Ctrl+X Alt+Dot</td>
<td>Make tag file</td>
</tr>
<tr>
<td>Ctrl+X Alt+Comma</td>
<td>Select tag file</td>
</tr>
<tr>
<td>F3</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
</tbody>
</table>

## GNU Emacs Debugging

<table>
<thead>
<tr>
<th>Keyboard Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Start/continue debugging</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
</tbody>
</table>
### GNU Emacs Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### GNU Emacs Macros

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X 'E'</td>
<td>Run last recorded macro</td>
</tr>
<tr>
<td>F3</td>
<td>Make and load current buffer</td>
</tr>
<tr>
<td>Ctrl+X Alt+N</td>
<td>Save last recorded macro</td>
</tr>
<tr>
<td>Ctrl+X '('</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+X ')''</td>
<td>End macro recording</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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):

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Key Sequence</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Alt+A..Alt+Z</td>
<td>Taken over by dialog manager for selecting controls</td>
</tr>
<tr>
<td><strong>GNU Emacs Command Line Keys</strong></td>
<td></td>
</tr>
<tr>
<td>F2, Alt+X</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Delete character</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete previous character</td>
</tr>
<tr>
<td>Tab, Ctrl+I</td>
<td>Insert tab character</td>
</tr>
<tr>
<td>Ctrl+J, Enter Ctrl+M</td>
<td>ENTER argument</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert toggle</td>
</tr>
<tr>
<td>Alt+\</td>
<td>Delete horizontal space</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Expand alias at cursor. Use <strong>alias</strong> command to define aliases.</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Cut previous word</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+W, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Y, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Paste recent clipboard</td>
</tr>
<tr>
<td>Ctrl+Alt+W</td>
<td>Append next clipboard</td>
</tr>
<tr>
<td>Alt+C</td>
<td>Capitalize word</td>
</tr>
<tr>
<td>Alt+L</td>
<td>Lowcase word</td>
</tr>
</tbody>
</table>
### GNU Emacs Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Transpose characters</td>
</tr>
<tr>
<td>Alt+T</td>
<td>Transpose words</td>
</tr>
<tr>
<td>Ctrl+N, Down</td>
<td>Retrieve previous argument</td>
</tr>
<tr>
<td>Ctrl+P, Up</td>
<td>Retrieve next argument</td>
</tr>
<tr>
<td>ESC, Ctrl+[</td>
<td>Complete and enter argument</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### GNU Emacs Miscellaneous

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1, Alt+?, Ctrl+-</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Ctrl+H</td>
<td>GNU Emacs Help</td>
</tr>
<tr>
<td>Alt+P</td>
<td>Configuration menu</td>
</tr>
<tr>
<td>F10</td>
<td>Main menu for mode</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+C</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Select from cursor to beginning of buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Select from cursor to end of buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Cancel</td>
</tr>
<tr>
<td>Ctrl+/, F9, Ctrl+X ‘U’</td>
<td>Undo</td>
</tr>
<tr>
<td>Shortcut</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Shift+F9</td>
<td>Redo</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Undo w/cursor grouping</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Transpose characters</td>
</tr>
<tr>
<td>Alt+T</td>
<td>Transpose words</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+T</td>
<td>Transpose lines</td>
</tr>
<tr>
<td>Alt+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Alt+L</td>
<td>Lowcase selection</td>
</tr>
<tr>
<td>Alt+C</td>
<td>Capitalize word</td>
</tr>
<tr>
<td>Ctrl+X 'F'</td>
<td>Set margins</td>
</tr>
<tr>
<td>Alt+Q</td>
<td>Reflow paragraph</td>
</tr>
<tr>
<td>Alt+S, Pad5</td>
<td>Center line within margins</td>
</tr>
<tr>
<td>Ctrl+Shift+F2</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Alt+)</td>
<td>Find matching start paren</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Center line within window</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Ctrl+X /</td>
<td>Alias change directory</td>
</tr>
<tr>
<td>F8, Ctrl+F8</td>
<td>Set macro variable</td>
</tr>
<tr>
<td>F4</td>
<td>Bind to key</td>
</tr>
<tr>
<td>F6</td>
<td>What is key</td>
</tr>
<tr>
<td>F7</td>
<td>Change directory</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+E</td>
<td>Shell</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+Z</td>
<td>Minimize editor</td>
</tr>
</tbody>
</table>
### GNU Emacs Argument/Repeating a Key

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+U</td>
<td>Select number of times to invoke a command. Sets argument-count.</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+O</td>
<td>Change number of blank lines at or before cursor to argument-count</td>
</tr>
<tr>
<td>Ctrl+X Ctrl+I, Ctrl+X Tab</td>
<td>Indent or unindent selected lines by argument-count characters</td>
</tr>
<tr>
<td>Alt+0..Alt+9</td>
<td>Select argument-count 0..9</td>
</tr>
<tr>
<td>Ctrl+0..Ctrl+9</td>
<td>Select argument-count 0..9</td>
</tr>
</tbody>
</table>

### ISPF Keys

#### ISPF Cursor Movement

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>Cursor left, if in first column, cursor moves into prefix area</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right, if at end of prefix area, moves to first column of line</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Top of buffer</td>
</tr>
</tbody>
</table>
### ISPF Keys

<table>
<thead>
<tr>
<th>Ctrl+End</th>
<th>Bottom of buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Begin line</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>F7</td>
<td>Page up</td>
</tr>
<tr>
<td>F8</td>
<td>Page down</td>
</tr>
<tr>
<td>F10</td>
<td>Page left</td>
</tr>
<tr>
<td>F11</td>
<td>Page right</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent or move to next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Move to previous tab stop, or if in first column, move to prefix area</td>
</tr>
<tr>
<td>Ctrl+J</td>
<td>Go to line</td>
</tr>
</tbody>
</table>

### ISPF Line Prefix Commands

<table>
<thead>
<tr>
<th>.label</th>
<th>Define line prefix label</th>
</tr>
</thead>
<tbody>
<tr>
<td>bnds</td>
<td>Insert left and right boundary ruler line</td>
</tr>
<tr>
<td>tabs</td>
<td>Insert tabs ruler line</td>
</tr>
<tr>
<td>cols</td>
<td>Insert column ruler line</td>
</tr>
</tbody>
</table>

784
## ISPF Inserting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character typed</td>
</tr>
</tbody>
</table>

## ISPF Inserting Text - Line Prefix Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i [n]</td>
<td>Insert n lines after this line</td>
</tr>
<tr>
<td>te [n]</td>
<td>Insert n lines for word-wrap text entry</td>
</tr>
<tr>
<td>tj</td>
<td>Join line</td>
</tr>
<tr>
<td>ts</td>
<td>Split line</td>
</tr>
<tr>
<td>mask</td>
<td>Insert new line mask</td>
</tr>
<tr>
<td>d [n]</td>
<td>Delete n lines</td>
</tr>
<tr>
<td>dd</td>
<td>Delete a block of lines</td>
</tr>
</tbody>
</table>

## ISPF Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
</tbody>
</table>
### ISPF Deleting Text - Line Prefix Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>d [n]</code></td>
<td>Delete n lines</td>
</tr>
<tr>
<td><code>dd</code></td>
<td>Delete a block of lines</td>
</tr>
</tbody>
</table>

### ISPF Searching

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F</td>
<td>Find</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Replace</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Find next occurrence</td>
</tr>
<tr>
<td>Ctrl+Shift+G</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>F5</td>
<td>Find next occurrence</td>
</tr>
<tr>
<td>F6</td>
<td>Repeat last change</td>
</tr>
<tr>
<td>Ctrl+I</td>
<td>Incremental search</td>
</tr>
<tr>
<td>Ctrl+Shift+I</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>

### ISPF Selection

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Select all</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Select block/column</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Deselect</td>
</tr>
<tr>
<td>Ctrl+X, Shift+F1</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del, Shift+F4</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
</tbody>
</table>

786
### ISPF Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Alt+=</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### ISPF Selection - Line Prefix Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>( [n], ( [n]</code></td>
<td>Shift block n columns left</td>
</tr>
<tr>
<td><code>) [n], )) [n]</code></td>
<td>Shift block n columns right</td>
</tr>
<tr>
<td><code>&lt; [n], &lt;&lt; [n]</code></td>
<td>Shift data n columns right</td>
</tr>
<tr>
<td><code>&gt; [n], &gt;&gt; [n]</code></td>
<td>Shift data n columns right</td>
</tr>
<tr>
<td><code>c [n]</code></td>
<td>Copy n lines</td>
</tr>
<tr>
<td><code>cc</code></td>
<td>Copy a block of lines</td>
</tr>
<tr>
<td><code>m [n]</code></td>
<td>Move n lines</td>
</tr>
<tr>
<td><code>mm</code></td>
<td>Move a block of lines</td>
</tr>
<tr>
<td><code>z [n]</code></td>
<td>Select n lines</td>
</tr>
</tbody>
</table>
### ISPF Keys

<table>
<thead>
<tr>
<th>shortcut</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>zz</code></td>
<td>Select a block of lines</td>
</tr>
</tbody>
</table>

### ISPF Clipboard

<table>
<thead>
<tr>
<th>shortcut</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Ctrl+C</code></td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td><code>Ctrl+Shift+C</code></td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td><code>Ctrl+K</code></td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td><code>Ctrl+Shift+V</code></td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td><code>Ctrl+V</code></td>
<td>Paste</td>
</tr>
<tr>
<td><code>Ctrl+Backspace</code></td>
<td>Cut line</td>
</tr>
<tr>
<td><code>Ctrl+E</code></td>
<td>Cut to end of line</td>
</tr>
<tr>
<td><code>Ctrl+Shift+K</code></td>
<td>Cut word</td>
</tr>
<tr>
<td><code>Ctrl+X</code></td>
<td>Cut selection</td>
</tr>
<tr>
<td><code>Ctrl+Shift+X</code></td>
<td>Append cut selection</td>
</tr>
</tbody>
</table>

### ISPF Clipboard - Line Prefix Commands

<table>
<thead>
<tr>
<th>short</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a [n]</code></td>
<td>Insert block after, repeat <code>n</code> times</td>
</tr>
<tr>
<td><code>b [n]</code></td>
<td>Insert block before, repeat <code>n</code> times</td>
</tr>
<tr>
<td><code>o [n]</code></td>
<td>Overlay <code>n</code> lines</td>
</tr>
<tr>
<td><code>oo</code></td>
<td>Overlay a block of lines</td>
</tr>
<tr>
<td><code>r [n]</code></td>
<td>Repeat a line <code>n</code> times</td>
</tr>
<tr>
<td><code>rr [n]</code></td>
<td>Repeat a block <code>n</code> times</td>
</tr>
<tr>
<td><code>z [n]</code></td>
<td>Select <code>n</code> lines</td>
</tr>
<tr>
<td><code>zz</code></td>
<td>Select a block of lines</td>
</tr>
</tbody>
</table>
## ISPF Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
</tbody>
</table>
### ISPF Keys

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>F12</td>
<td>Retrieve previous command from command line</td>
</tr>
<tr>
<td>Shift+F12</td>
<td>Retrieve next command from command line</td>
</tr>
</tbody>
</table>

### ISPF Files and Buffers

<table>
<thead>
<tr>
<th>Keyboard Mappings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2, Ctrl+O</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>F3, F4</td>
<td>Save (if autosave is on) and quit current buffer</td>
</tr>
<tr>
<td>F3, Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>F9</td>
<td>Next buffer or window</td>
</tr>
<tr>
<td>Shift+F9</td>
<td>Previous buffer or window</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>List buffers</td>
</tr>
</tbody>
</table>

### ISPF Windowing

<table>
<thead>
<tr>
<th>Keyboard Mappings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+H</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Ctrl+Tab, Ctrl+F6</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab, Ctrl+Shift+F6</td>
<td>Previous window</td>
</tr>
<tr>
<td>Ctrl+Shift+Z</td>
<td>Zoom window toggle</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Close window</td>
</tr>
</tbody>
</table>
### ISPF Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+F1</td>
<td>API Apprentice on current word</td>
</tr>
<tr>
<td>Alt+F2</td>
<td>Move window edge</td>
</tr>
<tr>
<td>Alt+F3</td>
<td>Create window edge</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Move</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Size</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Minimize</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

### ISPF Compiling and Programming Support

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the</td>
</tr>
<tr>
<td></td>
<td>symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Push a bookmark and go to the first reference to</td>
</tr>
<tr>
<td></td>
<td>the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Find next reference</td>
</tr>
<tr>
<td>Ctrl+Shift+G</td>
<td>Find previous reference</td>
</tr>
<tr>
<td>Ctrl+M</td>
<td>Build project</td>
</tr>
</tbody>
</table>
### ISPF Keys

<table>
<thead>
<tr>
<th>Keyboard Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Ctrl+Shift+Down</td>
<td>Next error</td>
</tr>
<tr>
<td>Ctrl+Shift+Up</td>
<td>Previous error</td>
</tr>
<tr>
<td>Ctrl+Shift+S</td>
<td>Set next error</td>
</tr>
<tr>
<td>Ctrl+Shift+E</td>
<td>List errors</td>
</tr>
<tr>
<td>Shift+F10</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Alt+1</td>
<td>Cursor to error/include file</td>
</tr>
<tr>
<td>Ctrl+Shift+M</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
</tbody>
</table>

### ISPF Debugging

<table>
<thead>
<tr>
<th>Keyboard Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
</tbody>
</table>
### ISPF Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### ISPF Macros

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F11</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Terminate recording &amp; run last recorded macro</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
</tbody>
</table>

### ISPF Selective Display

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>x [n]</td>
<td>Exclude n lines</td>
</tr>
<tr>
<td>xx</td>
<td>Exclude a block of lines</td>
</tr>
<tr>
<td>f [n]</td>
<td>Expose first n lines of excluded block</td>
</tr>
<tr>
<td>l [n]</td>
<td>Expose last n lines of excluded block</td>
</tr>
<tr>
<td>s [n]</td>
<td>Expose n lines at first indentation level</td>
</tr>
</tbody>
</table>

### ISPF Miscellaneous

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>SDK Help on word at cursor</td>
</tr>
<tr>
<td>Alt+F4</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Esc</td>
<td>Cancel or command line toggle</td>
</tr>
<tr>
<td>Ctrl+Z, Alt+Backspace</td>
<td>Undo</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Redo</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+J</td>
<td>Go to bookmark</td>
</tr>
<tr>
<td>Ctrl+1..Ctrl+0</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Ctrl+Shift+N</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Ctrl+]</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Change directory</td>
</tr>
<tr>
<td>Alt+F5</td>
<td>Restore MDI window</td>
</tr>
<tr>
<td>Alt+F7</td>
<td>Move MDI window</td>
</tr>
<tr>
<td>Alt+F8</td>
<td>Size MDI window</td>
</tr>
<tr>
<td>Alt+F10</td>
<td>Maximize MDI window</td>
</tr>
<tr>
<td>Enter</td>
<td>Execute line prefix commands or, if there are none, go to the next line.</td>
</tr>
</tbody>
</table>
### CodeWarrior Keys

#### CodeWarrior Cursor Movement

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Home</td>
<td>Begin line</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
</tbody>
</table>

#### ISPF Miscellaneous - Line Prefix Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc [n]</td>
<td>Lowcase n lines</td>
</tr>
<tr>
<td>lclc, lcc</td>
<td>Lowcase block of lines</td>
</tr>
<tr>
<td>md [n]</td>
<td>Make n data lines</td>
</tr>
<tr>
<td>mdmd, mdd</td>
<td>Make data lines</td>
</tr>
<tr>
<td>uc [n]</td>
<td>Upcase n lines</td>
</tr>
<tr>
<td>uclc, ucc</td>
<td>Upcase block of lines</td>
</tr>
<tr>
<td>tf</td>
<td>Reflow paragraph</td>
</tr>
</tbody>
</table>
### CodeWarrior Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Back indent text to previous tab stop</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Go to line</td>
</tr>
</tbody>
</table>

### CodeWarrior Inserting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Enter</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character typed</td>
</tr>
</tbody>
</table>

### CodeWarrior Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
</tbody>
</table>
# CodeWarrior Keys

## CodeWarrior Searching

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+F4, Ctrl+F</td>
<td>Find</td>
</tr>
<tr>
<td>Ctrl+R, Ctrl+=</td>
<td>Replace</td>
</tr>
<tr>
<td>F3</td>
<td>Find next occurrence</td>
</tr>
<tr>
<td>Shift+F3</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>Ctrl+I</td>
<td>Incremental search</td>
</tr>
<tr>
<td>Ctrl+Shift+I</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>

## CodeWarrior Selection

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Select all</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Select line</td>
</tr>
<tr>
<td>F8</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Deselect</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Alt+=</td>
<td>Execute commands in selection</td>
</tr>
</tbody>
</table>
### CodeWarrior Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### CodeWarrior Clipboard

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+C, Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Ctrl+V, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+X, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
</tbody>
</table>

### CodeWarrior Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
</tbody>
</table>
CodeWarrior Keys

**CodeWarrior Files and Buffers**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>F2, Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>F4</td>
<td>Save and quit current buffer</td>
</tr>
<tr>
<td>F7, Ctrl+O</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Resync after compare mismatch</td>
</tr>
</tbody>
</table>

**CodeWarrior Windowing**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+H</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Ctrl+Tab, Ctrl+F6</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab, Ctrl+Shift+F6</td>
<td>Previous window</td>
</tr>
<tr>
<td>Ctrl+Shift+Z</td>
<td>Zoom window toggle</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Close window</td>
</tr>
<tr>
<td>Alt+F1</td>
<td>API Apprentice on current word</td>
</tr>
<tr>
<td>Alt+F2</td>
<td>Move window edge</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Move</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Size</td>
</tr>
</tbody>
</table>
## CodeWarrior Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F9</td>
<td>Minimize</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

### CodeWarrior Compiling and Programming Support

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Push a bookmark and go to the first reference to the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>F3</td>
<td>Find next reference</td>
</tr>
<tr>
<td>Shift+F3</td>
<td>Find previous reference</td>
</tr>
<tr>
<td>Ctrl+M</td>
<td>Build project</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Ctrl+Shift+Down</td>
<td>Next error</td>
</tr>
<tr>
<td>Ctrl+Shift+Up</td>
<td>Previous error</td>
</tr>
<tr>
<td>Ctrl+Shift+S</td>
<td>Set next error</td>
</tr>
<tr>
<td>Ctrl+Shift+E</td>
<td>List errors</td>
</tr>
</tbody>
</table>
### CodeWarrior Keys

<table>
<thead>
<tr>
<th>shortcut</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F7</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Alt+1, Ctrl+D</td>
<td>Cursor to error/include file</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+M</td>
<td>Start concurrent process</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
</tbody>
</table>

### CodeWarrior Debugging

<table>
<thead>
<tr>
<th>shortcut</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Start/continue debugging</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>
## CodeWarrior Macros

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F11</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Terminate recording &amp; run last recorded macro</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>If editing dialog box or macro, load and run dialog box/macro.</td>
</tr>
</tbody>
</table>

## CodeWarrior Miscellaneous

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>F10</td>
<td>Menu bar</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>SDK Help on word at cursor</td>
</tr>
<tr>
<td>Alt+F4</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Esc</td>
<td>Cancel or command line toggle</td>
</tr>
<tr>
<td>Ctrl+Z, Alt+Backspace</td>
<td>Undo</td>
</tr>
<tr>
<td>Shift+F9</td>
<td>Undo with cursor motion grouping</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Redo</td>
</tr>
</tbody>
</table>
### CodeWright Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+J</td>
<td>Go to bookmark</td>
</tr>
<tr>
<td>Ctrl+1..Ctrl+0</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Ctrl+Shift+T</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Alt+F5</td>
<td>Restore MDI window</td>
</tr>
<tr>
<td>Alt+F7</td>
<td>Move MDI window</td>
</tr>
<tr>
<td>Alt+F8</td>
<td>Size MDI window</td>
</tr>
<tr>
<td>Alt+F10</td>
<td>Maximize MDI window</td>
</tr>
</tbody>
</table>

