
Visual SlickEdit[®] v7

User Guide

SlickEdit Inc.
3000 Aerial Center Parkway, Suite 120
Morrisville, NC 27560 USA
www.slickedit.com

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

Copyright © 1988-2002 SlickEdit Inc. All rights reserved.

Cover design copyright © by SlickEdit Inc.

Printed in the United States of America.

SlickEdit, Visual SlickEdit and Clipboard Inheritance are registered trademarks of SlickEdit Inc. Context Tagging, DIFFzilla, Dynamic Difference Editing and SmartPaste are trademarks of SlickEdit Inc. All other trademarks are the property of their respective owners. Protected by U.S. Patent 5,710,926.

ISBN 0-9716563-2-0

Contents

Introduction	1
About This Guide	1
Intended Audience	1
Terminology	1
Key Binding Conventions	1
Additional Visual SlickEdit Documentation	2
Contacting SlickEdit Inc.	2
Welcome to Visual SlickEdit	4
Overview	4
Visual SlickEdit Highlights	4
Development Language Support	7
Language Support for Main Features	7
Embedded Languages in HTML, Perl or UNIX Shell Scripts	9
Adding Support for Other Languages	10
Getting Started	13
Quick Start	13
File Type Association	13
Selecting an Emulation	13
Search Options	13
TAB Key and Indent Options	14
Selecting a Different Font	15
Redefining ENTER and Other Common Keys	15
Syntax Expansion Options	15
Setting up a Project	16
Building a Tag File for Run-Time Libraries	19
Auto Restore Options	20
Directory Aliases	20
Windows Word Help	21
UNIX Word Help	22
Network Backup Options	22
Invoking Visual SlickEdit	22
Invocation Options	23
Editing Files	25
The Screen Layout	27
Help Features	29
Tool Tips	30
Hypertext Help	30
Special Mouse Functions	30
Selecting Lines and Columns	30
Snapping Cursor to End of Line	31
Docking Toolbars	31
Copying and Moving Text	31
Changing Values on the Status Line	31
Accessing Sub-Menus	31
Organizing Your Code	32

Organization Overview	32
Creating Projects	32
Creating a Workspace and Project	33
Project Configuration Settings Dialog Box.....	34
Adding Files to a Project.....	34
Setting Directories	35
Setting Compile Commands	35
Pre- and Post-Build Commands	40
Open Command	41
Version Control Support	41
Analyzing Your Code.....	45
Analyzing Your Code Overview	45
Context Tagging(TM)	45
Context Tagging Overview	45
What is the Context Tagging Feature Set?	45
Creating and Updating Tag Files.....	50
Tagging Run-Time Libraries.....	52
Setting up Context Tagging for COBOL	53
Setting up Context Tagging for OS/390 Assembler.....	55
Dynamic Tagging.....	57
Tags for the Current File	57
Symbols Tab	57
Jumping to a Tag.....	59
Class Browser	60
Searching for Members or Classes.....	60
Viewing Accessible Members.....	61
Other Filtering Options	61
Viewing Symbol Properties	61
Viewing Base Classes	61
Viewing Derived Classes	61
Viewing Symbol References.....	62
Viewing Symbol Uses.....	62
Differencing Files with DIFFzilla(TM).....	63
Diffing Two Files.....	63
Diffing Source Trees.....	65
Diffing Symbols or Parts of Files.....	67
Diffing All Symbols of Two Files.....	68
Generating File Lists.....	68
Automatic Directory Mapping	69
3 Way Merge Editing	69
Working With Your Code	70
Working With Your Code Overview	70
Viewing and Editing Options.....	70
Hexadecimal View and Edit Mode	70
Searching for Binary Characters	71
Command Line and Text Box Editing	71
Completion.....	72
Command Line Argument Retrieval.....	73
Shelling	73

Changing the Current Directory.....	74
Windowing.....	74
Text Operations.....	76
Selecting Text	76
Operating on Selected Text.....	78
Word/Variable Completion.....	81
Syntax Expansion and Indenting.....	81
SmartPaste(TM).....	83
Moving Text Among Files With Clipboards.....	83
Spell Checking.....	85
Begin/end Structure Matching	87
Calculating Mathematical Expressions	88
Calculator Dialog Box	88
MATH Commands.....	89
Recording and Using Macros.....	90
Sorting.....	93
Sort Dialog Box	93
Sort Commands.....	94
Search and Replace	94
Incremental Searching.....	95
Find Dialog Box.....	96
/ Command.....	98
Searching for the Next Occurrence	102
Replace Dialog Box	102
c Command	103
Multiple File Search and Replace	106
Regular Expressions.....	107
Unicode Category Specifications for Regular Expressions.....	118
Search and Replace with Regular Expressions	122
Tagged Search Expressions.....	123
Minimal Versus Maximal Matching	125
Bookmarks	125
Setting a Bookmark.....	126
Go to Bookmark.....	127
Printing.....	127
Inserting a Formfeed	127
Print Setup Dialog Box	128
Print Dialog Box	128
Text Mode Print Dialog Box.....	131
Improving the Readability of Your Code.....	134
Readability Overview.....	134
Formatting Options	134
Margins and Tabs.....	134
Justification Style.....	136
Format Line.....	136
Color Coding.....	137
Code Beautifiers.....	139
HTML Beautifier	139
XML Beautifier.....	139

C/C++/Java/JavaScript/C# Beautifier	140
v1x File Format	140
Building and Debugging Your Code.....	146
Building and Debugging Overview	146
Building from within the Editor	146
Viewing Build Errors	146
Adding Support for Other Error Message Formats	147
Build Methods for GNU C/C++	148
Java and GNU C/C++ Debugger.....	148
Starting the Debugger	149
Debug Key Bindings	149
Setting Breakpoints	150
Setting Conditional Breakpoints	150
Avoiding Stepping Into Runtime Functions and Methods	151
Setting Exception Breakpoints (Java Debugger Only).....	151
Debugger Toolbars.....	151
Attaching to a Remote VM (Java Debugger Only)	152
Attaching to Running Process (GNU C++ Debugger Only)	153
Managing Your Files.....	154
Managing Your Files Overview	154
Editing Record Files.....	154
Alias Facility	154
Defining Directory Aliases	154
Extension-Specific Aliases.....	156
Accessing And Editing FTP Files	159
Starting a Connection.....	159
FTP Client.....	159
Opening FTP Files	160
Visual SlickEdit File Manager	160
List Files Dialog Box	160
dir and list Commands	161
Fileman Mode Keys.....	162
Selecting Files and Operating on Files.....	163
Repeat Command on Selected Dialog Box	165
How Visual SlickEdit Finds External Files	166
Search Order for Executable Files	166
Using Visual SlickEdit for Web Development	167
Web Overview	167
Using XML	167
XML Feature Overview	167
Validating and Well-formedness Checking of XML Documents	167
Automatic Validating for XML Documents	168
Beautifying XML.....	168
DTD Caching.....	168
Opening DTD Files from XML in One Step.....	169
URL Mappings.....	169
Toggling Between Begin and End XML Tags.....	170
Using Unicode.....	170
Unicode Overview	171

What are Encodings For?	172
Unicode File Recognition	172
Opening Unicode Files.....	173
Surrogate Support	173
Converting Unicode to UCN.....	173
Unicode Limitations.....	174
Implementation Overview.....	175
Using HTML	175
HTML Feature Overview.....	175
Beautifying HTML	176
Open URL.....	176
HTML Editing	176
Toggling Between Begin and End HTML Tags.....	177
Selecting a Block of Text.....	177
Hiding a Block of Text.....	177
Using Visual SlickEdit with Visual C++ or Visual Studio .NET	178
Visual Studio Overview	178
Opening Visual C++ or Visual Studio .NET Workspace Files	178
Compiling Your Project in Visual SlickEdit.....	178
Special Features on OS/390 and z/OS.....	179
OS/390 Features Overview	179
OS/390 Features	179
Edit Multiple Data Sets	179
Performance	179
Language Type Determination.....	179
Language Support	180
Other Features.....	184
Data Sets	186
Opening Data Sets.....	186
Searching for Data Sets and PDS Members with the Open Dialog.....	188
Searching for Data Sets and PDS Members with the Listds Command.....	189
Searching for Data Sets with the Edit Command.....	190
Multi-file Search and Replace.....	191
Data Set Utilities	192
Job Utilities	196
Invoking Visual SlickEdit from TSO	198
Using Visual SlickEdit ISPF Emulation.....	198
Compiling COBOL, Assembly, PL/I, and REXX Applications.....	199
Testing Your Application.....	200
Differencing Data Sets	200
Preserving Column of Line Numbers.....	203
OS/390 Version Control.....	203
Configuring Visual SlickEdit	204
General Configuration Dialog Box	205
General Tab.....	205
Search Tab	210
Select Styles Tab.....	211
Special Characters Tab.....	212
More Tab.....	212

Exit Tab.....	215
Virtual Memory Tab	215
Extension Options Dialog Box.....	216
Extension Options Indent Tab.....	217
Extension Options Word Wrap Tab	218
Extension Options General Tab	219
Extension Options Comments Tab.....	221
Extension Options Advanced Tab.....	222
Changing Key Bindings	224
Bind Command to Key Dialog Box	224
Editing Key Binding Source	226
Unbinding a Key	227
Determining the Key(s) a Command is Bound to	227
Determining the Function of a Key	227
Redefine Common Keys Dialog Box.....	227
ENTER Key Configurations	227
BACKSPACE Key Configurations.....	228
DEL Key Configurations	228
HOME Key Configurations	228
File Options Dialog Box	230
Load Tab	230
Save Tab	234
Backup Tab	235
AutoSave Tab.....	236
File Filters Tab.....	238
Color Settings Dialog Box	238
Font Configuration Dialog Box.....	240
Configuration Files.....	240
User Configuration Files	240
System Configuration Files.....	242
Configuration Changes not on Configuration Menu	243
Saving Disk Space with Tab Compression	243
Configuration Macro Variables.....	243
Setting Environment Variables	244
Setting Environment Variables in vslick.ini.....	246
set Command	246
Slick-C Language	248
Overview	248
Getting Started by Example	253
A Key Bindable Command	253
An Event Driven Dialog Box	253
A Batch Macro.....	255
Compiling and Loading Macros.....	256
Language Constructs	256
Comments	256
String Literals.....	257
Numeric Constants.....	258
Defining Constants.....	259
Mathematical Operators	260

Assignment Operator	262
Scoping and Declaring Variables.....	262
Type Casting	271
How Typeless Container Variables Work.....	272
The IF Statement.....	273
Statement Block	273
Loops	274
Switch Statement	275
Functions.....	277
defmain. Writing Slick-C Batch files.....	284
Preprocessing.....	285
Pragma	286
Including Header Files	287
Defining Controls.....	288
Defining Events and Event Tables	289
Module Initializations	292
Debugging Macros	293
Finding Procedures	294
Finding Runtime Errors	294
Creating Dialog Boxes with the Dialog Editor.....	294
Dialog Box Creation Overview.....	294
Dialog Editor Summary	295
Creating a Form	297
Adding Controls.....	297
Adding Event Handlers.....	299
Loading and Running the Form	299
Saving a Form.....	300
Adding a Cancel Button.....	300
Adding an OK Button and Closing a Dialog Box	300
Displaying Dialog Boxes	301
Clipboard Inheritance(R)	306
Dialog Box Inheritance Order.....	309
Objects and Instances.....	311
The Active Object	311
The Active Form	311
Instance Expressions.....	311
Using Functions as Methods	312
Built-in Controls	313
Label Control	313
Spin Control.....	313
Text Box Control	314
Editor Control	315
Frame Control	315
Command Button Control.....	315
Radio Button Control	315
Check Box Control	316
Combo Box Control	316
List Box Control	319
Vscroll Bar Control.....	321

Hscroll Bar Control.....	321
Drive List Control.....	321
File List Box Control	322
Directory List Box Control	323
Picture Box Control	323
Gauge Control.....	323
Image Control	324
Adding Dialog Box Retrieval.....	326
Changing or Adding Menus	327
Menu Macro Programming.....	327
Creating and Editing Menu Resources.....	327
Defining a Menu Resource.....	327
Predefined Attributes For Auto-Enabling Commands	328
Macro Callbacks for Enabling Commands	329
Putting a Menu Bar on a Form.....	330
Displaying a Menu as a Pop-up	330
Common Dialog Boxes.....	331
String Functions Overview.....	331
Search Functions Overview	333
Buffer Input/Output Functions.....	336
Selection Functions Overview	337
Writing Selection Filters	337
Unicode and SBCS/DBCS Macro Programming.....	339
DLL Interface.....	342
Shelling Programs from a Slick-C Macro	342
Hooking Exit and Other Events.....	343
Invoking a Macro When Visual SlickEdit is Invoked.....	343
Invoking a Macro Before Visual SlickEdit Terminates	343
Command Line Interfaces.....	343
Command Line Arguments.....	343
get_string Procedure	344
Single Argument Prompting with Support for Prompt Style	344
State File Caching	345
Error Handling and the RC Variable	345
Typical Mistakes	345
Interpreter Error Handling.....	346
Event Names	347
Differences Between Slick-C and C++	349
Windows Structure.....	352
Converting SlickEdit 2.3/2.4 Macros and Configuration	355
Frequently Asked Questions	357
Glossary	367
Index.....	370
Reader's Comments	379

Introduction

About This Guide

This User Guide describes all of the features of Visual SlickEdit. The Slick-C Macro Programming guide is included in this User Guide. To best configure Visual SlickEdit and maximize your productivity, please read the short “Quick Start” chapter on page 13 or in the online help.

Intended Audience

This guide is intended for anyone who uses Visual SlickEdit. Because Visual SlickEdit is available for Windows, UNIX and OS/390 or z/OS platforms, and its functionality on each platform is similar, this guide is intended for use on *all* platforms.

NOTE: Any item in the guide or online help that refers to OS/390 also applies to z/OS.

Terminology

Definitions of common terminology used in this guide are as follows:

buffer	A file that has been loaded into Visual SlickEdit. Once a file has been loaded, you can safely perform modifications to the buffer without modifying the file on disk until you save the buffer.
edit window	A rectangular viewing area used to display and edit buffers.
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.” (Do not confuse this term with the Microsoft Windows® operating system.)
selection	A highlighted region of text typically operated by a command which affects only the region. In the dialog editor, your selection is indicated by eight square handles which surround the control.
clipboard	A temporary storage area used to transfer text or dialog box controls from one place to another. Since the Microsoft Windows operating system does not have adequate multiple clipboard support, Visual SlickEdit stores multiple text clipboards internally.

Key Binding Conventions

Throughout this guide the default (CUA) key bindings are given for your convenience. If you change Visual SlickEdit’s emulation or key bindings, please remember that references to key bindings in the guide may be different from your own key bindings.

Commands are specified in the following format: ("Search", "Find"). This means you activate the menu bar (press ALT), select the "Search" menu bar word, and select the "Find" menu item. The following keying conventions are used throughout this guide:

Action	Convention	Example
Type input from the keyboard	Input shown in boldface	Type vs
Press a key on the keyboard	Key shown in uppercase	Press ESC
Click a button on the screen	Button shown in quotation marks	Click "Save"
Click left or right mouse buttons	"L" or "R" precedes the word "button"	Click Rbutton
Press a key combination	Keys in uppercase, linked by "+"	CTRL+=
Optional value	Value enclosed in braces	{value}
Your specified value	Value shown in italics	<i>filename</i>

Additional Visual SlickEdit Documentation

In addition to this User Guide, the complete Visual SlickEdit documentation set includes the following:

- Online version of this User Guide
- Visual SlickEdit Installation Guide
- Individual Quick Reference Sheets for the following emulations:
 - Brief
 - CodeWarrior
 - CUA
 - Epsilon
 - GNU Emacs
 - ISPF
 - SlickEdit (Text Mode Edition)
 - vi
 - Visual C++

Contacting SlickEdit Inc.

You can access patches and macros from the SlickEdit Inc. Web site: www.slickedit.com.

Contact SlickEdit at:

Product Support:	+1 919.473.0070* support@slickedit.com
------------------	--

Sales: 800.934.3348
+1 919.473.0070
sales@slickedit.com

Fax: +1 919.473.0080

* NOTE: A current Maintenance and Support contract is required for telephone support past the 30 day warranty period.

Welcome to Visual SlickEdit

Overview

Visual SlickEdit combines a rich set of powerful software engineering tools with more configurability than any software product in any category. Visual SlickEdit has proven to be the best-of-breed in features and performance on every platform. With its multi-platform presence, Visual SlickEdit supports today's businesses – with their mix of languages, tools, hardware, and operating systems. Visual SlickEdit's Slick-C macro language and its multi-platform graphical development environment simplify product enhancements and enable rapid additions of new editor features such as Context Tagging™, SmartPaste™, DIFFzilla™, Dynamic Difference Editing™, Code Block Selection, and Macro Recording.

Visual SlickEdit Highlights

- **Context Tagging™.** Provides Auto List Members, Auto Parameter Info, context tag navigation, and context tag preview. See "Context Tagging™" on page 45 for details.
- **Class Browser.** Displays class members, optionally shows accessible inherited members, views base classes, searches for members, and more. See "Class Browser" on page 60.
- **Dynamic Tagging.** Automatically updates project tag files and extension-specific tag files. Tracks current function, tags in current file, and tag symbols for word at cursor. See "Dynamic Tagging" on page 57.
- **DIFFzilla™.** The ultimate in multi-file differencing:
 - View differences between two files.
 - View differences between source trees.
 - View differences between two symbols or parts of files.
 - View differences between all symbols of two files.
 - Dynamic Difference Editing™. Intra-line differences are color coded even as you type.
 - Generate file lists.
 - Automatic directory mapping.
 - Save/restore multi-file results.
 - Dialog box history (wildcards, paths, file specifications).

See "DIFFzilla™" on page 63.

- **FTP Support.** Includes a complete FTP client and lets you easily open and edit remote files via a FTP connection. See "FTP Support" on page 159.
- **Symbol References and Uses.** Support for viewing Symbol References and Uses for C++, Java, COBOL, and Slick-C. See "Viewing Symbol References" on page 62.

- **Java, GNU C/C++ Debugger.** Provides a rich set of debugging features for JDK 1.3 or higher. See "Java and GNU C/C++ Debugger" on page 148 for details.
- **Unicode Support.** Supports many encodings formats (UTF-8, UTF-16, UTF-32, optional signatures, big endian, little endian, and many code pages), level 1 Unicode regular expressions, encoding translations, composite characters, surrogate characters, and support for almost all of Visual SlickEdit's features. See "Using Unicode" on page 170. (This feature is available on Windows platforms only.)
- **XML Support.** A list of tags or attributes, based on your DTD, is automatically displayed as you type to assist you in entering valid XML. The DTD can be local or automatically downloaded via HTTP. Tags and attributes are automatically color coded for easier viewing based on your DTD. When you open an XML file, the encoding specification is automatically recognized. See "Using XML" on page 167.
- **Customizable Dockable Toolbars.** Project toolbar with tabs for Class Browser, tags in current file, file open, project files, and remote files. Output toolbar with tabs for tag symbol tracking, multi-file search output, compilation output, and symbol reference output. FTP Client toolbar for uploading and downloading files via FTP.
- **C/C++/C#/Java/JavaScript/HTML/XML Code Beautifier.** Beautifies your entire buffer or selected lines in the buffer. See "HTML Beautifier" on page 139 or "C/C++/Java/JavaScript/C# Beautifier" on page 140.
- **SmartPaste™.** When lines of text are pasted into a source file, the added lines are reindented according to the surrounding code. See "SmartPaste™" on page 83.
- **Hex Editing.** View and edit files in hex. Supports inserting and deleting bytes.
- **Selective Display.** Display only function headings, lines with/without search string, lines your compiler sees after preprocessing, collapse the current code block, collapse all comments, and more.
- **3 Way Merge Editing.** Automatically merges two revision files with a base file. Collisions are color coded and you can interactively add changes to the output file while the files are being merged. See "3 Way Merge Editing" on page 69.
- **Code Block Selection.** Selects the lines in the current function, **for** loop, **if** block, or other block structure with a single command. Then you can drag drop the code to another location.
- **API Apprentice.** Type part of an API function and let the editor help you pick the function and enter the function parameters. Databases available for WIN32, MFC, Java, Motif 1.2, X11 R5, Xt, and C-runtimes.
- **Smart Editing Features.** Syntax expansion/indenting (see page 15) and brace/structure matching features know the difference between comments, strings, and other text.

- **Complete Embedded Language Support.** Visual SlickEdit recognizes languages embedded in HTML, Perl, COBOL 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. See "Embedded Language Support" on page 9.
- **Smart HTML Color Coding.** Color codes attributes on a per-tag basis in a user-definable attribute color.
- **Integration for Popular Tools.** Visual SlickEdit supports many compiler packages and includes support for several version control systems.
- **Unlimited Undo and Redo.** Even undoes past saves.
- **No Limits.** Line length, file size, number of files, number of bookmarks, and number of windows only limited by system resources (i.e. disk space, memory).
- **Multi-file Search and Replace.** Search and replace across files. Also supports recursing subdirectories.
- **Multi-file Spell Checking.** Spell check comments and strings, the selection, the current word, or the whole document. Includes special HTML spell checking.
- **Line, Column, and Character Selections.** See "Text Operations " on page 76.
- **Command Line for the Seasoned Pros.** Supports argument retrieval and shelling commands just as you would type them at the operating system prompt. DOS commands such as **set**, **cd**, and **d:**(change drive) are built-in. Invoke many editor commands from the command line. See "Command Line and Text Box Features" on page 71.
- **Macro Recording.** Generates Slick-C source code you can edit, and supports dialog box interactions. Write Slick-C source code without understanding the language. See "Recording and Using Macros" on page 90.
- **Concurrent Process Buffer.** Provides a natural way to run command line compilers, linkers, and tools concurrently from within the editor. If you know how to use a shell command or operating system command line prompt to run a command line tool, just type in the name into the concurrent process buffer to run it. See "Building from Within the Editor" on page 146.
- **Multiple Clipboards.** Paste from a stack of your most recent clipboards. See "Moving Text Among Files With Clipboards" on page 83.
- **Programmable File Manager.** See "The Visual SlickEdit File Manager" on page 160.

- **Project Management.** Organize your work into a neat package. See "Creating Projects" on page 32.
- **Alias Facility.** Define aliases for directories which can be used by the Change Directory dialog box. Aliases can also be expanded with CTRL+SHIFT+O in any text box, combo box, or edit window. Even though aliases are typically used as shortcuts for long directories, they are also used to expand a short key sequence to any text string of any length. See "Directory Aliases" on page 20 and "Alias Facility" on page 154.
- **Fast Incremental Searching.** Quickly search for a string as you type. See "Incremental Searching" on page 95.
- **Advanced Text Box Editing.** Includes completion, aliases, multiple clipboards, cut line, cut word, uppercase word, and lowercase word. See "Command Line and Text Box Features" on page 71.
- **Flawless Emulations.** Emulates BRIEF, GNU Emacs, VI, ISPF, Epsilon, CodeWarrior and Visual C++.
- **Demand File Loading.** Edits bottom of a file, or anywhere in a file, without loading the entire file first.
- **Autosave.** Automatically backs up modified files at configurable intervals.
- **Document Math.** Just type some numbers into a file, select them, and use the **add** command ("Tools", "Add Selected Expr") to compute the sum. See "Calculating Mathematical Expressions" on page 88.
- **Multi-Platform Configuration.** Recorded macros are source code and binary compatible between Windows and UNIX. Configuration data, including key bindings, bitmaps, dialog box templates, and Slick-C macro Pcode, are compatible across all platforms (except IBM System/390 because it uses EBCDIC and not ASCII).
- **DLL Interface.** Add features with a C or C++ DLL. See "DLL Interface" on page 342 for more information.

Development Language Support

Language Support for Main Features

The following table indicates the languages that support a particular feature. Features that are not language-specific, such as DIFFzilla, are not listed here.

Feature	Supported by
Context Tagging	C++, C, C#, Java, XML, HTML, XML, JavaScript, VBScript, CFScript, JSP, PHP, *COBOL, *OS/390 Assembler, *PL/I, *CICS, Perl, Python, IDL, InstallScript, Delphi, Pascal, Ada, PV-WAVE, VHDL, Slick-C

Syntax expansion	C++, C, C#, Java, JavaScript, XML, VBScript, CFScript, JSP, PHP, *COBOL, REXX, PL/SQL, Transact SQL, Perl, Tcl, Python, IDL, InstallScript, Delphi, Fortran, Pascal, Ada, PV-WAVE, VHDL, Slick-C
Syntax indenting	C++, C, Java, C#, HTML, XML, JavaScript, VBScript, JSP, CFScript, JSP, PHP, *COBOL, REXX, PL/SQL, Transact SQL, Perl, Tcl, Python, IDL, InstallScript, Delphi, Fortran, Pascal, Ada, PV-WAVE, VHDL, Slick-C
SmartPaste	C++, C, Java, XML, *PL/I, JavaScript, JSP, AWK, IDL, InstallScript, PL/SQL, Perl, Python, PV-Wave, PHP, Tcl, Delphi, Pascal
Code Beautifier	C++, C, C#, Java, JavaScript, HTML, XML, JSP
Class Browser, Symbol references, Procs tab, Symbols tab, Tag navigation, Selective display	C++, C, C#, Java, HTML, JavaScript, VBScript, CFScript, JSP, PHP, *COBOL, *OS/390 Assembler, *PL/I, *CICS, REXX, JCL, PL/SQL, Transact SQL, Perl, Tcl, Python, IDL, InstallScript, Delphi, Fortran, Pascal, Ada, PV-WAVE, VHDL, Intel Assembly, Modula-2, Slick-C
Select code block, Hide code block	C++, C, C#, Java, HTML, JavaScript, PHP, PL/SQL, Perl, Tcl, IDL, InstallScript, Delphi, Pascal, Ada, Modula-2, Slick-C
Javadoc Editor	C++, C, C#, Java, JavaScript, IDL, InstallScript, tagdoc, Slick-C
Color Coding	Too numerous to list.

* Support for this language is licensed separately on all platforms except OS/390 or z/OS.

Embedded Languages in HTML, Perl or UNIX Shell Scripts

Visual SlickEdit recognizes languages embedded in HTML, COBOL, Perl scripts, and UNIX shell scripts. When editing embedded languages, all language-sensitive features are supported, including Context Tagging™, SmartPaste™, syntax expansion, syntax indenting, and color coding. In fact, the tagging features pick up embedded tags. For example, the “Procs” tab on the Project toolbar will display function names if any exist. Embedded language colors are user-definable.

Embedded Languages in HTML

Visual SlickEdit supports any embedded language in HTML, except HTML. However, Web browsers typically only support VBScript, JavaScript, and/or Java, while web servers typically support VBScript, Java, or PHP.

The following example illustrates VBScript, JavaScript, and Java embedded in HTML:

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

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

Embedded Languages in Perl or UNIX Shell Scripts

To allow Visual SlickEdit to recognize embedded source in a UNIX shell/Perl script, prefix your Here Document terminator with the color coding lexer name.

Perl Example:

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

The above example illustrates HTML embedded in a Perl script. Unknown languages are color coded in string color.

Adding Support for Other Languages

To set up language-specific color coding for a new language, use the Color Coding Setup dialog box ("Tools", "Configuration", "Color Coding..."). See "Color Coding" on page 137 for more information.

You can often add begin/end structure matching using the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab). You can define extension-specific aliases to provide or override syntax expansion (see "Alias Facility" on page 154 for more information). Other language support such as tagging, syntax expansion, and syntax indenting will require writing Slick-C or DLL source code.

To add simple tagging for another language, read the comment at the beginning of the macro module "tags.e" and look at the "prg_proc_search" function in "prg.e" (only 20 lines of code). For help adding Context Tagging for another language, please contact Product Support and they will send you some sample code.

NOTE: Implementing Context Tagging for a new language can take anywhere from 1 week to 3 months depending upon the language.

To add syntax expansion and indenting for other languages, follow these steps:

1. Use the "prg.e" macro as a template. 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.

NOTE: Do not use any space characters in these constants.

3. Change the name of the first 5 characters of the **_command** functions "**dbase_mode**", "**dbase_enter**", and "**dbase_space**" to use the value you gave the MODE_NAME constant in step 2.
4. Modify the **prg_expand_enter** function to provide the Enter key support you want.
5. Modify the **prg_expand_space** function to provide the SPACEBAR key support you want. If you can rely on extension-specific aliases, follow the comment in this function.
6. Use the **load** command ("Macro", "Load Module...") to load the new macro module you have written.

Steps 4 and 5 are difficult because they require you to have a good understanding of the Slick-C language and what this specific macro is doing. See "[Slick-C Language](#)" on page 248 for more information, or contact Product Support if you need help learning Slick-C. If you add useful language support please send it to SlickEdit Inc. so we can distribute it to other users.

Getting Started

Quick Start

This section covers important features and configuration items and is designed to help you get started using Visual SlickEdit quickly. The remaining chapters contain detailed information on all of the features of Visual SlickEdit.

File Type Association

Windows:

To set up file types which run Visual SlickEdit when you open them in Windows Explorer and File manager, use the Associate File Types dialog box ("Tools", "Associate File Types..."). If you associate workspace files (".vpw" extension) to Visual SlickEdit, Visual SlickEdit will restore your edit session and project when you open the ".vpw" file.

Selecting an Emulation

Because many programmers are already comfortable with the key assignments they are currently using, Visual SlickEdit allows you to select the emulation that most closely resembles what you are used to from the following supported emulations:

- Brief
- CodeWarrior
- Epsilon
- GNU Emacs
- ISPF
- SlickEdit
- VI
- Visual C++

If you do not have a preference, choose the default CUA emulation.

To change the initial emulation selection, access the Emulation dialog box ("Tools", "Configuration", "Emulation...") and select the desired emulation.

Search Options

Use the **Search tab** ("Tools", "Configuration", "General...", select "Search" tab) to set your default search options. There are many more options here than just the typical options for search case sensitivity and whether regular expression searching is turned on by default. For example, you might want the Find dialog box "Search For:" combo box to be initialized with the selected text or the word at the cursor. For more information, bring up the "Search" tab and press F1 for detailed help.

TAB Key and Indent Options

Inserting Real Tab Characters

If you want the TAB key to insert a tab character, turn on the "Indent with tabs" check box ("Tools", "Configuration", "File Extension Setup..."), choose the extension you want to change and select the "Indent" tab.

Inserting Spaces Instead of a Real Tab

If you want the TAB key to indent your text with spaces, turn off the "Indent with tabs" check box ("Tools", "Configuration", "File Extension Setup..."), select the "Indent" tab, and choose the extension you want to change). If you are in SlickEdit (this is not the default emulation) or ISPF emulation, you need to change the TAB key binding to "move-text-tab" and the SHIFT+TAB key bindings to "move-text-backtab" with the following steps:

1. Access the Bind Command to Key dialog box ("Tools", "Configuration", "Key Bindings...").
2. Type **move-text-tab** in the "Command" combo box.
3. Click "Add Key or Mouse Click..."
4. Press TAB.
5. Click "Bind".
6. Type **move-text-backtab** in the "Command" combo box.
7. Click "Add Key or Mouse Click..."
8. Press SHIFT+TAB.
9. Click "Bind".
10. Click "Done".

Changing the Tab Stops and Indent

By default, TAB indents your text by 4 characters for source code files. Typically you will want the tab stops to match the TAB key indent. However, to work properly with SUN's Java API source code, the tab stops need to be in increments of 8, but the TAB key indent ("Syntax Indent") should be set to 4. The "Syntax Indent" affects not only the Tab key, but also how much to indent for each code block level.

Follow these steps to change the amount the TAB key indents ("Syntax Indent"):

1. Access the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup..."), select the "Indent" tab, and choose the extension you want to change.
2. Change the "Syntax Indent:" text box to the amount you want the TAB key to indent.

3. (Recommended for all languages except COBOL) Change the "Tabs" text box to "+*value*" where *value* is the same value you used for the "Syntax Indent" text box. The "Tabs" text box ONLY affects how tab characters are expanded on the screen. This does not affect the indent when you press TAB, or the amount of indent for statements inside a code block.
4. Click OK.

NOTE: For COBOL files, the "Tabs" text box also affects the TAB key. Syntax indent still affects the indent for each code block level.

Indent Selection When Text Selected

If you want your TAB key to indent the selection when text is selected, turn on the "Indent selection when text selected" option ("Tools", "Configuration", "File Extension Setup..."), select the "Indent" tab, and choose the extension you want to change.

Selecting a Different Font

If you want to use a smaller or larger font than Visual SlickEdit's default font, access the Font Configuration dialog box ("Tools", "Configuration", "Font"), select "Window Text" in the combo box at the top of the dialog box, and choose a font.

Windows: Choose the "Terminal" font for the largest selection of attractive, fixed fonts. These are the fonts used for a command prompt.

UNIX: Choose Adobe Courier, B&H Lucida Typewriter, or *Width x Height* family fonts for a broad selection of fixed fonts.

Redefining ENTER and Other Common Keys

Because many users have a preference for the functions of ENTER, BACKSPACE, DEL, HOME, LEFT, RIGHT, UP, and DOWN, Visual SlickEdit includes a Redefine Common Keys dialog box specifically designed for changing the function of these keys ("Tools", "Configuration", "Redefine Common Keys..."). For detailed information on this dialog box, see "[Redefine Common Keys dialog box](#)" on page 227. For help on changing the function of other keys, see "[Changing Key Bindings](#)" on page 224.

Syntax Expansion Options

When you type a keyword such as **if** and press SPACEBAR in a source code file that Visual SlickEdit recognizes, a template like the one below is inserted:

```
if () {  
}
```

The above template is inserted for C, C++, Java, AWK, Perl, and Slick-C source files. Various syntax expansion options, such as the brace style, are accessed by clicking

"Options..." on the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup..."). If the default syntax expansion options do not provide what you want, or you want to add syntax expansion for a new keyword, define an extension-specific alias by accessing "Tools", "Configuration", "File Extension Setup...". Choose the extension you want to change and click "Alias...". See ["Extension-Specific Aliases"](#) on page 156 for more information.

To turn off syntax expansion, uncheck the "Syntax Expansion" check box ("Tools", "Configuration", "File Extension Setup...") select the "Indent" tab, and choose the extension you want to change.

Setting up a Project

Projects facilitate Context Tagging™ (see ["Context Tagging\(TM\)"](#) on page 45), class/tag browsing (See ["Class Browser"](#) on page 60), dynamic tagging, restoring edit sessions, compiling within the editor, loading files, version control, automatic tag database generation, and more. You should create a project now if you want to use some of these features. See "Creating a Project File" below. For more information on projects and workspaces see ["Creating Projects"](#) on page 32.

If you are using Visual C++ v5.0 or later, Visual Studio .NET or Tornado v2.0 or later, you should create your projects and workspaces in these products and then use the "Open Workspace" menu item ("Project", "Open Workspace...") to open a workspace created from one of these products. If you do not plan on creating projects in Visual SlickEdit, open a workspace from one of these products and skip to the "Compiling" section below. (See ["Using Visual SlickEdit with Visual C++ or Visual Studio .NET"](#) on page 178 for more information.)

Creating a Project File

To create a project:

1. Invoke the "New..." menu item on the Project menu.
2. Select the appropriate type of project.
3. Enter the project name.
4. Choose a directory location. If the directory you choose does not exist, you will be prompted to create it when you click OK. For OS/390 or z/OS the location must be within the Hierarchical File System (HFS). Your home directory is recommended.
5. Enter the executable name (optional).
6. Click OK.

Adding Files to Projects

To add files to your project:

1. Access the "Project Files" tab ("Project", "Project Properties...", select "Files" tab). (The top portion of the "Files" tab operates in a manner similar to the **standard Open dialog box**.)
2. Press CTRL+CLICK to select more than one file in the file list box or SHIFT+CLICK for a range.
3. Click "Add" to add the selected files to the "Project files" list box.

Compiling

The Build menu items "Compile" and "Build" execute the compile and build commands respectively for the current project. If you selected a compiler package, you can try these commands now. To change these commands and a few other project options, use the Project Properties dialog box ("Project", "Project Properties...", Select the Tools tab). The "Next Error" and "Previous Error" menu items on the Build menu allow quick navigation of compiler errors.

For Visual C++ v5.x and v6.x, the default compile command uses the "**nmake**" program which requires a makefile (".mak" extension). Visual C++ v5.x and v6.x do not automatically create a makefile for you. Use the "Export Makefile.." menu item ("Project", "Export Makefile...") in Visual C++ to create or update your makefile. For Visual C++ v5.x or higher, the default build and rebuild commands do not need a makefile.

Version Control

To configure a SCC version control system (Windows only):

1. Access the Version Control Setup dialog box ("Tools", "Version Control", "Setup..."), and select the desired system.
2. Click "Initialize Provider".
3. Click "Open Project" and fill in the Open Project dialog appropriately. (This dialog will vary from system to system.)
4. Click OK.

A project name should now appear under the list of SCC Version Control Systems. This version control system and project are now bound to the active Visual SlickEdit project. For more information, see "[Version Control Support](#)" on page 41.

To configure a command line version control system (most non-Windows version control systems are command line):

1. Access the Version Control Setup dialog box ("Tools", "Version Control", "Setup...")
2. Choose a command line version control interface
3. Click OK.

If you chose a command line version control interface, make sure the executables included with your version control system can be found in the PATH environment variable. Depending on the system you are using, you may need to fill in the "VCS Project" text box. For more information or if you are using a command line system that is not in the "Command line systems" list, see "[Version Control Support](#)" on page 41.

For help with support for your OS/390 version control system, please contact SlickEdit Inc.

Building a Tag File for Run-Time Libraries

If you are using Visual SlickEdit for COBOL or OS/390 Assembler source code, read “[Setting up Context Tagging for COBOL](#)” on page 53, or “[Setting up Context Tagging for OS/390 Assembler](#)” on page 55 respectively, instead of this section.

SlickEdit recommends that you create extension-specific tag files for include files of compiler packages and/or utility libraries. This will allow tagging features to work for all symbols, not just those symbols in your project. Context Tagging™ (see “[Context Tagging\(TM\)](#)” on page 45), which features Auto List Members and Auto Parameter Info, needs all symbol information in order to work properly.

Visual SlickEdit automatically builds a tag file for the run-time libraries of Perl, JavaScript, Tornado, InstallShield, PV-WAVE, TCL, and Slick-C, so it should be unnecessary to build tag files for the run-times of these languages. If you have already built a tag file for your run-times during installation, you can skip this section.

Windows: If you are using Visual C++, C++ Builder, any Java package, or COBOL, use the Automatic Tagging dialog box (“Search”, “Tag Files...”, “Auto Tag...”) to build a tag file for these run-time libraries. This is the same dialog used by the Visual SlickEdit install program.

UNIX: If you are using C, C++ or Java, use the Automatic Tagging dialog box (“Search”, “Tag Files...”, “Auto Tag...”) to build a tag file for these run-time libraries. This is the same dialog used by the Visual SlickEdit installation program.

If the Automatic Tagging dialog box does not help you, perform the following steps to set up an extension-specific tag file for your run-times. This tag file can be accessed by all projects which have files of the same language.

Building an Extension-Specific Tag File

To build an extension-specific tag file:

1. Display the Tag Files dialog box (“Search”, “Tag Files...”).
2. Click “Add Tag File...”.
3. Select the language from the list and click OK. For example:
 - Select “C” to add an extension-specific tag file for C and C++.
 - When the Add Tags Database dialog appears, set the directory to the root directory of the include files or source tree. (For Visual C++, this might be “c:\Program Files\DevStudio\VC\” or “c:\msdev\”. UNIX include source is almost always under “/usr/include”.)
 - Press ENTER to select “tags.vtg” (UNIX: “utags.vtg”) as the output file.
4. If the Add Tree dialog box is not already displayed, click “Add Tree...”. Enter one or more wildcard file specifications in the Files text box (i.e. *.c *.h *.cpp *.hpp).

5. Click OK to recursively add these files to the selected tag file.

Auto Restore Options

By default, Visual SlickEdit restores the files, current directory, and more from your previous edit session. Various Auto Restore options are located on the **General Configuration dialog box** ("Tools", "Configuration", "General...", select General tab). Here you can turn off restoring the current directory, turn off restoring project files, and more.

Directory Aliases

Directory aliases is a feature unique to Visual SlickEdit, and is particularly helpful if you are a touch typist. Directory aliases are typically 4 characters or fewer and consist of a short abbreviation of the last name in a long directory list. By using a directory alias when opening a file you do not have to type in long directories or click the mouse repeatedly in the directory list box. You can also specify aliases in the Change Directory dialog box.

Most users are constantly opening files from a small number of directories throughout the day, so take the time now to identify the directories you access most frequently and define aliases for them. For example, if you had a directory called "c:\version20\src\project2\", a good alias name might be "p2". For your compiler include files, define an alias called "inc" (or "vinc" (MS VC++), "binc" (C++ Builder), "dinc" (Delphi) if you have multiple compilers).

Defining a Directory Alias

Try out this unique feature and you should immediately see how much it improves the way you work. To define a directory alias:

1. Invoke the Alias Editor ("Tools", "Configuration", "Aliases...") and select global (non-extension specific) aliases stored in the "alias.slk" file. Click OK.
2. Click "New" to add a new alias and type in its value.
3. Click OK.

Using a Directory Alias

Now that you have defined a directory alias, here is how to try it out:

1. Access the Open dialog box ("File", "Open...").
2. Type in your alias in the "File Name" text box.
3. Press CTRL+SHIFT+O to expand the alias. (This works in every text box in Visual SlickEdit, in the concurrent process buffer, and in any buffer.)
4. Press ENTER to view the files in this directory or type the rest of the file name to open the file.

Embedding Environment Variables

If you keep your source code in a "version" directory tree, you can set an environment variable and embed the environment variable in the alias value. For example, if you have a directory called "c:\version20\src\project2\", define a "p2" alias and give it a value such as "%VERSION%\src\project2\". Type **set VERSION=c:\version20** on the Visual SlickEdit command line to set or create the VERSION environment variable.

Alternatively, you can have environment variables automatically set when you open a project. See "Open Command" tab on the [Project Properties dialog box](#) on page 34.

Windows Word Help

If you are using the default CUA emulation, you can get help for the word at the cursor by pressing CTRL+F1 (which executes the **wh** command). For C++ compiler packages, this includes help on the Windows API, Standard C library, and Microsoft C Foundation Classes (if you have MS VC++). In any emulation, you can click the "Question Mark" icon for help on the word at the cursor.

Before you can get word help, you need to set up a help index file ("vslick.idx") which is an index for multiple .HLP and .MVB help files. The install program usually builds the "vslick.idx" help index file for you. However, you should follow these steps to ensure you have help for the compiler packages you are using.

Building the Help Index File "vslick.idx"

1. Access the [Configure Index File dialog box](#) ("Help", "Configure Index File...").
2. Click "Scan" to automatically add help files (.hlp) on your PATH. This should also find MS VC++, C++ Builder, and Delphi help files.
3. If "Scan" did not pick up some help files you want, add your other help directories to the "Help File Path" text box. Click "Scan" again.
4. Click OK to build the "vslick.idx" file.

NOTE: Word help is not yet supported for Visual Studio .NET.

Multiple Help File Access

Usually, you only need one help index file for all your compiler packages. However, you may prefer to separate your help files into groups which you can access on different keys.

To change the default word help file name(s) for CTRL+F1 and CTRL+F2 (**wh** and **wh2** commands):

1. Set the "Word Help Filename(s):" on the [General Configuration dialog box](#) ("Tools", "Configuration", "General...", select General tab).
2. Specify the help files for the **wh** command (CTRL+F1), each separated with a plus sign. You can only specify one .IDX or .MVB file per command.

3. Use a semicolon to indicate the start of the help files for the **wh2** command (CTRL+F2) and specify its help files separated with a plus sign.

You can use another semicolon to indicate the start of the help files for the **wh3** command; however, this command is not bound to a key. Help files have the extension .MVB, .HLP, or .IDX. Do not use a plus sign to specify more than one .IDX or .MVB file.

Example:

```
vslick.idx;win31wh.hlp+mscxx.hlp+errors.hlp+vslick.hlp
CtrlF1.idx;CtrlF2.idx
vslick.idx;c:\msdev\help\vcbks40.mvb
```

NOTE: environment variables may be embedded in the word help filename(s) string by the notation %(envvar-name).

UNIX Word Help

If you are using the default CUA emulation, you can get **man** page help on the word at the cursor by pressing CTRL+F1 (which executes the **wh** command). When you are editing a Slick-C macro, pressing CTRL+F1 brings up macro language help for the word at the cursor. You will find this especially useful for getting help on macro source generated during macro recording.

Network Backup Options

Windows:

Sometimes the default backup style does not work correctly when a network file is saved. By default, backups are placed in the "**vslick\backup**" directory. For this backup style to work, make sure Visual SlickEdit has access rights to this directory and that this directory exists. If you set the backup directory to "**c:\vslick\backup**", ("Tools", "Configuration", "File Options...", select "Backup" tab, "Backup Dir") this will force all backups to be written to drive C:. You may prefer a different backup style. (See "**Save Tab**" on page 234 for more information.)

UNIX:

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

Visual SlickEdit only makes a backup of a file the first time you save it in a given edit session. Therefore, your backup file may not represent your last save. For more up-to-date backups, use the AutoSave feature. (See "**AutoSave Tab**" on page 236 for more information.)

Invoking Visual SlickEdit

Windows:

Double-click on the Visual SlickEdit icon created by the install program.

The command line syntax for invoking Visual SlickEdit is often useful. For example, you might want to set up multiple icons which open different Visual SlickEdit projects, or start the editor from a shell prompt.

The syntax for invoking Visual SlickEdit is as follows:

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

Invocation Options

The most common invocation *options* are as follows:

-#command	Execute <i>command</i> on active buffer. For example, "vs test.c -#bottom-of-buffer" places the cursor at the end of test.c.
+t	Creates a new file and bypasses auto-restore of files. See " Auto Restore Options " on page 20 for more information.
+ or -new	Indicates whether a new instance of the editor should be created, or if the existing instance should process the command line parameters (default is -new). +new creates a new instance.

The complete list of invocation *options* is as follows:

+ or -new	Indicates whether a new instance of the editor should be created or if the existing instance should process the command line parameters. +new creates a new instance. Default is -new.						
-sc config-path	Specifies the configuration directory. Visual SlickEdit will use this directory to find and save configuration files. Sets the VSLICKCONFIG environment variable to <i>config-path</i> ..						
-suo option-letters	OS/390: <i>option-letters</i> is a string of one or more of the following letters: <table><tr><td>m</td><td>Disable mouse over detection</td></tr><tr><td>t</td><td>Disable large toolbars</td></tr><tr><td>c</td><td>Disable cursor blink</td></tr></table> These options reduce CPU utilization and speed up start time of the OS/390 version.	m	Disable mouse over detection	t	Disable large toolbars	c	Disable cursor blink
m	Disable mouse over detection						
t	Disable large toolbars						
c	Disable cursor blink						
-sua	UNIX: Enables right ALT key for foreign characters.						
-supf kbdfile	UNIX: Specifies a keyboard file for mapping the keyboard at the X (modmap) level. Some UNIX systems have keyboard mappings which need to be modified so that keys like Backspace and Delete (and sometimes others) function properly. Use the xmapkeys program to generate the keyboard file.						

-sr <i>restore-path</i>	Specifies the directory containing the Visual SlickEdit auto-restore files. Sets the VSLICKRESTORE environment variable to <i>restore-path</i> .
-si <i>kmax_var_space max_Nofvars</i>	<i>kmax_var_space</i> specifies the amount of memory in kilobytes allocated for storing the contents of interpreter variables (default is 1000k. <i>max_Nofvars</i> specifies the maximum number of local, static, and global variables there may be (default is 10000).
-st <i>state_ksize</i>	Specifies the maximum amount of swappable state file data in "vslick.sta" (UNIX: "vslick.stu") to be kept in memory, in kilobytes. -1 specifies no limit. 0 specifies that the editor preload all state file data. The default is 200k.
-sm <i>max_file_size</i>	Specifies the maximum amount of buffer text (in megabytes) that may be edited at one time. Visual SlickEdit will automatically allocate more memory to allow for the editing of up to 2 gigabytes of files. However, you might want to use this option to get better performance.
-x <i>pcode_name</i>	Alternate state file (.sta) or pcode file (.ex).
-m <i>menu_file</i>	Name of menu resource to use for MDI menu bar.
-p <i>cmdline</i>	Execute command with arguments given and exit. No other options or file names can be specified after this option since the rest of the command line is assumed to be the program name and space delimited arguments for this option.
-r <i>cmdline</i>	Execute command with arguments given and remain resident. No other options or file names can be specified after this option since the rest of the command line is assumed to be the program name and space delimited arguments for this option.
<i>file1 file2 ...</i>	Files to edit. File names may contain wildcard characters (* and ?).
-#<i>command</i>	Execute <i>command</i> on active buffer. For example, vs test.c -#bottom-of-buffer places the cursor at the end of test.c. Use double quotes if your command has spaces (e.g. vs test.c "-#goto-col 50").
+ or -L[C]	Turn on/off load entire file switch. The optional C suffix specifies counting the number of lines in the file.
+ or -S	Turn on/off unmodified block swapping to spill file.
+<i>nnn</i>	Load binary file(s) that follow with a record width <i>nnn</i> .

+T [buf_name]	Start a default operating system format temporary buffer with name <i>buf_name</i> .
+TU [buf_name]	Start a UNIX format temporary buffer with name <i>buf_name</i> .
+TM [buf_name]	Start a MACINTOSH format temporary buffer with name <i>buf_name</i> .
+TD [buf_name]	Start a DOS format temporary buffer with name <i>buf_name</i> .
+ or -E	Turn on/off expand tabs to spaces when loading file. Default is off.

Editing Files

Visual SlickEdit has full support for file names up to 200 characters long. If you need to edit a file whose name contains space characters, place double quotes around the name.

When you specify a file to edit, Visual SlickEdit automatically recognizes whether its format is DOS, UNIX, or MACINTOSH and adjusts accordingly. Visual SlickEdit will not append an EOF character to a UNIX or MACINTOSH format text file when saved, regardless of the save options. When text in one text file is copied to a buffer with another text file format, Visual SlickEdit translates the line separation characters to the destination buffer's format.

If you specify a workspace file name with extension (".vpw"), Visual SlickEdit will **Auto Restore** the workspace. If you specify a project file name with extension (".vpj"), Visual SlickEdit will **Auto Restore** the project.

Examples:

vs project1.vpw	Auto Restore from workspace file.
vs project1.dsw	Auto Restore from a VC++ workspace file.
vs autoexec.bat config.sys	Edit two files.
vs "this is.c"	Edit a file with a space character.
vs test.c -#1000	Edit test.c and go to line 1000.
vs orders -#bottom-of-buffer -#/invoice/-	Edit "orders" file, go to bottom of buffer, and search backward for "invoice".
vs +70 test.exe	Edit binary file test.exe in record width 70.
vs *.c	Edit all C source files in the current directory.

Getting Started

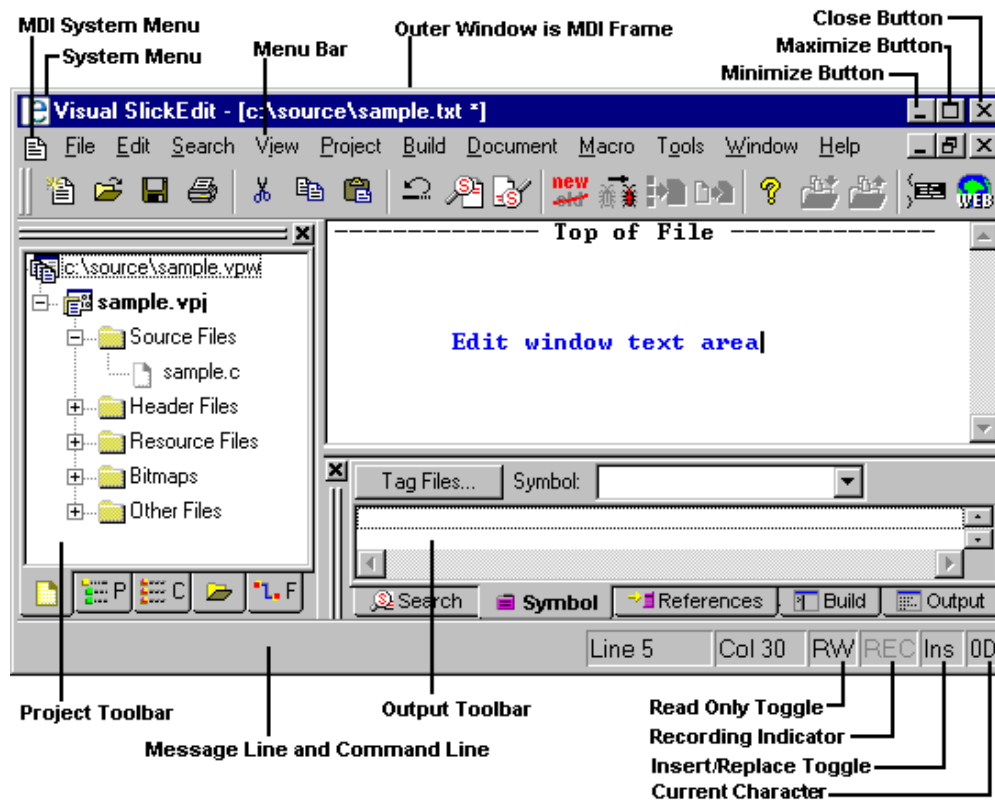
vs -r list \	Start the file manager (Windows) and list entire drive contents.
vs -x rich.sta autoexec.bat	Specify different state file and edit the file autoexec.bat

When Visual SlickEdit is invoked, it concatenates the value of the environment variable VSLICK before the command line you specify.

When Visual SlickEdit does not find a file you wish to edit, an empty buffer is created with that name.

The Screen Layout

Visual SlickEdit uses a Multiple Document Interface (MDI), common to Windows applications which support editing multiple documents.



The initial screen that appears when you invoke Visual SlickEdit might be somewhat different from the one displayed above because a file has already been opened and the window has been maximized. This will be explained in more detail later in this chapter.

NOTE: UNIX has a different minimize button, maximize button, restore button, system menu icon, and MDI system menu icon.

The outer window is called the MDI frame or window. If you want a non-clipped MDI child window you will need to start a new instance of Visual SlickEdit.

The MDI system menu displays various commands that operate on the MDI window. Single-click on the MDI system menu to display a menu of commands. The most commonly used item here is "Close" which is used to exit Visual SlickEdit. The MDI System Menu "close" option closes the current buffer; the System Menu "close" exits the application.

To expand the MDI window so that it fills the entire screen, access the "Maximize" menu item or click the MDI window maximize button. Once maximized, you can restore the MDI window to its original size by selecting the "Restore" command from the MDI system menu or by single-clicking the MDI window restore button. (The "maximize" button is changed to a "restore" button when the MDI window is maximized).

The restore button to the right of the menu bar operates on the edit window, not on the MDI window. The system menu to the left of the menu bar also operates on the edit window and not on the MDI window. To practice these functions perform the following steps:

1. Invoke Visual SlickEdit if you have not already done so.
2. From the "File" menu select the "Open" menu item by pressing ALT+F followed by the letter 'O' (menu items with underscores are known as selection characters). A dialog box with the title "Open" should appear.
3. Type **practice** into the "File name" text box.
4. Press ENTER or click OK. If the "practice" file does not exist, a new file will be created and given the name "practice".

At this point the editor should be displaying a smaller window inside the MDI window which includes the title "practice" (actual title will include path information). The smaller window is called the edit window or MDI child window, and has three buttons at the top, like the MDI window. Click on the edit window maximize button. Notice that the edit window now looks like the screen layout picture shown above. These additions to the menu bar operate on the edit window, not on the MDI window.

You can use the MDI minimize button to iconize the editor. **Windows:** Once iconized, you can click on Visual SlickEdit in the task bar at the bottom of your screen. **UNIX:** The way you restore an iconized application depends on the window manager you are using. Under the Motif and Open Look window managers you double-click on the icon to restore the application window.

Message Line and Command Line

The message line displays one-line information messages. Because the message line and the command line share the same screen space, when you click on the message line or invoke an editor command which requires the command line, the message line disappears and the command line appears.

Line and Column Indicators

To the right of the command line are the line and column indicators which tell you the current cursor position. You can click on these indicators to move the cursor to a new position.

Read Only Toggle

The "RW" letters indicate that the current buffer is read-write. When these letters change to "RO", it means that the buffer is read-only. You can click on the Read Only Toggle to toggle between modes.

Recording Indicator

Indicates whether or not you are recording a macro. See "[Recording and Using Macros](#)" on page 90 for more information.

Insert/Replace Toggle

The letters "Ins" to the right of the recording mode flag indicate that you are in insert mode. In insert mode, typing a character pushes characters at and after the cursor to the right. You will see the letters "Rep" to the right of the recording mode name when you are in replace mode. In replace mode, typing a character replaces the character under the cursor. The cursor changes shape.

Current Character

When editing an SBCS/DBCS mode file (non-Unicode file), the current character is displayed in hexadecimal format. If the current character is a double byte character (DBCS), then two bytes and its Unicode equivalent are displayed in hexadecimal (e.g. 95 74 U+4ED8). When editing a Unicode file, the current composite character is displayed in hexadecimal. This field is blank when the cursor is past the end of the line.

Scroll Bars

The scroll bars on the right and bottom sides of the edit window are not required. If you do not want scroll bars, use the [General Configuration dialog box](#) ("Tools", "Configuration", "General...", select General tab) to remove them.

Edit Window Text Area

The edit window text area displays the contents of files you edit. When the edit window is maximized, the buffer name is displayed at the top of the MDI window. Otherwise, the buffer name appears at the top of the edit window. A '*' character displayed to the right of the buffer name indicates a modified buffer.

Toolbars

In the default configuration, Visual SlickEdit displays the Standard toolbar (the button bar), the Project toolbar, and the Output toolbar. These toolbars contain the most commonly used editing features. You can access all toolbars, completely customize existing toolbars and even create your own personal toolbars with the Toolbar Customization dialog box ("View", "Toolbars").

Help Features

You can learn Visual SlickEdit while you are running the editor. The easiest way is to browse menu items in the menu bar. Menu items display keyboard shortcuts you can later memorize. As you move the cursor through menu items, a help message for the current menu item is displayed in the message line. You can press F1 to get more help for the current menu item. For help on most dialog boxes, press F1 while the dialog box is displayed.

For example, if you want to learn about the "Find..." menu item on the "Search" drop-down menu, follow these steps:

1. Press ALT to activate the menu bar.

2. Type **S** to select the search drop-down.
3. Move the menu item selection cursor down to the "Find..." menu item and press F1.

Tool Tips

If you do not know the function of a button on the button bar, move the mouse over the button and hold it there for a moment. Visual SlickEdit will display a description of the button's function.

Hypertext Help

Certain words are highlighted in the help for the "Find..." menu item, such as "Regular Expressions". This means you can get help on regular expressions by single-clicking on the highlighted text. The following are some ways you can use hypertext help:

- Under Windows or UNIX, click the "Back" button (below the menu bar of the help program) to return to the previous help item.
- TAB and SHIFT+TAB will move your cursor to the next and previous hypertext items.
- Click "Contents" to view the table of contents (which is almost identical to the table of contents of this guide).
- Click "Index" to see an index of help items. The contents of this entire guide, including the sections on the Slick-C language, are included in the hypertext help.
- The Help menu drop-down contains menu items for getting help on a summary of keys and Slick-C macro functions by category.
- If you remember the help item you are looking for, you can use the menu item ("Help", "Search") or the **help** command from the command line to display help on the item.

Special Mouse Functions

This section covers special mouse functions. The mouse is configured the same for all emulations.

Selecting Lines and Columns

If you are editing source code files, you will select complete lines quite often. A quick way to do this with the mouse is to click in the space between the left edge of the buffer and the edit window border. Click and drag the mouse in this area to select multiple lines. You can also triple-click on the text in a line to select the current line, and click and drag with the right

mouse button to create a column selection. Double-click selects a word and click. Use click-and-drag for character selection.

Snapping Cursor to End of Line

Most of the emulations will allow you to place the cursor past the end of a line if you click there. To disable this feature, set the macro variable **def_click_past_end** to 0 ("Macro...", "Set Macro Variable").

Docking Toolbars

Click in a toolbar (not on a control) and drag to dock or undock a toolbar. Double-clicking in a toolbar will also dock and undock the toolbar.