### CodeWright Cursor Movement

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Home</td>
<td>Begin line</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
</tbody>
</table>
# CodeWright Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Page left</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Page right</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Back indent text to previous tab stop</td>
</tr>
<tr>
<td>Ctrl+J</td>
<td>Go to line</td>
</tr>
</tbody>
</table>

## CodeWright Inserting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Enter</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character typed</td>
</tr>
</tbody>
</table>

## CodeWright Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Delete previous word</td>
</tr>
</tbody>
</table>
### CodeWright Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Del</td>
<td>Delete to end of line</td>
</tr>
<tr>
<td>Shift+Backspace</td>
<td>Delete word</td>
</tr>
</tbody>
</table>

#### CodeWright Searching

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F</td>
<td>Find</td>
</tr>
<tr>
<td>Ctrl+H</td>
<td>Replace</td>
</tr>
<tr>
<td>Ctrl+Shift+S, Ctrl+G</td>
<td>Find next occurrence</td>
</tr>
<tr>
<td>Ctrl+Shift+Q</td>
<td>Quick search</td>
</tr>
<tr>
<td>Ctrl+I</td>
<td>Incremental search</td>
</tr>
<tr>
<td>Ctrl+Shift+R</td>
<td>Repeat last replace</td>
</tr>
<tr>
<td>Ctrl+Shift+I</td>
<td>Reverse incremental search</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Stop search or search &amp; replace</td>
</tr>
</tbody>
</table>

#### CodeWright Selection

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Select all</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Select block/column</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Ctrl+I</td>
<td>Inclusive character</td>
</tr>
<tr>
<td>Ctrl+M</td>
<td>Non-inclusive character selection</td>
</tr>
<tr>
<td>Ctrl+W</td>
<td>Save selection</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
</tbody>
</table>
### CodeWright Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Alt+ =</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Shift+Cursor keys</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### CodeWright Clipboard

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+C, Ctrl+Ins</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>Ctrl+V, Shift+Ins</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+X, Shift+Del</td>
<td>Cut selection</td>
</tr>
</tbody>
</table>

### CodeWright Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Function Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards and optionally paste one</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle command line</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+D</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Del</td>
<td>Delete to end of line</td>
</tr>
<tr>
<td>Shift+Backspace</td>
<td>Delete word</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Delete previous word</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Delete line</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Shift+Cursor keys</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
</tbody>
</table>
## CodeWright Keys

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
</tbody>
</table>

### CodeWright Files and Buffers

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2, Ctrl+S</td>
<td>Save current buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+N</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>Ctrl+O</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Resync after compare mismatch</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Open new file or create new project</td>
</tr>
<tr>
<td>Ctrl+Shift+F</td>
<td>Insert file at cursor</td>
</tr>
</tbody>
</table>

### CodeWright Windowing

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Tab, Ctrl+F6</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+Tab, Ctrl+Shift+F6</td>
<td>Previous window</td>
</tr>
<tr>
<td>Ctrl+Shift+Z, Alt+F2</td>
<td>Zoom window toggle</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Close window</td>
</tr>
<tr>
<td>Alt+F3</td>
<td>Create window edge</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Move</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Size</td>
</tr>
</tbody>
</table>

### CodeWright Compiling and Programming Support
### CodeWright Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Alt+Comma, Alt+F1</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Shift+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Ctrl+Space</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Ctrl+Dot</td>
<td>Push a bookmark and go to the definition of the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+/</td>
<td>Push a bookmark and go to the first reference to the symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Ctrl+G, Ctrl+Shift+S</td>
<td>Find next reference</td>
</tr>
<tr>
<td>Ctrl+F9, Ctrl+M</td>
<td>Build project</td>
</tr>
<tr>
<td>Ctrl+F10, Shift+F10</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Ctrl+Shift+Down</td>
<td>Next error</td>
</tr>
<tr>
<td>Ctrl+Shift+Up</td>
<td>Previous error</td>
</tr>
<tr>
<td>Ctrl+Shift+S</td>
<td>Set next error</td>
</tr>
<tr>
<td>Ctrl+Shift+E</td>
<td>List errors</td>
</tr>
<tr>
<td>Alt+1</td>
<td>Cursor to error/include file</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+M</td>
<td>Start concurrent process</td>
</tr>
</tbody>
</table>

### CodeWright Debugging

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Start/continue debugging</td>
</tr>
</tbody>
</table>
### CodeWright Keys

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
<tr>
<td>F10</td>
<td>Step over</td>
</tr>
<tr>
<td>F11</td>
<td>Step into</td>
</tr>
<tr>
<td>Alt+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>Ctrl+Alt+B, Alt+F9</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Alt+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Alt+4, Ctrl+Alt+V</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Alt+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate threads window</td>
</tr>
</tbody>
</table>

### CodeWright Macros

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F7, Ctrl+F11</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>F8, Ctrl+F12</td>
<td>Terminate recording &amp; run last recorded macro</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro that is prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Use this to terminate infinite loops.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
</tbody>
</table>
# CodeWright Keys

## CodeWright Miscellaneous

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>SDK Help on word at cursor</td>
</tr>
<tr>
<td>Alt+Shift+Left</td>
<td>Back (like web browser)</td>
</tr>
<tr>
<td>Alt+Shift+Right</td>
<td>Forward (like web browser)</td>
</tr>
<tr>
<td>Alt+F4</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Esc</td>
<td>Cancel</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Line to top</td>
</tr>
<tr>
<td>F9</td>
<td>Line toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+G</td>
<td>Show functions headings</td>
</tr>
<tr>
<td>Ctrl+U, Ctrl+Z</td>
<td>Undo</td>
</tr>
<tr>
<td>Shift+F9</td>
<td>Undo with cursor motion grouping</td>
</tr>
<tr>
<td>Ctrl+Y, Alt+Ins</td>
<td>Redo</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+D</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+Shift+J</td>
<td>Set bookmark</td>
</tr>
<tr>
<td>Ctrl+1..Ctrl+0</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Ctrl+]</td>
<td>Find matching parenthesis</td>
</tr>
</tbody>
</table>
### Xcode Keys

#### Xcode Cursor Movement

<table>
<thead>
<tr>
<th>Keyboard Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift + Tab</td>
<td>Back indent text to previous tab stop</td>
</tr>
<tr>
<td>Command + Left arrow</td>
<td>Begin line</td>
</tr>
<tr>
<td>Command + Down arrow</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Ctrl + PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
<tr>
<td>Command + L</td>
<td>Go to line</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Alt + Right</td>
<td>Next word</td>
</tr>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
</tbody>
</table>
### Xcode Keys

<table>
<thead>
<tr>
<th>Hotkey</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Command+Down arrow</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
</tbody>
</table>

### Xcode Inserting Text

<table>
<thead>
<tr>
<th>Hotkey</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character types</td>
</tr>
</tbody>
</table>

### Xcode Deleting Text

<table>
<thead>
<tr>
<th>Hotkey</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
</tbody>
</table>

### Xcode Selection

<table>
<thead>
<tr>
<th>Hotkey</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+Right+Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Command+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Deselect</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Action</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
</tr>
<tr>
<td>Command+=</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Command+A</td>
<td>Select all</td>
</tr>
<tr>
<td>F8</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
</tbody>
</table>

**Xcode Searching**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+F</td>
<td>Find</td>
</tr>
<tr>
<td>Alt+Command+F</td>
<td>Find in files</td>
</tr>
<tr>
<td>Command+G</td>
<td>Find next occurrence</td>
</tr>
<tr>
<td>Command+Shift+G</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Replace</td>
</tr>
</tbody>
</table>
## Xcode Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>F3, Command+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Command+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards, optionally paste one</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Alt+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Command+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
</tbody>
</table>
### Xcode Keys

<table>
<thead>
<tr>
<th>Command/Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+E</td>
<td>Set search string</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
</tbody>
</table>

### Xcode Files and Buffers

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F7, Command+O</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>Alt+Command+Right arrow</td>
<td>Next Buffer</td>
</tr>
<tr>
<td>Command+N</td>
<td>New File</td>
</tr>
<tr>
<td>Alt+Command+Up arrow</td>
<td>Open associated file</td>
</tr>
<tr>
<td>Alt+Command+Left arrow</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Resync after compare mismatch</td>
</tr>
<tr>
<td>F4</td>
<td>Save and quit current buffer</td>
</tr>
<tr>
<td>F2, Command+S</td>
<td>Save current buffer</td>
</tr>
</tbody>
</table>

### Xcode Clipboard

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Command+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Command+X, Shift+Del</td>
<td>Cut selection</td>
</tr>
</tbody>
</table>
### Xcode Keys

<table>
<thead>
<tr>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+K</td>
<td>Select next condition</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards, optionally paste one</td>
</tr>
<tr>
<td>Command+V</td>
<td>Paste</td>
</tr>
</tbody>
</table>

### Xcode Windowing

<table>
<thead>
<tr>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+F1</td>
<td>API Apprentice on current word</td>
</tr>
</tbody>
</table>

### Xcode Macros

<table>
<thead>
<tr>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Terminate infinite loops.</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+F11</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Terminate recording, run last recorded macro</td>
</tr>
</tbody>
</table>

### Xcode Miscellaneous

<table>
<thead>
<tr>
<th>Keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esc</td>
<td>Cancel or command line toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+</td>
<td>Complete previous word/variable</td>
</tr>
</tbody>
</table>
## BBEdit Keys

**BBEdit Cursor Movement**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+Tab</td>
<td>Back indent text to previous tab stop</td>
</tr>
</tbody>
</table>

**BBEdit Keys**

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Command+D</td>
<td>Go to bookmark</td>
</tr>
<tr>
<td>Ctrl+Shift+D</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl-]</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Command+F10</td>
<td>Maximize MDI window</td>
</tr>
<tr>
<td>Command+F7</td>
<td>Move MDI window</td>
</tr>
<tr>
<td>Command+Shift+Z</td>
<td>Redo</td>
</tr>
<tr>
<td>Command+F5</td>
<td>Restore MDI window</td>
</tr>
<tr>
<td>Command+Q</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Command+F1</td>
<td>SDK Help on word at cursor</td>
</tr>
<tr>
<td>Ctrl+1..Ctrl+0</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Alt+F8</td>
<td>Size MDI window</td>
</tr>
<tr>
<td>Command+Z, Alt+Backspace</td>
<td>Undo</td>
</tr>
<tr>
<td>Shift+F9</td>
<td>Undo with cursor motion grouping</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Command+F7</td>
<td>Move MDI window</td>
</tr>
</tbody>
</table>
### BBEdit Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+Left arrow</td>
<td>Begin line</td>
</tr>
<tr>
<td>Command+Down arrow</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>End</td>
<td>End line</td>
</tr>
<tr>
<td>Command+J</td>
<td>Go to line</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Alt+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
<tr>
<td>Alt+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Command+Down arrow</td>
<td>Top of buffer</td>
</tr>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
</tbody>
</table>

### BBEdit Inserting Text

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
</tbody>
</table>
### BBEdit Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character typed</td>
</tr>
</tbody>
</table>

### BBEdit Deleting Text

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
</tbody>
</table>

### BBEdit Selection

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+Right-Click</td>
<td>Copy selection to cursor</td>
</tr>
<tr>
<td>Command+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Command+Shift+A</td>
<td>Deselect</td>
</tr>
<tr>
<td>Command+=</td>
<td>Execute commands in selection</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Command+A</td>
<td>Select all</td>
</tr>
<tr>
<td>F8</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Command+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>Command+Shift+L</td>
<td>Select paragraph</td>
</tr>
</tbody>
</table>
### BBEdit Keys

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection C</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt; selection</td>
<td>Start or extend char/stream</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Unindent selection</td>
</tr>
</tbody>
</table>

### BBEdit Searching

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+F</td>
<td>Find</td>
</tr>
<tr>
<td>Alt+Command+F</td>
<td>Find in files</td>
</tr>
<tr>
<td>Command+G</td>
<td>Find in next occurrence</td>
</tr>
<tr>
<td>Command+Shift+G</td>
<td>Find previous occurrence</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Replace</td>
</tr>
</tbody>
</table>

### BBEdit Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Space</td>
<td>Complete argument</td>
</tr>
<tr>
<td>F3, Command+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Keyboard Shortcut</td>
<td>Function</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Command+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>?</td>
<td>List arguments</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards, optionally paste one</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Alt+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Command+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Command+Shift+V</td>
<td>Paste previous clipboard</td>
</tr>
<tr>
<td>Alt+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Command+E</td>
<td>Set search string</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt; selection</td>
<td>Start or extend char/stream selection</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
</tbody>
</table>

**BBEdit Files and Buffers**

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F7, Command+O</td>
<td>Edit a file or find buffer</td>
</tr>
</tbody>
</table>
### BBEdit Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F6</td>
<td>File compare</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>Alt+Command+Right arrow</td>
<td>Next buffer</td>
</tr>
<tr>
<td>Command+N</td>
<td>New file</td>
</tr>
<tr>
<td>Alt+Command+Shift+N</td>
<td>New file from clipboard</td>
</tr>
<tr>
<td>Command+Shift+N</td>
<td>New file from selection</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Open associated file</td>
</tr>
<tr>
<td>Alt+Command+Left arrow</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Resync after compare mismatch</td>
</tr>
<tr>
<td>F4</td>
<td>Save and quit current buffer</td>
</tr>
<tr>
<td>F2, Command+S</td>
<td>Save current buffer</td>
</tr>
</tbody>
</table>

### BBEdit Clipboard

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Command+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Copy word to clipboard</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Command+X, Shift+Del</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards, optionally paste one</td>
</tr>
<tr>
<td>Command+V</td>
<td>Paste</td>
</tr>
</tbody>
</table>

### BBEdit Windowing
### BBEdit Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+F1</td>
<td>API Apprentice on current word</td>
</tr>
<tr>
<td>Command+W, Ctrl+F4</td>
<td>Close window</td>
</tr>
<tr>
<td>Alt+Command+W</td>
<td>Close All Windows</td>
</tr>
<tr>
<td>Command+F10</td>
<td>Maximize</td>
</tr>
<tr>
<td>Command+M</td>
<td>Minimize</td>
</tr>
<tr>
<td>Command+F2</td>
<td>Move window edge</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>Next window</td>
</tr>
<tr>
<td>Ctrl+Shift+F6</td>
<td>Previous window</td>
</tr>
<tr>
<td>Command+F8</td>
<td>Size</td>
</tr>
<tr>
<td>Command+Shift+®</td>
<td>Split window horizontally</td>
</tr>
<tr>
<td>Command+/</td>
<td>Zoom window toggle</td>
</tr>
</tbody>
</table>

### BBEdit Compiling and Programming Support

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+M</td>
<td>Build project</td>
</tr>
<tr>
<td>Shift+F10</td>
<td>Compile current buffer</td>
</tr>
<tr>
<td>Ctrl+Dot, F5, Alt+Esc</td>
<td>Complete symbol</td>
</tr>
<tr>
<td>Command+D</td>
<td>Cursor to error/include file</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Execute project</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Expand extension specific alias</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand global alias</td>
</tr>
<tr>
<td>Command+G</td>
<td>Find next reference</td>
</tr>
<tr>
<td>Command+Shift+G</td>
<td>Find previous reference</td>
</tr>
<tr>
<td>Ctrl+Shift+E</td>
<td>List errors</td>
</tr>
</tbody>
</table>
### BBEdit Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+Dot</td>
<td>List symbols</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>For next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Shift+PgDn</td>
<td>To page up/down in comment</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Alt+Command+Down arrow</td>
<td>Next error</td>
</tr>
<tr>
<td>Alt+Comma</td>
<td>Parameter Info</td>
</tr>
<tr>
<td>Ctrl+PgUp/Ctrl+PgDn</td>
<td>next/previous definition</td>
</tr>
<tr>
<td>Shift+PgUp/Ctrl+PgDn</td>
<td>page up/down in comment</td>
</tr>
<tr>
<td>Command+Comma</td>
<td>Pop a pushed bookmark</td>
</tr>
<tr>
<td>Alt+Command+Up arrow</td>
<td>Previous error</td>
</tr>
<tr>
<td>Command+Dot</td>
<td>Push a bookmark, go to definition of symbol at cursor</td>
</tr>
<tr>
<td>Shift+Command+/</td>
<td>Push a bookmark, go to first reference to symbol at cursor</td>
</tr>
<tr>
<td>Ctrl+Shift+S</td>
<td>Set next error</td>
</tr>
<tr>
<td>Ctrl+Shift+M</td>
<td>Start concurrent process</td>
</tr>
</tbody>
</table>

### BBEdit Debugging

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Command+B</td>
<td>Activate breakpoints window</td>
</tr>
<tr>
<td>Command+7, Ctrl+Alt+C</td>
<td>Activate call stack</td>
</tr>
<tr>
<td>Ctrl+Command+H</td>
<td>Activate threads window</td>
</tr>
<tr>
<td>Command+4</td>
<td>Activate variables window</td>
</tr>
<tr>
<td>Command+3, Ctrl+Alt+W</td>
<td>Activate watch window</td>
</tr>
<tr>
<td>Ctrl+Shift+F9</td>
<td>Clear all breakpoints</td>
</tr>
</tbody>
</table>

826
### BBEdit Keys

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+F5</td>
<td>Restart debugging</td>
</tr>
<tr>
<td>Ctrl+F10</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>Command+PadStar</td>
<td>Show next statement</td>
</tr>
<tr>
<td>F5</td>
<td>Start/continue debugging</td>
</tr>
<tr>
<td>Command+Shift+I</td>
<td>Step into</td>
</tr>
<tr>
<td>Command+Shift+T</td>
<td>Step out</td>
</tr>
<tr>
<td>Command+Shift+O</td>
<td>Step over</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>Alt+Command+/</td>
<td>Toggle breakpoint enable</td>
</tr>
<tr>
<td>Command+/</td>
<td>Toggle breakpoint</td>
</tr>
</tbody>
</table>

### BBEdit Macros

<table>
<thead>
<tr>
<th>Shortcuts</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
<tr>
<td>Ctrl+Shift+F12,&lt;key&gt;</td>
<td>Stops macro recording and binds macro to &lt;key&gt; (which can be 0-9, A-Z, or F1-F12).</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Halt Slick-C® macro prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl+Alt+Shift+F2</td>
<td>Halt Slick-C macro that is executing. Terminate infinite loops.</td>
</tr>
<tr>
<td>F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Ctrl+F11</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Terminate recording, run last recorded macro</td>
</tr>
</tbody>
</table>

### BBEdit Miscellaneous
<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esc</td>
<td>Cancel or command line toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Ctrl+Shift+Dot</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl+Shift+O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Command+D</td>
<td>Go to bookmark</td>
</tr>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase word</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Command+F10</td>
<td>Maximize MDI window</td>
</tr>
<tr>
<td>Command+F7</td>
<td>Move MDI window</td>
</tr>
<tr>
<td>Command+Shift+Z</td>
<td>Redo</td>
</tr>
<tr>
<td>Command+F5</td>
<td>Restore MDI window</td>
</tr>
<tr>
<td>Command+Q</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Command+F1</td>
<td>SDK Help on word at cursor</td>
</tr>
<tr>
<td>Ctrl+1..Ctrl+0</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Alt+F8</td>
<td>Size MDI window</td>
</tr>
<tr>
<td>Command+Z, Alt+Backspace</td>
<td>Undo</td>
</tr>
<tr>
<td>Shift+F9</td>
<td>Undo with cursor motion grouping</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
</tbody>
</table>