Copying and Moving Text

You can copy or move selected text to the mouse pointer by using CTRL+SHIFT+RButton to copy the selected text and CTRL+RButton to move the selected text. You might prefer this method to the drag drop method because you can use the scroll bars to copy or move the text a long distance. In addition, you can copy the selected text into an entirely different buffer/window.

Example:

1. Select some text in a buffer.
2. Use the scroll bars to go to the bottom of the buffer.
3. Press CTRL+SHIFT+RButton to copy the selected text to the cursor.

Changing Values on the Status Line

You can click on the Line, Col, RW/RO, and Ins/Rep indicators near the bottom of Visual SlickEdit's MDI window to change their value.

Accessing Sub-Menus

Clicking with the right mouse button often brings up a menu with items specific to the window where you clicked. For example, right clicking inside the edit window brings up a list of commonly used editing functions. Context-sensitive menus are supported in edit windows, all toolbars, the Diff dialog box, the multi-file Diff Output dialog box, and in the Tag Files dialog box.

Organizing Your Code

Organization Overview

This section covers features that will help you organize your code. If you are using Visual C++, be sure to read ["Using Visual SlickEdit with Visual C++ or Visual Studio .NET"](#) on page 178.

Creating Projects

What is a Project?

A project is a great way of organizing your work into a neat package. This package includes a working directory, Compile/Build command lines (where you specify what to execute for your Compile/Build), which source files you want as part of the project, and a few other things. Once you supply the editor with this information, you can do compiles/builds from within Visual SlickEdit and have your error messages automatically processed. A tag file for all of the project's source files is automatically created and maintained so that you can navigate your source code and use the Context Tagging™ features (see ["Class Browser"](#) on page 60, [Dynamic Tagging](#) on page 57 and ["Context Tagging\(TM\)"](#) on page 45). A few other dialogs take advantage of your project source files, such as the Load Files (Project menu), Find, Replace, and Print dialog boxes.

The Build menu allows you to execute the various project commands. In addition, the button bar contains buttons to execute the Build and Compile project commands. If you are using one of the supported version control systems (see ["Version Control Support"](#) on page 41), you can easily keep your version control project in sync with Visual SlickEdit projects. The data for each project is stored in a text file which has the extension ".vpj".

If you do not want to create a project and you are building something that only contains one source file, you can define an extension-based project. Extension-based projects are based on the extension (file type) of the current file. However, the working directory is ignored for extension-based projects. All extension-based projects are stored in the file "project.slk" (UNIX: "uproject.slk"). To define an extension-based project, use the [Extension Options dialog box](#) ("Tools", "Configuration", "File Extension Setup...", select the "Advanced" tab, select the extension, and click the "Extension Specific Project" button).

When you create a project, you must either create a workspace or add the new project to the current workspace.

What is a Workspace?

The purpose of a workspace is to allow you to share projects in multiple workspaces and to define dependencies between projects. For example, building Project A might depend on building Project B first, which is a utility library linked with Project A. You might find that you only need one project for each of your workspaces. In this scenario, workspaces do not help you.

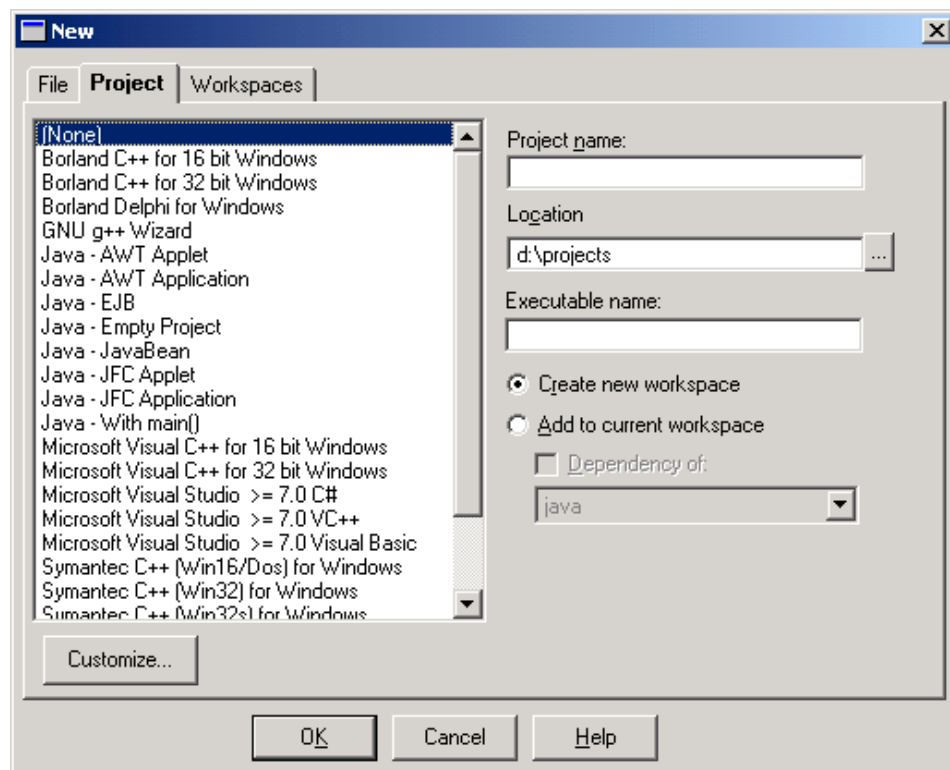
An additional feature with workspaces is compatibility with other vendors' workspaces. Visual SlickEdit supports opening Visual C++ and Tornado workspaces, allowing more seamless integration with these products. The data for each workspace is stored in a text file which has the extension ".vpw".

Creating a Workspace and Project

If you are using Visual C++ or Tornado, you can create your workspaces and projects in these products and then open their workspace ("Project", "Open Workspace...") in Visual SlickEdit.

To create a workspace and project:

1. Invoke the "New..." menu item from the Project menu. The New dialog box will appear:



Note that some packages ignore the executable name.

UNIX: The UNIX and OS/390 versions list different packages.

2. Select the appropriate package (or choose "None").

3. Enter the project name.
4. Choose a directory.
5. Enter the executable name and click OK.

You might see a different list of packages than the ones displayed.

Use the "Customize" button to modify or create project package configurations. System-defined packages are stored in "prjpacks.slk" (UNIX: "uprjpack.slk"). User-modified or additional packages are stored in "usrpacks.slk".

Project Configuration Settings Dialog Box

The Project Configurations Settings dialog box ("Build", "Configurations...") allows you to view, add, and delete project configurations. The most common use of project configurations is to allow you to build a debug or release version without having to define a new project. Changing the configuration will have no effect unless you either have different build commands, or use the "%b" (current configuration) or "%bd" (object directory) escape sequence in your build commands. You can set the active configuration with the Set Active Configuration menu item ("Build", "Set Active Configuration").

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

If you are using a make program, define a macro such as "CFG" which is the configuration you want to build. Then add code to your makefile to check for this macro and perform some different statements like choosing different compile options and a different directory for object files. The makefiles exported from Visual C++ already define a "CFG" macro. For a standard make program you will need to use the "**name=value**" syntax when passing a macro to the make program.

Adding Files to a Project

Use the Project Properties dialog box ("Project", "Project Properties...") and select the "Files" tab to add source files to a project. The top portion of the **Files tab** operates in a similar manner to the **Standard Open dialog box**. Press CTRL+Click to select more than one file in the files list box, then click "Add" to add the selected files into the "Project Files" list box.

Once you have added source files, a project tag file has already been built for you. Your tag files are automatically updated when you make edits. Now you can place your cursor on a call to a function (or another tag type depending on your language) and press CTRL+PERIOD (or use "Search", "Go To Definition..." menu item) to go to a tag definition. To "pop" back to where you were, simply press CTRL+COMMA (or use "Search", "Pop Bookmark..." menu item).

Setting Directories

To set the working directory, references file and include file search directories for the current project, use the project **Directories** tab on the Project Properties dialog box ("Project", "Project Properties...").

Working directory

When a project is set active, the current directory is set to the working directory (if specified). This information is stored per project and not per configuration.

Windows: In addition, when you start a compile, the current directory is changed to the Working Directory. To turn this feature off, use the "Compile Options" tab on the Project Properties dialog box ("Project", "Properties..", select the "Compile Options" tab) and uncheck the "Change to project directory" check box.

References file

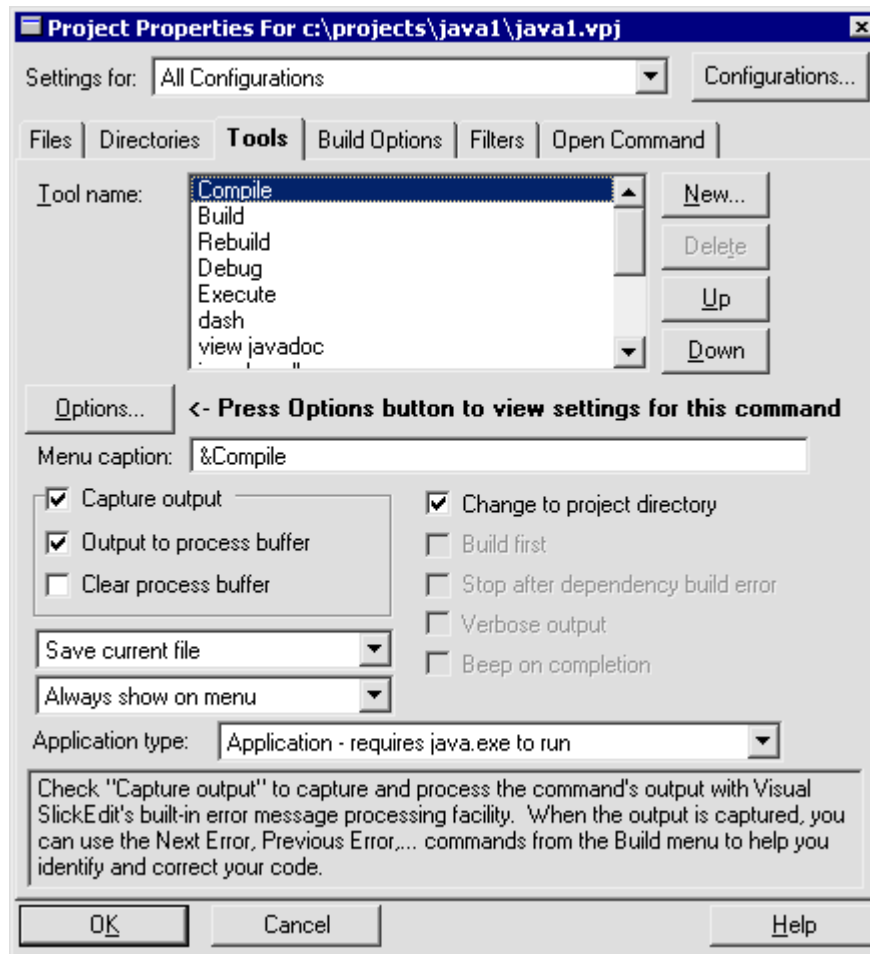
You only need to fill in this text box if you are using Microsoft Visual C++ and you prefer to use a Visual C++ ".bsc" database file instead of the Visual SlickEdit tag database when viewing references. If you open a Visual C++ v5.0 or later workspace and you have configured Visual C++ to generate a ".bsc" database file, Visual SlickEdit will automatically configure this field. See "**Viewing Symbol References**" on page 62 for more information.

Include directories

Specifies the directories the **cursor_error** (ALT+1) and **next_error** (CTRL+SHIFT+DOWN) commands will search when trying to open a file. For COBOL and OS/390 Assembler, this list of directories (or PDSs for OS/390) is used to find copy books or macros. You might want to add some of your own include directories here before your compiler's include directories. Although it is not obvious from looking at the Project dialog box, you can specify environment variables with the syntax **%(EnvVarName)**.

Setting Compile Commands

You can set the Compile, Build, Rebuild, Debug, Execute, and user-defined commands for the current project on the "Tools" tab of the Project Properties dialog box ("Project", "Project Properties...").



You should see settings similar to those above if you selected a "Java" package. For this Compile tool, click "Options..." to view the settings for this command. When the "Options..." button is not displayed, there is a text box below the "Tool name" list box which displays the command line executed for the selected tool.

Configurations...

Projects can have multiple configurations. The "Settings for" combo box allows you to select the configuration to modify. The "All Configurations" option allows you to change the settings for all your configurations. All settings in the Project Properties dialog are per configuration except the working directory, open command, and filters. Click "Configurations..." to view, add, or delete configurations. The most common use for multiple configurations is to have a debug and a release version. To do this, you need to either configure your build commands to place the object code in different directories or perform a rebuild after switching configurations. In addition, the build commands for the debug version need to specify any debug options. See "[Project Configuration Settings Dialog Box](#)" on page 34 for more information.

Tool Name

Use the "Tool Name" combo box to select the tool for which you want to define the command line. Click "New..." to create a new tool to appear on the project menu.

Options...

Not always present. This button displays a dialog for setting options for the command which gets executed for the tool specified in the "Tool name" combo box.

For Java, Visual SlickEdit provides a Java Options dialog which allows you to easily change Java tool options, including your class path. Click "Options..." when present to customize Java tool options. The "Options..." button lets you customize options supported by "javac", "javadoc", and "jar". To change the compiler from "javac" to another compiler (such as "sj" or "jikes"):

1. Click "Options..."
2. Click the "Compiler" tab
3. Click the "Other" tab
4. Fill in the "Compiler name"

vsbuild

vsbuild is a utility program used to compile files in a Visual SlickEdit project and process dependencies between projects. This tool is intended to help Visual SlickEdit implement its project support. It has a built-in "make" facility for Java and C++, performs project dependencies, and processes pre- and post-build commands. For example, if "file1.java" references "file2.java" which references "file3.java" and "file3.java" is modified, then when you invoke the Build command "file1.java", "file2.java", "file3.java" will be recompiled. Invoking **vsbuild** with no parameters displays invocation options.

New...

Allows you to add a new tool to the Project Menu. You can have different tools for each project.

Delete

Deletes the current tool.

Command line

Not always present. This text box defines the command line which gets executed for the tool specified in the "Tool name" combo box. This field is initially blank when you modify settings for "All Configurations" and the settings differ for different configurations. Click the button to the right of this text box for a list of escape sequences you can use to build your command line (such as "%f" which inserts the current buffer name).

OS/390: The following are some sample command line compiles:

c89 -o %n.o -c %n.c	Compile C source HFS file or data set.
c89 -o %n.o %n.s	Compile Assembler source HFS file or data set.
c++ -+ -o %n.o %n%.e	Compile C++ source HFS file or data set.

NOTE: **%n.<ext>** and **%f.<ext>** have very special processing when applied to data sets. In general, given `"/ds.name.sourcext/member"`, **%n.<ext>** is translated to `"/ds.name.<ext>(member)"`. However, for convenience, **".o"** and **".s"** are converted to **".obj"** and **".asm"** respectively which is the standard convention for c89 and C++.

Limitations: c89 requires that the data set containing C source has a **".c"** extension. c89 requires that a data set containing assembler source have a **".asm"** extension. If this is too restrictive for you, use the `vscomp.rexx` script instead to run the C compiler. Create a new project and select the appropriate C compiler for sample command line strings.

UNIX: If you are running text mode utilities, you might want to prefix the program name with **xterm -e** or **dos -w** (this waits for a key press) so you can see your program's output. You should not do this when either "Output to process buffer" or "Capture output" is checked because your program's output cannot be captured. OS/390 does not yet provide an xterm program.

Menu caption

This text box defines the menu item text which appears on the Project menu. Prefix the selection character with an ampersand (&) character to choose the selection character. If you have run out of selection letters, try using numbers. For example, **&1 MyTool** picks 1 as the selection character.

Output to process buffer

This check box specifies that the command be run in Visual SlickEdit's concurrent process buffer. The "Build" tab of the Output toolbar displays the process buffer. The concurrent process buffer has the following limitations:

- You cannot run graphical applications in the concurrent process buffer.
- Only programs that use standard in and standard out to read and write data can run in the concurrent process buffer. This is true for most compilers.
- Some programs which use standard in and standard out will not run properly in the concurrent process buffer because they use ANSI escape sequences or do not flush standard output data before prompting for input.
- **Windows:** Some 16-bit programs will not run in the concurrent process buffer. This is more common under Windows 98/ME. Most of the 16-bit programs that will not run in the concurrent process buffer under Windows 98/ME will run in the concurrent process buffer under Windows 2000/NT.

- **Windows 2000/NT:** Alternate command shells are not supported. Only CMD.EXE is supported.
- **Windows 98/ME:** Only Visual SlickEdit's command shell (slkshell) is supported.
- **Windows 98/ME:** A .process icon will appear on your task bar.

Capture output

This check box specifies that the output of the command will be captured and processed with Visual SlickEdit's built-in error message processing facility. The **next_error** (CTRL+SHIFT+DOWN) and **prev_error** (CTRL+SHIFT+UP) commands are used to go to the next and previous compilation error positions respectively.

UNIX: Output of text mode programs that are executed using **xterm** cannot be captured.

Clear process buffer

When this is checked, the process buffer output is cleared before the command is executed. The "Build" tab of the Output toolbar displays the process buffer.

Save current file

When this is on, the current file is saved before the command is executed.

Save all files

When this is on, all modified files are saved before the command is executed.

Save workspace files

When this is on, modified workspace files are saved before the command is executed.

List modified files

When this is on, a selection list of modified files is displayed which allows you to choose files to save before the command is executed.

Save none

When this is on, no files are saved before the command is executed.

Always show on menu

When this is on, this command will always show on the build menu.

Hide if no command line

When this is on, the menu item will not be shown if the command line is blank. This feature is intended to save space on the build menu for blank command lines.

Never show on menu

When on, this command will never be shown on the build menu.

Change to project directory

When this is on, the current directory in the concurrent process buffer, and for Visual SlickEdit, is set to the project working directory before the command is executed.

Build first

When this is checked, the build command is executed before this command. If the build completes with a non-zero return code, this command is not executed. This option requires the use of the **vsbuild** utility and will not work for build commands that do not return a valid return code. If your build command returns a zero return code, your command is executed even if your build actually failed.

Windows: This option does not support batch programs under Windows 95/98/ME. Under Windows NT, batch programs return the return code of the last command executed. For commands which will not execute in the concurrent process buffer, prefix your command with “**start**” (e.g. **start debug\myprogram.exe**)

UNIX: If your build command is a shell script, make sure it returns a zero return code for a successful and a non-zero return code for an unsuccessful build. For commands which will not execute in the concurrent process buffer, prefix your command with “**xterm -e** “ (e.g. **xterm -e debug/myprogram**).

NOTE: At the time this Guide was written, OS/390 UNIX System Services did not provide an xterm program. Contact SlickEdit Inc. to find out if this is still the case.

Stop after dependency or build error

When this is checked and the current project depends on other projects, the **vsbuild** utility is used to build your projects and check for error codes. When the **vsbuild** program detects an error, it does not continue building other dependencies.

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

Verbose output

When this is checked, the **vsbuild** utility is used to build your projects. Detailed information about the commands that are executed will be output during the build.

Beep on completion

When this is checked, the **vsbuild** utility is used to build your projects and to sound a status beep upon completion. A single beep indicates a successful build. Two beeps indicate an error occurred.

Run in an X terminal

(UNIX only) When this is checked, the command is run in an X terminal. This is useful for running programs which are full screen console applications such as VI.

Pre- and Post-Build Commands

The “Build commands” tab allows you to run programs and/or execute Visual SlickEdit commands before or after a build. This information is per configuration. Each line can contain a program to execute or a Visual SlickEdit command. For example, the **set** command could be used to set environment variables (See “[Setting Environment](#)”).

Variables” on page 244 for more information). The contents of the “Build commands” tab is disabled for extension-based projects.

Open Command

The “Open Command” tab allows you to enter Visual SlickEdit commands that are executed when you activate a project. This information is stored per project; not per configuration. Each line contains a Visual SlickEdit command just as you would type it on the Visual SlickEdit command line.

You can set environment variables in the concurrent process buffer with the **set** command (see “**Setting Environment Variables**” on page 244 for more information). For example, the command **set xxx=yyy** sets the environment variable “xxx” to the value “yyy” (this automatically supports different UNIX shells). Use **concur_command** to send a command string to the concurrent process buffer (e.g. **concur_command export name=value**). This tab is disabled for extension-based projects.

Version Control Support

Visual SlickEdit supports the following version control systems:

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

What is SCC?

SCC is the **Source Code Control** interface specification, originally designed by Microsoft. This interface allows for direct communication between a version control system and another application (in this case, Visual SlickEdit). Here are several guidelines for using SCC:

- If you are using a system that supports SCC, you should use the SCC support, as it provides tighter integration.
- If your system does not have an SCC interface installed, you should contact the manufacturer to be sure that an SCC interface is not available.
- If your version control system has an SCC interface, but does not seem to behave properly, please contact SlickEdit Inc. Product Support.

SCC Version Control Systems (Windows only): Visual SlickEdit supports interfacing with version control systems that have support for an SCC interface. If you are using PVCS, you should install the SCC interface support because it does not install SCC support by default. SourceSafe automatically installs SCC support.

The following version control systems have SCC interfaces, but if they do not appear in the SCC Providers list (or the SCC Providers list does not appear), you might need to install the support separately:

- ComponentSoftware RCS
- ClearCase
- MKS Source Integrity
- StarTeam
- CCC/Harvest

Opening an SCC Version Control System Project

To open an SCC version control project:

1. Go to the Version Control Setup dialog ("Tools", "Version Control", "Setup.."). Be sure that "SCC providers" is checked. If only the "Command line systems" frame appears, no SCC systems are installed, or Visual SlickEdit cannot find installed SCC.
2. Select the desired system, and click "Initialize Provider".
3. Click "Open Project".
This will show an Open Project dialog that will vary from system to system.

Here are some pointers for the systems where the desired items are not obvious:

- Source Integrity: For "SI Project Filename", give the whole path to the "project.vpj" file. For "Sandbox Path", give the path where the files are located on the local machine.
- Perforce: For "Client Name", give the name of the Perforce depot (for example, "//depot"). For "Local Path", give the path where the files are located on the local machine.
- StarTeam: For "StarTeam Project Name", type the name of the StarTeam project. For "Local Path", give the path where the files are located on the local machine. You will then be prompted for more information by StarTeam.

Once you have opened an SCC version control project, it will be bound to the currently active Visual SlickEdit project. When you restart Visual SlickEdit, or switch to this project from another project, this version control project will automatically be activated.

Command Line Version Control Systems

If you are on a non-Windows platform, or are using a version control system that does not support SCC, the version control system must have a command line interface. To set up a command line version control system:

1. Go to the Version Control Setup dialog ("Tools", "Version Control", "Setup.."). Be sure that "Command line systems" is checked.
2. Select the desired system.
3. Click "Setup" and look at each command to see that it matches the options you want. (If you are not sure what any of the %<character> macros expand to, click on the arrow to the right of the text box to view a list.)
4. Be sure that the executable for each VC Command is in your PATH.

NOTE: Please contact SlickEdit Inc. with any additional features you would like in the version control support, and any other version control systems you would like supported. See "**Error! Reference source not found.**" at the end of the guide.

PVCS Command Line Interface Notes

If you are using PVCS, there is typically no need to switch version control projects since the source files are placed in the same directory as the archive files. In some PVCS configurations, you will want to set some environment variables when you switch Visual SlickEdit projects. To do this, use the "Project Properties..." menu item on the Project menu, select the "Open Command" tab, and enter one or more **set** statements to set the environment variables. You will need to close and reopen the project for the project macro to be executed. Access Version Control commands from the Version Control Menu ("Tools", "Version Control").

Version Control Operations

The **History**, **Difference**, **Lock**, and **Properties** commands operate on the current buffer. For SCC version control systems, Visual SlickEdit simply calls the SCC provider's function for these. For command line version control systems, Visual SlickEdit runs the specified command, captures the output, and displays it.

The **Check In**, **Get**, **Check Out**, **Unlock**, **Add**, and **Remove** commands show a dialog box and operate on all files selected. This dialog will allow you to easily choose from files currently open, the files in the current project, or files in the current workspace. Also, for SCC version control systems, you can select from files that are valid for that command. For example, when using the **Check In** command, you can click "Available" to view all files that can currently be checked in.

Use the **Manager** command to bring up the version control system's user interface. For many command line systems there will not be a program to call for this.

Menu items that appear grayed out for a command line version control system are blank. For SCC version control systems, "Lock" is always grayed because the SCC interface does not make allowances for a lock command. It is possible to receive an "Operation

not supported” message when running some commands if an SCC version control system does not implement an interface to that operation.

Analyzing Your Code

Analyzing Your Code Overview

This chapter describes features for analyzing your code, including Context Tagging™, DIFFzilla™ and 3 Way Merge Editing.

Context Tagging(TM)

Context Tagging Overview

The Context Tagging feature set performs expression type, scope, and inheritance analysis. Built on Visual SlickEdit's powerful tagging facility, Context Tagging features work with your source code, not just standard APIs. See "[Context Tagging\(TM\) Limitations](#)" in the online help for a list of limitations for various languages. Expect to see Context Tagging for more languages in the future.

In order for all Context Tagging features to work, some setup is required:

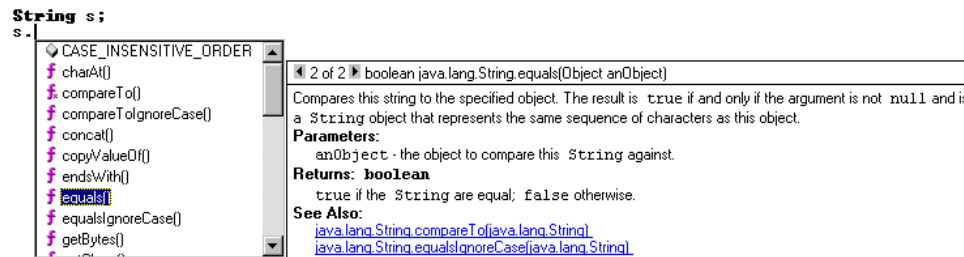
- For information on setting up Context Tagging for COBOL, see "[Setting up Context Tagging for COBOL](#)" on page 53.
- For information on setting up Context Tagging for OS/390 Assembler, see "[Setting up Context Tagging for OS/390 Assembler](#)" on page 55.
- For information on setting up Context Tagging for other languages, see "[Creating and Updating Tag Files](#)" on page 50.

What is the Context Tagging Feature Set?

Auto List Members

Visual SlickEdit automatically lists members when you type a member access operator ('.', '>', and '::' for C++, '.' for Java, "**IN**" and "**OF**" for COBOL). You can also access this feature on demand by pressing ALT+PERIOD. This can help you with identifiers when there is no member operator (list locals, global variables, current class members, etc.). For example, for C "**str**<ALT+PERIOD>" will help you find a string function. If you prefer to only list members on demand, turn Auto List Members off ("Tools", "Configuration", "General", uncheck "Auto list members").

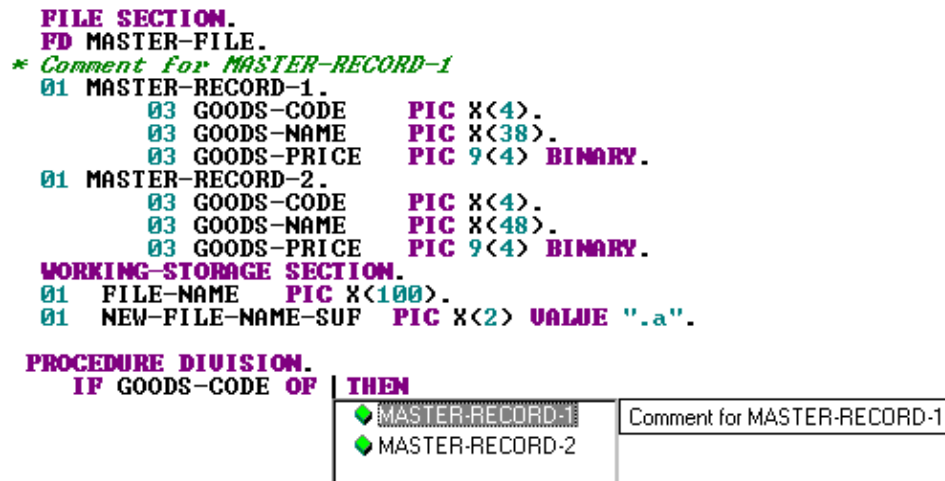
The example below shows what would appear after typing a period when entering Java source. Notice that the Javadoc comments are displayed in a mini-HTML browser. You must install Java API source if you want to view comments for the Java API runtime libraries. If you click on a URL, the default HTML browser is started. Clicking on other hypertext links will navigate 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 what would appear after typing a period when entering C++ source. The **stack** class is one of the C++ standard template library classes.



This example shows what would appear after typing "OF " when entering COBOL source.



Context Tag Completion

Press CTRL+SPACE to complete (type the rest of) the current symbol you are entering. If a unique match is not found, a list is displayed allowing you to select the exact match.

Auto Parameter Info

Visual SlickEdit automatically displays the prototype for a function when you type a function operator ('(' for most languages), and highlights the current argument within the displayed prototype. For C++, Parameter Info is also automatically displayed when you type a template argument operator ('<'). For COBOL, statement syntax help is displayed when the cursor is inside a statement for which help is provided. To access Auto Parameter Info or Auto Syntax Help on demand, press ALT+COMMA. If you prefer to only get Parameter Info or syntax help on demand, turn Auto Parameter Info and/or Auto Syntax Help off ("Tools", "Configuration", "General", uncheck "Auto Parameter Info" and/or "Auto syntax help"). For XML and HTML, it will show tag and attributes for that tag.

The example below illustrates pressing ALT+COMMA inside the argument list of the Java API String method **startsWith**. Notice that the Javadoc comments are displayed in a mini-HTML browser. You must install Java API source if you want to view comments for the Java API runtime libraries. If you click on a URL, the default HTML browser is started. Clicking on other hypertext links will navigate within the comment window. The **startsWith** method has two overloads which accept different arguments. Press CTRL+PGDN or CTRL+PGUP to select the next or previous occurrence.

```
String s;
if (<s.startsWith(<>> <
```

1 of 2 boolean java.lang.String.startsWith(String prefix)

Tests if this string starts with the specified prefix.

Parameters:
 prefix - the prefix.