**Visual Studio Default Keys**
<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ctrl+Alt+W,[1-4]</td>
<td>Activate Watch view</td>
</tr>
<tr>
<td>Ctrl+Alt+G</td>
<td>Activate Registers view</td>
</tr>
<tr>
<td>Ctrl+Alt+M,[1-4]</td>
<td>Activate Memory view</td>
</tr>
<tr>
<td>Ctrl+Alt+C</td>
<td>Activate Call Stack view</td>
</tr>
<tr>
<td>Ctrl+Alt+V,A or Ctrl+Alt+V,a</td>
<td>Activate Autos view</td>
</tr>
<tr>
<td>Ctrl+Alt+V,L or Ctrl+Alt+V,l</td>
<td>Activate Locals view</td>
</tr>
<tr>
<td>Ctrl+Alt+V,T or Ctrl+Alt+V,t</td>
<td>Activate Members view</td>
</tr>
<tr>
<td>Ctrl+Alt+B</td>
<td>Activate Breakpoints view</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Activate Threads view</td>
</tr>
<tr>
<td>F3</td>
<td>Find Next Occurrence</td>
</tr>
<tr>
<td>Shift+F3</td>
<td>Find Previous Occurrence</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Transpose Adjacent Characters</td>
</tr>
<tr>
<td>Ctrl+Shift+T</td>
<td>Transpose Adjacent Words</td>
</tr>
<tr>
<td>Alt+Shift+T</td>
<td>Transpose Adjacent Lines</td>
</tr>
<tr>
<td>Alt+F3,B or Alt+F3,b</td>
<td>Toggle Search Option: Search Backwards</td>
</tr>
<tr>
<td>Alt+F3,C or Alt+F3,c</td>
<td>Toggle Search Option: Match Case</td>
</tr>
<tr>
<td>Alt+F3,R or Alt+F3,r</td>
<td>Toggle Search Option: Use Regex</td>
</tr>
<tr>
<td>Alt+F3,W or Alt+F3,w</td>
<td>Toggle Search Option: Match Whole Word</td>
</tr>
<tr>
<td>Ctrl+F</td>
<td>Find</td>
</tr>
<tr>
<td>Keyboard Shortcut</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Ctrl+Shift+F</td>
<td>Find in Files</td>
</tr>
<tr>
<td>Ctrl+H</td>
<td>Replace</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Replace in Files</td>
</tr>
<tr>
<td>Shift+Alt+Enter</td>
<td>Fullscreen Mode</td>
</tr>
<tr>
<td>Ctrl+K, Ctrl+N</td>
<td>Goto Next Bookmark</td>
</tr>
<tr>
<td>Ctrl+K, Ctrl+P</td>
<td>Goto Previous Bookmark</td>
</tr>
<tr>
<td>Ctrl+K, Ctrl+L</td>
<td>Clear All Bookmarks</td>
</tr>
<tr>
<td>Ctrl+M, Ctrl+P</td>
<td>Show All</td>
</tr>
<tr>
<td>Ctrl+M, Ctrl+U</td>
<td>Show Selection</td>
</tr>
<tr>
<td>Ctrl+M, Ctrl+H</td>
<td>Hide Selection</td>
</tr>
<tr>
<td>Ctrl+K, Ctrl+U</td>
<td>Remove Comment</td>
</tr>
<tr>
<td>Ctrl+R, Ctrl+W</td>
<td>Toggle Viewing of Whitespace</td>
</tr>
<tr>
<td>Ctrl+K, Ctrl+K</td>
<td>Toggle Bookmark</td>
</tr>
<tr>
<td>Ctrl+K, Ctrl+W</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>Ctrl+PadMinus</td>
<td>Goto Previous Document</td>
</tr>
<tr>
<td>Ctrl+Shift+PadMinus</td>
<td>Goto Next Document</td>
</tr>
<tr>
<td>Ctrl+F12</td>
<td>Goto Definition</td>
</tr>
<tr>
<td>Ctrl+Alt+J</td>
<td>Activate Symbols view</td>
</tr>
<tr>
<td>Ctrl+Alt+A</td>
<td>Activate Build view</td>
</tr>
<tr>
<td>Ctrl+Alt+O</td>
<td>Activate Output view</td>
</tr>
<tr>
<td>Ctrl+Alt+R</td>
<td>Goto URL</td>
</tr>
<tr>
<td>Ctrl+Alt+L</td>
<td>Activate Projects view</td>
</tr>
<tr>
<td>Shift+Alt+A</td>
<td>Add File to Project</td>
</tr>
</tbody>
</table>
### Mac OS X Keys

#### Mac OS X Cursor Movement

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift + Tab</td>
<td>Back indent text to previous tab stop</td>
</tr>
<tr>
<td>Command+Left arrow, Ctrl+A</td>
<td>Begin line</td>
</tr>
<tr>
<td>Command+Down arrow, Ctrl+End</td>
<td>Bottom of buffer</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Bottom of window</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Cursor down</td>
</tr>
<tr>
<td>Left arrow</td>
<td>Cursor left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Cursor right</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Cursor up</td>
</tr>
<tr>
<td>End, Ctrl+E</td>
<td>End line</td>
</tr>
<tr>
<td>Ctrl+J</td>
<td>Go to line</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent to next tab stop</td>
</tr>
<tr>
<td>Option+Right, Ctrl+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>PgDn</td>
<td>Page down</td>
</tr>
<tr>
<td>PgUp</td>
<td>Page up</td>
</tr>
<tr>
<td>Option+Left, Ctrl+Left</td>
<td>Previous word</td>
</tr>
</tbody>
</table>
### Mac OS X Keys

<table>
<thead>
<tr>
<th>Ctrl+Home, Command+Up arrow</th>
<th>Top of buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+PgUp</td>
<td>Top of window</td>
</tr>
</tbody>
</table>

#### Mac OS X Inserting Text

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>Insert a line</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>Open a new line below current line</td>
</tr>
<tr>
<td>Ctrl+Shift+Enter</td>
<td>Open a new line above current line</td>
</tr>
<tr>
<td>Shift+Enter</td>
<td>Insert a line (no syntax expansion)</td>
</tr>
<tr>
<td>Shift+Space</td>
<td>Insert a space (no syntax expansion)</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character types</td>
</tr>
</tbody>
</table>

#### Mac OS X Deleting Text

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete char before cursor</td>
</tr>
<tr>
<td>Del</td>
<td>Delete char under cursor</td>
</tr>
</tbody>
</table>

#### Mac OS X Selection

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Backspace, Del</td>
<td>Delete selection</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Deselect</td>
</tr>
<tr>
<td>Mac OS X Keys</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Ctrl+Right-Click</td>
<td>Move selection to cursor</td>
</tr>
<tr>
<td>Command+A</td>
<td>Select all</td>
</tr>
<tr>
<td>F8</td>
<td>Select character/stream</td>
</tr>
<tr>
<td>Ctrl+L, Command+L</td>
<td>Select line</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>Ctrl+W, Double-Click</td>
<td>Select word</td>
</tr>
<tr>
<td>Command+]</td>
<td>Indent selection</td>
</tr>
<tr>
<td>Command+[</td>
<td>Unindent selection</td>
</tr>
<tr>
<td>Shift+F7</td>
<td>Shift selection left</td>
</tr>
<tr>
<td>Shift+F8</td>
<td>Shift selection right</td>
</tr>
<tr>
<td>Right-Click &amp; Drag</td>
<td>Start block/column selection</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Shift+&lt;Cursor keys&gt;</td>
<td>Start or extend char/stream selection</td>
</tr>
</tbody>
</table>

### Mac OS X 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 |
### Mac OS X Keys

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+R</td>
<td>Replace in files</td>
</tr>
</tbody>
</table>

#### Mac OS X Command Line and Text Box Editing

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Option+Command+Comma</td>
<td>Complete argument</td>
</tr>
<tr>
<td>Command+C, Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Esc</td>
<td>Cursor to command line toggle</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Command+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+K</td>
<td>Cut word</td>
</tr>
<tr>
<td>Shift+Click</td>
<td>Extend selection</td>
</tr>
<tr>
<td>Ins</td>
<td>Insert/overwrite toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards, optionally paste one</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase selection</td>
</tr>
<tr>
<td>Option+Right</td>
<td>Next word</td>
</tr>
<tr>
<td>Command+V, Ctrl+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Option+Left, Ctrl+Left</td>
<td>Previous word</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Quote next character</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Retrieve next command</td>
</tr>
<tr>
<td>Up arrow</td>
<td>Retrieve previous command</td>
</tr>
<tr>
<td>Triple-Click</td>
<td>Select line</td>
</tr>
<tr>
<td>Double-Click</td>
<td>Select word</td>
</tr>
</tbody>
</table>
### Mac OS X Keys

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+E</td>
<td>Use selection for search</td>
</tr>
<tr>
<td>Click &amp; Drag</td>
<td>Start char/stream selection</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
</tbody>
</table>

### Mac OS X Files and Buffers

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command+O, Ctrl+O</td>
<td>Edit a file or find buffer</td>
</tr>
<tr>
<td>Ctrl+=</td>
<td>File compare</td>
</tr>
<tr>
<td>Ctrl+Shift+B</td>
<td>List buffers</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Next Buffer</td>
</tr>
<tr>
<td>Command+N</td>
<td>New File</td>
</tr>
<tr>
<td>Ctrl+Backtick</td>
<td>Open associated file</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>Previous buffer</td>
</tr>
<tr>
<td>Command+S, Ctrl+S</td>
<td>Save current buffer</td>
</tr>
</tbody>
</table>

### Mac OS X Clipboard

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Shift+X</td>
<td>Append cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+C</td>
<td>Append selection to clipboard</td>
</tr>
<tr>
<td>Command+C, Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Cut line</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Cut to end of line</td>
</tr>
<tr>
<td>Command+X, Ctrl+X</td>
<td>Cut selection</td>
</tr>
<tr>
<td>Ctrl+Shift+Command+Right</td>
<td>Select next expression</td>
</tr>
<tr>
<td>Ctrl+Shift+V</td>
<td>List clipboards, optionally paste one</td>
</tr>
</tbody>
</table>
## Mac OS X Keys

<table>
<thead>
<tr>
<th>Command + V</th>
<th>Paste</th>
</tr>
</thead>
</table>

### Mac OS X Windowing

<table>
<thead>
<tr>
<th>Option + Tab</th>
<th>Next window</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option + Shift + Tab</td>
<td>Previous window</td>
</tr>
<tr>
<td>Command + M</td>
<td>Minimize window</td>
</tr>
<tr>
<td>Command + Backtick</td>
<td>Cycle through application windows</td>
</tr>
</tbody>
</table>

### Mac OS X Macros

<table>
<thead>
<tr>
<th>Ctrl + Shift + Space</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + Option + Command + S</td>
<td>Halt Slick-C® macro prompting for a key with get_event()</td>
</tr>
<tr>
<td>Ctrl + Option + Command + T</td>
<td>Halt Slick-C macro that is executing. Terminate infinite loops.</td>
</tr>
<tr>
<td>F4, F12</td>
<td>Make and load current macro buffer</td>
</tr>
<tr>
<td>Shift + F4</td>
<td>Start/end macro recording</td>
</tr>
<tr>
<td>Ctrl + F12</td>
<td>Terminate recording, run last recorded macro</td>
</tr>
</tbody>
</table>

### Mac OS X Miscellaneous

<table>
<thead>
<tr>
<th>Esc</th>
<th>Cancel or command line toggle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + Shift + Space</td>
<td>Complete more</td>
</tr>
<tr>
<td>Ctrl + Shift + Period</td>
<td>Complete next word/variable</td>
</tr>
<tr>
<td>Ctrl + Shift + Comma</td>
<td>Complete previous word/variable</td>
</tr>
<tr>
<td>Ctrl + Shift + O</td>
<td>Expand alias at cursor</td>
</tr>
<tr>
<td>Keyboard shortcut</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Ctrl+Shift+J</td>
<td>Toggle bookmark</td>
</tr>
<tr>
<td>Ctrl+Shift+N</td>
<td>Activate Bookmarks view</td>
</tr>
<tr>
<td>F1</td>
<td>Help for mode or context</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Hex display toggle</td>
</tr>
<tr>
<td>Ctrl+Shift+L</td>
<td>Lowcase selection</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Upcase word</td>
</tr>
<tr>
<td>Ctrl+]</td>
<td>Match parenthesis</td>
</tr>
<tr>
<td>Option+F10, Command+F10</td>
<td>Maximize MDI window</td>
</tr>
<tr>
<td>Command+F7</td>
<td>Move MDI window</td>
</tr>
<tr>
<td>Command+Shift+Z</td>
<td>Redo</td>
</tr>
<tr>
<td>Option+F5</td>
<td>Restore MDI window</td>
</tr>
<tr>
<td>Command+Q</td>
<td>Safe exit</td>
</tr>
<tr>
<td>Ctrl+1..Ctrl+0</td>
<td>Set bookmark 1..0</td>
</tr>
<tr>
<td>Command+Z, Option+Backspace</td>
<td>Undo</td>
</tr>
</tbody>
</table>
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` (UNIX®: `vslick.stu`), 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Example Name</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespaces</td>
<td><code>se.example</code></td>
<td>Lowercase, with an underscore or dot to separate multiple words.</td>
</tr>
<tr>
<td>Classes</td>
<td><code>ExampleName</code></td>
<td>Mixed case, first letter must be capitalized, all caps only acceptable for acronyms like &quot;FTP&quot;.</td>
</tr>
<tr>
<td>Interfaces</td>
<td><code>IExampleName</code></td>
<td>Like class names, but with &quot;I&quot; prefix.</td>
</tr>
<tr>
<td>Enums</td>
<td><code>ExampleName</code></td>
<td>Like class names (idea of &quot;E&quot; prefix rejected).</td>
</tr>
<tr>
<td>Enum Flags</td>
<td><code>ExampleFlags</code></td>
<td>Like enums, but ends with &quot;Flags&quot;</td>
</tr>
</tbody>
</table>
## Differences Between Slick-C® and C++

<table>
<thead>
<tr>
<th>Type</th>
<th>Example Name</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(idea of “F” prefix rejected).</td>
</tr>
<tr>
<td>Member Funcs</td>
<td>exampleName</td>
<td>Mixed case with the first letter lowercase.</td>
</tr>
<tr>
<td>Member Vars</td>
<td>m_exampleName</td>
<td>Mixed case, first letter lowercase, and an “m_” prefix.</td>
</tr>
<tr>
<td>Properties</td>
<td>m_exampleName</td>
<td>Same as member variables (should not distinguish from var).</td>
</tr>
<tr>
<td>Class Vars</td>
<td>s_exampleName</td>
<td>Mixed case, first letter lowercase, and an “s_” prefix.</td>
</tr>
<tr>
<td>Namespace Vars</td>
<td>g_exampleName</td>
<td>Like member vars except with a “g_” prefix.</td>
</tr>
<tr>
<td>Namespace Funcs</td>
<td>example_name</td>
<td>Lowercase with words separated with underscores.</td>
</tr>
<tr>
<td>Global Vars</td>
<td>&lt;anything&gt;</td>
<td>No rules.</td>
</tr>
<tr>
<td>Global Funcs</td>
<td>&lt;anything&gt;</td>
<td>No rules.</td>
</tr>
<tr>
<td>Typedefs</td>
<td>&lt;anything&gt;</td>
<td>No rules.</td>
</tr>
</tbody>
</table>
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:

```c
struct PHONERECORD {
    _str name;
    _str PhoneNumber;
};

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:

```c
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.

**Labeled Loops**

The break and continue statements accept an optional label parameter so that you can break a specific loop (like Java).

Example:

```c
outerlabel:
    for (;;) {
        for (;;) {
```
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:

```c
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

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.

Capability not Supported by Slick-C®

- You cannot define your own classes, methods, or inheritance. The **class**, **public**, **private**, and **new** keywords are not supported. Classes in Slick-C will not require a `delete` to free objects.
- 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.
- **enum** 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:

   ```c
c_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 **Window → SlickEdit Preferences → Keyboard → 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:

```c
#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:

```csh
#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 **Esc**.

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:

```csh
#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:

```c
#include "slick.sh"
void defmain()
{
    message("Hello World");
}
```

2. Save the file as `hellow.e`, then press `Esc` to open the SlickEdit® command line.

3. Type `hellow`, and press `Enter`.

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
• Reserved Words and Keywords
• Comments
• String Literals
• Numeric Literals
• Defining Constants Using #define
• Namespaces

Identifiers

A variable or identifier may contain any of the characters "A-Za-z$_0-9" and must start with one of the characters "A-Za-z_$".

Reserved Words and Keywords

The following keywords are reserved in the Slick-C® language:

• _command
• _notinit
• _reinit
• _str
• arg
• auto
• boolean
• break
• case
Reserved Words and Keywords

- 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
The following keywords are reserved for built-in functions:

- `_a2e`
- `_asc`
- `_assert`
- `_callmethod`
- `_callmethod`
- `_chr`
Reserved Words and Keywords

- _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
• _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
Reserved Words and Keywords

- index_callable
- index2event
- insert_name
- isinteger
- isnumber
- keyin
- last_index
- lastpos
- length
- list_bindings
- lowercase
- 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
Reserved Words and Keywords

• translate
• strappend
• strcmp
• stricmp
• strieq
• strip
• strrev
• substr
• togglecase
• trace
• translate
• upcase
• verify

The following keywords are reserved for future use:

• catch
• finally
• throw
• try

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. SlickEdit® reserves all identifiers starting with an underscore (_) for internal use.
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:

```cpp
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.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>Bell character (7)</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace character (8)</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed character (12)</td>
</tr>
<tr>
<td>\n</td>
<td>New line character (10)</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return (13)</td>
</tr>
<tr>
<td>\t</td>
<td>Tab character (9)</td>
</tr>
<tr>
<td>\v</td>
<td>Vertical tab character (11)</td>
</tr>
<tr>
<td>?</td>
<td>Question mark character</td>
</tr>
<tr>
<td>'</td>
<td>Single quote character</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote character</td>
</tr>
<tr>
<td>\</td>
<td>Backslash character</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\a</code></td>
<td>Bell character (7)</td>
</tr>
<tr>
<td><code>\b</code></td>
<td>Backspace character (8)</td>
</tr>
<tr>
<td><code>\f</code></td>
<td>Form feed character(12)</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>New line (10)</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>Carriage return (13)</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Tab character (9)</td>
</tr>
<tr>
<td><code>\v</code></td>
<td>Vertical tab character (11)</td>
</tr>
<tr>
<td><code>\x dd</code></td>
<td>Hexadecimal character code dd</td>
</tr>
<tr>
<td><code>\ddd</code></td>
<td>Decimal character code dddd</td>
</tr>
</tbody>
</table>
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 \texttt{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:

\[
[+|-] \text{digits } [.][\text{digits}][\text{E}[+|-]\text{digits}]
\]

or

\[
[+|-] \text{[.][digits]}[\text{E}[+|-]\text{digits}]
\]

There may be blank spaces before and after the leading sign.

Example:

\begin{verbatim}
  4.04
  4e2
  4e2
  4E-2
  4E-2
\end{verbatim}

Defining Constants Using \texttt{#define}

Slick-C® supports the \texttt{#define} preprocessor directive. The \texttt{#define} directive is for defining constants or in-line functions. Use the following syntax to define the constant or in-line function:

\[
\texttt{#define name\{(param1, param2,\})} \texttt{value}
\]

Use a backslash at the end of a line to indicate that the \texttt{value} text continues to the next line. Any occurrence of \texttt{name} is replaced with the text \texttt{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:

```c
#define MAXLINES 15
#define MAXLINESP1 (MAXLINES+1)
#define max(a,b) (((a) >= (b)) ? (a) : (b))
#define min(a,b) (((a) <= (b)) ? (a) : (b))

void main() {
    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:

```c
const MAXLINES = 15;
const MAXLINESP1 = (MAXLINES+1);
const SEARCHKEY = "<Search>";
```

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.
- 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:

```c
// 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:

```c
// 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**
- **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.
- **Interfaces**
- **Classes**
- **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 `boolean`.
- **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.

See also Implicit Conversion to Strings.

Enumerated Types
Slick-C® enumerated types are very much like C enumerated types, with the exception of having relaxed type checking with respect to arithmetic and bit operations.