Returns:
 true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise. Note also that true will be returned if the argument is an empty string or is equal to this String object as determined by the [equals\(Object\)](#) method.

Throws:
 java.lang.NullPointerException - if prefix is null.

Since:
 JDK1.0

The example below illustrates pressing ALT+COMMA inside the argument list of the WIN32 API function **CreateWindowEx**.

```
hwnd=CreateWindowEx(0,
    szAppName,
    "The Hello Program",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,CW_USEDEFAULT,
    CW_USEDEFAULT,CW_USEDEFAULT,
    NULL,
    NULL,
    WINUSERAPI HWND WINAPI CreateWindowExA(DWORD dwExStyle,LPCSTR lpClassName,
    LPCSTR lpWindowName,DWORD dwStyle,
    int X,int Y,int nWidth,int nHeight,HWND hwndParent,
    HMENU hMenu,HINSTANCE hInstance,
    LPVOID lpParam);
```

This example shows what would appear after typing "USING " when entering COBOL source.

CALL SUMCALC USING "19991231" OUTPUT-DAYS|

SUMCALC::SUMCALC USING INPUT-DATE, OUTPUT-DAYS

This example shows what would appear after typing ALT+COMMA followed by ALT+PERIOD when entering COBOL source.

input-output section.

file-control.

select input-file-1 assign to "test1.dat".

select output-file-2 assign to "test2.dat".

procedure division.

open output output-file-1 sharing no other.

OPEN { (INPUT or OUTPUT or I-O or EXTEND)
[SHARING WITH {ALL OTHER or NO OTHER or READ ONLY}]
[TIMEOUT {AFTER arithmetic-expression-1 SECONDS
or RETRY {arithmetic-expression-2 TIMES or NO LIMIT}}]
{file-name-1 [{REVERSED or WITH NO REWIND}] }... }....

 input-file-1

 output-file-2

Returns **assign to "test2.dat"**

Auto List Parameters

When you type a function operator ('(' for most languages), a list of compatible variables and expressions for the current argument is displayed. Auto List Parameters can also be used instead of Auto List Members, in assignment statements ("x=<Alt+Comma>") and when listing members of a class or struct. For performance reasons, not all possible variables and expressions are listed. Press ALT+PERIOD if the symbol you want is not listed. To access Auto List Parameters on demand, press ALT+COMMA. If you prefer to only list parameters on demand, turn off Auto List Params ("Tools", "Configuration", "General", uncheck "Auto List Parameters").

The following is a C++ example that shows Auto List Parameters after a function operator is typed:

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

    CalculateArea(|
> int CalculateArea(RECT *pRect)
```

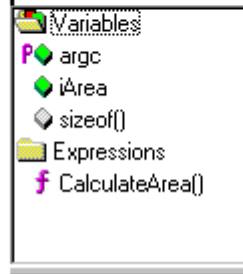


Notice that “argc”, “argv”, and “iArea” are not listed because their types do not match.

The example below illustrates pressing ALT+COMMA after an assignment operator:

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

    iArea=|
>
```



Notice that “Rect,” “pRect,” and “argv” are not listed because their types do not match.

Context Tag Navigation

Source code context is automatically analyzed so you can quickly navigate to the definition of an identifier, including local and global variables, class members, and functions. Press CTRL+PERIOD to go to the definition of a tag under the cursor. To "pop" back to where you were, press CTRL+COMMA (or use the "Search", "Pop Bookmark..." menu item).

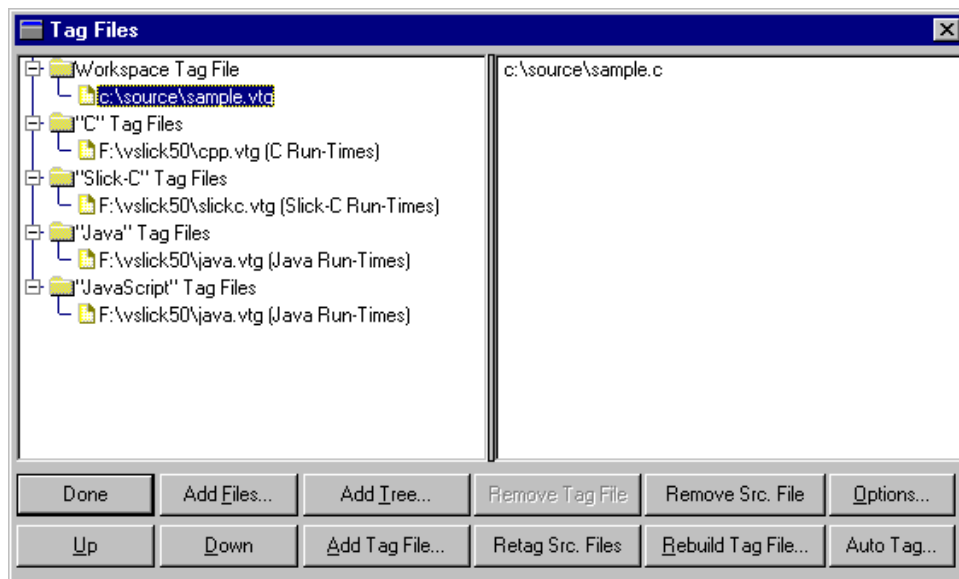
Context Tag Preview

The "Symbols" tab of the Output toolbar (docked on the bottom by default) displays the symbols matching the word under the cursor and utilizes context information. If only one tag is found, the source file containing the tag is displayed. Otherwise, multiple symbol matches are listed. When a source file is displayed in the "Symbols" tab, you can double-click on it to edit the file. When multiple matches are listed, double-click on the line to go to that symbol. Alternatively, you can enter a symbol to view in the "Symbol:" combo box of the "Symbols" tab. Right click in the editor control of the "Symbols" tab to show an option menu of tag types to show/hide.

Creating and Updating Tag Files

Many of Visual SlickEdit's source code analysis features, such as Context Tagging™ (see "Context Tagging(TM)" on page 45), class/tag browsing (See "Class Browser" on page 60), and dynamic tagging, require a tag file. Visual SlickEdit automatically creates and maintains a tag file for files in your workspace. However, you will want to create extension-specific tag files for compiler includes or other libraries of code that you have. Once a tag file is created, it is updated in the background when you make edits. If you modify some source files using another tool, you might want to rebuild the tag file.

Use the **Tag Files dialog box** ("Search", "Tag Files...") to create or update a tag file. A picture of the Tag Files dialog box is displayed below:



Tag file names have the extension ".vtg". By separating tag files for different languages into extension-specific tag file sections, the **Context Tagging(TM)** features can correctly look up symbol information for the current file you are editing. However, the **Class Browser** and the Go To Definition dialog box are not extension-specific and access all tag files. Refer to "Tagging Options" in the online help to change settings for background tagging, tag on save, incremental references, etc.

Tagging Run-Time Libraries

SlickEdit recommends that you create extension-specific tag files for include files of compiler packages and/or utility libraries. This will allow tagging features to work for all symbols, and not just those symbols in your project. Context Tagging™ (see "**Context Tagging(TM)**" on page 45), which features Auto List Members and Auto Parameter Info, needs all symbol information in order to work properly.

Visual SlickEdit automatically builds a tag file for the run-time libraries of Perl, JavaScript, Tornado, InstallShield, PV-WAVE, TCL, and Slick-C, so it should be unnecessary to build tag files for the run-times of these languages. If you have already built a tag file for your run-times during installation, you can skip this section. If you are using TCL, Perl, or Python and the compiler cannot be found in your PATH (or registry for Windows), you need to build tag files for these run-time libraries.

Windows:

If you are using Visual C++, C++ Builder, any Java package, or COBOL, access the Automatic Tagging dialog box ("Search", "Tag Files...", "Auto Tag...") to build a tag file for these run-time libraries. This is the same dialog used by the Visual SlickEdit install program.

UNIX:

If you are using C, C++ or Java, access the Automatic Tagging dialog box ("Search", "Tag Files...", "Auto Tag...") to build a tag file for these run-time libraries. This is the same dialog used by the Visual SlickEdit installation program.

If the Automatic Tagging dialog box does not help you, perform the following steps to set up an extension-specific tag file for your run-times which can be accessed by all projects which have files of the same language:

Building an Extension Specific Tag File

1. Display the Tag Files dialog box ("Search", "Tag Files...").
2. Click "Add Tag Files...".
3. When the Add Tag File dialog box appears, click "Extension Specific Tag File" and select the language from the combo box. For example, select "C" to add an extension specific tag file for C and C++.
4. Click OK. When the Add Tags Database dialog appears, set the directory to the root directory of the include files or source tree. For Visual C++, this might be "c:\Program Files\DevStudio\VC\" or "c:\msdev\". UNIX include source is almost always under **"/usr/include"**.
5. Press ENTER to select **"tags.vtg"** (UNIX: **utags.vtg**) as the output file.

3. If the Add Tree dialog box is not already displayed., click "Add Tree...". Enter one or more wildcard file specifications in the Files text box (i.e. *.c *.h *.cpp *.hpp).
4. Click OK to recursively add these files to the selected tag file.

Setting up Context Tagging for COBOL

Windows:

COBOL support is licensed separately; therefore, if you did not purchase support for COBOL, this feature will be disabled.

All of the Context Tagging features for COBOL, except Auto Parameter Info, are provided by scanning your COBOL source file and the copy books that are included. This information is used by Auto List Members, Context Tag Completion, Context Tag Navigation, Context Tag Preview, and the "Procs" tab of the Project Toolbar. Auto Parameter Info for COBOL commands and intrinsic functions are provided by the COBOL built-ins file created during product installation. In order to provide Parameter Info for subroutines, you must build a tag file that will hold linkage information from the subroutine's point of view.

OS/390:

If you are working with data sets, the term path or directory has the same meaning as a partitioned data set.

To set up all of the Context Tagging features for COBOL, except for Auto Parameter Info, you need to specify a search path for your COBOL copy books. There are three ways to configure the search path for copy books:

1. Set a VSLICKCOBOLPATH environment variable.
(This is useful if you want to specify the same copy path or a default copy path for all your COBOL programs.)
 - Separate multiple directories with a semicolon (UNIX: colon). You can use the "-set" option (see [Invocation Options](#) on page 23) to configure this environment variable.
 - In addition, **OS/390 users** can add a line to the *TSOID.VSLICK.MISC(ENV)* data set to configure this environment variable:

VSLICKCOBOLPATH=//TSOID.PDS1://TSOID.PDS2

2. Set the extension-specific include directories.
(This is useful if you want to specify the same copy path or a default copy path for all your COBOL programs.)
 - Make sure no project is open (invoke the "Project", "Close" menu item if necessary).

- Display the project properties dialog (“Project”, “Project Properties...”).
 - Select the “Directories” tab.
 - Set the “Include directories” text box. Separate multiple directories with a semicolon (UNIX: colon). You must have a COBOL program open or the Project pull-down menu does not allow access to Project Properties. When a COBOL program is open and no project open, your settings are stored in the extension-specific project **.cob**.
3. Set the project include directories.
(This allows you to open a specific Visual SlickEdit project to get a specific set of copy paths.)
- Create or open an existing project.
 - Display the project properties dialog (“Project”, “Project Properties...”).
 - Select the “Directories” tab.
 - Set the “Include directories” text box.

Use the (“Project, “New...””) to create a new project. Separate multiple directories with a semicolon (UNIX: colon).

If you want Parameter Info for subroutines, you must build a tag file. See “**Creating and Updating Tag Files**” on page 50 for information on creating a tag file. When a tag file is created for the COBOL subroutines available to a COBOL program, the subroutine names are listed when the CALL statement is typed. As a subroutine is highlighted, any comments that appear at the top of the subroutine program (above the IDENTIFICATION DIVISION) will be displayed. After the USING option of the CALL statement is entered, the list of variable names in the subroutine’s PROCEDURE DIVISION USING statement is displayed.

For non-COBOL subroutines, you should create a dummy built-ins file to hold the parameters in COBOL format so a tag file can be built. Use the COBOL alias **docsubrtn** to generate the lines needed for each subroutine. Gather the following information for each non-COBOL subroutine:

1. Brief comment on what the subroutine does
2. The name of the copybook that serves as the common area between the calling program and the subroutine (if any)
3. The standard COBOL field names passed to the subroutine from the caller’s point of view (if any). You will be prompted for 4 fields. If you have more passed fields see step 5 below.
4. If there are no standard field names that should be passed, consider using a name that describes the attributes of the fields such as PIC-XXX-FIELD or PIC-S999V99-COMP-3.

5. If there are more than 4 passed fields for the subroutine, add additional lines with the field names using the existing lines as examples.

Create an HFS file called “noncobol subroutines.cob”. This file can reside in your home directory or a directory that is available to all team members. The contents of the file should look like the following lines:

```
* SUBRTN1 comments
* Interface copybook = copybookname
PROGRAM-ID. SUBRTN1.
PROCEDURE DIVISION USING PIC-X8-FIELD .

* SUBRTN2 comment
* Interface copybook = copybookname
PROGRAM-ID. SUBRTN2.
PROCEDURE DIVISION USING PIC-99-FIELD .

* SUBRTN3 comment.
* Interface copybook = copybookname
PROGRAM-ID. SUBRTN3.
PROCEDURE DIVISION USING PIC-X-FIELD
                        PIC-XX-FIELD
                        PIC-XXX-FIELD
                        PIC-XXXX-FIELD
                        PIC-XXXXX-FIELD .
```

Create a tag file containing COBOL subroutine modules and the file “noncobol subroutines.cob”. This will result in all subroutines, COBOL and non-COBOL, being available when you enter the CALL command. The comments above the PROGRAM-ID line will appear when the program name is highlighted in the pull-down list.

If you use field names that are descriptive of the field type that should be passed to the subroutine, then you can compare the field type required to the field type of the available fields in the calling program when you get the Auto List Members (automatically or via ALT+PERIOD).

Windows:

If you are using a Windows COBOL compiler, use the Automatic Tagging dialog box ("Search", "Tag Files...", "Auto Tag...") to build a tag file for your compiler run-time libraries. This is the same dialog used by the Visual SlickEdit install program.

Setting up Context Tagging for OS/390 Assembler

Windows:

OS/390 Assembler support is licensed separately; therefore, if you did not purchase support for OS/390 Assembler, this feature will be disabled.

All of the Context Tagging for OS/390 Assembler is provided by scanning your Assembler source file and included macros. This information is used by Auto List Members, Auto Parameter Info, Context Tag Completion, Context Tag Navigation, Context Tag Preview, and the “Procs” tab of the Project Toolbar. Auto Parameter Info displays help for macro arguments.

OS/390:

If you are working with data sets, the term path or directory has the same meaning as a partitioned data set.

To set up all of the Context Tagging features for OS/390 Assembler, you need to specify a search path for your macros and copy files. There are three ways to configure the search path:

1. Set a VSLICKASM390PATH environment variable.
(This is useful if you want to specify the same search path or a default search path for all your Assembler source files.)
 - Separate multiple directories with a semicolon (UNIX: colon). You can use the “-set” option (see [Invocation Options](#) on page 23) to configure this environment variable.
 - In addition, OS/390 users can add a line to the *TSOID.VSLICK.MISC(ENV)* data set as follows to configure this environment variable:

```
VSLICKASM390PATH=//TSOID.PDS1://TSOID.PDS2
```

2. Set the extension-specific include directories.
(This is useful if you want to specify the same search path or a default search path for all your Assembler source files.)
 - Make sure no project is open (invoke the “Project”, “Close” menu item if necessary).
 - Display the project properties dialog (“Project”, “Project Properties...”).
 - Select the “Directories” tab.
 - Set the “Include directories” text box.

Separate multiple directories with a semicolon (UNIX: colon). You must have an Assembler source file open or the Project pull-down menu does not allow access to Project Properties. When an Assembler source file is open and no project is open, your settings are stored in the extension specific project **.asm390**.

3. Set the project include directories.
(This allows you to open a specific Visual SlickEdit project to get a specific set of search paths.)
 - Create or open an existing project.
 - Display the project properties dialog (“Project”, “Project Properties...”).
 - Select the “Directories” tab.
 - Set the “Include directories” text box.

Use the (“Project, “New...””) to create a new project. Separate multiple directories with a semicolon (UNIX: colon).

Dynamic Tagging

Most dynamic tagging features require tag files. See "[Creating and Updating Tag Files](#)" on page 50.

Visual SlickEdit's tagging is dynamic because it tags your source files in the background – even for those you are currently editing. Visual SlickEdit tracks the current function, tags in the current file, and symbols for matching the word at the cursor.

Tags for the Current File

The “Procs” tab on the Project toolbar (docked on the left by default) displays the symbols for the current source file. In addition, it indicates the current function. This feature does NOT require a tag file.

By default, symbols in the current file are sorted by name. If you want the items sorted by line number, right click in the Procs window and select the "Sort by Line Number" menu item.

The “Procs” tab is also useful for switching buffers. All the buffers you are editing are listed, so you can double-click on the buffer to switch to it. Double-click on a function name to move the cursor to the beginning of that function in the edit window.

Symbols Tab

The “Symbols” tab of the Output toolbar (docked on the bottom by default) displays the tags for the word under the cursor. If only one tag is found, the source file containing the tag is displayed. Otherwise, the tag symbols are listed.

- When a source file is displayed in the “Symbols” tab, double-click on it to edit the file.
- When multiple tag symbols are listed, double-click on the tag symbol to go to that tag. Alternatively, you can enter a symbol to view in the “Symbol:” combo box of the “Symbols” tab.
- Right click inside the “Symbols” tab to show an option menu of tag types to show/hide.

The following options apply to the “Symbol” tab when the focus is in the current buffer and Visual SlickEdit is attempting to display the tag currently under the cursor. Only extension-specific tag files are searched. If the current editing mode does not support tagging, then all tag files are searched.

- **Case-Sensitive** – If this is turned on, Visual SlickEdit will attempt to find the symbol using a case-sensitive match first. If no matches are found, it will automatically revert to a case-insensitive search.
- **Tag Types** – Allows you to filter out certain types of tags that you do not want to be displayed in the preview window. The primary purpose of this function is to help resolve tags with the same name but different purposes, such as the definition and declaration of a function in C, or the name of a class and a class constructor in Java. If a search using the filters results in no matches, a second search is done, ignoring the tag type filters.

When the focus is in the “Symbol:” text box on the “Output Symbol” tab, you can perform the following search operations to find tags in the current context.

- **Incremental search** – matches tags as you type. First, it searches for an exact match to the tag, without regard for the current context. If no exact match is located, it searches for a prefix match.
- **Prefix matching** – can be forced by typing "*" at the end of the symbol prefix you wish to match. For example, type "XmSet*" to list all symbols starting with the string "XmSet", including XmSet itself (if such a symbol is defined).
- **Class members** – can be searched by first typing the qualified class name followed by :: and a symbol prefix. For example, to see the members of the ANSI C time structure (struct tm), you would type "tm::tm_". If no matches are found for the tag in the given class, the class name is ignored.

When the focus is not in the current buffer, the “Symbol” tab may also perform a function if it is visible. In most of these cases, it is functioning as a “Preview” window, allowing you to look briefly at a tag before actually jumping to it or using it in code.

Focus	Display
Procs tab	If the focus is in the “Procs” tab (on the Project toolbar), the currently selected tag will be shown. The tag type filters are disregarded.
Class browser tab	If the focus is in the “Class browser” tab (on the Project toolbar), the currently selected tag will be shown. The tag type filters are disregarded. In the case of functions and function prototypes, if the currently selected item is a function, the function will be shown in the preview window; if it is a prototype, the prototype will be shown. Thus, what is shown in the preview window depends on what is shown in the class browser,

which is typically whatever appears in the class definition.

Inheritance tree dialog, Class Parents

If the focus is in the tree control on the left-hand side of the Inheritance tree dialog (the class list), the currently selected class will be shown, if it can be resolved.

Inheritance tree dialog, Members

If the focus is in the tree control on the right-hand side of the Inheritance tree dialog (the member list), the currently selected item will be shown, which is the item as it appears within the class definition.

Symbol Uses/Calling Tree

If the focus is in the Calling Tree dialog, the currently selected item will be shown in the symbol window, just as it is for the class browser.

List members dialog visible

When the list members dialog is visible, either a result of pressing ALT+PERIOD or Auto List Members, the currently selected item is previewed in the symbol window.

Jumping to a Tag

You do not need to use the “Symbols” tab to search for a tag. If there is a tag symbol at the cursor (for example, a function name), place your cursor on the tag name and invoke the **push_tag** (or **f**) command (CTRL+PERIOD or "Search", "Go To Definition...").

You can also use the Go To Definition dialog box to go to a specific tag. This dialog box will list all tags which match the prefix you have typed so far. Deselect “Prefix Match” to match tags using a regular expression or a substring search. If you prefer to use the command line, use the **f** command and use the completion keys (SPACEBAR and '?') to assist you when entering a tag name. For example, if you are tagging the C run-time library, type "**f str?**" on the Visual SlickEdit command line for a list of tag names starting with "str" (such as strcpy, strcmp, etc.).

The **push_tag** command sets a bookmark in the file you are currently editing so that you can easily return to your edit position. To go back to the previous edit position, press CTRL+COMMA.

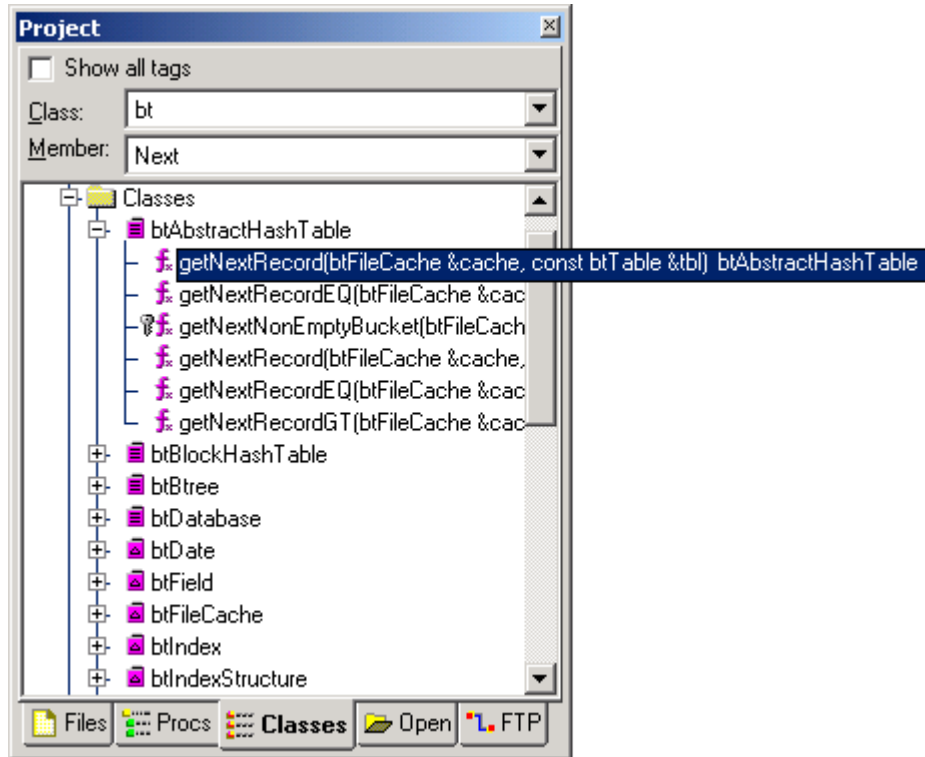
Use the **find_proc** (or **fp**) command ("Macro", "Find Slick-C Proc...") to navigate to a Slick-C tag symbol. If you are editing a Visual SlickEdit macro, you can invoke the **push_tag** command (CTRL+PERIOD) to find the symbol at the cursor. The **push_tag**

command actually calls the **find_proc** command with the symbol name at the cursor to perform the task.

Class Browser

This feature requires tag files. See [Creating and Updating Tag Files](#) on page 50.

The class browser is located on the “Classes” tab of the Project toolbar as shown below. (Project toolbar, select “Classes” tab)



The Project toolbar is docked on the left side of the editor by default. If the Project toolbar is not open, use the Toolbar Customization dialog box ("View", "Toolbars...") to make it visible. If you move the mouse over the bitmap displayed to the left of the tree item (such as the "+" bitmap), a tool tip is displayed indicating what the bitmap means.

Searching for Members or Classes

The "Class Filter" and "Member Filter" combo boxes allow you to specify a regular expression string which can limit the display to only classes that match the class filter and members which match the member filter. Separate multiple regular expression search strings with a space. For example:

- The member filter "**Begin End**" will find members that contain Begin or End anywhere in the name.
- The member filter "**^Begin**" will find members that start with Begin.

In addition, you can use the Find Tag dialog box (right click in the class browser window and select "Find Tag...") to help you search for a specific symbol in the Class Browser.

The Class Browser contains options for turning off display of the class and member filter combo boxes to save space (right click in the class browser window, "Filters", "Class Filter" or "Member Filter"). No class or member filters are applied if the combo boxes are not visible.

Viewing Accessible Members

By default, the class browser does not show you inherited members that this class can access. To view inherited members in addition to other members, set the "Inherited" option on the Class Browser Filter Options dialog box (right click in the class browser window and select "Filters", "Filtering Options...") to "Don't Care" (gray).

You can temporarily disable filtering options and class and member filters by turning on the "Show all tags" check box.

Other Filtering Options

The Class Browser Filter Options dialog box has options for filtering of members based on member attributes (right click in the class browser window, "Filters", "Filtering Options..."). Bring up the Class Browser Filter Options dialog box and press F1 for detailed information.

Viewing Symbol Properties

To see various attributes, return type, and arguments for a particular symbol, right click on the symbol in the class browser and select the "Properties..." menu item.

Viewing Base Classes

To see what classes are inherited by a particular class, right click on the class and select the "Base Classes..." menu item.

Viewing Derived Classes

To see what classes are derived from a particular class, right click on the class and select the "Derived Classes..." menu item.

Viewing Symbol References

While most of the tag navigation features deal with a symbol definition, you can use the Symbol References feature to see all places where a particular symbol was referenced. Visual SlickEdit currently supports Symbol References for C++, Java, COBOL, Slick-C, InstallScript, and Delphi. You can either view symbol references with the class browser or for the symbol at the cursor.

To view symbol references in the class browser:

1. Create a project (See "[Creating Projects](#)" on page 32) or open an existing project.
2. Click on the "Classes" tab of the Project toolbar.
3. Right click on the function or method for which you want to see references, and select "References" from the class browser context menu.

To view symbol references for the symbol at the cursor:

1. Create a project (See "[Creating Projects](#)" on page 32) or open an existing project.
2. Open a source file and press CTRL+SLASH on the symbol at the cursor (or right click on the symbol, and choose "Go To Reference").

NOTE: If you upgraded from a previous version of Visual SlickEdit, Ctrl+Slash is likely bound to another command. Make sure CTRL+SLASH is bound to the **push_ref** command ("Tools", "Configuration", "Key Bindings...").

3. Press CTRL+G or CTRL+SHIFT+G to go to the next or previous occurrence respectively. Alternatively, you can select occurrences from the references list box.

Typically, you only want to view references that occur in your project files, and not run-time libraries, which can be very large. For this reason, Visual SlickEdit does not automatically generate references 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 Tag Files dialog box ("Search", "Tag Files..."), choose the tag file, right click to display the context menu, and select "Generate References".

Viewing Symbol Uses

The Symbol Uses feature allows you to see which symbols (variables, functions, methods, classes, etc.) are used by a specific function or method. Visual SlickEdit currently supports Symbol Uses for C++, Java, COBOL, Slick-C, InstallScript, and Delphi.

Perform the following steps to view the symbols that a particular function or method uses:

1. Create a project (See "[Creating Projects](#)" on page 32) or open an existing project.
2. Click on the "Classes" tab of the Project toolbar.
3. Right click on the function or method for which you want to see uses, and select "Calls or Uses" from the class browser context menu.

Differencing Files with DIFFzilla(TM)

Visual SlickEdit's file differencing feature, "DIFFzilla™", helps you merge changes from one version of a file to another, or helps you determine what changes have been made to a file. Using the DIFFzilla™ dialog box, you can difference two files or source trees.

DIFFzilla™ provides the following file differencing features:

- View differences between two files
- View differences between source trees
- View differences between all symbols of two files
- Intra-line differences are color-coded as you type with Dynamic Difference Editing™
- Generate file lists
- Automatic directory mapping
- Save/restore multi-file results
- Dialog box history (wildcards, paths, file specifications)

Diffing Two Files

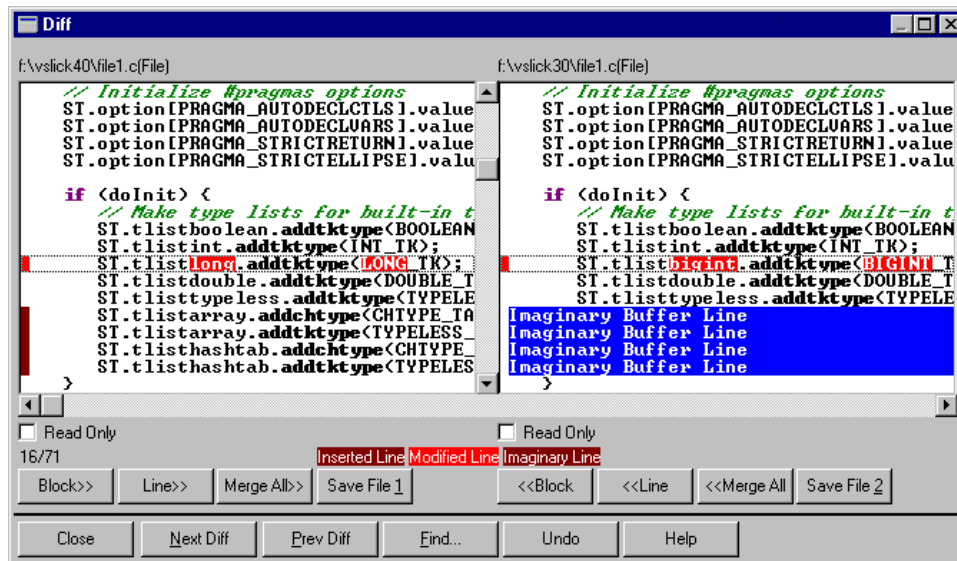
When you diff two files, you can view the differences side-by-side or interleaved. In either case, the output is color coded to make it easier for you to see the changes. The side-by-side output is much more powerful because you can transfer changes from one file to the other just by clicking on a button. You can even edit the files without exiting from the DIFFzilla dialog box.