```c
enum BasicOptions {
    OPTION1=1,
    OPTION2,
    OPTION3,
};
```

In addition, Slick-C enumerated types introduce enumerated type flags, a convenient way to create a set of bit flags.

```c
enum_flags OptionFlags {
    FLAG1=0x4,
    FLAG2, // 0x8
    FLAG3, // 0x10
    FLAGS_ALL=FLAG1|FLAG2|FLAG3
};
```

## 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:

```c
[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:

```c
array.__makeempty(); // Empty the array.
array=null; // Empty the array. Same as above.
```

You can delete and insert items into an array using the `_deleteel()` and `_insertel()` built-in methods, respectively.

A Slick-C® class instance can be indexed using array syntax provided that the class implements the
**sc.lang.Indexable** 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.
- Array initializers are not supported for local variables.

Example:

```cpp
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;

defmain()
{
    t=gai; // Copy all the array elements into a local container
    t[t._length()]=45; // Add another array element.
    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:

```cpp
[static] TypeName variable1[] [= {s1=>e1, s2=>e2, ...}],
variable2[] [= {s1=>e1, s2=>e2, ...}] ...
```

You can delete an item from a hash table using `deleteel()`. 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:

```cpp
#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;
    }
};

defmain() {
    boolean 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;
}
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;
};
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:

```c
[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:

```c
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
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:
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:

```c
def main()
{
    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:

```c
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 variable's 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:

```sc
#include "sc/lang/IIndexable.e"

class PerfectSquares : sc.lang.IIndexable {
    typeless _array_el(int i) {
        return i*i;
    }
};

defmain()
{
    PerfectSquares ps;
    say("defmain: 3^2="ps[3]);
    say("defmain: 16^2="ps[16]);
}
```

Below is an example of IHashIndexable:

```sc
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];
    }
};
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:

```c
typeless t;
t=1;    // Store an integer.
    // Convert the contents of the variable t to a
    // floating pointer 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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>! e1</td>
<td>Logical NOT. Result is 1 if e1 evaluates to 0. Otherwise the result is 0.</td>
</tr>
<tr>
<td>~ e1</td>
<td>Bitwise complement.</td>
</tr>
<tr>
<td>- e1</td>
<td>Negation.</td>
</tr>
<tr>
<td>+ e1</td>
<td>No change.</td>
</tr>
<tr>
<td>++ v1</td>
<td>Increments the variable v1 and returns the result.</td>
</tr>
<tr>
<td>v1 ++</td>
<td>Returns the value of v1 and then increments the variable v1.</td>
</tr>
<tr>
<td>-- v1</td>
<td>Decrements the variable v1 and returns the result.</td>
</tr>
<tr>
<td>v1 --</td>
<td>Returns the value of v1 and then decrements the variable v1.</td>
</tr>
</tbody>
</table>

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 for all platforms.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Assign right operand to left operand.</td>
</tr>
<tr>
<td>:=</td>
<td>Declare new variable with type matching right operand and assign it the value of the right operand.</td>
</tr>
<tr>
<td>+=</td>
<td>Add left operand to right operand and assign to left operand.</td>
</tr>
<tr>
<td>-=</td>
<td>Subtract right operand from left operand and assign</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>to left operand.</td>
<td></td>
</tr>
<tr>
<td>Divide left operand by right operand and assign to left operand.</td>
<td></td>
</tr>
<tr>
<td>Multiply left operand with right operand and assign to left operand.</td>
<td></td>
</tr>
<tr>
<td>Bitwise OR left operand with right operand and assign to left operand.</td>
<td></td>
</tr>
<tr>
<td>Bitwise XOR left operand with right operand and assign to left operand.</td>
<td></td>
</tr>
<tr>
<td>Bitwise AND left operand with right operand and assign to left operand.</td>
<td></td>
</tr>
<tr>
<td>If expression e1 is TRUE (not the string 0), expression e2 is returned. Otherwise, expression e3 is returned.</td>
<td></td>
</tr>
<tr>
<td>Logical AND. If left hand expression is false, right-hand expression is not evaluated.</td>
<td></td>
</tr>
<tr>
<td>Logical OR. If left hand expression is true, right-hand expression is not evaluated.</td>
<td></td>
</tr>
<tr>
<td>Bitwise OR.</td>
<td></td>
</tr>
<tr>
<td>Bitwise XOR.</td>
<td></td>
</tr>
<tr>
<td>Bitwise AND.</td>
<td></td>
</tr>
<tr>
<td>Equal. Performs a numeric or string comparison depending on the operands. This function is NOT identical to the C <code>strcmp</code> function (see <code>==</code> 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.</td>
<td></td>
</tr>
<tr>
<td>Greater than. Performs a numeric or string comparison depending on the operands. See <code>==</code></td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal. Performs a numeric or string comparison depending on the operands. See <code>==</code> operator.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than. Performs a numeric or string comparison depending on the operands. See <code>==</code> operator.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal. Performs a numeric or string comparison depending on the operands. See <code>==</code> operator.</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal. Performs a numeric or string comparison depending on the operands. See <code>==</code> operator.</td>
</tr>
<tr>
<td>:=</td>
<td>Exactly equal. Always performs string comparison. This is equivalent to the C expression: <code>(strcmp(a,b)==0)</code></td>
</tr>
<tr>
<td>:!=</td>
<td>Not exactly equal. Always performs string comparison.</td>
</tr>
<tr>
<td>:&lt;=</td>
<td>Exactly less than or equal. Always performs string comparison.</td>
</tr>
<tr>
<td>&lt;:</td>
<td>Exactly less than. Always performs string comparison.</td>
</tr>
<tr>
<td>:&gt;</td>
<td>Exactly greater than. Always performs string comparison.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Exactly greater than or equal. Always performs string comparison.</td>
</tr>
<tr>
<td>instanceof</td>
<td>Can be used to test if a class instance derives from a specific class or interface. It can be used in two ways: <code>x instanceof MYCLASS</code>, or <code>x instanceof &quot;MYCLASS&quot;</code>. &quot;MYCLASS&quot; does not need to be a constant string, and <code>x</code> may be a typeless container variable. Slick-C’s <code>instanceof</code> is slightly more powerful than Java’s, since the right operand can be a string value rather than just a class name. Otherwise, it is essentially the same concept.</td>
</tr>
</tbody>
</table>
### Mathematical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:+</td>
<td>Concatenation.</td>
</tr>
<tr>
<td>:+=</td>
<td>Append right operand (string) to the end of the left operand (string) and assign to the left operand. This is similar to the built-in <code>strappend()</code> function, but more convenient to use.</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Bitwise shift left.</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Bitwise shift right.</td>
</tr>
<tr>
<td>+</td>
<td>Addition.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction.</td>
</tr>
<tr>
<td>/</td>
<td>Division with possible floating point result.</td>
</tr>
<tr>
<td><code>intdiv</code></td>
<td>Division with integer result.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication.</td>
</tr>
<tr>
<td>%</td>
<td>Modulo (integer remainder).</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Example</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(1.0==1)</code></td>
<td><code>== true</code></td>
</tr>
<tr>
<td><code>(1e2==100)</code></td>
<td><code>== true</code></td>
</tr>
<tr>
<td><code>(1e2:==100)</code></td>
<td><code>== false</code></td>
</tr>
<tr>
<td><code>(&quot; abc &quot;:=&quot;abc&quot;)</code></td>
<td><code>== true</code></td>
</tr>
<tr>
<td>Example</td>
<td>Operator</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>(&quot; abc &quot;::&quot;abc&quot;)</td>
<td>== false</td>
</tr>
<tr>
<td>(&quot; abc &quot;::=!&quot;abc&quot;)</td>
<td>== true</td>
</tr>
<tr>
<td>(&quot; 1 &quot;:==1)</td>
<td>== true</td>
</tr>
<tr>
<td>(&quot; 1 &quot;:==1)</td>
<td>== false</td>
</tr>
<tr>
<td>(&quot;abc&quot;:&lt;&quot;def&quot;)</td>
<td>== true</td>
</tr>
<tr>
<td>1 2</td>
<td>:=&quot;12&quot;</td>
</tr>
<tr>
<td>1 ( 2)</td>
<td>:=&quot;12&quot;</td>
</tr>
<tr>
<td>pow(4,2)</td>
<td>==16</td>
</tr>
<tr>
<td>5%2</td>
<td>==1</td>
</tr>
<tr>
<td>5/2</td>
<td>==2</td>
</tr>
<tr>
<td>5/2.0</td>
<td>==2.5</td>
</tr>
<tr>
<td>5 intdiv 2.0</td>
<td>==2</td>
</tr>
<tr>
<td>5&amp;2</td>
<td>==0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>(10&lt;7)</td>
<td>== false</td>
</tr>
<tr>
<td>(10:&lt;7)</td>
<td>== true</td>
</tr>
</tbody>
</table>
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:

```c
// 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";
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 boolean, 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 empty 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:

boolean globalboolean=true;
int globalint;
double globaldouble;
defmain()
{
    // Will print message "globalboolean=1 globalint=0 globaldouble=0".
    message("boolean="globalboolean" "globalint" "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:

defmain()
{
    int i;
    double d;
    d=1.2;
    i=(int)d;  // i gets the value 1, NOT 1.2
    typeless t;
    t=1.2;
    i=t;      // Here i gets 1.2 BUT
    boolean 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
Declaring Variables With auto

Slick-C® supports type inference using the **auto** keyword. The syntax for auto variable declarations is:

```
```

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;       // boolean 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.

```
890
```
Examples:

```c
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:

```plaintext
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:

```plaintext
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:

```java
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:

```java
{
    statement1;
    statement2;
    ...
}
```

Example:

```java
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.Iterable, 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:
break

Loops are exited with the `break` primitive. The `break` 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.

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:

```plaintext
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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>variable_name</code></td>
<td>Output variable.</td>
</tr>
<tr>
<td>.</td>
<td>Null output variable.</td>
</tr>
<tr>
<td><code>nnn</code></td>
<td>Number specifying new parse column.</td>
</tr>
<tr>
<td><code>+ nnn</code></td>
<td>Amount to increment parse column relative to start of last string found or last column setting.</td>
</tr>
</tbody>
</table>
### parse Statement

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- nnn</td>
<td>Amount to decrement parse column relative to start of last string found or last column setting.</td>
</tr>
</tbody>
</table>
| 'text'[.search_options] | String constant to search for. If found, parse column becomes first character after text. Otherwise parse column becomes first character after length of string being parsed. search_options is an optional expression that may evaluate to a string of one or more of the option letters U, R, B, I, and Y:  
  - U specifies UNIX regular expressions.  
  - R specifies SlickEdit regular expressions.  
  - B specifies Brief regular expressions.  
  - L specifies Perl regular expressions.  
  - & 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).  
  See the topic "regular expressions" in the SlickEdit® Help system for more information. |
| (expression)[.search_options] | String expression 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. |

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:

```c
[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.

```c
[ result =] id( expr1, expr2, ...);
```
In the above example, expr1 matches the type of id1 and expr2 matches the type of id2, etc.

Example:

```c
int increment(int x)
{
    return x+1;
}
boolean 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:

```c
boolean proc(int &p1,_str p2,_str (&list)[],int (*&pfn)(int))
```
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);
}
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.
}
```
The \texttt{\_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 (Window \textrightarrow SlickEdit Preferences \textrightarrow Keyboard \textrightarrow Key Bindings).

The syntax for defining a command is:

\begin{verbatim}
\_command [TypeName | void] name1[,name2 [,name3... ]]( [ArgDecl1, ArgDecl2, ...
) 
  [name_info(const_exp)]
  {
    statements
  }
\end{verbatim}

\texttt{TypeName} specifies the return type of the command (see \texttt{Types}). If \texttt{TypeName} or \texttt{void} is not specified, the return type is \texttt{typeless}. When the \texttt{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 \texttt{ArgDecls} is the same as for declaring a variable, except that the \texttt{static} keyword may not be used. In addition, an \& before the \texttt{id} declares a call by references parameter. Call by reference array and hash table parameters require parentheses around the \& and the \texttt{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 \texttt{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 \\\texttt{"/"}. SlickEdit uses the slash to define a search command.

Example:

\begin{verbatim}
// 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);
}
\end{verbatim}
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 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 (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.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSARG2_CMDLINE</td>
<td>Command supports the command line.</td>
</tr>
<tr>
<td>VSARG2_CMDLINE</td>
<td>VSARG2_CMDLINE allows a fundamental mode key binding to be inherited by the command line.</td>
</tr>
<tr>
<td>VSARG2_MARK</td>
<td>ON_SELECT event should pass control on to this command and not deselect text first. Ignored if command does not require an editor control.</td>
</tr>
<tr>
<td>VSARG2_QUOTE</td>
<td>Indicates that this command must be quoted when called during macro recording. Needed only if command name is an invalid identifier or keyword.</td>
</tr>
<tr>
<td>VSARG2_LASTKEY</td>
<td>Command requires <code>last_event</code> value to be set.</td>
</tr>
<tr>
<td>Flag</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VSARG2_MACRO</td>
<td>This is a recorded macro command. Used for completion.</td>
</tr>
<tr>
<td>VSARG2_TEXT_BOX</td>
<td>Command supports any text box control. VSARG2_TEXT_BOX allows a fundamental mode key binding to be inherited by a text box.</td>
</tr>
<tr>
<td>VSARG2_NOEXIT_SCROLL</td>
<td>Do not exit scroll caused by using scroll bars. Ignored if command does not require an editor control.</td>
</tr>
<tr>
<td>VSARG2_EDITORCTL</td>
<td>Command allowed in editor control. VSARG2_EDITORCTL allows a fundamental mode. Key binding to be inherited by a non-MDI editor control.</td>
</tr>
<tr>
<td>VSARG2_NOUNDOS</td>
<td>Do not automatically call _undo('s'). Require macro to call _undo('s') to start a new level of undo.</td>
</tr>
<tr>
<td>VSARG2_READ_ONLY</td>
<td>Command allowed when editor control is in strict read only mode. Ignored if command does not require an editor control</td>
</tr>
<tr>
<td>VSARG2_ICON</td>
<td>Command allowed when editor control window is iconized. Ignored if command does not require an editor control.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_EDITORCTL</td>
<td>Command requires an editor control.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_MDI_EDITORCTL</td>
<td>Command requires MDI editor control.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_AB_SELECTION</td>
<td>Command requires selection in active buffer.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_BLOCK_SELECTION</td>
<td>Command requires block/column selection in any buffer.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_CLIPBOARD</td>
<td>Command requires editorctl clipboard.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_FILEMAN_MODE</td>
<td>Command requires active buffer to be in fileman mode.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_TAGGING</td>
<td>Command requires &lt;ext&gt;_proc_search/find-tag support.</td>
</tr>
</tbody>
</table>
Defining a Command

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSARG2_REQUIRES_SELECTION</td>
<td>Command requires a selection in any buffer.</td>
</tr>
<tr>
<td>VSARG2_REQUIRES_MDI</td>
<td>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).</td>
</tr>
</tbody>
</table>

Example:

```c
#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 guiEnumerate()
    name_info(',VSARG2_REQUIRES_EDITORCTL|VSARG2_REQUIRES_AB_SELECTION)
{
    ...
}
// This command supports completion on multiple filenames.
_command 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 `constExp` 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:

```c
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.

• 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:

```c
[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.

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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PCDOS</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
### Constant

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td>Non-zero if current operating system is UNIX compatible.</td>
</tr>
<tr>
<td><strong>NT</strong></td>
<td>Non-zero if the current operating system is Microsoft Windows NT® compatible.</td>
</tr>
<tr>
<td><strong>VERSION</strong></td>
<td>Version number of SlickEdit®.</td>
</tr>
<tr>
<td><strong>COLUMN</strong></td>
<td>Number of characters from start of current line.</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>Current file name.</td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>Current line number.</td>
</tr>
<tr>
<td><strong>PATH</strong></td>
<td>Current file name and path.</td>
</tr>
<tr>
<td><strong>DATE</strong></td>
<td>Current date.</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>Current time.</td>
</tr>
</tbody>
</table>

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:

```c
#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
```

### #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.

<table>
<thead>
<tr>
<th>OptionName</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autodecl</td>
<td>On</td>
<td>Enables autodeclvars and autodeclctls. See those options for more information.</td>
</tr>
<tr>
<td>autodeclctls</td>
<td>On</td>
<td>When enabled, the compiler attempts to automatically declare control variables. This option is automatically enabled when autodecl, pedantic, strict, or strict2 is enabled.</td>
</tr>
<tr>
<td>autodeclvars</td>
<td>On</td>
<td>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.</td>
</tr>
<tr>
<td>deprecation</td>
<td>Off</td>
<td>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</td>
</tr>
<tr>
<td>OptionName</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>it as an error. A function is considered as deprecated if it has a Javadoc function comment containing the <code>@deprecated</code> tag. Deprecation is automatically enabled when <code>pedantic</code> is enabled. Note that using the <code>deprecation</code> 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.</td>
</tr>
<tr>
<td>pedantic</td>
<td>Off</td>
<td>Enabling this option automatically enables all existing and future strict syntax and type-checking options. Unlike other Slick-C pragmas, the meaning of <code>pedantic</code> could be augmented in future releases of SlickEdit. This means that if you use the <code>pedantic</code> 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.</td>
</tr>
<tr>
<td>redeclvars</td>
<td>Off</td>
<td>This is used to generate code for variables without having the type information. When enabled, any variable can be redeclared as a typeless variable.</td>
</tr>
<tr>
<td>strict</td>
<td>Off</td>
<td>This option is used to turn on a high level of type checking and syntax enforcement in the Slick-C translator. It automatically enables <code>autodeclctls</code>, <code>autodeclvars</code>, <code>strictnumbers</code>, <code>strictparens</code>, <code>strictsemicolons</code>, and <code>strictstrings</code>. We recommend using <code>#pragma option(strict, on)</code></td>
</tr>
<tr>
<td>OptionName</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in user-written macros because it gives the best combination of high level of error checking and forward compatibility.</td>
</tr>
<tr>
<td>strict2</td>
<td>Off</td>
<td>Second generation of strict Slick-C compilation checks. Automatically enables all options that strict enables, plus strictarglists, strictincludes, and twopass.</td>
</tr>
</tbody>
</table>
| 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:  
void first_char(s) { 
    return substr(s,1,1); 
}  
This option is automatically enabled when pedantic or strict2 is enabled. |
| strictboolean    | Off           | 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:  
boolean b = 0;  
This option is automatically enabled within classes and namespaces. |
<p>| 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. |</p>
<table>
<thead>
<tr>
<th>OptionName</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>This option is automatically enabled when <strong>pedantic</strong> or <strong>strict2</strong> is enabled.</td>
</tr>
<tr>
<td>strictincludes</td>
<td>Off</td>
<td>When enabled, verifies that all <strong>#import</strong> and <strong>#include</strong> statements precede any real code in the current module. This is required for <strong>twopass</strong> compilation. This option is automatically enabled when <strong>pedantic</strong> or <strong>strict2</strong> is enabled.</td>
</tr>
<tr>
<td>strictnames</td>
<td>Off</td>
<td>When enabled, enforces naming conventions for symbols declared in Slick-C classes, namespaces, and enumerated types. See <a href="#">Slick-C® Naming Conventions</a> for more information. This option is automatically enabled when <strong>pedantic</strong> is enabled.</td>
</tr>
<tr>
<td>strictnumbers</td>
<td>Off</td>
<td>When enabled, Slick-C numeric constants are treated strictly as integer or double precision floating point types, rather than than typeless variables. This makes it possible to use precise type inference with integer types, like <code>i := 0;</code>. This option is automatically enabled when <strong>pedantic</strong> is enabled. It is also automatically enabled within classes and namespaces.</td>
</tr>
<tr>
<td>strictparens</td>
<td>Off</td>
<td>Use this pragma for more readable code. When enabled, parentheses must be given on all built-in functions. This option is automatically enabled when <strong>pedantic</strong>, <strong>strict</strong>, or <strong>strict2</strong> is enabled.</td>
</tr>
<tr>
<td>strictprotos</td>
<td>Off</td>
<td>When enabled, all function calls require the function to be</td>
</tr>
<tr>
<td>OptionName</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>strictreturn</td>
<td>On</td>
<td>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.</td>
</tr>
<tr>
<td>strictsemicolons</td>
<td>Off</td>
<td>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 is enabled.</td>
</tr>
<tr>
<td>strictstrings</td>
<td>Off</td>
<td>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 &quot;0&quot; to an integer variable. This option is automatically enabled when pedantic, strict, or strict2 is enabled.</td>
</tr>
<tr>
<td>twopass</td>
<td>Off</td>
<td>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.</td>
</tr>
</tbody>
</table>

#pragma
OptionName | Default Value | Description |
---|---|---|
| | | option is automatically enabled when pedantic or strict2 is enabled. |

#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:

```c
#region Region_1
void Test() {} 
void Test2() {} 
void Test3() {} 
#endregion Region_1

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:

```plaintext
#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:

```c
[_nocheck] ObjectName ControlName;
```

Or you can use:

```c
[_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] ObjectName ControlName; declaration is only permitted outside the scope of a function. The [_nocheck] _control ControlName; declaration already supports local procedure scope.

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 boolean 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.
  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:

```c
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 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.

Example:
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 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:

```
def eventtab 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.
def eventtab 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.
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`:

```c
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:

```c
int gmarkid= -1;
definit()
{
   // If this is an editor invocation,
   if (arg(1)!="L") {
      gmarkid=-1; // indicate no mark is allocated.
   }
}
```

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:

```sh
#include "slick.sh"
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_EXIT;
    }
    // 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: vtw) 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.

<table>
<thead>
<tr>
<th>Alias Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>messageNwait(\n: \c);</td>
</tr>
<tr>
<td>mb</td>
<td>_message_box(\n: \c);</td>
</tr>
</tbody>
</table>

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.
### 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 view is displayed listing the call stack at the time of the error. Double-click in this view to view source for a call stack entry. The `find_error` command (Macro $\rightarrow$ 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 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 view 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 view is not active by default in debugging mode. To display it, from the main menu, click **Debug → Windows → Loaded Classes**. See "Loaded Classes view" 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:

• DialogEditorSummary
• 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
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
Aligning Controls

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.
<table>
<thead>
<tr>
<th>Assignment</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right mouse click</td>
<td>Displays menu with various dialog editor commands.</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>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.</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Loads form and saves into state file. Under UNIX, this may just list source for the form that can be executed.</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Loads form.</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copies selected controls.</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Pastes controls from the clipboard.</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Cuts selected controls.</td>
</tr>
<tr>
<td>Ctrl+A</td>
<td>Selects all controls with same parent as the already selected control(s).</td>
</tr>
<tr>
<td>Tab</td>
<td>Deselects all controls and selects next control in tab order (p_tab_index).</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Deselects all controls and selects previous control in tab order.</td>
</tr>
<tr>
<td>Left mouse click</td>
<td>Double-click (on control) displays Select an Event Function dialog box for adding or modifying event handlers.</td>
</tr>
</tbody>
</table>

### Miscellaneous Menu Items

The table below shows the miscellaneous menu items.
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
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 event table to an inherited event table (event table not belonging to the control and possible 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 (UNIX: vslick.stu).
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.

```c
#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 **lbutton_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:
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:

```bash
show cmdline
```

The function call syntax is:

```bash
show (cmdline [,arg1 [,arg2 ... [argN]]])
```
cmdline is a string in the format:

```markdown
[option] form_name
```

option can be one of the options in the table below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mdi</td>
<td>Keep the form on top of the MDI window.</td>
</tr>
<tr>
<td>-app</td>
<td>Keep the form on top of the SlickEdit® application window. This allows the MDI window to be displayed on top of the form.</td>
</tr>
<tr>
<td>-xy</td>
<td>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).</td>
</tr>
<tr>
<td>-hidden</td>
<td>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.</td>
</tr>
<tr>
<td>-nocenter</td>
<td>Do not center the form.</td>
</tr>
<tr>
<td>-new</td>
<td>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.</td>
</tr>
<tr>
<td>-reinit</td>
<td>(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</td>
</tr>
</tbody>
</table>
Option | Description
--- | ---
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:**

```bash
// This example requires that you create a form called form1 with a command button and load this file.
#include "slick.sh"
_command mytest()
{
    result=show("-modal form1");
    if (result=="") {
        return(COMMAND_CANCELLED_RC);
    }
    message("_param1="_param1" _param2="_param2");
}

defeventtab form1
ctl.command1.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
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"
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) {

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=0`) until the form is closed. In addition, the `_delete_window` function can be used to return a value (see the previous example).

Modeless example:

```sh
#include "slick.sh"
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=1;
    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:

```sh
#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;  
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® 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.

// 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
  // ctlopenwidth.

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='';
    ctlopenwidth.p_text='';
  }

  static 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.
  ctopenlinesep.on_change()
  {
    if (p_text!='') {
      ctlopenwidth.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.ctlopenwidth.
  ctlopenwidth.on_change()
  {
    if (p_text!='') {
      ctopenlinesep.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 `lbutton_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.lbutton_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.
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
```
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_combobx`. The `_ul2_combobx` 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.

```plaintext
ctltext1.p_text="test";  // Assuming ctltext1 has been declared globally or locally,
                       // lookup the ctltext1 control of the active form to get
x=_control ctltext1;    // Put the window id of the "ctltext1" control of the active
                       // form in the variable x.
```

956
// The variable x does not have to be declared. There are cases where the control keyword is not needed. It is better to always use it so you don't have to worry.

x.p_text="test"; // Set the p_text property of the object referenced by the instance expression x.

(x+1-1).p_text="test"; // Same as previous statement. This shows that any valid Slick-C language expression may be used to get the window id.

x.(x+1-1).x.p_text="test"; // Same as the previous statement but wastes more code space.

// This shows that multiple dots ("." ) may be used in an instance expression.

form_wid=p_active_form; // Get the window id of the active form.
form_wid.ctltext1.p_text="test"; // Lookup ctltext1 as if the object referred to by the variable form_wid was the active object.

p_next.p_next.p_prev.p_prev.p_text="test"; // Waste some code space and access the p_text property of the active object.

p_window_id=_control ctltext1; // Make the ctltext1 control the active object.

p_text="test"; // Access the p_text property of the active object.

_cmdline.p_text="test"; // _cmdline is a constant window id defined in "slick.sh".

// Set the command line p_text property to "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.

```c
#include "slick.sh"
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 \texttt{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 \texttt{Up, Down, Left,} and \texttt{Right} arrow keys.

On the Dialog Editor, click the \texttt{Insert Label Control} button \includegraphics[width=1cm]{label_button.png} 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:

```c
#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(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()
```
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:

```c
#include "slick.sh"
_cmd 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 (Window → SlickEdit Preferences → Keyboard → 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":

```c
#include "slick.sh"
defeventtab form1;
cyllist1.on_change(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=1;  // Enable
            the OK button.
        } else if(!ctlok.p_enabled)
            ctlok.p_enabled=0;  // Disable the OK button.
    }
}
```

The following example illustrates how to resize a dialog box based on the longest item in a list box:

```c
#include "slick.sh"
defeventtab form1;
cyllist1.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_width-
_dx2lx(form_width.p_xyscale_mode,form_width.p_client_width);

p_width=list_width;
form_width.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"
def eventtab form1;
c_tllist1.on_create()
{
    _lbadd_item("item1");
    _lbadd_item("item2");
    p_no_select_color=1;
    p_enabled=0;
    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 "ctlok", and combo box named "ctlcombo1":

def eventtab form1;
ctlok.lbutton_up()
{
    // When the OK button is pressed, you need to save combo box retrieve
    // information.
    _append_retrieve(_control ctlcombo1,ctlcombo1.p_text);
}
ctlok.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:

def eventtab 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"
def eventtab 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.lbutton_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 "ctlok", a combo box
named "ctlcombo1", and another command button named "ctlcommand1":

#include "slick.sh"
def eventtab form1;
ctlok.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=ctlcombol._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.
    ctlcombol.p_cb_list_box._lbadd_item("Hello")
    ctlcombol.p_cb_list_box._lbadd_item("Open");
    ctlcombol.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 "$" specifies that an exact match should be found and
    // not a prefix match.
    int status=ctlcombol._cbi_search("","$");  
    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.
        ctlcombol.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.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGE_OTHER</td>
<td>The p_text property changed, probably because of typing.</td>
</tr>
<tr>
<td>CHANGE_CLINE</td>
<td>The p_text property changed because selected line in list box changed and the list was visible.</td>
</tr>
<tr>
<td>CHANGE_CLINE_NOTVIS</td>
<td>The p_text property changed because a key was pressed which scrolls the list (Up, Down, PgUp, PgDn) while the list was invisible.</td>
</tr>
<tr>
<td>Reason</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CHANGE_CLINE_NOTVIS2</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP_UP</td>
<td>After combo list box is made invisible.</td>
</tr>
<tr>
<td>DROP_DOWN</td>
<td>Before combo list box is made visible.</td>
</tr>
<tr>
<td>DROP_INIT</td>
<td>Before retrieve next/previous. Used to initialize list box before it is accessed.</td>
</tr>
<tr>
<td>DROP_UP_SELECTED</td>
<td>Mouse released while on valid selection in list box and list is visible.</td>
</tr>
</tbody>
</table>

Example:

```
#include "slick.sh"
defeventtab form1;
ctlcomboBox.on_drop_down(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:

```c
#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:

```c
#include "slick.sh"
defeventtab form1;
ctlcombo1.on_change(reason)
{
    if (reason==CHANGE_DRIVE) {
        message("Item selected from list. Current drive is now ",_dvidrive());
    }
}
```
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.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGE_SELECTED</td>
<td>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.</td>
</tr>
<tr>
<td>CHANGE_FILENAME</td>
<td>The _filename() function was called which changed the file names listed.</td>
</tr>
</tbody>
</table>

Example:

```sh
#include "slick.sh"
defeventtab form1;ctlcommand1.lbutton_up()
{
    ctl(list1._flfilename("*.bat","c:\")
}ctl(list1.on_change(reason)
{
    if (reason==CHANGE_FILELNAME) {
        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":

```c
#include "slick.sh"
defeventtab form1;
ctllist1.on_change(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(_dlpath(),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:

```c
// 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 boolean 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 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:

```c
#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("\n");
  }
  p_picture=index;
  p_command="gui_find";
```
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".

```c
#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=0;
    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.
                }
            }
```
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;
clok.on_create()
{
    // Retrieve the previous response to this dialog box.
    _retrieve_prev_form();
}
clok.lbutton_up()
{
    _save_form_response();
    p_active_form._delete_window(1);
}
```
Menus

}
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 (Window → SlickEdit Preferences → Keyboard → 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 (Window → SlickEdit Preferences → 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
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 (UNIX: vslick.stu) 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:

```plaintext
_menu menu_name {
    submenu menu_item, help_command, help_message, categories {
        menu_item, command, categories, help_command, help_message
    }
    submenu
}endsubmenu
}
The table below contains the menu items and their definitions:

<table>
<thead>
<tr>
<th>Menu item</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>menu_item</td>
<td>Menu item name in double quotes. Use &amp; to choose selection character.</td>
</tr>
<tr>
<td>command</td>
<td>Any editor command. Places the cursor on the command line and press ? to list all editor commands.</td>
</tr>
<tr>
<td>help_command</td>
<td>Command to be executed when F1 is pressed. Usually it is a help or popup_imessage command.</td>
</tr>
<tr>
<td>categories</td>
<td>Specifies zero or more help categories in double quotes. Multiple help categories are separated with</td>
</tr>
<tr>
<td>help_message</td>
<td>A single line message in double quotes displayed on message line.</td>
</tr>
</tbody>
</table>

Example of a menu definition:

```plaintext
_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|nr|only", "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:

```plaintext
#include slick.sh
```
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:

```c
#include "slick.sh"
static boolean 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 the editor control does not have a selection:
    if (!target_wid.select_active2()) {
        //say('disabled at h4');
        return(MF_GRAYED);
    }
    //say('h6');
    if (gSomeOtherState) {
        //say('disabled at h6');
        return(MF_GRAYED);
    }
}
```
Placing a Menu Bar on a Form

The following sample code shows how to add a menu on a form as a menu bar:

```c
#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);
}
```

Creating and Editing Menu Resources
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:

```bash
#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.

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_textbox_form</td>
<td>Displays a variable number of text boxes or combo boxes.</td>
</tr>
<tr>
<td>_sellist_form</td>
<td>Displays a list box, an optional combo box, and a variable number of command buttons.</td>
</tr>
<tr>
<td>_open_form</td>
<td>Used to open and save files that does not have the advanced controls.</td>
</tr>
<tr>
<td>_edit_form</td>
<td>Used to open and save files that has the advanced controls used for the <strong>File → Open</strong> dialog box.</td>
</tr>
<tr>
<td>_font_form</td>
<td>Used to prompt for a font.</td>
</tr>
<tr>
<td>_choose_font</td>
<td>(Non-UNIX platforms only) Dialog box built-in to operating system used to prompt for a font.</td>
</tr>
<tr>
<td>_printer_setup</td>
<td>(Non-UNIX platforms only) Dialog box built-in to operating system used for printer setup.</td>
</tr>
</tbody>
</table>

If a key displays a dialog box, you can find out the command the key executes by using the Key Bindings options screen (**Window → SlickEdit Preferences → 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `_str center (_str string,int width [,str pad_ch])` | Returns `string` padded evenly on left and right with spaces or a character you choose with the optional argument `pad_ch`.
| `_dec2hex (long number [,int base])` | Returns `number` converted to `base` specified.
| `_str expand_tabs (_str string [,int start [,int count [,str option]]])` | Very similar to `substr` function except that this function supports tab characters very well.
| `_str field(_str string,int width)` | Returns `string` padded with trailing spaces to `width` characters.
| `long hex2dec(_str number [,int base])` | Returns `number` converted to `base` specified.
| `_str indent_string(int width)` | If indent with tabs is on, a string of tabs of length `width` is returned. Otherwise, a string of spaces of length `width` is returned.
| boolean isalnum(_str ch) | Returns non-zero value if `ch` is a numeric or alphabetic character.
| boolean isalpha(_str ch) | Returns non-zero value if `ch` is an alphabetic character.
| boolean isdigit(_str ch) | Returns non-zero value if `ch` is a numeric character.
| boolean isinteger(_str string) | Returns non-zero value if `string` is a valid int. If `string` is floating point number, 0 is returned.
| boolean isnumber(_str string) | Returns non-zero value if `string` is a valid double (floating pointer number).
| `_str last_char(_str string)` | Returns last character of `string`. If `string` is null, the space character is returned.
| `int lastpos(_str needle [,str haystack [,int start [,str options]]])` | Returns the position (1..length(`haystack`)) of the last occurrence of `needle` in `haystack`. If `needle` is
## String Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>not found, 0 is returned. Regular expressions are supported.</strong></td>
<td></td>
</tr>
<tr>
<td>int <strong>length</strong>(_str string) )</td>
<td>Returns the number of characters in string.</td>
</tr>
<tr>
<td>_str <strong>lowcase</strong>(_str string) )</td>
<td>Returns string converted to lowercase.</td>
</tr>
<tr>
<td>_str <strong>number2onoff</strong>(_str number) )</td>
<td>Returns off if number==0. Otherwise on is returned.</td>
</tr>
<tr>
<td>_str <strong>number2yesno</strong>(_str number) )</td>
<td>Returns N if number==0. Otherwise Y is returned.</td>
</tr>
<tr>
<td>parse <strong>expr with template</strong></td>
<td>Breaks apart the expression expr given into variables that appear in template, and much more. See parse Statement for more information.</td>
</tr>
<tr>
<td><strong>boolean parseoption</strong>(_str &amp; cmdline , _str option_ch) )</td>
<td>Strips + or - option from cmdline. Returns non-zero number if option_ch was found.</td>
</tr>
<tr>
<td>int <strong>pos</strong>(_str needle , _str haystack , [int start [ , _str options ]]]))</td>
<td>Returns the position (1..length(haystack)) of the first occurrence of needle in haystack. If needle is not found, 0 is returned. Regular expressions are supported.</td>
</tr>
<tr>
<td><strong>boolean setonoff</strong>(_str &amp; name , _str value) )</td>
<td>Sets name to 1 or 0 corresponding to value=on or value=off. Returns 0 if input value is valid. Displays message if value is not on or off.</td>
</tr>
<tr>
<td><strong>boolean setyesno</strong>(int &amp; name , _str value) )</td>
<td>Sets name to 1 or 0 corresponding to value=Y, Yes or value=N, No. Returns 0 if input value is valid. Displays message if value is not Y or Yes, N or No.</td>
</tr>
<tr>
<td>_str <strong>translate</strong>(_str string , _str replace_string , _str search_string , _str search_options) )</td>
<td>Returns string with all occurrences of search_string replaced with replace_string.</td>
</tr>
<tr>
<td>_str <strong>strieq</strong>(_str string1 , _str string2) )</td>
<td>Returns true if string1 matches string2 when case is ignored.</td>
</tr>
<tr>
<td>_str <strong>strip</strong>(_str string , _str ltb [ , _str strip_char ]))</td>
<td>Returns string stripped of leading and/or trailing strip_char.</td>
</tr>
<tr>
<td>_str <strong>strip_filename</strong>(_str filename , 'P'</td>
<td>'D'</td>
</tr>
<tr>
<td>_str <strong>strip_last_word</strong>(_str &amp; line) )</td>
<td>Returns the last space delimited word in line. The</td>
</tr>
</tbody>
</table>
## Search Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>str strip_options</strong>(_str cmdline , _str &amp; options)</td>
<td>Returns cmdline without words that start with the characters -, +, or [. options variable is set to stripped option words.</td>
</tr>
<tr>
<td><strong>str substr</strong>(_str string , int start [, int length [:str pad]])</td>
<td>Returns length characters of string beginning at start. By default, length defaults to rest of string. If length is greater than length of string, the return string is padded with blanks or pad character if specified.</td>
</tr>
<tr>
<td><strong>str translate</strong>(_str string [:str output_table [:str input_table [:str pad]]])</td>
<td>Returns string with characters translated according to arguments.</td>
</tr>
<tr>
<td><strong>str upcase</strong>(_str string)</td>
<td>Returns string converted to uppercase.</td>
</tr>
<tr>
<td><strong>int verify</strong>(_str string , _str reference [, M] [,int start])</td>
<td>Returns the position (1..length(string)) of first character not matching or matching a character in reference. 0 is returned on failure.</td>
</tr>
<tr>
<td><strong>str word</strong>(_str string , int Nth)</td>
<td>Returns the Nth space or tab-delimited word in string. Is returned if the Nth word does not exist.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gui_find</code></td>
<td>Displays Find and Replace view open to the Find tab, and performs search using the <code>find</code> or <code>_mffind</code> functions.</td>
</tr>
<tr>
<td><code>gui_replace</code></td>
<td>Displays Find and Replace view open to the Replace tab, and performs search using <code>gui_replace2</code> or <code>_mfreplace</code> functions.</td>
</tr>
<tr>
<td><code>gui_replace2</code></td>
<td>Performs a search and replace based on arguments given. This function is very similar to the <code>replace</code> function, except that this function uses a dialog box to prompt the user where to replace.</td>
</tr>
<tr>
<td><code>find_next</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>find</code></td>
<td>Performs search based on arguments given.</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>Performs a replace based on arguments given. The user is prompted where to replace through the message line.</td>
</tr>
<tr>
<td><code>_mffind</code></td>
<td>Performs a multiple file and buffer search based on the arguments given.</td>
</tr>
<tr>
<td><code>_mfreplace</code></td>
<td>Performs a multiple file and buffer search based on the arguments given.</td>
</tr>
<tr>
<td><code>search</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

```c
def main()
{
    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

        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.
    for (;;)
```
Search Functions

{
    if (status) {
        break;
    }

get_line(line); // Place the cursor at the end of the line so no
                 // more occurrences can be found on this 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"

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(temp_view_id);

    if (orig_view_id=="") {

return("\n");
}
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(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();
}

Search Functions

990
// 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_alloc_selection</td>
<td>Returns a handle to a selection.</td>
</tr>
<tr>
<td>_free_selection</td>
<td>Frees a selection associated with the selection handle given. Note that it cannot free the active selection. To free the active selection, use _show_selection first.</td>
</tr>
<tr>
<td>_show_selection</td>
<td>Used to make another selection the active selection.</td>
</tr>
<tr>
<td>_duplicate_selection()</td>
<td>Returns the actual handle number of the active selection.</td>
</tr>
</tbody>
</table>

Example:

```c
// 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.

```sh
#include "slick.sh"

_str hex_filt(string);

def main()
{
    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(string)
{
    line="";
    for (i =1;i<=length(string);++i) {
        line=line:+dec2hex(_asc(substr(string,i,1))):",";
    }
    return(line);
}
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:

```c
// 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:

```c
// 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.
Function | Definition
--- | ---
_default_option(VSOPTIONZ_SPECIAL_CHAR_ XLAT_TAB) | All other options for this function are UTF-8.

All seek functions: goto_point(), _QROffset(), _GoToROffset, _nrseek(), point(), and seek() | All seeking is done on raw data. Buffers need to be loaded in the same raw format so that seek functions work.

All _rawXXX() or XXX_raw() functions | Unlike the C API, the Slick-C functions get_text() and _expand_tabs() 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos()</td>
<td>When p_rawpos appended to options argument.</td>
</tr>
<tr>
<td>lastpos()</td>
<td>When p_rawpos appended to options argument.</td>
</tr>
<tr>
<td>upcase()</td>
<td>When p_UTF8 property given as second argument.</td>
</tr>
<tr>
<td>lowcase()</td>
<td>When p_UTF8 property given as second argument.</td>
</tr>
<tr>
<td>parse</td>
<td>When p_rawpos appended to options of search argument.</td>
</tr>
</tbody>
</table>
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()`, `vsQSelectedText()` - 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:

```c
// 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 \texttt{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 \texttt{arg(1)}. For example, invoking the edit command \texttt{e file1 file2} calls the \texttt{e} command with \texttt{file1 file2} in \texttt{arg(1)}. The \texttt{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 \texttt{arg} function with no parameters returns the number of parameters with which the command or procedure was called.