Undo, copy/paste, syntax expansion/indenting, SmartPaste™, Auto List Members, Auto Parameter Info and many of your emulation key mappings work when editing in the DIFFzilla dialog box. Unlike other less powerful differencing tools, DIFFzilla has Dynamic Difference Editing™. When you type or make any edit, lines are rediffed so you see the new intra-line differences.

To diff two source files:

1. Invoke the DIFFzilla™ dialog box (Ctrl+= or "Tools", "File Difference...").
2. Enter the first file in the "Path 1" text box and 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".
3. If you want interleaved output instead of side-by-side difference editing, click on the "Instead of an interactive dialog, output one buffer with the differences" check box on the "Options" tab

Below is a picture of the side-by-side difference editor:

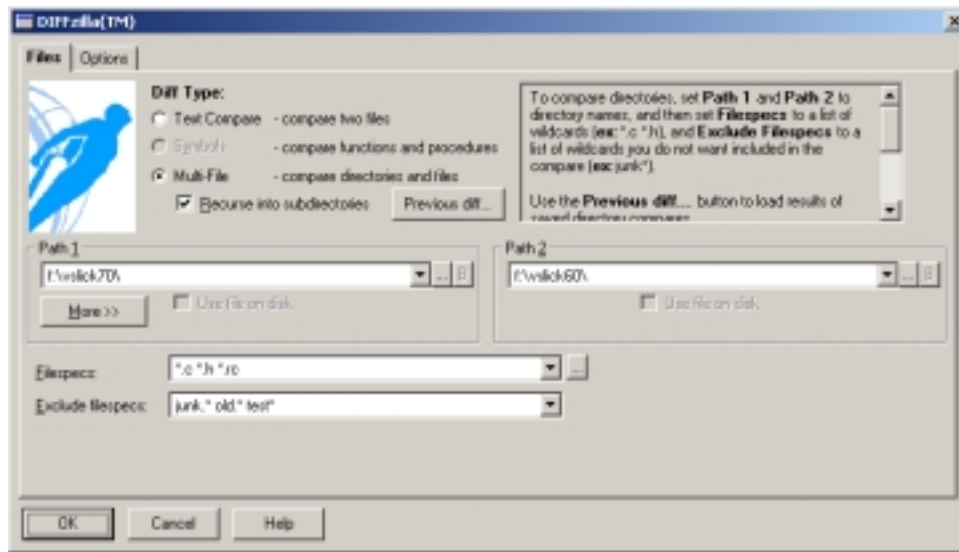


Diffing Source Trees

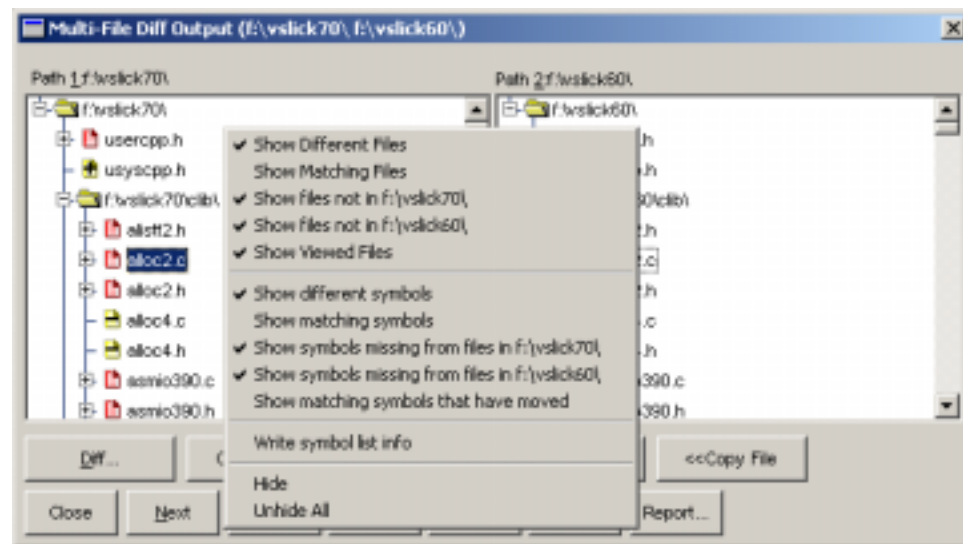
To diff two source trees:

1. Invoke the DIFFzilla™ dialog box (Ctrl+= or "Tools", "File Difference...").
2. Enter the two directories in the "Path 1" and "Path 2" text boxes.
3. Fill in the "Filespecs" text box with the files you want processed.
4. Turn on the "Recursive" check box to compare subdirectories.

The following dialog box shows differencing two source trees. All .c, .h, and .rc files are processed excluding files which match the "Exclude filespecs".



The results from differencing with these parameters are shown below:



If a file exists in one tree but not the other, a "+" bitmap is displayed in the one tree and a "-" in the other. You can customize which files you want to view with the context menu. To bring up the context menu, right click in the left or right tree. If you move the mouse over the bitmap displayed to the left of the tree item (such as the "+" bitmap), a tool tip is displayed indicating what the bitmap means.

The buttons perform the following functions when displayed:

Copy File

Copies the selected files from left to right or right to left. Press CTRL+Click to select multiple files. This button is shown when the selected files differ or when the selected file only exists in the current source tree ("+" bitmap shown). The copy button is also displayed when the selected item is a directory which only exists in the current source tree. In this case, when you click Copy, you are prompted as to whether you want to copy the directory source tree recursively.

Diff...

Shows current files in the difference editor when the selected files differ.

View...

Shows current files in the difference editor when the selected files match (not shown in above picture).

Del File

Deletes the selected files on the other side. Press CTRL+Click to select multiple files. This button is shown when the selected file only exists in the other source tree ("- " bitmap shown. Not shown in above picture).

Next

Jumps to next mismatch in both source trees.

Prev

Jumps to previous mismatch in both source trees.

Save...

Lets you save a diff state file (.dif) which you can load later with the "Previous diff..." button on the **DIFFzilla(TM) dialog box**. This is especially useful when you have not completed merging files and you want to continue at a later time. In addition, you can generate a file list. See "**Generating File Lists**" below.

Refresh

Rediffs modified files or all files.

Options...

Displays **DIFFzilla(TM) dialog box Options Tab**. Options include ignoring spaces, skipping leading comments, and expanding tabs.

Report...

Displays a report of the operations you performed in this dialog including file copies, file deletes, and diffs where changes were saved. In addition, you may optionally save the report.

Diffing Symbols or Parts of Files

DIFFzilla allows you to diff a selected range of lines from two files or the same file. This is very useful for differencing a piece of code which has been moved into a different part of a different file. The "Instead of an interactive dialog, output one buffer with the differences labeled" option is not supported when diffing a selected range of lines.

To diff a selected range of lines from two source files:

1. Invoke the DIFFzilla™ dialog box (Ctrl+= or "Tools", "File Difference...").
2. Enter the first file in the "Path 1" text box. Click the "More>>" button and fill in the start and end line numbers next to "Line Range" label.
3. Enter the second file in the "Path 2" text box and fill in the start and end line numbers next to "Line Range" label. When differencing a symbol definition, click "Symbols..." to help you enter the line number range. If the file names only differ by path, you only need to specify the path for "Path 2".

Diffing All Symbols of Two Files

DIFFzilla allows you to diff all the symbols of two different files. This feature is most useful for diffing files with symbols that have been moved around. It is also helpful for easily diffing specific symbols instead of the entire file.

To diff all the symbols of two source files:

1. Invoke the DIFFzilla™ dialog box (Ctrl+= or "Tools", "File Difference...").
2. Enter the first file in the "Path 1" text box.
3. Enter the second file in the "Path 2" text box. If the file names only differ by path, you only need to specify the path for "Path 2".
4. Select the "Symbols" radio button under "Diff type" and click OK. You do not need to turn on the "Diff all symbols" check box when performing a multi-file diff because mismatching files will have a '+' in front of them so you can diff all symbols.

CAUTION: Be very careful when diffing all symbols. Some symbol blocks are not yet picked correctly. For example, for C++/Java/C# variable initializations are not yet handled correctly. For example:

```
struct MYSTRUCT {
    int x;
    int y;
};

MYSTRUCT VariableDefinition={ // Symbol definition stops here
    0,1
}; // really should end here
```

Generating File Lists

One of the features of the "Save..." button in the Multi-File Diff Output dialog is to write a list of files which optionally include different files, matching files, and files which do not exist in the other tree. For example, suppose you wanted to e-mail updated source code and/or binaries to someone, but you did not want to send all of it because it might be 10 MB of compressed data. If you knew what files were changed you could just compress and send those files. This is a job for DIFFzilla™:

1. Use the DIFFzilla™ dialog box to difference the new source tree with the original source tree.
2. Click "Save..."

3. When the Multi-File Diff Output dialog box appears, turn on "Save Path 1 Filelist", "Include different files", and "Include files not in Path2". All other check boxes should be off.
4. Click OK and select an output file for your list. Do not enter an extension on your output file. ".lst" will be appended to your output file name.
5. Now you can use a program like PKZIP25 to zip up your files (e.g. **pkzip25 -path -add file.zip @file.lst**).

Automatic Directory Mapping

By default, the DIFFzilla™ dialog box automatically updates the "Path 2" text box with a directory, based on previous responses to the dialog. For example, if you previously entered "**f:\vslick40\bitmaps**" into the "Path 1" text box and "**\\server\dan\vslick40\bitmaps**" into the "Path 2" text box, then "**f:\vslick40**" is mapped to "**\\server\dan\vslick40**". The next time you enter "**f:\vslick40\macros**" into the "Path 1" text box, "**\\server\dan\vslick40\macros**" is automatically entered into the "Path 2" text box.

To turn this option off, click "Options..." on the DIFFzilla™ dialog box and uncheck "Automatic directory mapping".

3 Way Merge Editing

3 Way Merge is 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. If there are any conflicts (because both people edited the same area of code), a dialog box is displayed which lets you select the change you want in the output file. The output can be viewed side-by-side or interleaved. The side-by-side output is more powerful because you can pick the change you want by clicking on a button.

To start a 3 Way Merge, use the ("Tools", "File Merge...") menu item.

Working With Your Code

Working With Your Code Overview

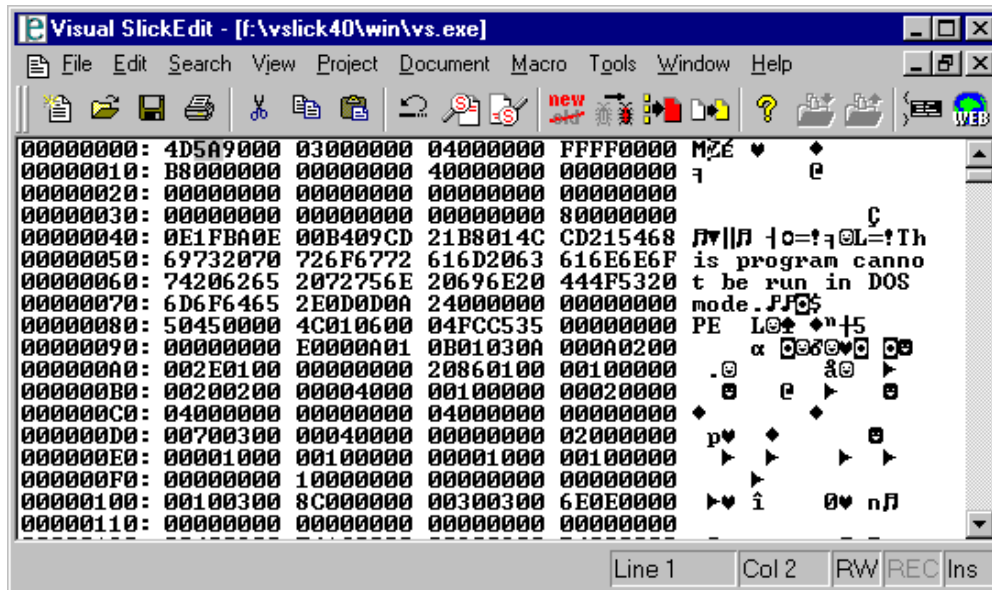
Visual SlickEdit offers many features to help you efficiently work with your code, including several styles of editing, the ability to record and use macros, powerful search and replace capabilities, various sorting and **spell checking** features.

Viewing and Editing Options

With Visual SlickEdit, there are several different options for viewing and editing your files.

Hexadecimal View and Edit Mode

To view a binary or text file in a hex/text view mode, use the "Hex" toggle command on the View menu. If you close a file while viewing it in hex mode, the next time you edit the file, it will be displayed in hex/text view mode. When the cursor is in the hex data, you can overwrite or insert hex nibbles (characters 0..F). When the cursor is in the text data, you can overwrite or insert text characters as you would when you are editing a text file.



All of Visual SlickEdit's search and replace commands work while hex editing. Only character selections are displayed when in hex editing mode.

Searching for Binary Characters

The easiest way to search for a sequence of binary characters is to use regular expressions because they allow you to specify hex or decimal characters.

UNIX or Brief Regular Expression Examples

```
\x0d\x0a\x01\x02
\d13\d10\d1\d2
```

SlickEdit Regular Expression Examples

```
\x0d\x0a\x01\x02
\13\10\1\2
```

Hex Mode Key Bindings

Toggle cursor between hex data on left and text data on right.	TAB, SHIFT+TAB
Move cursor to beginning of hex/text line.	HOME
Move cursor to last character of hex/text line.	END
Delete a byte to the left of the cursor and move the cursor to the left.	BACKSPACE
Delete the byte under the cursor.	DELETE

The keys above override key bindings for your emulation. Most of your other emulation keys will perform the same operation. However, keys that are bound to the commands **top_of_buffer**, **bottom_of_buffer**, **page_up**, **page_down**, **begin_line**, **end_line**, **begin_line_text_toggle**, **cursor_left**, or **cursor_right** will perform hex/text cursor motion.

Command Line and Text Box Editing

For online help on command line and text box editing keys, press ALT+H, 'K', and select "Command Line and Text Box Editing".

Quick Summary

The command line is a text box control just like the text boxes that appear in various dialog boxes. The following is a summary of the text box editing keys:

Insert mode toggle.	INS
Expand partially typed parameter or insert a space.	SPACEBAR
List matches to partially typed parameter.	? (Type " Ctrl+Q ? " to insert a question mark.)
Expand alias. See " Alias Facility " on page 154.	CTRL+SHIFT+O
Cut to end of line.	CTRL+E
Cut line.	CTRL+BACKSPACE
Copy word to clipboard.	CTRL+K
Cut word.	CTRL+SHIFT+K
Lowercase word.	CTRL+SHIFT+L
Previous word.	CTRL+LEFT
Next word.	CTRL+RIGHT

Paste.	CTRL+V
Cut.	CTRL+X
Copy.	CTRL+C
Append cut.	CTRL+SHIFT+X
Append to clipboard.	CTRL+SHIFT+C
List clipboards.	CTRL+SHIFT+V
Select text between cursor and beginning of line.	SHIFT+HOME
Select text between cursor and end of line.	SHIFT+END
Extend selection to mouse position.	SHIFT+Click
Delete previous character or selection.	BACKSPACE
Delete character under cursor or selection.	DEL
Move cursor left.	LEFT
Move cursor right.	RIGHT
Move cursor to end of line.	END
Move cursor to beginning of line.	HOME
Select word.	Double-click
Select line.	Triple-click

The Visual SlickEdit command line also supports retrieval and allows you to run external programs or DOS internal commands.

Completion

Completion is a mechanism which helps you type a command or an argument to a command. When you press SPACEBAR in one of these areas, Visual SlickEdit attempts to insert as much of the argument as possible, thereby “completing” the argument for you. Visual SlickEdit's command line and dialog input fields support completion.

To try this feature follow these steps:

1. Press ALT+M and 'O' to display the Open Form dialog box.

2. Type `_open` and press SPACEBAR. You will see the word "`_open`" expanded to "`_open_form`".

You can use the question mark key to display a selection list of possible matches to an argument you have partially typed. This is especially useful when you are not sure what you are looking for.

3. Press ALT+M and 'O' to display the Open Form dialog box again.
4. Type `_c?`. You should now see a selection list of dialog boxes (also called forms) that start with "`_c`". You can select an item in the list by moving your cursor to an entry and pressing ENTER.

Some dialog boxes which support completion are: Open, Save As, Go To Definition, Change Directory, and Bind Command to Key. When you press SPACEBAR while entering an argument, you may see the message "Word expanded as much as possible". Type "?" to list the matches.

Completion not only enables you to type in arguments faster but also can help you remember the argument.

Command Line Argument Retrieval

The term "argument retrieval," sometimes called "argument history," means that arguments you give to commands are saved for later use. When you are prompted for an argument on the command line, press the up and down arrow keys to retrieve your previous or next argument. For most commands, the last argument you entered is automatically displayed.

Shelling

You can use the Visual SlickEdit command line near the bottom of the MDI window to execute any program. Click on the command line or press ESC to toggle the cursor to the command line. Enter the program name and arguments and press ENTER.

When you enter a command that the editor does not recognize as internal, a path search is performed to find an external program to execute. To invoke a program whose name contains space characters, enclose the name in double quotes. For example, "`this is`" would execute a program called "`this is.exe`" if it existed.

UNIX:

You will need to prefix your command with "`dos` " if your program name starts with a leading '/' character (specifying a path most likely) because Visual SlickEdit will think you are performing a search.

Executing the command `dir` (or `ls`) from the command line will invoke Visual SlickEdit's internal `dir` command. To bypass an internal command, prefix the command with "`dos` ". You can execute the DOS DIR command by typing "`dos -w dir`" and pressing ENTER. The "`-w`" option specifies that you want to view the output of the `dir` command before the window

is closed. To get an operating system prompt, type the command "**dos**" with no arguments or select the menu item ("Tools", "OS Shell").

IMPORTANT FOR UNIX: To execute a text mode program and view the resulting output, you must use the "**-w**" or "**-t**" options which runs the program specified in an xterm window. The "**-w**" option waits for a key press before the program terminates.

For example, "**dos -w sgrep**" will display help on the **sgrep** program. If you do not need to wait for a key press, use the "**-t**" option. For example, "**dos -t man cc**" will display help on the UNIX **cc** program. You can change the xterm and specify xterm options that Visual SlickEdit uses by setting an **VSLICKXTERM** environment variable. 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. See "[Setting Environment Variables](#)" on page 244 for more information.

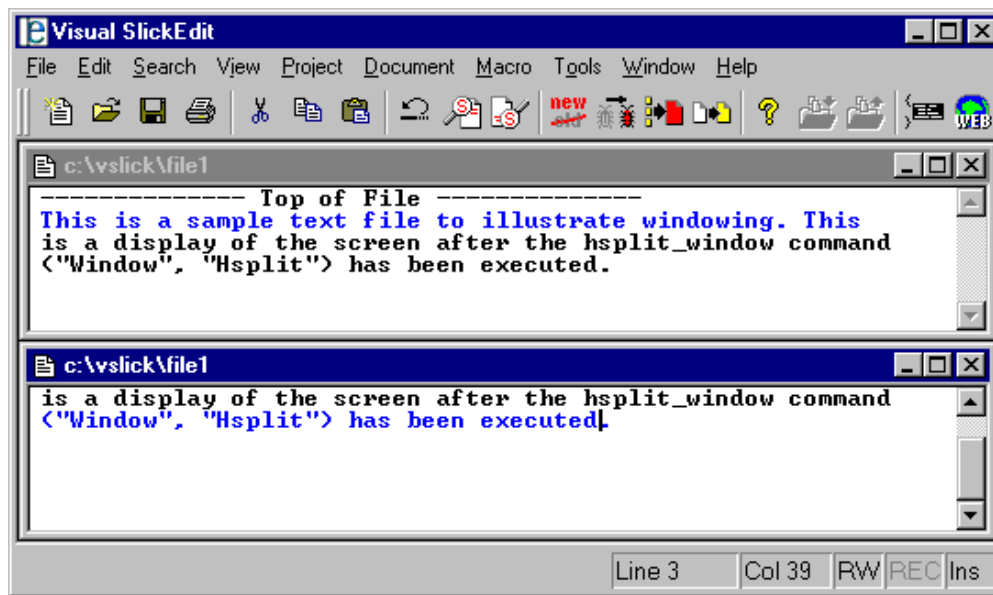
Changing the Current Directory

You can use the command line to change the current directory or drive. Press ESC to toggle the cursor to the command line and type "**cd** " followed by the directory you want to make active.

You can also use the [Change Directory dialog box](#) ("File", "Change Directory...") to change the current directory. Both methods support completion while entering the directory argument. A faster way to change directory is to use aliases (see "[Directory Aliases](#)" on page 20).

Windowing

One typical use for windowing is to view two parts of the same buffer at once. This allows you to scroll up and down the current window's buffer position while the buffer position in the other window remains the same. The buffer name of the current window is highlighted so that you know it is active. Below is a picture of the screen after the **hsplit_window** command (CTRL+H or "Window", "HSplit") has been executed.



After a window is split vertically or horizontally, you can use the mouse to move the adjoining edge to size both windows at once. To close the current window, double-click on the edit window system menu, or press CTRL+F4 to execute the **close_window** command. If the windows are still tiled, the **close_window** command will have the effect of unsplitting a window. The space once taken by the current window will be used by an adjoining tile.

You can use the **link_window** command ("Window", "Link Window") to select a different buffer to view in the current window. This is an easy way to view two specific buffers without the windows overlapping. Split a window and then use the **link_window** command to view a different buffer.

The number of windows that can be created is only limited by memory. However, if you split a window too many times, the window will become too small to split again.

The following commands create new windows:

create_tile	(ALT+F3) Allows you to create a new window by pointing to the edge for the new window with cursor keys. The current window is split into two equally sized windows.
hsplit_window	(CTRL+H or "Window", "Hsplit") Splits the current window horizontally into two equally sized windows.
vsplit_window	("Window", "Vsplit") Splits the current window vertically into two equally sized windows.

The following commands manipulate windows:

next_window	(Ctrl+Tab "Window", "Next") Switches to the next window.
prev_window	(Ctrl+Shift+Tab "Window", "Previous") Switches to the previous window.
change_window	Allows you to point to an adjoining tile with the cursor keys to the window you wish to change to.
window_above	Switches to window tile above the current window if one exists.
window_below	Switches to window tile below the current window if one exists.
window_left	Switches to window tile to the left of the current window if one exists.
window_right	Switches to window tile to the right of the current window if one exists.
move_edge	Press ALT+F2 to move the adjoining edge of a tiled window. Point to the window edge to move with the cursor keys, move the cursor to the new edge position, and press ENTER.
delete_tile	Delete an adjacent tiled window. Point to the edge of the window you wish to delete.
zoom_window	("Window", "Zoom Toggle")Zoom or unzoom the current window.
one_window	("Window", "One Window")Delete all windows except the current window. The current window is expanded to the largest possible size.
duplicate_window	("Window", "Duplicate Window")Create another window view of the current buffer.
tile_windows	("Window", "Tile")Resize and move all windows so that they do not overlap.
cascade_windows	("Window", "Cascade")Resizes and moves all windows so that more window tiles may be seen.

Text Operations

Selecting Text

There are three types of selections:

- Line selections: for processing complete lines of text. You should use line selections when editing source code because they work best with the **SmartPaste(TM)** feature (see "**SmartPaste(TM)**" on page 83).
- Block selections: for processing columns of text.
- Character selections: for processing sentences of text which do not start or end on line boundaries.

Steps to select the text:

1. Select the beginning of the text with one of the commands **select_line** (CTRL+L or "Edit", "Select...", "Line"), **select_block** (Ctrl+B or "Edit", "Select...", "Block") or **select_char** (F8 or "Edit", "Select...", "Char") which correspond to line select, block (column) select, and character select respectively.
2. Move the cursor to the end of the text you want to work with. The text between the start of the selection and your current cursor position should be selected.

You can also select text by clicking the left button and dragging. While the left button is still down, clicking the right button switches the selection type from CHAR to BLOCK to LINE to CHAR. Holding down the SHIFT key and clicking the left button extends a selection to the cursor. A double-click of the left button selects a word. A triple-click of the left button selects a line. Click and drag using the right mouse button to create a BLOCK type selection.

The following keys select text:

Deselect text.	CTRL+U
Start line selection	CTRL+L
Start block or column selection	CTRL+B
Start or extend character selection to right.	SHIFT+RIGHT
Start or extend character selection to left.	SHIFT+LEFT
Start or extend character selection up one line.	SHIFT+UP
Start or extend character selection down one line.	SHIFT+DOWN
Start or extend character selection up one page.	SHIFT+PGUP
Start or extend character selection down one page.	SHIFT+PGDN
Start or extend character selection to top of buffer.	CTRL+SHIFT+HOME
Start or extend character selection to bottom of buffer.	CTRL+SHIFT+END

See "**Special Mouse Functions**" on page 30 for information on selecting and operating on selected text with the mouse.

Visual SlickEdit provides different selection styles found in non-CUA compliant editors such as SlickEdit, Brief, and PE. Use the **General Configuration dialog box** ("Tools", "Configuration...", "General...", select the Select Styles tab) to choose a different selection style.

Operating on Selected Text

Now that the text has been selected, you can invoke a key or command that operates on the selected text. The following is a list of commands that operate on selected text. Where two commands are displayed, the first command is the command line version of the command, and the second is the graphical version of the command.

Command	Key Combination	Description
cut	(CTRL+X or "Edit", "Cut")	Deletes the selected text and copies it to the clipboard.
paste	(CTRL+V or "Edit", "Paste")	Inserts clipboard at cursor.
mou_paste	(Middle Mouse Button)	Inserts clipboard at cursor.
mou_click_copy	(CTRL+LButton)	Drags and copies the selected text. Click within the selected text and hold the left button down while moving the mouse to a new location. Line selections are inserted after the current line by default. If you want line selections inserted before the current line, change the line insert style. You can access the line insert style by ("Tools", "Configuration", "More Options...", "Line Insert Style").
mou_move_to_cursor	(CTRL+RButton)	Moves the selected text to the cursor.
mou_copy_to_cursor	(CTRL+SHIFT+RButton)	Copies the selected text to the cursor.
mou_click	(LButton)	Drags and moves the selected text. You must click within the selected text, and keep the left button down while moving the mouse to a new location.
list_clipboards	(CTRL+SHIFT+V or "Edit", "List Clipboards")	Allows you to select a clipboard from a list of your most recently used (15 is the default maximum) clipboards to insert at the cursor.
copy_to_clipboard	(CTRL+C or "Edit", "Copy")	Copies selected text to the clipboard.

append_to_clipboard	(CTRL+SHIFT+C or "Edit", "Append to Clipboard")	Appends selected text to the clipboard.
append_cut	(CTRL+SHIFT+X or "Edit", "Append Cut")	Deletes the selected text and appends it to the clipboard.
copy_to_cursor		Copies selected text to the cursor. Line selections are inserted after the current line by default. If you want line selections inserted before the current line change the line insert style. The line insert style may be accessed by ("Tools", "Configuration", click on the More tab, "Line insert style").
move_to_cursor	("Edit", "Move")	Moves selected text to the cursor.
fill_selection, gui_fill_selection	("Edit", "Fill...")	Fills the selected text with a character you are prompted for.
indent_selection	(TAB or "Edit", "Indent")	Indents the selected text by the "indent for each level" or by one tab stop. Indenting will be with tab or space characters depending upon whether the "Indent with tabs" check box is on. See "Extension Options dialog box" on page 216 for more information.
unindent_selection	(SHIFT+TAB or "Edit", "Unindent")	Unindents each line of selected text by the "indent for each level" or by one tab stop. Not all editing modes have an "indent for each level". See "Extension Options dialog box" on page 216 for more information.
lowcase_selection	("Edit", "Other", "Lowcase")	Converts the selected text to lowercase.
upcase_selection	("Edit", "Other", "Uppcase")	Converts the selected text to uppercase.
shift_selection_left	(SHIFT+F7 or "Edit", "Other", "Shift Left")	

		Shifts the text within the selection to the left. Supports line and block selections. If a character selection is used, it is converted to a line selection.
shift_selection_right	(SHIFT+F8 or "Edit", "Other", "Shift Right")	Shifts the text within the selection to the right. Supports line and block selections. If a character selection is used, it is converted to a line selection.
overlay_block	("Edit", "Other", "Overlay Block")	Overlays selected text at the cursor. Supports block selection only.
adjust_block	("Edit", "Other", "Adjust block")	Overlays selected text at cursor and fills the source selected text with blanks. Supports block selection only.
reflow_selection	("Document", "Format Selection")	Word wraps the text within the selected area. Line selections are word wrapped with current margin settings. Block selections are word wrapped within the columns of the block. Character selection is not supported.
execute_selection	(ALT+=)	Executes each line or sub-line of the selected text as if entered on the command line.
put <i>filename</i>, gui_write_selection	("File", "Write Selection...")	Writes the selected text to <i>filename</i> . Use the ("File", "Insert a file...") to insert a file.
append <i>filename</i>, gui_append_selection		Appends the selected text to the file <i>filename</i> .
sort_within_selection	("Tools", "Sort...", "Sort Within Selection...")	Sorts lines within a selected area.

sort_on_selection	("Tools", "Sort...", "Sort on Selection...") Sorts lines of text based on text within columns specified.
add	("Tools", "Add Selected Expr") Adds selected text and inserts result below the last line of the selection. Addition is performed for each adjacent line. If no operator exists between two adjacent numbers, addition is assumed. See "Calculating Mathematical Expressions" on page 88 for information on expression syntax.

Word/Variable Completion

If you often type the same variable name, the word/variable completion feature will probably save you a lot of typing time. This example shows how word/variable completion works:

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

Now press <CTRL+SHIFT+COMMA><CTRL+SHIFT+SPACE><CTRL+SHIFT+SPACE> to get the following result:

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

Note that in this example, the first prefix match of "pW" that was found was the one we wanted. When it is not, press CTRL+SHIFT+COMMA again to retrieve a previous one. This feature takes advantage of the fact that most variables are local variables which have a declaration and at least one usage. The following is a summary of word completion keys:

Complete previous occurrence	CTRL+SHIFT+COMMA
Complete next occurrence	CTRL+SHIFT+PERIOD
Complete more	CTRL+SHIFT+SPACE

Syntax Expansion and Indenting

Visual SlickEdit provides syntax expansion for the following languages:

- C
- C++
- Java
- Delphi

- Pascal
- AWK
- REXX
- Perl
- JavaScript
- PV-WAVE
- COBOL
- DBASE
- FORTRAN
- Ada
- Modula-2
- PHP
- Python
- Tcl
- VHDL
- Slick-C
- HTML

When you type a keyword such as **for** and press SPACEBAR, a template is inserted. This is called syntax expansion. For the C language, you would see the following text expansion:

```
for() {  
}
```

Loop, if and switch/case structures are expanded as well. 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 have typed, a list of possible keyword matches is displayed. To get the C template above you could just type "f" followed by SPACEBAR. To define a new keyword and expansion value, use the Alias Editor dialog box ("Tools", "Configuration", "File Extension Setup..."), choose the extension you want to change, click "Alias..."). SPACEBAR will only expand an alias if it is the first word on a line.

Options

You can access various syntax expansion options such as the brace style, and Minimum Expandable Keyword Length by pressing the "Options..." button on the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup..."). If the default syntax expansion options do not provide what you want or you want to add syntax expansion for a new keyword, define an extension-specific alias ("Tools", "Configuration", "File Extension Setup...", select the "General" tab, choose the extension you want to change, click "Aliases..."). See "**Extension-Specific Aliases**" on page 156 for more information.

To turn off syntax expansion, uncheck the "Syntax Expansion" check box ("Tools", "Configuration", "File Extension Setup...", select the "Indent" tab, choose the extension you want to change).

Syntax Indenting

When you press ENTER while editing a source file, Visual SlickEdit will indent to the next level if the cursor is moved inside a structure block. This is called syntax indenting. For

example, if you edit a C file and the cursor is on a line containing "for (;;){" and you press ENTER, a new line will be inserted and the cursor will be indented four spaces in from the 'f' character in **for**. The syntax indent amount may be changed in the **Extension Options dialog box** ("Tools", "Configuration", "File extension setup..", select the "Indent" tab, pick a file extension, "Syntax Indent").

SmartPaste(TM)

When you paste lines of text into a source file, SmartPaste reindents the added lines according to the surrounding code. For example, if you edit 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. You can use the mouse to copy and move lines and still take advantage of SmartPaste.

Currently SmartPaste does not attempt to reindent source code if the clipboard text is not indented at all, or if it starts with preprocessing. There are a few other cases where SmartPaste will not perform reindenting. If you have some good ideas for making SmartPaste smarter, please contact SlickEdit.

Visual SlickEdit supports SmartPaste™ for the following languages:

- C
- C++
- Java
- Delphi
- AWK
- Perl
- JavaScript
- PL/I
- PL/SQL
- Python
- PV-WAVE
- Pascal
- Tcl
- PHP
- Slick-C macro language

Moving Text Among Files With Clipboards

For online help for clipboard keys, press ALT+H, 'K' , and select "Clipboard". For information on selecting text see "**Text Operations**" on page 76.

You can use clipboards to copy or move text from anywhere to anywhere, including a file you are editing, the command line, a dialog text box, or another application that supports text clipboards, such as a word processor. When you invoke a cut or copy command, a clipboard is

created. You can insert this clipboard back into your buffer by pressing CTRL+V. This means you can actually use cutting text as another means of copying text.

Press CTRL+K ("Edit", "Copy word") to copy the current word to the clipboard. Then use CTRL+V to paste it anywhere. Pressing CTRL+K multiple times in succession creates one clipboard. All clipboard-related commands are available on the Edit drop-down menu.

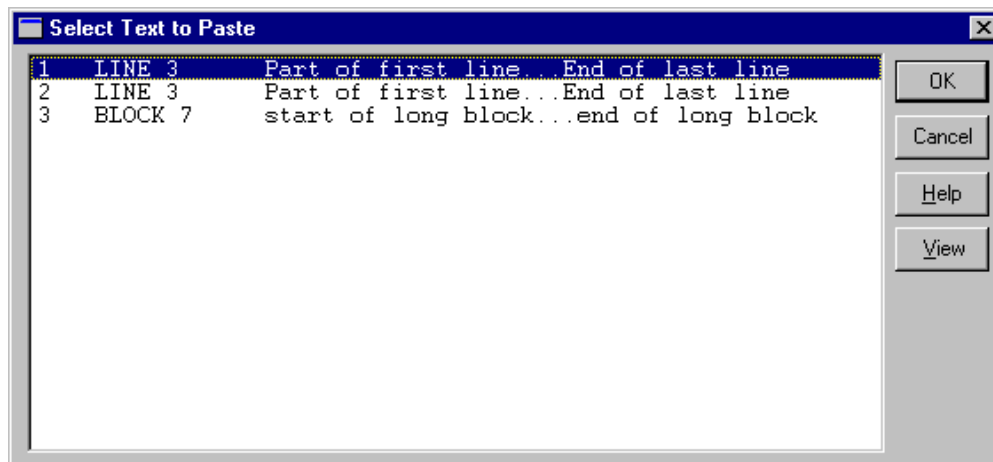
To try this feature, follow these steps:

1. Go to the beginning of a line with some text and press CTRL+E to erase to end of line.
2. Press CTRL+V to paste the text back in.
3. Move your cursor somewhere else in the buffer and press CTRL+V. You have just made a copy of a line of text without selecting anything first.

Pressing the same cut key multiple times in succession creates one clipboard. If you press CTRL+SHIFT+K three times to cut three words, you will get one clipboard that you can insert with CTRL+V. This is true for cut line (CTRL+BACKSPACE) and erase to end of line (CTRL+E) as well.

NOTE: If you are using BRIEF emulation and you want a clipboard created when you cut text, bind the commands **cut_word**, **cut_end_line**, and **cut_line** to the appropriate keys. See "[Changing Key Bindings](#)" on page 224 for more information.

Visual SlickEdit keeps a stack of the last 15 (default maximum) of your most recently used clipboards. You can change the maximum number of clipboards in the **General Configuration dialog box** ("Tools", "Configuration", "General...", select General tab). When you press CTRL+SHIFT+V ("Edit", "List Clipboards..." or **list_clipboards** command), you will see a dialog box that looks like the following picture:



- The numbers on the far left are used to help you move the selection cursor.
- The word following the number is the clipboard type and is not important when inserting a clipboard into the command line or a text box.
- The number following the clipboard type indicates the number for complete or partial lines of text in the clipboard.

Only clipboards of one line can be inserted into the command line or text box.

Both CTRL+V and CTRL+SHIFT+V key sequences insert clipboard text into the text area or the command line. The result of inserting a clipboard into the text area varies depending on the clipboard type. A LINE type clipboard is inserted after the current line by default. If you want LINE type clipboards inserted before the current line, change the line insert style ("Tools", "Configuration", "General...", "More" tab, "Line Insert Style..."). A BLOCK type clipboard is inserted before the current character and pushes over all text intersecting with the block. No lines are inserted. A CHAR type clipboard is inserted before the current character.

The View button allows you to look at the complete text for the selected condensed clipboard. While viewing the clipboard, you can copy all or part of it to the operating system clipboard.

Spell Checking

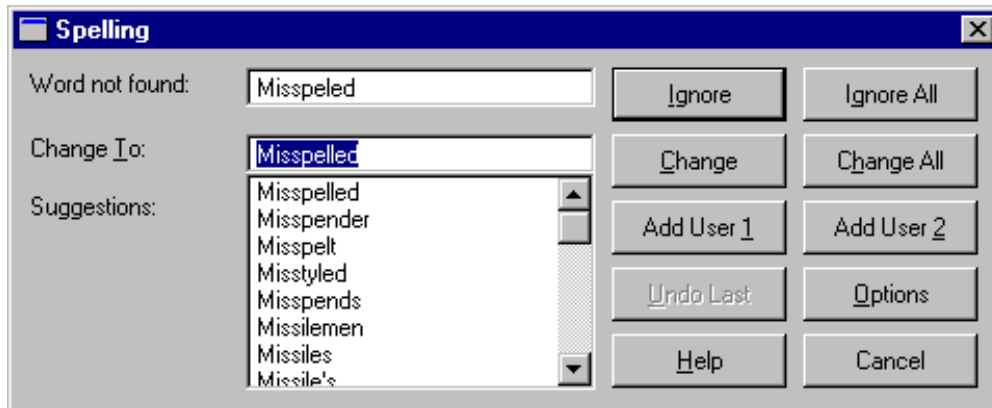
The Spell Checking menu ("Tools", "Spell Check") contains all of Visual SlickEdit's spell checking features, including spell checking from the cursor, spell checking a selection, spell

checking the current word, spell checking multiple files, and access to the **Spell Options dialog box**. When a misspelled or repeated word is found, the Spelling dialog box is displayed.

Spelling Dialog Box

("Tools", "Spell Check" or **spell_check** command)

Below is a picture of the Spelling dialog box:



Spell checking options are as follows:

Ignore

Ignores the misspelled or repeated word and continues spell checking.

Ignore All

Ignores all instances of the misspelled word until the Spelling dialog box is closed.

Change

Replaces the misspelled word with the word in the "Change To" text box and continues spell checking.

Change All

Replaces the misspelled word with the word in the "Change To" text box for the current occurrence and all additional occurrences found during the current spell checking session.

Add User 1

Adds the misspelled word to the user list 1 of correctly spelled words and continues spell checking. These words are saved in the text file "userdct1.dct" by default.

Add User 2

Adds the misspelled word to the user list 2 of correctly spelled words and continues spell checking. These words are saved in the text file "userdct2.dct" by default.

Undo Last

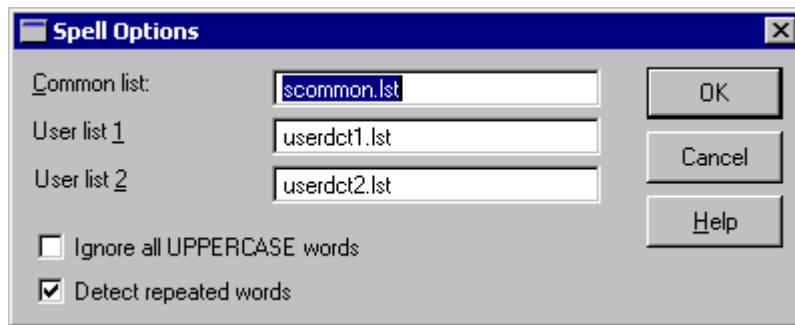
Undoes the last spell checking modification. This undo operation is limited to the last change made.

Options

Displays the Spell Options dialog box. See Spell Options Dialog Box below.

Spell Options Dialog Box

Below is a picture of the Spell Options dialog box ("Tools", "Spell Check", "Spell Options..." or **spell_options** command):

**Common List**

Words in this file are words that are found most frequently and words of length two or less. The common word list speeds up spell checking when most of the words in the buffers being checked are contained in the common word list.

User List 1

Specifies a text file which contains additional words that you want to be considered as correctly spelled words. Each line can contain only one word.

User List 2

Specifies a text file which contains additional words that you want to be considered as correctly spelled words. Each line can contain only one word.

Begin/end Structure Matching

When the cursor is on a begin or end keyword pair for one of the supported languages and you invoke the `find_matching_paren` command (CTRL+] by default or "Tools", "Match Paren"), the cursor is placed on the opposite end of the structure. Visual SlickEdit supports begin/end structure matching for the following languages:

- C++
- C
- Java
- Perl
- JavaScript
- PL/SQL
- PL/I
- AWK
- Delphi
- REXX
- Assembler
- Modula-2
- DBASE
- Pascal
- InstallScript
- Tcl
- PHP
- Slick-C

The **find_matching_paren** command also supports matching parenthesis pairs {},[], and (). See "**Paren match style**" on page 213 for information on additional parenthesis matching features. Use the **Extension Options dialog box** to view the begin/end pairs for any language ("Tools", "Configuration", "File Extension Setup...", select the "Advanced" tab, select the extension). You can add begin/end structure matching support for other languages by specifying a pseudo regular expression. See "**Extension Options dialog box**" on page 216 for more information.

Calculating Mathematical Expressions

Calculator Dialog Box

("Tools", "Evaluate Expr..." or **calculator** command)

Choose the "Evaluate Expr..." menu item on the Other menu to invoke the calculator. The calculator is somewhat different than other calculators. It lets you type in a mathematical expression from the keyboard or by clicking buttons, including parentheses. Almost all the editing keys you are familiar with, such as undo, next word, and previous word are supported. The calculator uses a slightly enhanced C expression syntax. The calculator does support specifying binary numbers and allows just an 'x' prefix when specifying hexadecimal numbers.

For example: to add the decimal numbers 135 and 288, simply type "**135+288=**". Pressing the '=' character evaluates the expression and places the result on the next line. Once you have the result, click on the Hex, Dec, Oct, Bin button to see the result in a different base.

Expressions with Mixed Bases

To add hex FF with octal 77 with binary 111 with decimal 99, perform the following steps:

1. Click the Hex button and type or click FF.
2. Type or Click +.
3. Click the Octal button and type or click 77.
4. Type or Click +.
5. Click the Bin button and type or click 111.
6. Type or Click +.
7. Click the Dec button 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

You can evaluate mathematical expressions by selecting expressions in a buffer and executing the **add** command ("Tools", "Add Selected Expr") or by executing one of the **math** commands on the Visual SlickEdit 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 ("Tools", "Calculate expression...") evaluates the Slick-C language *expression* given and places the results in the message line. Unlike the Slick-C language, you can specify octal numbers by prefixing the number with a 0, and specify binary numbers by prefixing the number with 'b'. If no operator is specified between two unary expressions, addition is assumed. The characters '\$' and ';' 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.

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.

&,	bitwise AND, bitwise OR
^	xor
+, blank(s), -	addition, implied addition, subtraction
*, /, %	multiplication, division, remainder
**	power

Hexadecimal numbers are prefixed with the characters "0x" or just "x". Octal numbers are prefixed with the character "O" or digit "0".

NOTE: Not all Slick-C language operators are supported.

Examples:

math 2.5*2	Multiplies 2.5 times 2.
math 5/2	Divides 5 by 2.
mathx 255	Converts 255 to hexadecimal.
math xFF	Converts hexadecimal FF to decimal.
math o77	Converts octal 77 to decimal.
matho 255	Converts 255 to octal.
math 077+0xff+10	Adds octal 77, hex FF, and 10.

If overflow or underflow occurs, the message "Numeric overflow or underflow" will appear on the message line. Floating point numbers may have up to a 32-digit mantissa and a 9-digit exponent.

You can type mathematical expressions into a buffer and evaluate them with the **add** command. This feature is known as document math. The **add** command ("Tools", "Add Selected Expr") 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. Because it is so easy to use, you might prefer to use this Visual SlickEdit feature for simple spreadsheet tasks.

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, what its first divisor is. The **nextprime** command takes a decimal number as an argument and finds the next greater prime number.

Recording and Using Macros

Visual SlickEdit allows you to create your own Slick-C macros without understanding the Slick-C language. All you do is turn macro recording on, use the editor's various features, and turn macro recording off. After the macro is created, you can save it, bind it to a key, and even edit it. Even your responses to dialog boxes are recorded as Slick-C source.

Recording Macros

Below is a summary of the Macro menu items related to macro recording:

Record Macro	(CTRL+F11) Starts or ends macro recording.
--------------	--

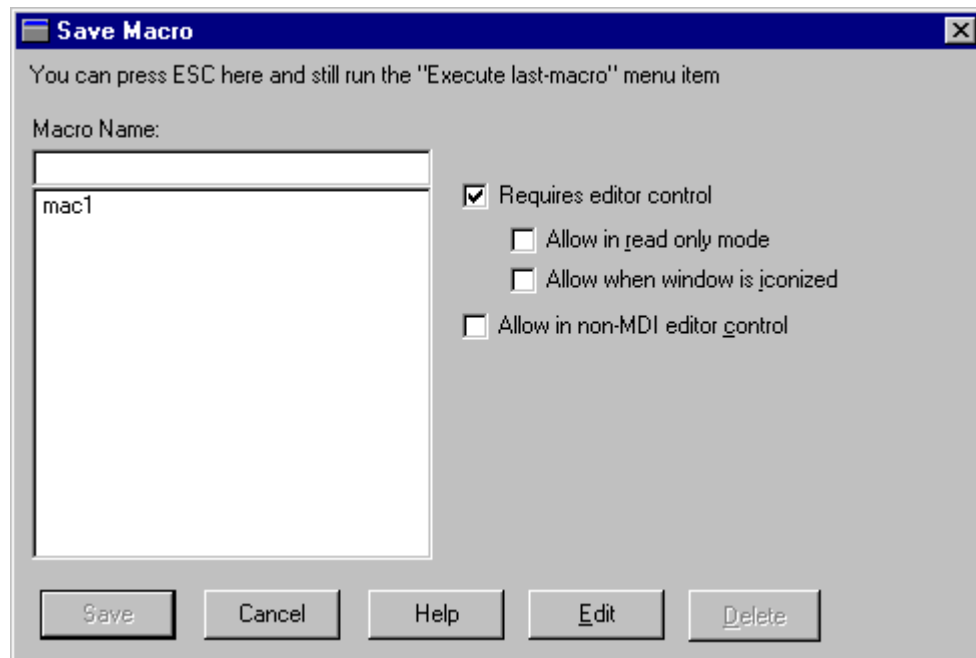
Execute last-macro	(CTRL+F12) Executes the last recorded macro.
Save last-macro	Saves the last recorded macro under a name you specify.
List Macros...	List recorded macros. Edit, run, or bind a macro to a key.

Once macro recording is turned on, the Recording Macro menu item on the Macro menu changes to Stop Recording Macro and the REC status indicator turns red. Press CTRL+F11 ("Macro", "Stop Recording Macro") to terminate macro recording. Alternatively, you can terminate macro recording and immediately execute the macro by pressing CTRL+F12. Press ESC to terminate recording.

Perform the following steps to record a simple macro:

1. Press CTRL+F11 to start recording.
2. Press the Right arrow key three times.
3. Press CTRL+F11 to end recording.

After performing these steps, the Save Macro dialog box ("Macro", "Save last-macro..." or **gui_save_macro** command) will appear.



Specify the name for your macro in the "Macro Name" text box. The various check boxes specify when your macro is allowed to be executed. The following paragraphs describe how to decide which check boxes to check.

Requires editor control

Check this box if your macro can only operate if the target is an editor control.

Allow in read only mode

Check this box if your macro does not modify the current buffer.

Allow when window is iconized

You will probably NOT want this box checked if your macro modifies the current buffer. Whether to check this box is more a matter of personal taste.

Allow in non-MDI editor control

Check this box if your macro should be allowed in a non-MDI editor control. This is typical for commands which require an editor control but do not open or close editor windows/buffers.

Edit

If you press this button, you can view the source code that has been generated without giving the macro a name. After viewing the source you can use the "Save last-macro" menu item on the Macro menu to give this macro a name. The Bind Command to Key dialog box will not be displayed if you press this button.

For this example, follow these steps:

1. Turn on the "Allow in read only mode" check box.
2. Turn on the "Allow in non-MDI editor control" check box.
3. Type "**example**" for the "Macro Name".
4. Click "SAVE".

The next dialog box you will see is the **Bind Command to Key dialog box**. For this example press ALT+E to edit the Slick-C source code. For more information on the **Bind Command to Key dialog box**, see "**Changing Key Bindings**" on page 224. You should now see some Slick-C source for the macro you just created.

```
#include "slick.sh"
_command example() name_info('VSARG2_MACRO|VSARG2_MARK|
                           VSARG2_READ_ONLY|VSARG2_REQUIRES_EDITORCTL)
{
    _macro('R',1)
    cursor_right(3)
}
```

You can use the online help to find out more about the functions being called. For help on the "**cursor_right**" function, place your cursor somewhere on the function and press F1, or click on the question mark icon on the button bar.

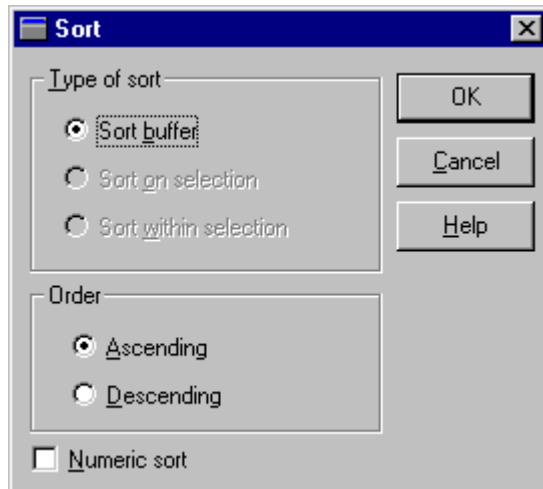
Sorting

Visual SlickEdit uses a stable quicksort algorithm to sort text. It is recommended that at least half the text be in memory for best speed results.

Sort Dialog Box

("Tools", "Sort..." or **gui_sort** command)

Below is a picture of the Sort dialog box ("Tools", "Sort..."):



Sort buffer

When on, the entire contents of the current buffer are sorted.

Sort on selection

When on, 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 on, the selected text is sorted. Text outside a block or column selection is not moved. Sort On Selection and Sort Within Selection have the same effect except when sorting a block or column selection.

Order

In an ascending sort, the lowest text item sorted is placed at the top.

Numeric sort

When on, a numeric compare is performed. Visual SlickEdit supports C-style floating point numbers with up to 32-digit mantissa.

Remove duplicate lines

When on, removes adjacent lines that are identical. This feature does not yet fully support column selection (it always compares complete lines).

Case sensitive

When on, the sort is case sensitive.

Sort Commands

If you prefer using the command line, you can sort text with one of the following commands (syntax for the commands is also shown):

sort_buffer [A][D][I][E] [-N]-F]

sort_on_selection [A][D][I][E] [-N]-F]

sort_within_selection [A][D][I][E] [-N]-F]

You can specify the following multiple options:

A	Sort in ascending order.
D	Sort in descending order.
I	Case insensitive sort (Ignore case).
E	Case sensitive sort (Exact case which is the default).
-N	Numeric sort. C-style floating point numbers with up to 32-digit mantissa are supported.
-F	Filename sort.

You can sort the current buffer with the **sort_buffer** command.

You can sort the text within a selected area with the command **sort_within_selection**. See "[Text Operations](#)" on page 76 for help on selecting text. The **sort_within_selection** command supports line and block selections only.

To sort on a column field, press CTRL+B to select an area of text and invoke the command **sort_on_selection**. See "[Text Operations](#)" on page 76 for help on selecting text. The **sort_on_selection** command supports line and block selections only.

The commands **sort_within_selection** and **sort_on_selection** have the same effect when a line selection is used. When a block selection is used, the **sort_within_selection** command will only sort text inside the left and right edge of the block selection, whereas the **sort_on_selection** command will sort the lines over which the block selection extends, based on the text within the block selection.

Search and Replace

For online help about searching keys, press ALT+H, 'K', and select "Searching".

Visual SlickEdit provides a rich set of searching capabilities, many of which are available on the Search menu ("Search"). Besides the commands found on the Search menu ("Search"), there is an incremental search command (**i_search**) which will search for a string as you type it. Both the Find and Replace dialog boxes support searching multiple files and/or buffers. In addition, you can use the file manager to perform a search and replace on multiple files. You can also use the **grep** (UNIX: **sgrep**) utility program to search through multiple files.

Incremental Searching

The fastest way to search for a string in Visual SlickEdit is to use incremental searching. During an incremental search, Visual SlickEdit will search for a string as you type it. You rarely need to type the entire string to get to the first occurrence. The **i_search** command (CTRL+I) starts a forward incremental search. The **reverse_i_search** command (CTRL+SHIFT+I) starts a reverse incremental search. The following keys take on a different definition during an incremental search:

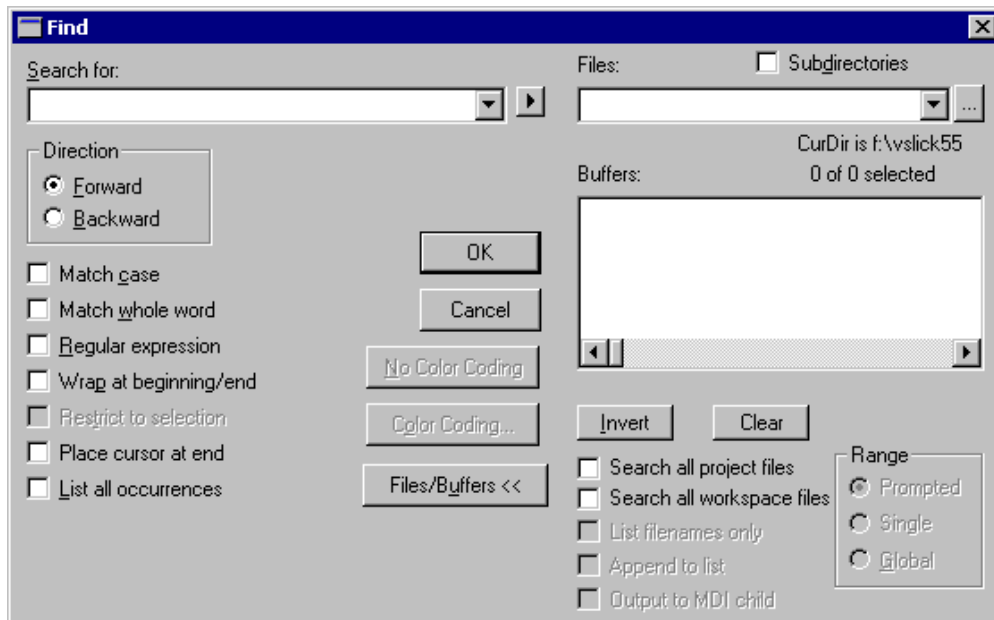
Search in reverse for the next occurrence of the search string.	CTRL+R
Search forward for the next occurrence of the search string.	CTRL+S
Toggle regular expression pattern matching on/off. (See " Regular Expressions " on page 107 for more information.)	CTRL+T
Toggle word searching on/off. (The default word characters are "A-Za-z0-9_ \$" and can be changed with the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab). See " Search and Replacing Variables " on page 102 for information on changing the names of variables.)	CTRL+W
Toggle case sensitivity. The key bound to the BRIEF emulation command case_toggle will also toggle the case sensitivity.	CTRL+C
Toggle searching within selection.	CTRL+M
Toggle incremental search mode.	CTRL+O
Quote the next character typed.	CTRL+Q

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 before entering a search string.

Find Dialog Box

The Find dialog box (CTRL+F or "Search", "Find...") searches for the string you specify. Press and hold CTRL+ALT+SHIFT to terminate a long search. The picture below is how the Find dialog box appears after pressing the "File/Buffers" button:



When the Find dialog box first appears, the options you used for your last search are displayed. This provides you a way to repeat your last search. In addition, you can press F7 and F8 to retrieve previous and next responses, respectively, to this dialog box.

Search For

Enter the string you want to search for here. You can retrieve previous search strings from this combo box list.

NOTE: ISPF search expressions cannot be used here.

Direction

Set the appropriate radio button Forward or Backward to perform a search toward the bottom of the buffer or the top of the buffer, respectively.

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 can be changed in

the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab).

Regular expression

When checked, a regular expression search is performed. See "**Regular Expressions**" on page 107 for more information.

Wrap at beginning/end

When checked, the search will always be performed on the entire buffer, starting from the cursor.

Restrict to selection

Limits the search to the selected text. This check box is disabled if no text is selected in the current buffer.

Place cursor at end

When checked, the cursor is placed at the end of the occurrence found.

List all occurrences

When checked, a list of all occurrences of the search string found in the current buffer is displayed. You can choose to go to a specific occurrence.

No Color Coding

Clears color coding search options.

Color Coding

Displays **Color Coding Search Options dialog box**. This feature is only supported for languages for which Visual SlickEdit has color coding elements defined. One way to use this dialog is to search for something that does not start in a comment or string. For example, if you use **printf** or some other function to help debug your programs, turn off "Comment" and "String" to search for a function not in a comment or string. Note that only the color coding for the first character is checked. You need to combine word searching with "Keyword" color coding to search for the keyword **"if"**.

Files/Buffers

This button expands or unexpands the dialog box. When the dialog box is expanded, and the Files text box and/or the Buffers list box have something specified, a multiple file and/or buffer search is performed.

Subdirectories

Indicates whether the search for a file specified in the Files text box traverses subdirectories.

Files

Specifies files on disk to be searched. If the file has already been opened (i.e. it is a buffer), the buffer is searched. You can specify wildcards ("*.c *.h") in this text box.

Buffers

Specifies one or more buffers to search through. Press CTRL+Click to select multiple buffers.

Search all project files

When checked, all files in the current project are searched in addition to other files/buffers specified.

CAUTION: You might end up searching a file twice if you specify other files/buffers in addition to your project files.

Search all workspace files

When checked, all files in the current workspace are searched in addition to other files/buffers specified.

CAUTION: You might end up searching a file twice if you specify other files/buffers in addition to your workspace files.

List filenames only

When checked, only file names and not occurrences are listed in the multi-file search output.

Append to List

When checked, multi-file search information is appended to the existing multi-file search results.

Output to MDI Child

When checked, output is sent to an MDI child edit window instead of the “Search” tab of the Output toolbar.

Prompted

When checked, you are prompted whether to continue searching when an occurrence is found.

Single

When checked, your cursor is placed on the first occurrence found, but the remaining files are not searched.

Global

When checked, all files are searched for occurrences without prompting.

Invert

Click on the Invert button to select all buffers in the Buffers list box that are currently not selected and deselect all files which are selected.

Clear

Deselects all buffers in the buffer list.

/ Command

If you prefer using the command line to perform a search, you can use the `/` or `find` commands which provide you with the same functionality as the Find dialog box. Press ESC to toggle the cursor to the command line.