Example:

```c
#include "slick.sh"

// This command supports completion on a filename followed by an
// environment variable argument.
_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);
}
```

The string constant expression given to the \texttt{name_info} keyword is used for argument completion, restricting when the command can be executed, and a few other options.

get_string Procedure

The \texttt{get_string} procedure reads a single argument from the user.

Example:

```c
#include "slick.sh"
_command 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.
    }
    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:

```c
#include "slick.sh"
_command 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:

```c
_exit_cleanup_stuff()
{
    messageNwait("Got here");
}
```
State File Caching

By default, a module, event, dialog box template, or picture from `vslick.sta` (UNIX: `vslick.stu`) 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 (Window → SlickEdit Preferences → 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

<table>
<thead>
<tr>
<th>Window Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_window_x, p_window_y</code></td>
<td>Top left coordinates of window.</td>
</tr>
<tr>
<td><code>p_tile_id</code></td>
<td>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 <code>tile_id</code>. A zoomed window has a negative <code>tile_id</code>.</td>
</tr>
<tr>
<td><code>p_x</code></td>
<td>The top left x position of window.</td>
</tr>
<tr>
<td><code>p_y</code></td>
<td>The top left y position of window.</td>
</tr>
<tr>
<td><code>p_height</code></td>
<td>Window height.</td>
</tr>
<tr>
<td><code>p_width</code></td>
<td>Window width.</td>
</tr>
<tr>
<td><code>p_view_id</code></td>
<td>Pointer to active view.</td>
</tr>
<tr>
<td><code>p_next (next_window)</code></td>
<td>Window id of next window.</td>
</tr>
<tr>
<td><code>p_prev (prev_window)</code></td>
<td>Window id of previous window.</td>
</tr>
<tr>
<td><code>p_child</code></td>
<td>Window id of child window.</td>
</tr>
</tbody>
</table>

### View Properties

<table>
<thead>
<tr>
<th>View Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>block, line within block</code></td>
<td>Accessible via <code>point</code> and <code>goto_point</code>.</td>
</tr>
</tbody>
</table>
## Buffer Properties

<table>
<thead>
<tr>
<th>View Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_line</td>
<td>Line number of current line.</td>
</tr>
<tr>
<td>p_col</td>
<td>Column position within current line (1..2 billion).</td>
</tr>
<tr>
<td>p_left_edge</td>
<td>Column scroll position.</td>
</tr>
<tr>
<td>p_cursor_x</td>
<td>Text cursor x position.</td>
</tr>
<tr>
<td>p_cursor_y</td>
<td>Text cursor y position.</td>
</tr>
<tr>
<td>p_window_id</td>
<td>Window id.</td>
</tr>
<tr>
<td>p_buf_id</td>
<td>Buffer id.</td>
</tr>
</tbody>
</table>

## Buffer Properties

<table>
<thead>
<tr>
<th>Buffer Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_buf_name</td>
<td>Name of buffer.</td>
</tr>
<tr>
<td>p_buf_flags</td>
<td>Indicates whether a buffer is hidden and may specify other buffer options.</td>
</tr>
<tr>
<td>p_Noflines</td>
<td>Number of lines in file.</td>
</tr>
<tr>
<td>p_modify</td>
<td>Non-zero indicates buffer has been modified.</td>
</tr>
<tr>
<td>pMargins</td>
<td>String containing left, right, and new paragraph margins (1..2 billion).</td>
</tr>
<tr>
<td>p_tabs</td>
<td>String containing up to 2 billion tab stops.</td>
</tr>
<tr>
<td>p_mode_name</td>
<td>Name of current mode.</td>
</tr>
</tbody>
</table>
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:

```c
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 with 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;
}

Defining Stack Routines
struct TABLE_ENTRY {
    _str name;
    int value;
};

// defmain is the main entry pointer for a Slick-C batch/script macro.
defmain()
{
    TABLE_ENTRY table[];

    // Table file should exist.
    // NOTE: (TABLE_ENTRY []) is type compatible with (typeless []).
    status = ReadTable("table", table);
    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`:

```c
#include 'slick.sh'

_command int proc_search(...) name_info(','VSARG2_READ_ONLY|
               VSARG2_REQUIRES_EDITORCTL|
               VSARG2_MARK)
{

    // Save the original cursor position to restore later.

    typeless original_position;
    save_pos(original_position);

    // Prompt the user for a search string, and search options.
```

1012
Searching for a String Within a Current Function

_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
selection.

if ( status ) {
   // If the search string was not found, deselect and return the cursor
to
   // the original position.
   deselect();
   restore_pos(original_position);
}
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 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>(Default) Forward search.</td>
</tr>
<tr>
<td>-</td>
<td>Reverse search.</td>
</tr>
<tr>
<td>&lt;</td>
<td>(Default) Place cursor at beginning of string found.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Place cursor after end of string found.</td>
</tr>
<tr>
<td>E</td>
<td>(Default) Case-sensitive search.</td>
</tr>
<tr>
<td>I</td>
<td>Case-insensitive search.</td>
</tr>
<tr>
<td>M</td>
<td>Search within visible mark.</td>
</tr>
<tr>
<td>H</td>
<td>Find text in hidden lines.</td>
</tr>
<tr>
<td>R</td>
<td>Search for SlickEdit® regular expression.</td>
</tr>
<tr>
<td>U</td>
<td>Interpret string as a UNIX regular expression.</td>
</tr>
<tr>
<td>B</td>
<td>Interpret string as a Brief regular expression.</td>
</tr>
<tr>
<td>N</td>
<td>(Default) Do not interpret search string as a regular search string.</td>
</tr>
<tr>
<td>@</td>
<td>No error message.</td>
</tr>
<tr>
<td>W</td>
<td>Limits search to words such as variable names.</td>
</tr>
<tr>
<td>,</td>
<td>Delimiter to separate ambiguous options.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_str cur_word( int &amp; start_col [, _str from_cursor ] )</code></td>
<td>Gets the current word at cursor.</td>
</tr>
<tr>
<td>int delete_line()</td>
<td>Deletes the current line.</td>
</tr>
<tr>
<td>void _delete_text( int len )</td>
<td>Delete len bytes starting from the cursor position.</td>
</tr>
<tr>
<td>void get_line( _str &amp; line )</td>
<td>Retrieves current line.</td>
</tr>
<tr>
<td><code>_str get_text([int count [,int seek_pos ]])</code></td>
<td>Gets a stream of text starting at current line.</td>
</tr>
<tr>
<td>void keyin( _str string )</td>
<td>Inserts string of characters as if typed from the keyboard.</td>
</tr>
<tr>
<td>void insert_line( _str line )</td>
<td>Inserts line after current line.</td>
</tr>
<tr>
<td>void _insert_text( _str string )</td>
<td>Inserts string at cursor position.</td>
</tr>
<tr>
<td>void replace_line( _str line )</td>
<td>Replaces current line.</td>
</tr>
</tbody>
</table>

Common Functions for Navigating Buffers

The table below contains functions that can be used for navigating buffers.
<table>
<thead>
<tr>
<th>Function</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>int up( [int num ] )</td>
<td>Moves cursor up num lines, or one line if no value passed in.</td>
</tr>
<tr>
<td>int down( [int num ] )</td>
<td>Moves cursor down num lines, or one line if no value passed in.</td>
</tr>
<tr>
<td>void left()</td>
<td>Moves cursor one position to the left.</td>
</tr>
<tr>
<td>void right()</td>
<td>Moves cursor one position to the right.</td>
</tr>
<tr>
<td>void top()</td>
<td>Places cursor at first line and first column of buffer.</td>
</tr>
<tr>
<td>void bottom()</td>
<td>Places cursor at end of last line of buffer.</td>
</tr>
<tr>
<td>void _begin_line()</td>
<td>Places cursor at the beginning of the current line.</td>
</tr>
<tr>
<td>void _end_line()</td>
<td>Places cursor after the end of the current line.</td>
</tr>
</tbody>
</table>

### 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:

```c
_command escape_slash() {

    _str myLine;

    getline(myLine); // Set string szLine to the current line.

    myLine = stranlaze(myLine, "\\\\", \\\\"); // Replace slash with double slashes.

    replace_line(myLine); // Replace the line in the buffer.
}
```

The above command accepts the following line of code:

```c
myDirectory = "C:\Data\Corporate\Internal";
```
and replaces it with:

```c
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:

```c
_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 (**Display** → **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 (**Window** →
SlickEdit Preferences → Languages → [Language Category] → [Language] → General).

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 **Display** → **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 **Display** → **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 **Display** → **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 **Display** → **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:

   ```c
   #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...
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 **Window** → **SlickEdit Preferences** → **Keyboard** → **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`
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..\255 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:

```c
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;
ctlcombo1.on_change()
{
}
ctlcombo1."c-s-a"() // Define event handler for Ctrl+Shift+A
{
}
ctlcombo1."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") ) {
                ...
            }
    }
}
```

---

**Keys**

1027
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_load_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.
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.

**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.

**Parameter Info**

Automatically displays the prototype for a function when you type a function operator, and highlights the current argument within the displayed prototype.

**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, 920
#pragma, 914
#region, 920
3 Way Merge Dialog, 509
[Language] File Extensions dialog, 587
[Language] Referenced in Languages dialog, 587
__command, 904
__control, 956
__inherit, 926
__str center, 984

A
About SlickEdit Core, 20
ActionScript Formatting Options, 286
Active form, 956
Active object, 956
Ada Beautifier dialog box, 267
Ada Formatting Options, 266
Adaptive Formatting, 176
Adaptive Formatting Preferences
(Language-Specific), 592
Adaptive Formatting Results dialog, 179
Adaptive Formatting Statistics dialog, 182
Add (Code Templates Add New Item dialog box), 212
Add File dialog box (Code Templates), 210
Add import, 286
Add New Item dialog box (Code Templates), 211
Add New Language dialog, 259
Add Parameter dialog box (Code Templates), 211
Add Tag dialog, 311
Add Tag File dialog, 106
Adding a File Type Filter, 633
Adding completion to command, 905
Advanced tab (FTP Preferences), 640
after header, 56
alias
Surround With, 234
Alias Preferences (Language-Specific), 601
aliases, 189
creating from selection, 198
directory aliases, 190
escape sequences, 193
expansion, 189
expansion option, 604
language-specific aliases, 191
parameter prompting, 197
align on equal, 293
align on parens, 293
Aligning Controls, 939
All Languages Preferences, 585
Allow drag/drop of text, 565
AND operator, 883
anonymous e-mail address, 640
Anonymous Unions, 873
ANSI C Formatting Options, 286
Ant Options, 274
Appearance Preferences, 530
Application Preferences, 633
arg built-in, 903
Argument completion preferences for, 604
Argument Declarations, 902
Array length method, 867
Arrays, 867
Assignment Operator, 892
Associate with language, 262
Associative array variables, 879
At left margin, 595
At level of indent, 596
auto, 589
auto caps mode, 588
auto deselect, 575
Auto Enable Properties dialog box, 504
auto increment search results window, 485
Auto List Compatible Parameters, 100
auto list members, 98
auto merge, 510
auto parameter information, 99
Auto read only, 630
Auto reload, 629
Auto reload all files if current file changed, 630
Auto reload current file only, 630
Auto reload timeout (ms), 630
Auto Symbol Translation, 330
Auto symbol translation (option), 308
auto validate on open, 308
Auto-Close Options
Options, 579
Auto-Close Options (Language-Specific) Options, 607
Auto-Complete, 185
Auto-Complete Preferences (Language-Specific), 602
Auto-display parameter, 610
Auto-insert matching parameter, 610
Auto-list compatible parameters, 611
Auto-list compatible values, 605
Auto-list members, 605
Auto-Updated Tag Files, 103
autodecl, 915
autodeclctls, 915
autodeclvars, 915
automatic directory mapping, 363
automatic width (comment wrapping), 599
Automatically expand doc comments, 596
AutoSave activated, 631
AutoSave directory, 632
AutoSave File Preferences, 631
Available SlickEdit® Core Views, 44
AWK Formatting Options, 286

B
Background Process, 114
Background tagging of open files, 569
Background tagging of other files, 569
Backspace in replace mode, 559
Backspace key, 559
Backspace over tab, 559
backward, 483
Base Classes dialog box, 161
base file, 364
basic editing, 116
  cut, copy, paste, move, 127
batch macro, 852
BBEdit emulation keys, 819
  clipboard, 824
  command line and text box editing, 822
  compiling and programming, 825
  cursor movement, 819
  debugging, 826
deleting, 821
files and buffers, 823
inserting, 820
macros, 827
miscellaneous, 827
searching, 822
selecting, 821
windowing, 824
beautifier, 251
Beautifier Settings

Export/Import Preferences, 654
beautify, 290
Beautify Menu, 508
Beautifying code, 251
before footer, 56
begin end structure matching, 139
  setting match style, 140
viewing/defining, 140
binary character searching, 390
Bind Key dialog, 558
bind macro to key, 430
bitmaps (adding to image control), 973
bitmaps (adding to list box), 964
Bitwise AND, 883
Bitwise OR, 883
Bitwise XOR, 883
Block Comments
  configuration, 596
block cursor option, 531
Block insert mode, 134
block selections, 120
block statement, 893
blog, 6
Bookmark Stack dialog, 492
bookmarks, 238
  named, 238
  pushed, 241
boolean, 892
Bounds, 689
breadcrumbs, 241
break (primitive), 897
Brief emulation keys, 736
  clipboard, 738
  command line, 739
  compiling and programming, 741
cursor movement, 736
debugging, 742
deleting, 737
files and buffers, 740
inserting, 737
macros, 744
miscellaneous, 744
searching, 738
selecting, 743
text box editing, 739
windowing, 741
Brief regular expressions, 383
  examples, 389
Buffer cache size, 634
Build tag file (C++ Compiler), 279
Build workspace tag file with references, 571
building tag files, 106
built-in functions, 910
byte offset navigation, 143

C
C# Formatting Options, 286
c/c++ beautifier, 275
C/C++ Compiler Properties, 278
C/C++ preprocessing, 279
cache size (for buffers), 634
cache size (for tagging), 634
calculating (see mathematics), 422
calculator dialog box, 422
call tree, 160
Cancel button (adding), 943
case for hex values, 320
case sensitive, 371
case statement, 900
casting types, 888
Categories (Code Templates Add New Item dialog box), 211
Categories (Template Manager dialog box), 209
CFML Formatting Options, 319
CFScript Formatting Options, 286
Ch Formatting Options, 286
changing emulations, 63
Changing the Configuration Directory, 673
can be selected, 118
inclusive option, 575
characters (identifiers), 614
characters (inserting literal), 134
Check Box control, 962
CICS Formatting Options, 281
Class Exclusion Manager dialog, 148
Class view, 146
  exclusion manager, 148
  filtering hierarchy pane, 148
  filtering/sorting members pane, 149
clear bookmark, 240
Clear Modified Lines, 167
Click past end of line, 566
Clipboard Inheritance(R), 951
clipboard (how to use), 129
Close after find/replace, 578
Closing a dialog box, 943
closing files
  automatically, 138

COBOL Formatting Options, 281
code folding, 170
code navigation, 137
Code Templates, 202
  Add New Item dialog box, 211
  Creating a multi-file template, 214
  Creating templates, 204
  Export/Import Preferences, 654
  Global Substitution Parameters, 210
  Locating Templates, 212
  Manually creating a template, 212
  Metadata file reference, 215
  Organizing templates, 207
  Pre-defined substitution parameters, 205
  Substitution parameters, 205
  Template Manager dialog box, 208
  Template Options dialog box, 210
  User templates (Locating), 212
CodeWarrior emulation keys, 795
  clipboard, 798
  command line and text box editing, 798
  compiling and programming, 801
cursor movement, 795
debugging, 802
deleting, 796
defining templates, 800
inserting, 796
macros, 803
miscellaneous, 803
searching, 797
selecting, 797
windowing, 800
CodeWright emulation keys, 804
  clipboard, 807
  command line and text box editing, 807
  compiling and programming, 809
cursor movement, 804
debugging, 810
deleting, 805
defining templates, 809
inserting, 805
macros, 811
miscellaneous, 812
searching, 806
selecting, 806
windowing, 809
collapse code block, 171
Color Coding
  adding keywords, 95
advanced configuration, 96
Comments Tab, 620
configuration, 96
creating support for a new language, 95
introduction, 83
Language Tab, 618
Numbers Tab, 614
Preferences (Language-Specific), 612
Strings Tab, 617
Tags Tab, 623
Tokens Tab, 613
use, 95
Color Coding Search
Preferences dialog box, 482
Color Inheritance, 85
Color Picker dialog, 84
Color Preferences, 532
Color Settings dialog box, 532
colors, 83
Color Coding, 83
customizing, 83
for embedded languages, 86
of special characters, 168
schemes, 85
Symbol Coloring, 83, 86
Columns per Line, 447
Combo Box control, 965
Command Button control, 962
Command Line, 43
command line
(see SlickEdit Core command line), 46
Command line prompting, 561
Command Line Switches, 48
commands, 50
binding to keys, 69
defining with Slick-C, 903
see also SlickEdit Core command line, 46
viewing associated key bindings, 48
Commands, Built-Ins, Defs (Differences), 910
Comment Preferences (Language-Specific), 593
comment styles (C++ language constructs), 861
Comment width, 462
Comment Wrap Preferences (Language-Specific), 597
comment wrapping, 250
enable, 598
reflowing comments, 250
sync vertical line, 599
width settings, 598
comments, 245
block comments, 245
commenting lines, 245
configuration, 245
configuring block comments, 594
configuring doc comments, 596
configuring line comments, 595
creating doc comments, 246
doc comment examples, 247
Doxygen, 248
editing preferences, 596
Javadoc, 247
line comments, 245
reflowing, 250
removing, 245
string editing, 250
wrapping, 250
XMLdoc, 247
Comments tab (Beautifier), 293
Comments tab (Color Coding Setup), 620
common keys (redefining), 54
Compare a Local Backup History Element with the Current Version Using DIFFzilla®, 366
Compare file contents before auto reload, 629
Compare Two Backup History Elements Using DIFFzilla®, 366
comparing files, 357
comparing folders (directories), 358
comparing parts of files, 359
comparing symbols, 359
Compiler Properties (Language-Specific), 625
Compiling macros, 930
completion, 185
Adding to command, 905
Auto-Close, 187
word/variable, 188
Completion (Context Tagging), 101
Completion choice, 606
Concatenation, 885
certainty level, 182
config (see configuration), 673
configuration, 673
changes not on menu, 667
configuration variables, 669
environment variables, 665
set command, 668
system config files, 675
table of sys config files, 675
table of user config files, 674
default local directory, 640
DefaultName element (Code Templates metadata file reference), 216
defeventtab, 924
defeventtab primitive, 924
defining constants, 863
defining event tables, 926
defining functions, 901
defining keys, 924
defining procedures, 901
Defining Special Characters, 168
definit, 928
defload, 928
defmain, 911
Delete, 276
delete bookmark, 240
Delete Code Block dialog, 237
Delete key, 559
Delete Menu, 450
delete selection, 575
Deleting a template (Template Manager dialog box), 209
deleting code blocks, 237
deprecation, 915
Derived Classes dialog, 161
Description element (Code Templates metadata file reference), 217
deselect after paste, 575
Details tab (Template Manager dialog box), 209
Dialog Box Retrieval, 975
dialog boxes (forms for macros), 983
Dialog Editor, 937
Dialog Editor Properties dialog box, 938
Dialogs, 43
Diff dialog box, 356
diff options tab, 516
Diff Setup dialog box, 516
diffing (see DIFFzilla), 356
DIFFzilla, 356
  comparing folders (directories), 358
  comparing parts of files, 359
  comparing symbols, 359
  comparing two files, 357
dialog, 510
directory mapping, 363
dynamic difference editing, 356
Files tab, 511
generating file lists, 361
options, 514
path info, 511
  types, 511
directory aliases, 190
  creating, 190
  embedding env variables, 191
  using, 190
Directory List Box control, 970
Directory Location, 673
directory mapping, 363
Display after idle, 573
Display Menu, 471
display tool tips, 78
Displaying Dialog Boxes, 944
do (loop), 894
do not upload, 640
Doc Comment Editor dialog, 248
doc comments, 246
  configuration, 596
document math, 424
documentation, 6
documentation conventions, 7
documentation feedback, 6
Doxygen comments, 248
drag and drop text, 129
drag/drop, 565
Draw box only, 532
Drive List control, 969
Dynamic Surround, 229
E
eat spaces after, 295
Edit, 276
Edit Alias Parameter dialog, 602
Edit Menu, 448
drag other copy unicode as ucn menu, 662
Edit Other Menu, 450
Edit Select Menu, 449
Editing an existing template (Template Manager dialog box), 208
Editing Preferences, 562
editing recorded macros, 431
Editor Control, 961
Elements (Code Templates metadata file reference), 216
else on same line as }, 291
embedded asp dialect, 320
embedded language color, 86
embedded languages, 14
  HTML, 15
  Perl, 15
  UNIX, 15
 embedding env variables in aliases, 191
 Emulation Preferences, 555
 emulations, 62
  changing, 63
  determining keys/functions, 64
  supported, 62
  tables of keys, 709
 Emulations (list), 62
 Enable auto-completion, 603
 enable comment wrap, 598
 Enable soft wrap, 600
 encoding, 660
  SBCS/DBCS, 660
  Unicode, 660
  UTF-8, 660
 Encoding (global option), 628
 END command saves the file, 689
 End key, 559
 end of line, 168
 Enter Alias Parameter dialog, 197
 Enter key, 559
 Enter New Alias Name dialog, 192
 Enter New Lexer Name dialog, 95
 Enter New Tag Name dialog, 338
 Enter places cursor in prefix area, 689
 Entire block comment (Comment Wrap tab), 462
 entity reference translation, 330
 Enumerate dialog box, 454
 Enumeration, 122
 environment variables, 665
  set command, 668
  setting in vslick.ini, 667
  VSLICKINCLUDE, 921
  VSLICKPATH, 921
  VST, 915
 Epsilon emulation keys, 746
  argument and repeating a key, 757
  clipboard, 749
  command line, 754
  compiling and programming, 751
  cursor movement, 746
  debugging, 752
  deleting, 748
  files and buffers, 750
  inserting, 747
  macros, 753
  miscellaneous, 755
  searching, 748
  selecting, 749
  text box editing, 754
  windowing, 751
 escape sequences, 193
 evaluating regular expressions, 381
 Event driven dialog (event tables), 926
 Event tables, 924
 event-driven dialog boxes, 848
 events and event tables, 924
 examples of Brief reg expressions, 389
 examples of Perl reg expressions, 397
 examples of SlickEdit reg expressions, 405
 examples of UNIX reg expressions, 412
 Executing Eclipse Commands From the SlickEdit® Editor, 74
 Executing programs from macro, 998
 Exit SlickEdit Core on AutoSave, 632
 expand code block, 171
 Expanded text, 604
 Export Groups, 651
 export key bindings, 71
 Export Preferences, 650
 export to HTML, 318
 expressions (unary operators in), 882
 Extend line comments, 597
 extension-specific aliases
   creating, 192
   escape sequences, 193
   parameter prompting, 197
 Extensionless File Manager, 263, 583
 Extract method (Quick Refactoring), 253

F
 f command, 138
 Fast auto read only, 630
 Fast line count on partial load, 628
 Feature Notification, 114
 Feature Notifications
   options, 635
 features (new), 5
 File element (Code Templates metadata file reference), 217
 File Extension Manager, 261, 583
 file filters, 632
 File History and Backups, 366
 File List Box control, 970
file lists (generating), 361
File locking, 628
File Menu, 439
File Preferences, 625
File Preferences (Language-Specific), 624
file search order, 678
files
   auto-close, 138
   comparing parts of, 359
   comparing two files, 357
   configuration files and directories, 673
   encoding, 660
   file filters, 632
   FTP, 374
   generating lists, 361
   merging, 363
   navigating between, 142
Files element (Code Templates metadata file reference), 219
files menu, 439
Files of Type Filter Preferences, 632
Files tab (Template Manager dialog box), 209
Fill Selection, 124
Filtering marked text, 994
filter_selection procedure, 994
find
   see also searching, 344
Find and Replace view, 479
Find in Files tab (Find and Replace), 483
Find references incrementally (faster), 571
Find Symbol view, 490
Find tab (Find and Replace), 481
find_error command, 932
find_proc command, 931
firewall proxy tab, 641
First line is top, 595
fixed right margin (comment wrapping), 599
fixed width (comment wrapping, 598
Floating point numbers, 863
follow characters, 614
Font Configuration dialog box, 552
Font Preferences, 552
fonts, 80
   changing, 80
   recommended, 81
   UNIX, 82
footer (printing), 56
for (loop), 894
force parens on return, 295
foreach (loop), 895
foreground search, 486
form feed after last page, 446
format file, 625
Format Menu, 457
Formatting Preferences (Language-Specific), 591
Fortran Formatting Options, 281
forums, 6
fp command, 138
Frame control, 962
FTP, 372
   configuration options, 374
   connecting, 373
   disconnecting, 374
   FTP Profile Manager, 372
   opening files, 374
   views, 372
ftp connection profile, 372
FTP Default Preferences, 639
FTP Default Preferences Advanced Tab, 640
FTP Default Preferences Debug Tab, 643
FTP Default Preferences Firewall/Proxy Tab, 641
FTP Default Preferences SSH/SFTP Tab, 643
FTP Preferences Advanced Tab, 640
ftp preferences firewall/proxy tab, 641
ftp preferences ssh/sftp tab, 643
FTP view, 372
function braces, 291
Function Prototypes, 909
functions (defining), 901
functions (finding), 910

G
Gauge control, 971
General Appearance Preferences, 530
General Editing Preferences, 563
General Preferences (Language-Specific), 586
generate file list, 361
generate references, 157
get_string procedure, 1000
Global Alias Preferences, 579
global aliases, 190
GNU Emacs emulation keys, 772
   argument and repeating a key, 783
   clipboard, 775
   command line, 780
   compiling and programming, 777
cursor movement, 772
debugging, 778
deleting, 774
files and buffers, 776
inserting, 773
macros, 779
miscellaneous, 781
searching, 774
selecting, 774
text box editing, 779
windowing, 777
Go to Bookmark feature, 240
go to definition, 137
navigation option, 611
go to reference, 137
Go to references only lists references, 571
Grid Settings dialog, 502

H
hash tables, 868
hash tables editing, 500
header (printing), 56
header files (Slick-C), 841
Header Files, Including, 920
Help Menu, 656
help resources, 6
help system, 7
Hex character code, 862
hex view, 172
hexadecimal display, 135
hexadecimal numbers, 863
Hide mouse pointer, 532
Highlight matches, 483
Highlight matching blocks, 565
Highlight matching symbols under cursor, 612
Highlight references in editor, 571
highlight symbols, 355
History retrieval, 578
Home key, 560
HOME Key Configurations, 54
Home places cursor on command line, 689
host name of firewall, 642
HTML, 318
beautifying, 321
browser configuration, 318
exporting current file, 318
HTML Beautifier Advanced tab, 328
HTML Beautifier Attributes Values tab, 326
HTML Beautifier Comments tab, 327
HTML Beautifier dialog box, 321
HTML Beautifier Indent tab, 322
HTML Beautifier Schemes tab, 330
HTML Beautifier Tags tab, 323
HTML Comments dialog, 328
HTML Formatting Options, 319
HTML symbol translation, 330

I
iconized windows, 498
Identifiers, 614
if statement, 892
Ignore forward class declarations, 611
ignore spaces, 510
Image control, 972
import key bindings, 71
Import options (Java Imports), 286
Import Preferences, 652
Imports, 286
Imports Menu, 459
Including macro header files, 920
Incremental search highlighting, 578
incremental searching, 345
indent case, 292
indent first level of code, 292
indent for each level, 292
indent inside block, 295
indent preprocessing, 295
indent stand alone comments, 293
indent style, 589
indent with tabs, 292
indenting tab, 291
indicator (named bookmark), 239
indicator (pushed bookmark), 242
Inheritance, 951
Inherited Code Found dialog box, 942
Initial NumLock state, 561
Initialize search string, 578
Initialize with default preferences, 578
initializing macro modules, 928
Insert Form Source dialog box, 941
Insert Literal dialog box, 134
insert mode, 564
Insert open parenthesis for functions, 605
insert real indent, 588
Insert space after comma, 611
inserted attribute (casing), 320
inserted tag (casing), 320
Inserting characters, 134
Inserting control characters, 58
Installation, 17
Installed templates (Locating), 212
Installing SlickEdit® Core, 17
Installing with the Update Site, 17
Installing with ZIP Archive, 18
InstallScript Formatting Options, 286
Instances, 956
Internet Protocol (IP), 639
ISPF Copy Lines, 696
ISPF Delete Lines, 697
ISPF Emulation, 688
ISPF emulation keys, 783
  clipboard, 788
  command line and text box editing, 789
  compiling and programming, 791
cursor movement, 783
debugging, 792
deleting, 785
files and buffers, 790
inserting, 785
macros, 793
miscellaneous, 793
selecting, 786
selective display, 793
windowing, 790
ISPF Emulation Options, 688
ISPF Exclude Lines, 703
ISPF Expose First Lines, 697
ISPF Expose Last Line, 698
ISPF Expose Next Level of Code, 700
ISPF Insert After, 695
ISPF Insert Before, 695
ISPF Insert Bounds Ruler, 696
ISPF Insert Columns Ruler, 696
ISPF Insert Lines, 697
ISPF Insert Mask Line, 699
ISPF Insert Tabs Ruler, 701
ISPF Insert Text, 701
ISPF Join Lines, 702
ISPF Line Command A, 693
ISPF Line Command B, 693
ISPF Line Command Delete, 693
ISPF Line Command Exclude, 694
ISPF Line Command First, 693
ISPF Line Command Uppercase, 694
ISPF Line Commands, 694
ISPF Line Labels, 694
ISPF Lowercase Lines, 698
ISPF Make Data Lines, 699
ISPF Move Lines, 698
ISPF Overlay Lines, 700
ISPF Repeat Lines, 700
ISPF Select Lines, 703
ISPF Shift Lines Left or Right, 694
ISPF Split Line, 702
ISPF Unsupported Primary Commands, 704
ISPF Uppercase Lines, 702

J
J# Formatting Options, 286
Java, 283
Java Beautifier, 290
Java Formatting Options, 286
Java Organize Imports, 286
Java Organize imports options, 286
Java Refactoring, 255
Javadoc Beautifier dialog box, 298
Javadoc comments, 247
Javadoc Editor dialog box, 298
JavaScript Formatting Options, 286
JCL Formatting Options, 281
Join comments, 597
JSP TagLib Formatting Options, 319
Jump over tab characters, 567
jumping to a tag, 137
justification dialog box, 600
justified, 600
justify style, 600

K
keep alive default FTP, 641
Key Binding Preferences, 555
key bindings, 66
  binding macros, 430
  creating, 69
definitions of terms, 66
emulations, 62
export/import, 71
key message delay, 73
unbinding, 71
viewing associated commands, 48
key definition, 48
Key message delay, 561
key shortcuts in text boxes, 52
Keyboard and Mouse Preferences, 554
Keyboard Preferences
  advanced, 560
Keys Help, 64
Keyword case, 267
Keywords, 604

L
Label control, 959
Language Constructs, 853
Language Manager, 258, 582
Language Preferences, 580
Language Setup, 262
language support, 9
Language tab (Color Coding Setup), 618
language-specific aliases, 191
creating from selection, 198
Largest file to AutoSave, 632
Last line is bottom, 595
lcase, 133
Leave selected, 578
left and respace, 600
Left and Right, 595
lexer name, 261
lexer name (creating in VLX file), 680
Licensing, 18
Light bulb, 604
Line Comment
color coding, 622
Line format, 625
line hex, 172
Line insert style, 564
line navigation, 143
line numbers, 168, 590
line selections, 119
Line wrap, 567
Lines per Page, 446
Linux (printing on), 57
List all occurrences, 483
List Box control, 963
list clipboards, 129
list clipboards dialog box, 129
List command line completions, 532
List include files after typing #include, 605
List Members, 98
List of matches, 604
literal characters (inserting), 134
Load as Binary, 624
load command, 930
Load entire file, 629
Load File, 276
Load File Preferences, 626
Load Module dialog box, 433
Load partially for large files, 629
Load partially when files are larger than, 629
Loading macros, 930
Locals, 603
Locating templates, 212
Location, 595
Location (Code Templates Add New Item dialog box), 212
look in, 481
loop (loop), 895
Loops, 893
lower case file names, 320

M
Mac Option/Alt key behavior, 560
Mac OS X
writing selection filters, 994
Mac OS X emulation keys, 831
clipboard, 835
column line and text box editing, 834
cursor movement, 831
deleting, 832
files and buffers, 835
inserting, 832
macros, 836
miscellaneous, 836
searching, 833
selecting, 832
windowing, 836
Macro Menu, 495
macro prompting, 1000
macros
programmable, 433
recorded, 428
macros (batch), 434
margins (setting), 600
margins tab, 56
Match block comment setting, 462
Match case, 481
Match case (default), 577
match highlighting, 355
Match whole word, 481
Match whole word (default), 577
math (see mathematics), 422
Mathematical Operators, 882
mathematics, 422
document math, 424
expressions with mixed bases, 422
math commands, 423
math commands (examples), 424
overflow/underflow, 424
prime numbers, 425
using the Calculator, 422
Max files, 628
Max number of tags per file, 569
Max size of files to tag, 569
Max undo steps, 628
Maximum candidates for list parameters, 572
Maximum class/struct shown in list members, 571
Maximum clipboards, 564
Maximum functions found by parameter help, 571
Maximum items found in references search, 572
Maximum response time for list members (ms), 571
Maximum response time for list parameters (ms), 572
Maximum search results output (KB), 578
Maximum symbols, 573
Maximum tags found in symbol search, 572
Maximum word completion, 573
Menu Customizations
   Export/Import Preferences, 653
Menu Editor dialog box, 502
Menu Item Alias dialog box, 708
Menus, 42
menus (changing or adding), 977
Merge dialog box, 363
merging files, 363
Methods, 958
Minimum prefix, 606
Minutes before restarting, 569
Modal and modeless dialog boxes, 948
mode name, 587
modified lines
   reset, 631
   viewing, 167
Modify parameter list (Quick Refactoring), 253
modifying doc comment templates, 248
modifying selected text, 122
Module Initializations, 928
modules (Slick-C), 433
mouse pointer option, 532
Mouse Preferences
   advanced, 560
   mouse selection, 575
Moving Controls, 939
multi-file diff output dialog box, 361
multi-file undo/redo, 355
Multi-line Comment
   color coding, 622
   multiple file search and replace, 483