The syntax of the `/` command is:

`/string[/OptionLetters]`

The syntax of the **find** command is:

`find /string[/OptionLetters]`

OptionLetters is one or more of the following option letters:

E	Exact case.
I	Ignore case.
-	Reverse search.
M	Limit search to selection.
<	If found, place cursor at beginning of word.
>	If found, place cursor at end of word.
R	Interpret search string as a SlickEdit regular expression. See " SlickEdit Regular Expressions " on page 111 for syntax of regular expression.
U	Interpret search string as UNIX regular expression. See " UNIX Regular Expressions " on page 108 for syntax of UNIX regular expressions.
B	Interpret string as a Brief regular expression. See " Brief Regular Expressions " on page 114 for syntax of Brief regular expressions.
N	Do not interpret search string as a regular expression.
P	Wrap to beginning/end when string not found.
W	Limit search to words. Used to search for variables. See " Search and Replacing Variables " on page 102 for information on changing the names of variables.
W= <i>SlickEdit-regular-expression</i>	Specifies the valid characters in a word. The default value is [A-Za-z0-9_\$]. To change the word characters for a specific extension, use the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab).
W:P	Limit search to word prefix. For example, a search for "pre" matches "pre" and "prefix" but not "supreme" or "supre".
W:PS	Limit search to strict word prefix. For example, a search for "pre" matches "prefix" but not "pre", "supreme" or "supre".
W:S	Limit search to word suffix. For example, a search for "fix" matches "fix" and "suffix" but not "fixit".
W:SS	Limit search to strict word suffix. For example, a search for "fix" matches "suffix" but not "fix" or "fixit".
H	Allow finding search string in hidden lines.
Y	Binary search. This allows start positions in the middle of a DBCS or UTF-8 character. This option is useful when editing binary files (in SBCS/DBCS mode) which may contain characters which look like DBCS but are not. For example, if you search for the character 'a', it will not be found as the second character of a DBCS sequence unless this option is specified.
,	(comma) Delimiter to separate ambiguous options.
<i>XCCLetters</i>	

Requires the first character of search string NOT be one of the color coding elements specified. For example, "XCS" requires that the first character not be in a comment or string. *CCLetters* is a string of one or more of the following color coding element letters:

O	Other
K	Keyword
N	Number
S	String
C	Comment
P	Preprocessing
L	Line number
1	Symbol 1
2	Symbol 2
3	Symbol 3
4	Symbol 4
F	Function color
V	No save line
A	Attribute

CCCLetters

Requires the first character of search string to be one of the color coding elements specified. See *CCLetters* above.

You can set default search options with the "Search" tab ("Tools", "Configuration", "General...", select Search tab).

If you use the **find** command, the first non-blank character is used as the delimiter. See previous section for information on option letters.

Examples:

/if/xcs,w	Search for word-delimited string "if" where the first character is not in a comment or string. (For most languages this will find the if keyword.)
/if/ck,w	Search for word-delimited string "if" where the first character is a keyword. (For most languages this will find the if keyword.)
/xyz/-	Search for string "xyz" using default search case in reverse.
find /xyz/-	Search for string "xyz" using default search case in reverse. NOTE: prefixing the string "/xyz/-" with find has no effect on this command because the first non-blank is a delimiter.
find \$xyz\$-	Search for string "xyz" using default search case in reverse. The find command can change the string delimiter which in this case is not necessary.
find \$/\$	Search forward for the character slash. The find command has been used so that the string delimiter could be changed to \$ which allows searching for the / character.
find \$/\$-	Search backward for the character slash.
/i/wxcs	Search for the variable "i" not in comment and not in a string.
/xyz/e	Search for "xyz" in exact case.

When no parameters are specified, you are prompted for the string parameter. While you enter the string parameter, the following keys take on a new meaning:

CTRL+R	Search in reverse for the next occurrence of the search string.
CTRL+S	Search forward for the next occurrence of the search string.
CTRL+T	Toggle regular expression pattern matching on/off. The key bound to the BRIEF emulation command re_toggle will also toggle regular expression pattern matching. (See " Regular Expressions " on page 107 for more information.)
CTRL+W	Toggle word searching on/off. The default word characters are "A-Za-z0-9_". To change the word characters for a specific extension, use the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab). See " Search and Replacing Variables " on page 102 for information on changing the names of variables.
CTRL+C	Toggle case sensitivity. The key bound to the BRIEF emulation command case_toggle will also toggle the case sensitivity.
CTRL+M	Toggle searching within selection.
CTRL+O	Toggle incremental search mode.
CTRL+Q	Quote the next character typed.

Searching for the Next Occurrence

After using any of the search commands, you can search for the next and previous occurrences by pressing CTRL+G ("Search", "Next occurrence") and CTRL+SHIFT+G ("Search", "Previous occurrence") respectively.

Replace Dialog Box

The Replace dialog box (CTRL+R or "Search", "Replace...") performs a search and replace. This command searches for the string specified in the "Search For:" text box and replaces it with the string specified in the "Replace With:" text box. Press and hold CTRL+ALT+SHIFT to terminate a long search. This dialog box is identical to the **Find dialog box** except for the Replace All button and the Preserve case check box. See "**Find Dialog Box**" on page 96 for information on other options of this dialog box.

Click Replace All instead of OK if you do not want to be prompted when replacing. When an occurrence of the search string is found after clicking OK, a dialog box with Yes, No, Last, Go, and Quit buttons is displayed. Type the appropriate letter from the following list:

- **Y** to do the replace
- **N** to skip this occurrence and search for the next occurrence
- **L** to do the replace and cancel the search
- **G** to replace all occurrences without prompting any more
- **Q** to cancel the search

Preserve Case

When "Preserve Case" is checked, each occurrence found is checked for all lowercase, all uppercase, first letter 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.

When the Replace dialog box first appears, the options you used for your last search are displayed, enabling you to repeat your last Replace operation. You can also press F7 and F8 to retrieve previous and next responses, respectively, to this dialog box.

Search and Replacing Variables

You can use the **Replace dialog box** to replace occurrences of variables. For example, to replace occurrences of the variable "i" to "something_more_meaningful" follow these steps:

1. Press CTRL+R
2. Type **i** in the Search For text box
3. Type **something_more_meaningful**
4. Check the "Match Whole Word" check box
5. Press ENTER

c Command

If you prefer using the command line to perform a search and replace, you can use the **c** command which provides you with the same functionality as the **Replace dialog box**. Press and hold CTRL+ALT+SHIFT to terminate a long search. Press ESC to toggle the cursor to the command line.

The syntax of the **c** command is:

```
c /string1/string2[/OptionLetters]
```

OptionLetters can consist of one or more of the following letters:

E	Exact case.
I	Ignore case.
-	Reverse search.
M	Limit search to selection.
V	Preserve case. When specified, each occurrence found is checked for all lowercase, all uppercase, first word capitalized or mixed case. The replace string is converted to the same case as the occurrence found except when the occurrence found is mixed case (possibly multiple capitalized words). In this case, the replace string is used without modification.
<	If found, cursor is positioned at beginning of word.
>	If found, cursor is positioned at end of word.
R	Regular expression search and replace. See " SlickEdit Regular Expressions " on page 111 for syntax of regular expression.
U	UNIX regular expression search and replace. See " UNIX Regular Expressions " on page 108 for syntax of UNIX regular expressions.
B	Interpret string as a Brief regular expression. See " Brief Regular Expressions " on page 114 for syntax of Brief regular expressions.
W	Limits search to words. Used for search and replacing variables.
W=SlickEdit-regular-expression	Specifies the valid characters in a word. The default value is [A-Za-z0-9_\$]. To change the word characters for a specific extension, use the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab).
W:P	Limits search to word prefix. For example, searching for "pre" matches "pre" and "prefix" but not "supreme" or "supre".
W:PS	Limits search to strict word prefix. For example, searching for "pre" matches "prefix" but not "pre", "supreme" or "supre".
W:S	Limits search to word suffix. For example, searching for "fix" matches "fix" and "sufix" but not "fixit".
W:SS	Limits search to strict word suffix. For example, searching for "fix" matches "sufix" but not "fix" or "fixit".
*	Change all occurrences without prompting.
H	Allow finding search string in hidden lines.
Y	Binary search. This allows start positions in the middle of a DBCS or UTF-8 character. This option is useful when editing binary files (in SBCS/DBCS mode) which may contain characters which look like DBCS

but are not. For example, if you search for the character 'a', it will not be found as the second character of a DBCS sequence unless this option is specified.

, (comma) Delimiter to separate ambiguous options.

XCCLetters

Requires that the first character of a search string NOT be one of the color coding elements specified. For example, "XCS" requires that the first character not be in a comment or string. *CCLetters* is a string of one or more of the following color coding element letters:

O	Other
K	Keyword
N	Number
S	String
C	Comment
P	Preprocessing
L	Line number
1	Symbol 1
2	Symbol 2
3	Symbol 3
4	Symbol 4
F	Function color
V	No save line
A	Attribute

CCCLetters

Requires the first character of search string to be one of the color coding elements specified. See *CCLetters* above.

You can set default search options with the "Search" tab ("Tools", "Configuration", "General...", select Search tab).

If the '*' option is not specified, you will be prompted with the message "Yes/No/Last/Go/Quit/Suspend?" for each occurrence of the "Search for:" string. You can take one of the following actions by pressing the corresponding key:

Action	Key
Make change and continue searching.	'Y' or SPACE
No change and continue searching.	'N' or BACKSPACE
Make change and stop searching.	'L' or .
Make change and change the rest without prompting.	'G' or !
Exit command.	'Q' or ESC

(By default, the cursor is NOT restored to its original position. If you want the cursor restored to its original position, turn on the "Restore cursor after replace" check box on the "Search" tab ("Tools", "Configuration", "General...",

select Search tab).

Suspend change and replace. (Invoke the resume command to restart the search and replace from the new cursor position. Only one level of suspension is allowed, therefore you must issue the resume command before you can use the suspend option in a new search and replace command. This is an advanced option which allows you to edit the area where the string was found any way you like and then resume the command.)	'S'
Exit command and restore cursor to its original position.	CTRL+G
Search in reverse for next occurrence of search string.	CTRL+R
Search forward for next occurrence of search string.	CTRL+S
Toggle regular expression pattern matching on/off. (The key bound to the BRIEF emulation command re_toggle will also toggle regular expression pattern matching. See " Regular Expressions " on page 107 for more information.)	CTRL+T
Toggle word searching on/off. (To change the word characters for a specific extension, use the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab).	CTRL+W
Toggle case sensitivity. (The key bound to the BRIEF emulation command case_toggle will also toggle the case sensitivity.)	CTRL+C
Toggle searching within selection.	CTRL+M
Display help on Replace dialog box .	F1 or '?'

No space character is needed between the 'C' and the '/' delimiter UNLESS you use a delimiter other than '/'. You can use any character except space as the delimiter.

Examples:

<code>c \$/\$\$</code>	Replace occurrences of forward slashes with back slashes.
<code>c/x/y/m</code>	Replace occurrences of x in the selected area with y using default search case sensitivity.
<code>c \$x\$y\$m</code>	Replace occurrences of x in the selected area with y using default search case sensitivity. The string delimiter \$ has been used requiring a space character after the C.
<code>c/x/y/e*</code>	Replace lowercase occurrences of x with y without prompting.
<code>c/i/something_more_meaningful/w</code>	Replace occurrences of the variable i with something_more_meaningful.
<code>c/i/j/w=[A-Za-z]</code>	Replace occurrences of the word i with j and specify valid characters in a word to be alphabetic characters.

Multiple File Search and Replace

The **Find dialog box** and the **Replace dialog box** both allow you to perform searches across multiple files and/or buffers. Click on the "File/Buffers >>" button to view these additional options of the Find and Replace dialog boxes. Alternatively, you can perform a search and replace while in the Visual SlickEdit file manager. This allows you to be more selective about which files to operate on:

1. Use the **fileman** command ("File", "File Manager") to start the file manager
2. Select the files with SPACEBAR or the file manager select menu ("File", "File Manager", "Select")
3. Invoke the **fileman_replace** command (CTRL+SHIFT+G or "File", "File Manager", "Global Replace...")

You might prefer using the **grep** (UNIX: **sgrep**) utility in the concurrent process buffer to perform a search (without replace) through multiple files instead of using the **Find dialog box**.

1. Use the **start_process** command (CTRL+SHIFT+M or "Build", "Activate Process") to start the concurrent process buffer
2. Run the **grep** (UNIX: **sgrep**) program to search the files (run the program with no arguments for help on arguments)

For example, the command **grep -w buffer_list *.c** will perform a word search for the variable `buffer_list` in all files with extension ".c". Since the `grep` program outputs the file name, line number, and column position, you can use the **next_error** (CTRL+SHIFT+DOWN or "Build", "Next Error"), or **cursor_error** command (ALT+1, Double-click, or "Build", "Go to Error/Include") to place your cursor in the file(s) containing the occurrences.

Another way to perform a search through multiple files is to run the **grep** (UNIX: **sgrep**) program from the Visual SlickEdit command line and prefix the command with "**dos -e**". For example, the command **dos -e grep -w buffer_list *.c** will perform a word search for the variable `buffer_list` in all files with extension ".c". If any occurrences are found, your cursor will be placed in the file containing the first occurrence found after the `grep` (UNIX: `sgrep`) program finishes searching.

Regular Expressions

Sometimes searching for a string literal is too limiting. For instance, you cannot search for a quoted string, a blank line, a word starting at the beginning of a line, or two words separated by any number of spaces. A regular expression can describe these search strings and many more.

All search commands support regular expressions. The Find and Replace dialog boxes have a Regular Expression check box for turning on regular expression searching. CTRL+T toggles regular expression searching on/off while an incremental search is in progress (see "[Incremental searching](#)" on page 95). The search commands "/" and "find" take 'R' and 'U' options to interpret the search string as a regular expression (see "[/ command](#)" on page 98). The search and replace command "**c**" also takes 'R', 'U', and 'B' options to specify a regular expression (see "[c command](#)" on page 103).

Visual SlickEdit supports three types of syntax of regular expressions:

- UNIX (see "[UNIX Regular Expressions](#)" on page 108)
- SlickEdit (see "[SlickEdit Regular Expressions](#)" on page 111)
- Brief (see "[Brief Regular Expressions](#)" on page 114).

All syntax types have the same features. In order to accomplish this, SlickEdit Inc. made several enhancements to the UNIX and Brief syntaxes. Select the regular expression syntax from the "Search" tab on the General Configuration dialog box (Other, "Configuration", "General...").

For all syntax types: When you specify Unicode characters in a regular expression (**\x{hhhh}** or **[p{Spec}]**) to be used on a SBCS/DBCS mode buffer, the characters are converted to the active code page (at least the ones that exist). This works well; however, currently SBCS/DBCS mode character sets do not fully support DBCS. DBCS character set matching only checks the lead byte. If Unicode support is not active (it is not active under UNIX), Unicode characters above 127 are not translated. When searching in a Unicode buffer, "**\x{FF}**" is converted to UTF-8 and will match the corresponding Unicode character. "**\xFF**" will not match anything because it is not a valid UTF-8 character. When searching in a SBCS/DBCS mode buffer, "**\x{FF}**" is converted to the active code page and will match a

SBCS or DBCS character. “\xFF” is not converted and will match one byte that is not part of a DBCS sequence.

UNIX Regular Expressions

UNIX regular expressions are defined as follows:

<code>^</code>	Matches beginning of line.
<code>\$</code>	Matches end of line.
<code>.</code>	Matches any character except newline.
<code>X+</code>	Maximal match of one or more occurrences of X. See " Minimal Versus Maximal Matching " on page 125 for more information on minimal and maximal matching.
<code>X*</code>	Maximal match of zero or more occurrences of X.
<code>X?</code>	Maximal match of zero or one occurrences of X.
<code>X{n1}</code>	Match exactly <i>n1</i> occurrences of X.
<code>X{n1,}</code>	Maximal match of at least <i>n1</i> occurrences of X.
<code>X{,n2}</code>	Maximal match of at least 0 occurrences but not more than <i>n2</i> occurrences of X.
<code>X{n1,n2}</code>	Maximal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
<code>X+?</code>	Minimal match of one or more occurrences of X.
<code>X*?</code>	Minimal match of zero or more occurrences of X.
<code>X??</code>	Minimal match of zero or one occurrences of X.
<code>X{n1}?</code>	Matches exactly <i>n1</i> occurrences of X.
<code>X{n1,}?</code>	Minimal match of at least <i>n1</i> occurrences of X.
<code>X{,n2}?</code>	Minimal match of at least 0 occurrences but not more than <i>n2</i> occurrences of X.
<code>X{n1,n2}?</code>	Minimal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
<code>(?!X)</code>	Search fails if expression X is matched. The expression <code>^(?!if)</code> matches the beginning of all lines that do not start with “if”.
<code>(X)</code>	Matches sub-expression X and specifies a new tagged expression. See " Tagged Search Expressions " on page 123 for more information. No more tagged expressions are defined once an explicit tagged expression number is specified as shown below.
<code>(?dX)</code>	Matches sub-expression X and specifies to use tagged expression number <i>d</i> where $0 \leq d \leq 9$. No more tagged expressions are defined by the sub-expression syntax "(X)" once this sub-expression syntax is used. This is the best way to make sure you have enough tagged expressions.
<code>(?:X)</code>	Matches sub-expression X but does not define a tagged expression.
<code>X Y</code>	Matches X or Y.
<code>[char-set]</code>	Matches any one of the characters specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression <code>[A-Z]</code> matches any uppercase letter. '\ may be used inside the square brackets to define literal characters or define ASCII characters. For example, <code>\" specifies a literal dash character. The expression <code>[\d0-\d27]</code> matches ASCII character codes 0..27. The expression <code>[]</code> matches a right bracket. In SlickEdit regular expressions, <code>[]</code> matches no characters. In both syntaxes, the expression <code>[]</code> matches a right bracket. The expression <code>[\^]</code> matches a '^' character but this</code>

	does not work for SlickEdit regular expressions. In both syntaxes, <code>[^]</code> matches a '^' character.
<code>[^char-set]</code>	Matches any character not specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression <code>[^A-Z]</code> matches all characters except uppercase letters.
<code>[char-set1 - [char-set2]]</code>	Character set subtraction. Matches all characters in <i>char-set1</i> except the characters in <i>char-set2</i> . For example, <code>"[a-z-[qw]]"</code> matches all English lowercase letters except 'q' and 'w'. <code>"[p{L}-[qw]]"</code> matches all Unicode lowercase letters except 'q' and 'w'.
<code>[char-set1 & [char-set2]</code>	Character set intersection. Matches all characters in <i>char-set1</i> that are also in <i>char-set2</i> . For example, <code>"[x{0}-\x{7f}&[p{L}]]"</code> matches all letters between 0 and 127.
<code>\x{hhhh}</code>	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhhh</i> .
<code>\p{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeCategorySpec</i> . Where <i>UnicodeCategorySpec</i> uses the standard general categories specified by the Unicode consortium. For example, <code>"[p{L}]"</code> matches all letters. <code>"[p{Lu}]"</code> matches all uppercase letters. See Unicode Category Specifications for Regular Expressions on page 118.
<code>\P{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeCategorySpec</i> . For example, <code>"[P{L}]"</code> matches all characters that are not letters. This is equivalent to <code>"[^\p{L}]"</code> . <code>"[P{Lu}]"</code> matches all characters that are not uppercase letters. See Unicode Category Specifications for Regular Expressions on page 118.
<code>\p{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeIsBlockSpec</i> . Where <i>UnicodeIsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, <code>"[p{isGreek}]"</code> matches Unicode characters in the Greek block. See Unicode Character Blocks for Regular Expressions on page 119.
<code>\P{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeIsBlockSpec</i> . For example, <code>"[P{isGreek}]"</code> matches all characters that are not in the Unicode Greek block. This is equivalent to <code>"[^\p{isGreek}]"</code> . See Unicode Character Blocks for Regular Expressions on page 119.
<code>\xhh</code>	Matches hexadecimal character <i>hh</i> where $0 \leq hh \leq 0xff$.
<code>\dddd</code>	Matches decimal character <i>ddd</i> where $0 \leq ddd \leq 255$.
<code>\c</code>	Specifies cursor position if match is found. If the expression <code>xyz\c</code> is found the cursor is placed after the <i>z</i> .
<code>\n</code>	Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file. Use <code>"\d10"</code> if you want to match a 10 character.
<code>\r</code>	Matches carriage return (ASCII 13).
<code>\t</code>	Matches tab character.
<code>\f</code>	Matches form feed character.
<code>\om</code>	Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end of line characters. For example, <code>"\om.+"</code> matches the rest of the buffer.

WARNING: Test your regular expression file before using your regular expression on a large file. This option may cause the editor to use a lot of memory.

<code>\ol</code>	Turns off multi-line matching (default). You can still use "\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.	
<code>\char</code>	Declares character after slash to be literal. For example, "*" represents the star character.	
<code>\:char</code>	Matches predefined expression corresponding to <i>char</i> . The predefined expressions are:	
<code>\:a</code>	<code>[A-Za-z0-9]</code>	Matches an alphanumeric character.
<code>\:b</code>	<code>([\t]+)</code>	Matches blanks.
<code>\:c</code>	<code>[A-Za-z]</code>	Matches an alphabetic character.
<code>\:d</code>	<code>[0-9]</code>	Matches a digit.
<code>\:f</code>	<code>([^\[\]:\V<> =+;, \t"]+)</code>	Windows: Matches a filename part.
<code>\:f</code>	<code>([^\t"]+)</code>	UNIX: Matches a filename part.
<code>\:h</code>	<code>([0-9A-Fa-f]+)</code>	Matches a hex number.
<code>\:i</code>	<code>([0-9]+)</code>	Matches an integer.
<code>\:n</code>	<code>(([0-9]+(\.[0-9]+) \.[0-9]+)([Ee](\+ -)[0-9]+))</code>	Matches a floating number.
<code>\:p</code>	<code>(([A-Za-z]:)(\\ /)(\:f(\\ /))*\:f)</code>	Windows: Matches a path.
<code>\:p</code>	<code>((/)(\:f(/))*\:f)</code>	UNIX: Matches a path.
<code>\:q</code>	<code>(\"[^\"]*" '[^']*')</code>	Matches a quoted string.
<code>\:v</code>	<code>([A-Za-z_][A-Za-z0-9_\$]*)</code>	Matches a C variable.
<code>\:w</code>	<code>[A-Za-z]+)</code>	Matches a word.

NOTE: The "\:f" and "\:p" predefined expressions are not intended to support all operating system file names. Instead, they are intended to be useful in practical cases. For Windows, "\:f" is designed for FAT (DOS) file systems. The space character is not in \:f because it is more typically a file name separator.

The precedence of operators, from highest to lowest, is as follows:

- `+,*,?, {},+?,*?,??, {}?` (These operators have the same precedence.)
- concatenation
- `|`

Sample Regular Expressions:

^defproc	Matches lines that begin with the word defproc.
^definit\$	Matches lines that only contain the word definit.
^*name	Matches lines that begin with the string <code>"*name"</code> . Notice that the backslash must prefix the special character <code>*</code> .
[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.
[a-z-[qw]]	Character set subtraction. Matches all English lowercase letters except 'q' and 'w'.
[p{isGreek}&[p{L}]]	Character set intersection. Matches all Unicode letters in the Greek block.
\\x{6587}	Matches Unicode character with hexadecimal value 6587.
[p{L}-[qw]]	Character set intersection. Matches all Unicode letters in the Greek block.
[p{L}]-[qw]	Matches all Unicode letters except 'q' and 'w'.
[p{L}]	Matches all Unicode letters.
[p{Lul}]	Matches all Unicode uppercase and lowercase letters.
[P{L}]	Matches all Unicode characters that are not letters.
[p{isGreek}]	Matches all Unicode characters in the Greek block.

SlickEdit Regular Expressions

SlickEdit regular expressions are defined as follows:

^	Matches beginning of line.
\$	Matches end of line.
?	Matches any character except newline.
X+	Minimal match of one or more occurrences of X. See " Minimal Versus Maximal Matching " on page 125 for more information.
X#	Maximal match of one or more occurrences of X.
X*	Minimal match of zero or more occurrences of X.
X@	Maximal match of zero or more occurrences of X.
X:n1	Matches exactly <i>n1</i> occurrences of X. Use () to avoid ambiguous expressions. For example "a:9()1" searches for 9 a's followed by a 1.
X:n1,	Maximal match of at least <i>n1</i> occurrences of X.
X:n1,n2	Maximal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
X:*n1,	Minimal match of at least <i>n1</i> occurrences of X.

<code>X:*n1,n2</code>	Minimal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
<code>~X</code>	Search fails if expression X is matched. The expression <code>^~(if)</code> matches the beginning of all lines that do not start with if.
<code>(X)</code>	Matches sub-expression X.
<code>{X}</code>	Matches sub-expression X and specifies a new tagged expression. See " Tagged Search Expressions " on page 123 for more information.
<code>{#dX}</code>	Matches sub-expression X and specifies to use tagged expression number <i>d</i> where $0 \leq d \leq 9$.
<code>X Y</code>	Matches X or Y.
<code>[char-set]</code>	Matches any one of the characters specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression <code>[A-Z]</code> matches any uppercase letter. '\ ' may be used inside the square brackets to define literal characters or define ASCII characters. For example <code>"\"</code> specifies a literal dash character. The expression <code>[\0-\27]</code> matches ASCII character codes 0..27 . The expression <code>[]</code> matches no characters. In UNIX regular expressions <code>[]</code> matches a right bracket. In both syntaxes, the expression <code>[]</code> matches a right bracket. The expression <code>[\^]</code> matches a '^' character in both syntaxes.
<code>[~char-set]</code>	Matches any character not specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression <code>[~A-Z]</code> matches all characters except uppercase letters.
<code>^[char-set]</code>	Same as <code>[~char-set]</code> above.
<code>[char-set1 - [char-set2]]</code>	Character set subtraction. Matches all characters in <i>char-set1</i> except the characters in <i>char-set2</i> . For example, <code>"[a-z-[qw]]"</code> matches all English lowercase letters except 'q' and 'w'. <code>"[p{L}-[qw]]"</code> matches all Unicode lower case letters except 'q' and 'w'.
<code>[char-set1 & [char-set2]]</code>	Character set intersection. Matches all characters in <i>char-set1</i> that are also in <i>char-set2</i> . For example, <code>"[\x{0}-\x{7f}&[p{L}]]"</code> matches all letters between 0 and 127.
<code>\x{hhh}</code>	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhh</i> .
<code>\p{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeCategorySpec</i> . Where <i>UnicodeCategorySpec</i> uses the standard general categories specified by the Unicode consortium. For example, <code>"[p{L}]"</code> matches all letters. <code>"[p{Lu}]"</code> matches all uppercase letters. See Unicode Category Specifications for Regular Expressions on page 118.
<code>\P{UnicodeCategorySpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeCategorySpec</i> . For example, <code>"[P{L}]"</code> matches all characters that are not letters. This is equivalent to <code>"[^\p{L}]"</code> . <code>"[P{Lu}]"</code> matches all characters that are not uppercase letters. See Unicode Category Specifications for Regular Expressions on page 118.
<code>\p{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeIsBlockSpec</i> . Where <i>UnicodeIsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, <code>"[p{isGreek}]"</code> matches

Unicode characters in the Greek block. See **Unicode Character Blocks for Regular Expressions** on page 119.

`\P{UnicodeIsBlockSpec}`

(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** on page 119.

`\xhh` Matches hexadecimal character *hh* where $0 \leq hh \leq 0xff$.

`\ddd` Matches decimal character *ddd* where $0 \leq ddd \leq 255$.

`\gd` 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.

`\c` Specifies cursor position if match is found. If the expression `xyz\c` is found the cursor is placed after the *z*.

`\n` Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file. Use `"\d10"` if you want to match a 10 character.

`\r` Matches carriage return.

`\t` Matches tab character.

`\b` Matches backspace character.

`\f` Matches form feed character.

`\om` 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.

WARNING: Test your regular expression on a very small file before using your regular expression on a large file.

This option may cause the editor to use a lot of memory.

`\ol` 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.

`\char` Declares character after slash to be literal. For example, `\:` represents the colon character.

`:char` Matches predefined expression corresponding to *char*.
The predefined expressions are:

<code>:a</code>	<code>[A-Za-z0-9]</code>	Matches an alphanumeric character
<code>:b</code>	<code>([\t]#)</code>	Matches blanks
<code>:c</code>	<code>[A-Za-z]</code>	Matches an alphabetic character
<code>:d</code>	<code>[0-9]</code>	Matches a digit
<code>:f</code>	<code>([~\[]:\ <> =+;, \t"]#)</code>	Windows: Matches a filename part
<code>:f</code>	<code>([~/ \t"]#)</code>	UNIX: Matches a filename part.
<code>:h</code>	<code>([0-9A-Fa-f]#)</code>	Matches a hex number
<code>:i</code>	<code>([0-9]#)</code>	Matches an integer
<code>:n</code>	<code>((([0-9]#(.[0-9]#)).[0-9]#)([Ee](\+ -)[0-9]#))</code>	Matches a floating number
<code>:p</code>	<code>((([A-Za-z]\:)(\ /)):(f(\ /))@:f)</code>	Windows: Matches a path
<code>:p</code>	<code>((/)(:f(/))@:f)</code>	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

NOTE: The :f and :p predefined expressions are not intended to support all operating system file names. Instead, they are intended to be useful in practical cases. The space character is not in :f because it is more typically a file name separator.

The precedence of operators, from highest to lowest, is as follows:

- +, #, *, @, , , : , * (These operators have the same precedence.)
- concatenation
- |

Sample Regular Expressions:

^defproc	Matches lines that begin with the word defproc.
^definit\$	Matches lines that only contain the word definit.
^\\:name	Matches lines that begin with the string ":name". Notice that the backslash must prefix the special character ':'.
[t]	Matches tab and space characters.
[932]	Matches tab and space characters.
[x9x20]	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.

Brief Regular Expressions

Brief regular expressions are defined as follows:

%	Matches beginning of line.
<	Matches beginning of line.
\$	Matches end of line.
>	Matches end of line.
?	Matches any character except newline.
*	Minimal match of zero or more of any character except newline. This is the same as "?@".
X+	Minimal match of one or more occurrences of X. See " Minimal Versus Maximal Matching " on page 125 for more information.
X\.*	Maximal match of zero or more of any character except newline. This is the same as "?\:@".
X\:@	Maximal match of zero or more occurrences of X.
X\;+	Maximal match of one or more occurrences of X.