N
Name (Code Templates Add New Item dialog box), 212
Name (Code Templates Add Parameter dialog box), 211
Name element (Code Templates metadata file reference), 219
named bookmarks, 238
deleting/clearing, 240
indicator, 239
navigating (go to), 240
setting, 238
toggle, 239
namespaces, 279
name_info, 905
Navigate Menu, 477
navigate to named bookmark, 240
navigate to pushed bookmark (popping), 241
navigation, 137
   between symbols, 137
   between words, 138
cursor movements, 142
go to offset, 143
   in pages and files, 142
   in statements and tags, 141
subword navigation, 139
symbol browsing, 146
to a specific line, 143
URLs, 144
Network and Internet Preferences, 638
Network Settings, 638
never, 589
new configurations, 279
New Extension dialog, 262
new features, 5
New Formatting Scheme dialog, 336
Newline characters, 590
next word, 138
Next word style, 564
no space before paren, 291
non-mdi editor control, 498
none, 589
Notifications, 114
notifications
   options, 635
   no_code swapping, 1003
null, 867  
number lines every, 442  
number of copies, 442  
Number of lines to color above and below the current page, 573  
Number of off-page lines to color per pass (chunk size), 573  
Numeric constants, 863  
numeric overflow or underflow, 424  
numeric sort, 370  
umerical values, 320  
O  
Object Instances, 956  
Objective-C Formatting Options, 286  
offset navigation, 143  
OK button (adding), 943  
OMG IDL Formatting Options, 286  
Open Form dialog, 941  
Open Menu dialog box, 707  
open site, 642  
Opening SlickEdit® Core Views, 44  
opening Unicode files, 662  
Operators, 882  
Options  
  Export/Import Overview, 61  
  Outline View, 150  
  outline view, 150  
  Output View, 150  
  Options, 152  
output file, 510  
output style, 510  
Overview Ruler, 41  
P  
pad condition, 295  
Pad parentheses, 611  
Parameter element (Code Templates metadata file reference), 220  
Parameter info, 610  
Parameter Information, 99  
parameter prompting, 197  
Parameters element (Code Templates metadata file reference), 221  
paren style, 140  
Parenthesis matching style, 565  
parse statement, 897  
partial load (fast line count), 628  
Pascal Formatting Options, 300  
passive transfers, 643  
paste, 129  
pedantic, 916  
Perl Formatting Options, 286  
Perl regular expressions, 390  
  examples, 397  
Perspectives, 39  
PHP Formatting Options, 319  
Picture Box control, 971  
pictures (adding to image control), 973  
pictures (adding to list box), 964  
PL/I Formatting Options, 303  
PL/SQL Formatting Options, 303  
Place cursor at end, 578  
Pointer Variables, 879  
pop bookmark, 241  
Preferences  
  Export/Import, 650  
  see Options, 60  
Preferences dialog, 523  
Preferences dialog shortcuts, 526  
Preferences Export/Import, 650  
  Export Groups, 651  
Preferences History, 649  
Preferences Search, 525  
Prefix area width, 688  
Prefix matching (Preferences dialog), 526  
preprocessing for C/C++, 279  
Preserve column on top/bottom, 565  
Preserve trailing, 606  
preserve width on existing (comment wrapping),
preview (code templates add file dialog box), 211
preview view, 153
what is displayed, 155
preview window symbol lookup timeout (ms), 571
prime numbers, 425
print (see also printing), 55
print color, 442
print color coding, 442
print command, 447
print device, 447
print dialog (text mode), 444
print dialog box, 439
print filter, 447
print hex, 442
print schemes, 57
printing, 55
insert formfeed, 58
on linux, 57
on windows, 55
schemes, 57
procedures (defining), 901
procs and prototypes, 137
product improvement program, 21
product improvement program options, 637
product registration, 4
product support, 20
product support
contacting, 20
profile combo box, 276
profiling (slick-c), 932
programmable macros, 433
loading, 433
progress 4gl formatting options, 286
project templates
export/import preferences, 654
prompt for value (code templates add parameter dialog box), 211
prompt procedure, 1001
prompt replace dialog box, 486
prompt string (code templates add parameter dialog box), 211
prompting from macros, 1000
properties dialog box, 938
protect read-only, 565
protect read-only mode, 565
prototypes, 909
proxy settings, 645
pushed bookmarks, 241
bookmark stack, 242
indicator, 242
options, 242
popping, 241
pushing, 241
viewing, 242
python formatting options, 286
q
quick brace, 227
quick brace/unbrace one line statements, 288
quick extract method, 253
quick modify parameter list, 253
quick refactoring, 252
quick refactoring menu, 459
quick rename, 252
quick replace, 345
quick replace literal with constant, 253
quick search, 344
quick start,
quick start configuration wizard, 28
quotes, 321
quotes (numerical values), 320
quotes (single values), 321
quote_key command, 58
r
radio button control, 962
rc variable, 935
read only mode (macros), 498
read only mode (protect), 565
rebuilding tag files, 110
recognizing unicode files, 661
recorded macros, 428
binding to keys, 430
deleting, 432
execute_last_macro_key, 430
operations, 428
recording a new macro, 429
running, 431
saving and editing, 431
using to discover/control options, 432
redeclvars, 916
redefine common key preferences, 559
redo (multi-file), 355
redo replacement, 355
refactoring, 252
Refactoring results, 253
references (preferences), 571
References view, 156, 156
options, 157
reflow comment dialog box, 461
Reflow next, 565
reflowing comments, 250
Regex Evaluator, 381
Regex Evaluator view, 381
entering expressions, 382
entering test cases, 382
options, 382
registration, 4
Regular Expresions, 375
regular expressions
An Overview, 375
Brief, 383
Brief examples, 389
evaluating, 381
minimal versus maximal matching, 376
Perl, 390
Perl examples, 397
search and replace with, 354
SlickEdit, 399
SlickEdit examples, 405
syntax, 383
tagged search expressions, 377
Unicode category specifications, 414
Unicode character blocks, 416
UNIX, 406
UNIX examples, 412
Reinsert after current, 628
Reload on switch buffer, 630
reload prompt, 629
remove duplicate lines, 371
Rename (Quick Refactoring), 252
replace
see also replacing, 344
Replace in Files tab (Find and Replace), 488
Replace literal with constant (Quick Refactoring), 253
replace mode, 564
Replace params in target file (Code Templates Add File dialog box), 210
 Replace tab (Find and Replace), 486
replacing, 344
Find and Replace view, 352
minimal vs maximal matching, 376
quick replace, 345
replace and c commands, 349
see also searching, 344
undo/redo, 355
using regular expressions, 354
reset, 290
Reset modified lines, 631
Resetting the Configuration Directory, 673
Restore cursor after replace, 578
retagging workspace, 103
return statement, 902
revision one, 510
revision two, 510
REXX Formatting Options, 303
Right CTRL = Enter/Send, 689
router, 642
Ruby Formatting Options, 286
rule, symbol coloring, 87
Run-time error finding, 932
running recorded macros, 431
S
Same name, 632
Same name different extension, 632
Save after period of inactivity, 631
Save after period of time, 632
Save and Bind to Key, 498
Save File Preferences, 630
save macro dialog box, 497
Save Multi-File Diff Output dialog box, 362
save settings, 290
Save to, 632
Save to different directory, 632
Save/restore file position, 628
saving recorded macros, 431
SBCS/DBCS, 660
sc.lang.IHashIndexable, 870
sc.lang.INIndexable, 867
sc.lang.IToString, 876
scheme, 87
Schemes (Beautify), 295
schemes (colors), 85
schemes (printing), 57
schemes tab, 295
Scoping and Declaring Variables, 887
Screen Layout, 38
scroll bars, 78
Scroll style, 532
Scroll when, 532
search
see also searching, 344
search (go to offset), 143
Search backward, 578
search for, 481
Search for word matches if symbol is not found, 571
Search hidden text, 578
Search Menu, 479
Search Options
  Regular expression syntax, 577
Search Options dialog box, 481
search order, 678
  configuration files, 678
  executable files, 678
Search Preferences, 575
Search Results node, 525
Search results output, 353
searching, 344
  Find and Replace view, 352
  find and slash commands, 346
  Find Symbol view, 354
  finding all symbols in file, 355
  history retrieval, 578
  incrementally, 345
  initialization preferences, 578
  minimal vs maximal matching, 376
  multiple files, 483
  quick search, 344
  syntax-driven, 352
tagged search expressions, 377
undo/redo, 355
  using regular expression syntax, 577
  using regular expressions, 354
See F1 Index Help for more information., 523
Seek dialog, 143
seek offset position, 143
Select Mode dialog box, 258
Select Mouse Event dialog, 69
Select Symbol dialog, 477, 493
Select Text to Paste dialog, 453
Selected text (if exists), 578
selecting a code block, 234
Selecting a mode, 258
Selecting Controls, 937
selecting text, 117
  block insert mode, 134
  blocks (columns), 120
  characters, 118
  commands, 122
counting lines/characters, 120
drag-and-drop, 129
enumeration, 122
keyboard shortcuts, 118
lines, 119
  modifying a selection, 122
Selection (Comments), 462
selection (creating alias from), 198
selection color, 117
Selection Functions Overview, 992
Selection indicator, 120
selection only, 441
Selection Preferences, 574
selection restriction, 290
selection styles, 574
Selective Display, 170
Selective Display dialog box, 473
Selective Display Function headers, 475
Selective Display Multi-level, 475
Selective Display Preprocessor directives, 475
Selective Display Search Text, 474
Selective Display, Expand/collapse, 561
service name, 643
set command, 668
set named bookmark, 238
set pushed bookmark, 241
set scroll style, 78
Set Variable dialog box, 500
SETemplate element (Code Templates metadata file reference), 222
setting
  fonts, 80
Setting Breakpoints, 244
Setting Conditional Breakpoints, 244
Setting Java Exception Breakpoints, 244
Setting Properties, 938
SFTP, 372
sftp tab, 643
shell prompt, 49
shell scripts, 14
shelling, 49
Shelling from macro, 998
shortcuts in Preferences dialog, 526
shortcuts in text boxes, 52
Show auto reload timeout notifications, 630
Show categories, 604
show changes, 510
Show comments, 610
Show EOF character, 628
Show icons, 604
Show info for symbol under mouse, 612
Show parameters, 604
Show preview of symbols in tagging tool windows
after (ms), 571
Show preview of symbols in tagging tool windows
on mouse-over, 571
Simple Variables, 887
single line statements, 227
single word values, 320
Size limit for comparing contents (KB), 630
Sizing Controls, 939
Slick-C Batch Files, Writing, 911
Slick-C Debugger, 933
Slick-C Formatting Options, 286
Slick-C headers, 435
Slick-C module, 433
Slick-C Profiler dialog, 932
Slick-C profiling, 932
Slick-C variable, 434
SlickEdit book, 6
SlickEdit Core command line, 46
activating, 46
common commands, 50
history, 47
text box editing, 730
key phrases, 47
keyboard shortcuts, 47
prompting, 49
prompting (option), 561
shell commands, 49
SlickEdit Core emulation keys, 726
clipboard, 730
command line, 730
compiling and programming, 732
cursor movement, 726
debugging, 733
deleting, 728
files and buffers, 731
inserting, 727
macros, 734
miscellaneous, 735
searching, 728
selecting, 728
text box editing, 730
windowing, 732
SlickEdit regular expressions, 399
examples, 405
SlickEdit® Core Views, 44
SmartPaste, 174
Smooth horizontal scroll, 532
Snippets, 189
Soft Wrap, 600
SoftWrap, 169
tab characters, 370
tab commands, 371
tab dialog box, 370
tab on selection, 370
tab within selection, 370
SoftWrap, 169
tab characters, 370
SoftWrap, 600
SoftWrap, 169
tab characters, 370
Space always inserts space, 606
space between, 442
Space inserts longest unique prefix, 606
Space inserts longest unique prefix, 606
Special Character Preferences, 553
Special Characters, 590
special characters tab, 553
specifying tabs for a file extension, 174
spell check, 367
in multiple files, 368
operations, 367
running, 367
Spell Check Files dialog box, 368
Spell Check Menu, 460
Spell Check Preferences, 648
spelling dialog box, 367
Split line comments, 596
SQL Server Formatting Options, 303
spill file path, 634
Spin control, 960
Split strings, 597
Start after minutes idle, 569
Start after seconds idle, 569
Start characters, 614
Start in column, 596
Start mode, 564
Start wrapping on line, 598
starting FTP connection, 373
state file, 434
Statement Level Tagging, 102
statement navigation, 141
Status Area, 42
stopping FTP connection, 374
strict, 916
strict2, 917
strictarglists, 917
strictboolean, 917
strictellipsis, 917
strictincludes, 918
strictnames, 918
strictnumbers, 918
strictparens, 918
strictprotos, 918
strictreturn, 919
strictsemicolons, 919
strictstrings, 919
string concatenation, 885
String editing, 597
string editing (comments), 250
string literals, 861
string operators, 866
Strip trailing spaces, 631
struct, 870
structure matching, 139
setting match style, 140
viewing/defining, 140
subsystem, 643
subword navigation, 139
summary of keys, 62
support (for SlickEdit Core), 20
Suppress prompt unless modified, 629
surrogate support, 662
Surround With, 233
alias, 234
commands, 235
surrounding, 229, 229, 233
unsurrounding, 237
switch statement, 900
Switching Between Editors, 41
Symbol Browser Filter Options dialog box, 162
symbol browsing, 146
base/derived classes, 161
Class view, 146
Find Symbol view, 153
Outline view, 150, 338
Preview view, 153
References view, 156
symbol uses/call tree, 160
Symbols view, 157
Tag Properties view, 165
Symbol Coloring, 83, 86
Preferences, 539
rule, 87
Unidentified Symbols, 88
View Menu, 473
Symbol declaration, 604
symbol highlighting, 355
symbol navigation, 137
between multiple instances, 138
browsing symbols, 146
Find Symbol view, 138
more methods, 138
Symbol Properties view, 165
Symbol Translation Editor, 330
symbol use, 160
Symbol Uses/Calling tree dialog box, 160
Symbol view, 157
symbols
comparing, 359
Symbols, 603
Symbols view, 157
base/derived classes, 161
filter options, 162
filtering symbols, 158
options, 159
uses/calling tree, 160
Sync background colors, 534
sync extension, 290
sync vertical line (comment wrapping), 599
syntax expansion, 199
Syntax expansion, 604
Syntax expansion (Overview), 199
Syntax expansion on space, 605
syntax indent, 292
Syntax indent, 592
Syntax indent ruler, 532
syntax-driven search, 352
system configuration files, 675
System Requirements, 17
system state file, 434
Tab cycles choices, 606
Tab inserts longest, 606
Tab key, 173
tab size, 292
Tabs, 592
tabs dialog box, 174
Tabs ruler, 532
Tag file cache maximum, 572, 635
Tag file cache size, 572, 634
Tag file on save, 569
tag files, 279
building, 103
categories, 103
language-specific libraries, 106
rebuilding, 110
search order, 109
workspace files, 103
tag navigation, 141
tagging, 98
compiler libraries, 104
workspace files, 103
tagging (identifiers), 614
tagging (minutes before retagging), 569
tagging cache, 634
Tags tab (Color Coding Setup), 623
Target file name (Code Templates Add File dialog box), 210
Tcl Formatting Options, 286
Template file (Template Manager dialog box), 209
Template Manager dialog box (Code Templates), 208
TemplateContent element (Code Templates metadata file reference), 224
TemplateDetails element (Code Templates metadata file reference), 225
Templates list (Code Templates Add New Item dialog box), 211
Templates list (Template Manager dialog box), 209
testing regular expressions, 381
Text Box control, 961
text box editing keys, 52
text compare, 356
text editing
block insert mode, 134
inserting characters, 134
selections, 117
sorting, 370
Text Mode Print dialog box, 444
text selection, 575
The SlickEdit® Core Editor, 39
The Workbench, 38
The Workspace, 38
three way merge editing, 363
Timeout after (ms), 573
Tips on Finding Eclipse Command Information, 74
toggle bookmark, 239
Tool Preferences, 648
Toolbar and View Layout
  Export/Import Preferences, 653
Tools Menu, 507
Truncate file at EOF, 629
truncation, 588
two up, 442
twopass, 919>Type Casting, 888
typeless, 880
typeless variable declaration, 888
U
ucase, 133
UCN, 662
unbinding a key, 71
Unbrace, 227
uncommenting, 245
undo (multi-file), 355
undo replacement, 355
Unicode, 660
category specs for reg expressions, 414
character blocks for reg expressions, 416
converting to UCN, 662
file recognition, 661
implementation, 664
limitations, 662
opening files, 662
surrogate support, 662
Unidentified Symbols, 88
Uninstalling SlickEdit® Core, 19
union, 871
UNIX (recommended fonts), 82
UNIX regular expressions, 406
examples, 412
UNIX temp environment, 634
unload module, 433
unsurrounding, 229, 237
Update after (ms) idle, 573
Update after (ms) idle (0 implies no delay), 572
Update after idle, 573
Update workspace tag file on activate, 569
Update workspace tag file on open, 569
URI Scheme Preferences, 644
URL Mapping Preferences, 643
1053
url mappings, 317
URL navigation, 144
Use Clear key as NumLock, 561
Use Command+key for dialog hotkeys, 560
Use Command+key for menu drop-downs, 560
use hanging indent (comment wrapping), 599
use smart merge, 510
Use strict case-sensitivity , 606
Use undo, 628
user configuration files, 673
user preferences, 60
user state file, 434
User-Created Forms
  Export/Import Preferences, 653
User-Created Menus
  Export/Import Preferences, 653
User-Created Toolbars
  Export/Import Preferences, 653
User-Loaded Modules dialog, 496
User-Recorded Macros
  Export/Import Preferences, 654
UTF-8, 660
V
Value (Code Templates Add Parameter dialog box), 211
variable (Slick-C), 434
Variable editor, 500
Variable Editor dialog box, 500
Variable Initializations, Details, 888
Vertical line color, 532
Vertical line column, 532
view
  hex/line hex, 172
  line number, preferences, 590
  line numbers, 168
  special characters, 167
View Preferences (Language-Specific), 589
view special characters, 167
View text symbols (Special Characters tab), 553
viewing
  current line, 166
  modified lines, 167
viewing accessible members, 164
viewing symbol properties, 160
viewing symbol references, 156
viewing symbol uses, 160
Views, 42
Vim emulation keys, 757
  clipboard, 763
  command line and text box editing, 764
  compiling and programming, 767
  cursor movement, 757
  debugging, 768
  deleting, 761
  files and buffers, 765
  inserting, 760
  macros, 769
  miscellaneous, 770
  searching, 761
  selecting, 762
  windowing, 766
Vim Tutorial, 64
Virtual Memory Preferences, 634
visible lines only, 442
Visual C++ emulation keys, 718
  clipboard, 721
  command line and text box editing, 721
  compiling and programming, 723
  cursor movement, 718
  debugging, 724
  deleting, 721
  files and buffers, 722
  inserting, 719
  macros, 725
  miscellaneous, 725
  searching, 719
  selecting, 720
  windowing, 723
Visual Studio emulation keys, 828
VLX file, 679
  creating a lexer name, 680
  modifying to change color definitions, 680
Vscroll Bar control, 969
VSECLIPSECONFIG, 665
VSLICK, 665
VSLICKALIAS, 666
VSLICKBACKUP, 666
VSLICKBIN, 668
VSLICKBITMAPS, 666
VSLICKINCLUDE, 921
VSLICKLOAD, 666
VSLICKMACROS, 665
VSLICKMISC, 666
VSLICKPATH, 665
VSLICKRESTORE, 665
VSLICKTAGS, 666
VSLICKXNOBLINK, 667
VSLICKXNOPLUSNEWMSG, 667
VSLICKXTERM, 667
VSLICK_XIM, 666
VSNet emulation keys, 828
VST, 667
vstw program, 930
vstw.exe program, 930

W
Web Browser Setup, 318
Web Browser Setup Preferences, 646
web development, 307
welcome, 3
what is key, 48
When tab key reindents, 588
where is command, 48
while (loop), 896
wid_expression, 956
Window left margin, 531
Windows (printing on), 55
Windows temp environment, 634
word boundary, 601
word chars, 589
Word completion, 604
word navigation, 138
word wrap, 600
Word Wrap Preferences (Language-Specific), 599
Workspace tag file only, 569
wrap, 482
  SoftWrap, 169
Wrap at beginning/end, 578
Wrap line length, 628
wrapping (comments), 250

X
Xcode emulation keys, 813
  clipboard, 817
  command line and text box editing, 816
cursor movement, 813
deleting, 814
files and buffers, 817
inserting, 814
macros, 818
miscellaneous, 818
searching, 815
selecting, 814
windowing, 818
xcom command, 912
XEDIT line commands, 689
XEDIT Line Commands, 703
XML, 307
  beautifying, 309
  DTD caching, 317
toggle between begin/end tags, 318
turn off auto-validate, 672
XMLdoc Editor, 309
XML Beautifier Advanced tab, 316
XML Beautifier Attributes Values tab, 313
XML Beautifier Comments tab, 314
XML Beautifier dialog box, 309
XML Beautifier Indent tab, 310
XML Beautifier Schemes tab, 316
XML Beautifier Tags tab, 311
XML encoding, 660
XML Formatting Options, 307
XML symbol translation, 330
XML/HTML Formatting, 331
  Export/Import Preferences, 654
xml/html formatting
  content wrap, 466
  enabling, 332
  enabling for current document, 333
  schemes, 334
  settings, 338
  tag layout, 468
tags, 337
XML/HTML Formatting dialog, 331
XML/HTML formatting menu, 332
XML/HTML Formatting Scheme Configuration
dialog, 463
XMLdoc comments, 247
XMLDOC Editor dialog box, 309
XOR operator, 423