<code>X\:<i>n1</i></code>	Matches exactly <i>n1</i> occurrences of X. Use {} to avoid ambiguous expressions. For example "a:9{}1" searches for 9 a's followed by a 1.
<code>X\:<i>n1</i>,</code>	Maximal match of at least <i>n1</i> occurrences of X.
<code>X\;:<i>n2</i></code>	Maximal match of at least 0 occurrences but not more than <i>n2</i> occurrences of X.
<code>X\:<i>n1</i>,<i>n2</i></code>	Maximal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
<code>X\:<i>n1</i>?</code>	Match exactly <i>n1</i> occurrences of X.
<code>X\;<i>n1</i>,?</code>	Minimal match of at least <i>n1</i> occurrences of X.
<code>X\;:<i>n2</i>?</code>	Minimal match of at least 0 occurrences but not more than <i>n2</i> occurrences of X.
<code>X\;<i>n1</i>,<i>n2</i>?</code>	Minimal match of at least <i>n1</i> occurrences but not more than <i>n2</i> occurrences of X.
<code>\(X\)</code>	Matches sub-expression X but does not define a tagged expression.
<code>{X}</code>	Matches sub-expression X and specifies a new tagged expression. See "Tagged Search Expressions" on page 123 for more information.
<code>{@<i>dX</i>}</code>	Matches sub-expression X and specifies to use tagged expression number <i>d</i> where $0 \leq d \leq 9$. No more tagged expressions are defined by the sub-expression syntax "{X}" once this sub-expression syntax is used. This is the best way to make sure you have enough tagged expressions.
<code>X Y</code>	Matches X or Y.
<code>[<i>char-set</i>]</code>	Matches any one of the characters specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression [A-Z] matches any uppercase letter. '\' can be used inside the square brackets to define literal characters or define ASCII characters. For example "\-" specifies a literal dash character. The expression [\0-27] matches ASCII character codes 0..27. The expression [] matches a right bracket. In SlickEdit regular expressions [] matches no characters. In both syntaxes, the expression [\\] matches a right bracket.
<code>[~<i>char-set</i>]</code>	Matches any character not specified by <i>char-set</i> . A '-' character may be used to specify ranges. The expression [~A-Z] matches all characters except uppercase letters. The expression [~] matches any character except newline.
<code>[<i>char-set1</i> - [<i>char-set2</i>]]</code>	Character set subtraction. Matches all characters in <i>char-set1</i> except the characters in <i>char-set2</i> . For example, "[a-z-[qw]]" matches all English lower case letters except 'q' and 'w'. "[\p{L}-[qw]]" matches all Unicode lower case letters except 'q' and 'w'.
<code>[<i>char-set1</i> & [<i>char-set2</i>]]</code>	Character set intersection. Matches all characters in <i>char-set1</i> that are also in <i>char-set2</i> . For example, "[\x{0}-\x{7f}&[\p{L}]]" matches all letters between 0 and 127.
<code>\x{<i>hhh</i>}</code>	Matches up to 31-bit Unicode hexadecimal character specified by <i>hhh</i> .
<code>\p{<i>UnicodeCategorySpec</i>}</code>	(Only valid in character set) Matches characters in <i>UnicodeCategorySpec</i> . Where <i>UnicodeCategorySpec</i> uses the standard general categories specified by the Unicode consortium. For example, "[\p{L}]" matches all letters. "[\p{Lu}]" matches all uppercase letters. See Unicode Category Specifications for Regular Expressions on page 118.
<code>\P{<i>UnicodeCategorySpec</i>}</code>	

	(Only valid in character set) Matches characters not in <i>UnicodeCategorySpec</i> . For example, “[P{L}]” matches all characters that are not letters. This is equivalent to “[^p{L}]”. “[P{Lu}]” matches all characters that are not uppercase letters. See Unicode Category Specifications for Regular Expressions on page 118.									
<code>\p{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters in <i>UnicodeIsBlockSpec</i> . Where <i>UnicodeIsBlockSpec</i> one of the standard character blocks specified by the Unicode consortium. For example, “[p{isGreek}]” matches Unicode characters in the Greek block. See Unicode Character Blocks for Regular Expressions on page 119.									
<code>\P{UnicodeIsBlockSpec}</code>	(Only valid in character set) Matches characters not in <i>UnicodeIsBlockSpec</i> . For example, “[P{isGreek}]” matches all characters that are not in the Unicode Greek block. This is equivalent to “[^p{isGreek}]”. See Unicode Character Blocks for Regular Expressions on page 119.									
<code>\xhh</code>	Matches hexadecimal character <i>hh</i> where $0 \leq hh \leq 0xff$.									
<code>\ddd</code>	Matches decimal character <i>ddd</i> where $0 \leq ddd \leq 255$.									
<code>\d</code>	Defines a back reference to tagged expression number <i>d</i> . For example, “[abc]def0” matches the string “abcdefabc”. If the tagged expression has not been set, the search fails.									
<code>\c</code>	Specifies cursor position if match is found. If the expression <i>xyz</i> \c is found the cursor is placed after the <i>z</i> .									
<code>\n</code>	Matches newline character sequence. Useful for matching multi-line search strings. What this matches depends on whether the buffer is a DOS (ASCII 13,10 or just ASCII 10), UNIX (ASCII 10), Macintosh (ASCII 13), or user defined ASCII file. Use “\d10” if you want to match a 10 character.									
<code>\r</code>	Matches carriage return.									
<code>\t</code>	Matches tab character.									
<code>\b</code>	Matches backspace character.									
<code>\f</code>	Matches form feed character.									
<code>\om</code>	Turns on multi-line matching. This enhances the match character set, or match any character primitives to support matching end of line characters. For example, “[om? ” matches the rest of the buffer. WARNING: Test your regular expression on a very small file before using your regular expression on a large file. This option may cause the editor to use a lot of memory.									
<code>\ol</code>	Turns off multi-line matching (default). You can still use “\n” to create regular expressions which match one or more lines. However, expressions like “[?\b ” will not match multiple lines. This is much safer and usually faster than using the “\om” option.									
<code>\char</code>	Declares character after slash to be literal. For example, “[* ” represents the star character.									
<code>\:char</code>	Matches predefined expression corresponding to <i>char</i> . The predefined expressions are: <table><tr><td><code>\:a</code></td><td>[A-Za-z0-9]</td><td>Matches an alphanumeric character</td></tr><tr><td><code>\:b</code></td><td>[\t\#]</td><td>Matches blanks</td></tr><tr><td><code>\:c</code></td><td>[A-Za-z]</td><td>Matches an alphabetic character</td></tr></table>	<code>\:a</code>	[A-Za-z0-9]	Matches an alphanumeric character	<code>\:b</code>	[\t\#]	Matches blanks	<code>\:c</code>	[A-Za-z]	Matches an alphabetic character
<code>\:a</code>	[A-Za-z0-9]	Matches an alphanumeric character								
<code>\:b</code>	[\t\#]	Matches blanks								
<code>\:c</code>	[A-Za-z]	Matches an alphabetic character								

<code>\:d</code>	<code>[0-9]</code>	Matches a digit
<code>\:f</code>	<code>([~\[\]\:\/<> =+;, \t'"]#)</code>	Windows: Matches a filename part
<code>\:f</code>	<code>([~/\t'"]#)</code>	UNIX: Matches a filename part.
<code>\:h</code>	<code>([0-9A-Fa-f]#)</code>	Matches a hex number
<code>\:i</code>	<code>([0-9]#)</code>	Matches an integer
<code>\:n</code>	<code>(([0-9]#([0-9]#) [0-9]#)([Ee](\+ -)[0-9]#))</code>	Matches a floating number
<code>\:p</code>	<code>(([A-Za-z\:])(\\ /)(:f(\\ /))@:f)</code>	Windows: Matches a path
<code>\:p</code>	<code>((/)(:f(/))@:f)</code>	UNIX: Matches a path
<code>\:q</code>	<code>("["~\"]@\"["~']@')</code>	Matches a quoted string
<code>\:v</code>	<code>([A-Za-z_\$][A-Za-z0-9_\$]@)</code>	Matches a C variable
<code>\:w</code>	<code>([A-Za-z]#)</code>	Matches a word

NOTE: The `\:f` and `\:p` predefined expressions are not intended to support all operating system file names. Instead they are intended to be useful in practical cases. The space character is not in `\:f` because it is more typically a file name separator.

Sample Regular Expressions:

<code><defproc</code>	Matches lines that begin with the word <code>defproc</code> .
<code><definit></code>	Matches lines that only contain the word <code>definit</code> .
<code><*name</code>	Matches lines that begin with the string <code>"*name"</code> . Notice that the backslash must prefix the special character <code>*</code> .
<code>[\t]</code>	Matches tab and space characters.
<code>[\d9\d32]</code>	Matches tab and space characters.
<code>[\x9\x20]</code>	Matches tab and space characters.
<code>p?t</code>	Matches any three letter string starting with the letter 'p' and ending with the letter 't'. Two possible matches are "pot" and "pat".
<code>s*t</code>	Matches the letter 's' followed by any number of characters followed by the nearest letter 't'. Two possible matches are "seat" and "st".
<code>{for}{while}</code>	Matches the strings "for" or "while".
<code>^\:p</code>	Matches lines beginning with a file name.
<code>xy+z</code>	Matches x followed by one or more occurrences of y followed by z.

Unicode Category Specifications for Regular Expressions

Visual SlickEdit supports the Unicode consortium standard regular expression categories. The syntax for specifying categories is:

`\p{MainCategoryLetter Subcategories}`

which matches the categories specified, and:

`\P{MainCategoryLetter Subcategories}`

which matches all characters not in the categories specified.

The `\p` and `\P` notations can only be used inside a character set specification. *MainCategoryLetter* can be 'L', 'M', 'N', 'P', 'S', 'Z', or 'C'. The valid *Subcategories* depend on the *MainCategoryLetter* specified. If no *Subcategories* are specified, all are assumed. For example:

“`\p{L}`” matches all Unicode letters
“`\p{Lu}`” matches all uppercase and lowercase letters
“`\P{L}`” matches all characters that are not letters

The following table lists the valid subcategories for a specific main category. These character tables were generated using the “**UnicodeData-3.1.0.txt**” found on the Unicode consortiums Web site.

Subcategories	Description
Lu	Letter, Uppercase
Ll	Letter, Lowercase
Lt	Letter, Titlecase
Lo	Letter, Other
Mn	Mark, Non-Spacing
Mc	Mark, Spacing Combining
Me	Mark, Enclosing
Nd	Number, Decimal Digit
Nl	Number, Letter
No	Number, Other
Pc	Punctuation, Connector
Pd	Punctuation, Dash
Ps	Punctuation, Open
Pe	Punctuation, Close
Pi	Punctuation, Initial quote (may behave like Ps or Pe depending on usage)
Pf	Punctuation, Final quote (may behave like Ps or Pe depending on usage)
Po	Punctuation, Other
Sm	Symbol, Math
Sc	Symbol, Currency
Sk	Symbol, Modifier

So	Symbol, Other
Zs	Separator, Space
Zl	Separator, Line
Zp	Separator, Paragraph
Cc	Other, Control
Cf	Other, Format
Cs	Other, Surrogate
Co	Other, Private Use
Cn	Other, Not Assigned (no characters in the file have this property)

Unicode Character Blocks for Regular Expressions

Visual SlickEdit supports the Unicode consortium standard regular expression block categories. The syntax for specifying a character block is:

```
\p{IsBlockName}
```

which matches the characters in the block specified, and:

```
\P{IsBlockName}
```

which matches all characters not in the block specified.

The \p and \P notations may only be used inside a character set specification. For example, “[\p{isBasicLatin}]” matches all characters in the Greek block. “[\P{isBasicLatin}]” matches all characters that are not in the Greek block.

The following table lists the non-standard valid character block names. These character tables were generated from XML standards found at the World Wide Web Consortium’s Web site.

Block Name	Description
XMLNameStartChar	All characters that are valid for the start of an XML tag name.
XMLNameChar	All characters that are valid in an XML tag name.

The following table lists the valid character block names. These characters tables were generated using the “**blocks.txt**” found on the Unicode consortium’s web site.

Range	Block Name
0000..007F	BasicLatin
0080..00FF	Latin-1Supplement
0100..017F	LatinExtended-A
0180..024F	LatinExtended-B
0250..02AF	IPAExtensions
02B0..02FF	SpacingModifierLetters
0300..036F	CombiningDiacriticalMarks
0370..03FF	Greek
0400..04FF	Cyrillic
0530..058F	Armenian
0590..05FF	Hebrew

0600..06FF	Arabic
0700..074F	Syriac
0780..07BF	Thaana
0900..097F	Devanagari
0980..09FF	Bengali
0A00..0A7F	Gurmukhi
0A80..0AFF	Gujarati
0B00..0B7F	Oriya
0B80..0BFF	Tamil
0C00..0C7F	Telugu
0C80..0CFF	Kannada
0D00..0D7F	Malayalam
0D80..0DFF	Sinhala
0E00..0E7F	Thai
0E80..0EFF	Lao
0F00..0FFF	Tibetan
1000..109F	Myanmar
10A0..10FF	Georgian
1100..11FF	HangulJamo
1200..137F	Ethiopic
13A0..13FF	Cherokee
1400..167F	UnifiedCanadianAboriginalSyllabics
1680..169F	Ogham
16A0..16FF	Runic
1780..17FF	Khmer
1800..18AF	Mongolian
1E00..1EFF	LatinExtendedAdditional
1F00..1FFF	GreekExtended
2000..206F	GeneralPunctuation
2070..209F	SuperscriptsandSubscripts
20A0..20CF	CurrencySymbols
20D0..20FF	CombiningMarksforSymbols
2100..214F	LetterlikeSymbols
2150..218F	NumberForms
2190..21FF	Arrows
2200..22FF	MathematicalOperators
2300..23FF	MiscellaneousTechnical
2400..243F	ControlPictures
2440..245F	OpticalCharacterRecognition
2460..24FF	EnclosedAlphanumerics
2500..257F	BoxDrawing
2580..259F	BlockElements
25A0..25FF	GeometricShapes
2600..26FF	MiscellaneousSymbols
2700..27BF	Dingbats
2800..28FF	BraillePatterns
2E80..2EFF	CJKRadicalsSupplement
2F00..2FDF	KangxiRadicals
2FF0..2FFF	IdeographicDescriptionCharacters

3000..303F	CJKSymbolsandPunctuation
3040..309F	Hiragana
30A0..30FF	Katakana
3100..312F	Bopomofo
3130..318F	HangulCompatibilityJamo
3190..319F	Kanbun
31A0..31BF	BopomofoExtended
3200..32FF	EnclosedCJKLettersandMonths
3300..33FF	CJKCompatibility
3400..4DB5	CJKUnifiedIdeographsExtensionA
4E00..9FFF	CJKUnifiedIdeographs
A000..A48F	YiSyllables
A490..A4CF	YiRadicals
AC00..D7A3	HangulSyllables
D800..DB7F	HighSurrogates
DB80..DBFF	HighPrivateUseSurrogates
DC00..DFFF	LowSurrogates
E000..F8FF	PrivateUse
F900..FAFF	CJKCompatibilityIdeographs
FB00..FB4F	AlphabeticPresentationForms
FB50..FDFD	ArabicPresentationForms-A
FE20..FE2F	CombiningHalfMarks
FE30..FE4F	CJKCompatibilityForms
FE50..FE6F	SmallFormVariants
FE70..FEFE	ArabicPresentationForms-B
FEFF..FEFF	Specials
FF00..FFEF	HalfwidthandFullwidthForms
FFF0..FFFD	Specials
10300..1032F	OldItalic
10330..1034F	Gothic
10400..1044F	Deseret
1D000..1D0FF	ByzantineMusicalSymbols
1D100..1D1FF	MusicalSymbols
1D400..1D7FF	MathematicalAlphanumericSymbols
20000..2A6D6	CJKUnifiedIdeographsExtensionB
2F800..2FA1F	CJKCompatibilityIdeographsSupplement
E0000..E007F	Tags

Search and Replace with Regular Expressions

UNIX regular expressions: When regular expressions are turned on for a search and replace command, the backslash character ('\') has special meaning in the replace string. A '\' in the replace string has the same meaning as in the search string except that "\c" and "\:char" are not supported. See "UNIX Regular Expressions" on page 108 for a list of '\' options. See "Tagged Search Expressions" below for information on specifying tagged expressions in the replace string.

SlickEdit regular expressions: When regular expressions are turned on for a search and replace command, the characters '#' and '\' have special meaning in the replace string. A '\' in the replace string has the same meaning as in the search string except that "\c", "\:char", and "\gd" are not supported. See "SlickEdit Regular Expressions" on page 111 for a list of '\' options. See "Tagged Search Expressions" below for information on specifying tagged expressions in the replace string using the '#' character.

Brief regular expressions: When regular expressions are turned on for a search and replace command, the backslash character ('\') has special meaning in the replace string. A '\' in the replace string has the same meaning as in the search string except that "\c" and "\:char" are not supported. See "Brief Regular Expressions" on page 114 for a list of '\' options. See "Tagged Search Expressions" below for information on specifying tagged expressions in the replace string.

Replace dialog box Regular Expression Examples:

Search For: **hat\$**
Replace With: **cat**

Search for occurrences of the string "hat" which occur at the end of a line and replace it with "cat".

UNIX/SlickEdit Regular Expression

Search For: **^\n**
Replace With:

Brief Regular Expression

Search For: **<\n**
Replace With:

Delete blank lines.

UNIX/SlickEdit Regular Expression

Search For: **^\n\n**
Replace With: **\n**

Brief Regular Expression

Search For: **<\n\n**
Replace With: **\n**

Replace occurrences of two consecutive blank lines with one.

UNIX Regular Expression

Search for: `^a+$`

Replace with: `\d12`

SlickEdit Regular Expression

Search for: `^a+$`

Replace with: `\12`

Brief Regular Expression

Search for: `<a+$`

Replace with: `\d12`

Search for lines containing only one or more “a’s” and replace them with a form feed character.

Tagged Search Expressions

When you use regular expressions to search for a string, you will often want the replace string to depend on what was found. You can use tagged search expressions to insert parts of what is found into the replace string.

UNIX Regular Expressions: Use parentheses “()” to denote a tagged expression in the search string. The replace string specifies tagged expressions with a “\” followed by a tag group number 1-9. Count the left parenthesis “(” in the search string to determine a tagged expression number. The first tagged expression is “\1” and the last is “\0”.

SlickEdit Regular Expressions: Use curly braces “{}” to denote a tagged expression in the search string. The replace string specifies tagged expressions with a “#” followed by a tagged expression number 0-9. Count the left braces “{” in the search string to determine a tagged expression number. The first tagged expression is “#0”.

Brief Regular Expressions: Use curly braces “{}” to denote a tagged expression in the search string. The replace string specifies tagged expressions with a “\” followed by a tagged expression number 0-9. Count the left braces “{” in the search string to determine a tagged expression number. The first tagged expression is “\0”.

Regular Expression Examples:

UNIX Regular Expression

Search for: **(if|while)**

Replace with: **x\1y\2**

SlickEdit Regular Expression

Search for: **{if|while}**

Replace with: **x#0y#1**

Brief Regular Expression

Search for: **{{if}|while}}**

Replace with: **x\0y\1**

Replace occurrences of "if" and "while" with "xify" and "xwhiley". Unmatched groups are null. NOTE: the \2 (SlickEdit syntax #1,Brief syntax \1) is replaced with null.

UNIX Regular Expression

Search for: **^(.*?),(.*)\$**

Replace with: **\2,\1**

SlickEdit Regular Expression

Search for: **^{?*,}{?*\$}**

Replace with: **#1,#0**

Brief Regular Expression

Search for: **^{*},{*}\$**

Replace with: **\1,\0**

Reverse text on lines containing a comma. Lines with "abc,def" will be changed to "def,abc". Notice that the UNIX Regular Expression search string uses a minimal matching operator "*?" so that the comma actually matches the first comma in the line and not the last.

Minimal Versus Maximal Matching

If you are using tagged expressions or you are using regular expressions to perform a search and replace, you need to understand the difference between the minimal and maximal operators. Let's say you have a line of text which contains the following DOS file name:

```
\path1\path2\path3\name.ext
```

The regular expression "`^\\..*?\\`" (SlickEdit syntax "`^\\..*?\\`", Brief syntax "`<\\..*?\\`") will match the string "`\path1\`". The regular expression "`^\\..*\\`" (SlickEdit syntax "`^\\..*@\\`", Brief syntax "`<\\..*@\\`"), which uses the maximal operator, matches the string "`\path\path2\path3\`". As a rule of thumb, you will usually want to use the minimal matching operators "`*?"`" (SlickEdit syntax "`*`", Brief syntax "`@`") and "`+?"`" (SlickEdit/Brief syntax "`+`") after a less specific regular expression such as "`'`" (SlickEdit/Brief syntax "`?`").

You will usually want to use the maximal matching operators after a regular expression which matches something more specific. For example, to search for a string of digits and prefix each string of digits with a '\$' you would specify the following in the Replace dialog box:

Regular Expression Example:

UNIX Regular Expression

Search for: `([0-9]+)`

Replace with: `$1`

SlickEdit Regular Expression

Search for: `{[0-9]#}`

Replace with: `$#0`

Brief Regular Expression

Search for: `{[0-9]\: +}`

Replace with: `$\0`

If the minimal matching "`+?"`" (SlickEdit/Brief syntax "`+`") operator was used in the "search for" string instead of the maximal matching "`+`" (SlickEdit syntax "`#`", Brief syntax "`\: @`"), the above search and replace would prefix each digit in the file with a '\$' character, which is probably not what you want.

Bookmarks

Using bookmarks allows you to save your edit position, move around to other files and later return to your edit position. Visual SlickEdit provides stack-oriented bookmarks which can be pushed, popped and named. Bookmark names must consist of alphanumeric characters and are limited to 100 characters in length.

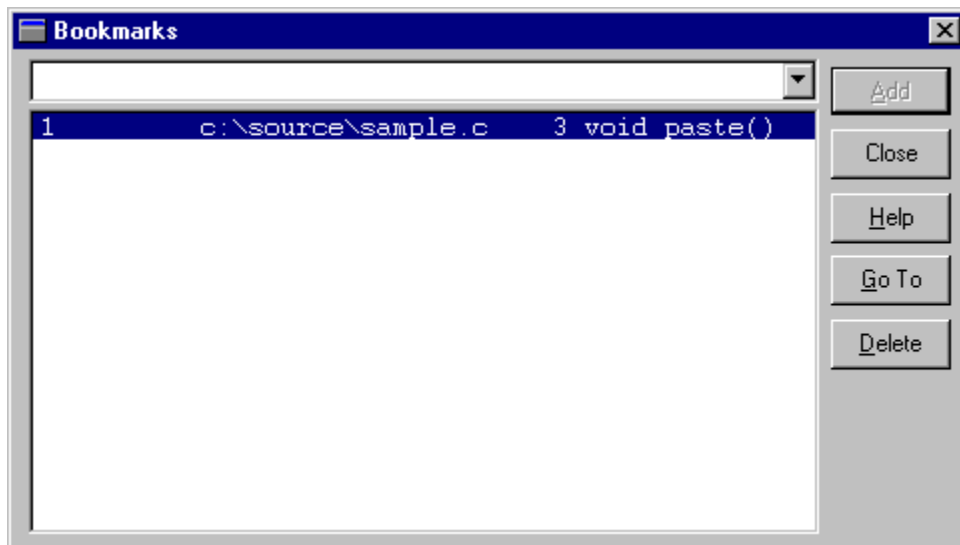
Setting a Bookmark

You can set a bookmark quickly with one of the keys CTRL+1, CTRL+2..CTRL+0. These keys are use to set a bookmark with names 1, 2..0 respectively. If the bookmark is already in use, the bookmarks are displayed in a menu with the cursor on the bookmark ID that is being set, allowing you to replace it or select a different one.

Alternatively, you can use the Bookmarks dialog box to specify your own name for a bookmark. The Bookmarks dialog box allows you to delete the currently selected bookmark in the list box.

If auto restore is on (default is on), bookmarks are saved when exiting Visual SlickEdit and restored for use in your next edit session.

The following is a picture of the Bookmarks dialog box ("Search", " Bookmarks..." or **set_bookmark** command):



The lower list box shows the bookmark name, buffer name, line number, and line of text where the bookmark was set.

set_bookmark command

If you like using the command line, use the **sb** or **set_bookmark** command to set a bookmark at the cursor location in the active buffer. The syntax for the command is:

```
sb [-r] [bookmark_name]
```

If a *bookmark_name* is not given on the command line, a list of the current bookmarks is displayed for reference and you are prompted to enter a *bookmark_name* or select a bookmark to replace.

If the `-r` (replace) option is used, the bookmark is replaced without prompting.

An automark function, which sets the next available bookmark, is enabled by entering a '#' or '##' as the bookmark name. For example:

sb	Pops up the Set Bookmark dialog box
sb 5	Sets a bookmark with an id of 5 at the cursor
sb ##	Sets a bookmark using the next 2-digit number available as the bookmark name

Go to Bookmark

To go to a bookmark, use the Bookmarks dialog box (CTRL+SHIFT+J or "Search", "Bookmarks..." or **set_bookmark** command).

goto_bookmark command

If you like using the command line, the **gb** or **goto_bookmark** command places the cursor at a specified bookmark.

gb [*id*]

If the *id* is not given or the *id* entered is not active, a menu with a list of the current bookmarks is displayed and you are prompted to select an *id*.

Examples:

gb	Displays the Go To Bookmark dialog box
gb 5	Goes to bookmark 5 if set or displays Go To Bookmark dialog box if not
gb 10	Goes to bookmark 10 if set or displays Go To Bookmark dialog box if not

You can switch to the next and previous bookmark by using the **next_bookmark** (**nb**) or **prev_bookmark** (**pb**) command.

Printing

This section describes the printing options available in Visual SlickEdit.

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...") to insert a formfeed or any other character. The menu item ("Tools", "Ascii Table") gives you access to an ASCII table of characters.

Print Setup Dialog Box

(**printer_setup** command)

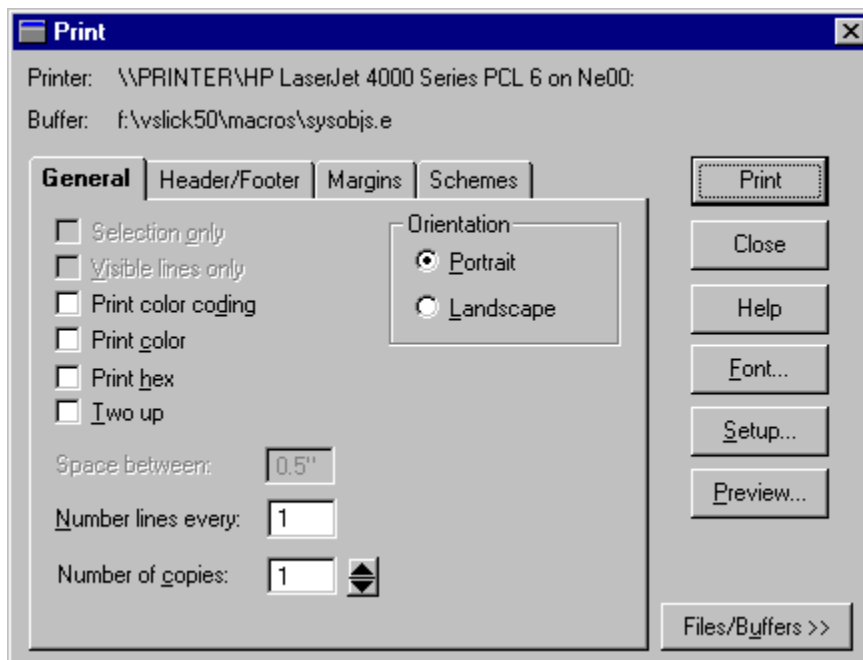
Windows: For Windows, the Print Setup dialog box ("File", "Print", press "Setup..." button) is built into the operating system and depends on the print driver you are using. This dialog box allows you to select the printer you want your documents printed to, landscape and portrait printing, output paper tray, and more.

UNIX: The UNIX version does not have a Print Setup dialog box.

Print Dialog Box

("File", "Print..." or **gui_print** command)

The Print dialog box ("File", "Print...") is used to print the current buffer or selection. Below is a picture of the Print dialog box:



This dialog box is not present in the UNIX version of Visual SlickEdit. See "Text Mode Print Dialog Box" on page 131.

General

Selection only

When on, only the selection is printed.

Visible lines only

When on, only visible lines are printed. This option allows you to print selective display.

Print color coding

When on, language-specific color coding is printed using the same font attributes (bold, italic, underline).

Print color

When on, language-specific color coding is printed using the same colors.

Print hex

When on, printed output looks like the hex display mode.

Two up

When on, 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

Windows only: 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 grayed unless the Two Up check box is checked.

Number lines every

When non-zero, lines at intervals of this value are printed with line numbers.

Number of copies

Specifies the number of copies to print.

Header/Footer**Left, Center, and Right header**

The left, center, and right header text boxes specify the title that appears at the top left, center, and right of each page respectively. When one of the escape sequences %f, %d, or %p, is found, it is replaced with the file name, date, or page number respectively.

Left, Center, and Right footer

The left, center, and right footer text boxes specify the title that appears at the bottom left, center, and right of each page respectively. When one of the escape sequences %f, %d, or %p, is found, it is replaced with the file name, date, or page number respectively.

Margins

Windows: Top, bottom, left and right margins, in inches, between the outer edge of the paper and the printed text. Specify 0 for all margins to print the maximum amount of text.

After header

This is the space in inches between the header and the first line on the page.

Before footer

This is the space in inches between the last line on a page and the footer.

Schemes

Allows you to define or select a printing scheme. For example, you might want to define a scheme which prints two up, landscape, and uses a small font. System schemes are stored in "print.ini". User-defined schemes or modified system schemes are stored in "uprint.ini".

Font

Allows you to select the font used by the printer.

Setup

Invokes the Print Setup dialog box.

Files/Buffers

Expands or condenses the dialog box. When the dialog box is expanded, you can specify to print multiple files and/or buffers.

Subdirectories

Affects whether searching for file specified in the Files text box traverses subdirectories.

Files

Specifies files on disk to print. If the file has already been opened (i.e. it is a buffer), the buffer is printed.

Buffers

Specifies one or more buffers to print. Press CTRL+Click to select multiple buffers.

Search all project files

When checked, all files in the current project are printed in addition to other files/buffers specified.

CAUTION: you might end up printing a file twice if you specify other files/buffers in addition to your project files.

Search all workspace files

When checked, all files in the current workspace are printed in addition to other files/buffers specified.

CAUTION: you might end up printing a file twice if you specify other files/buffers in addition to your workspace files.

Invert

Selects all buffers in the Buffers list box that are currently not selected and deselect all files which are selected.

Clear

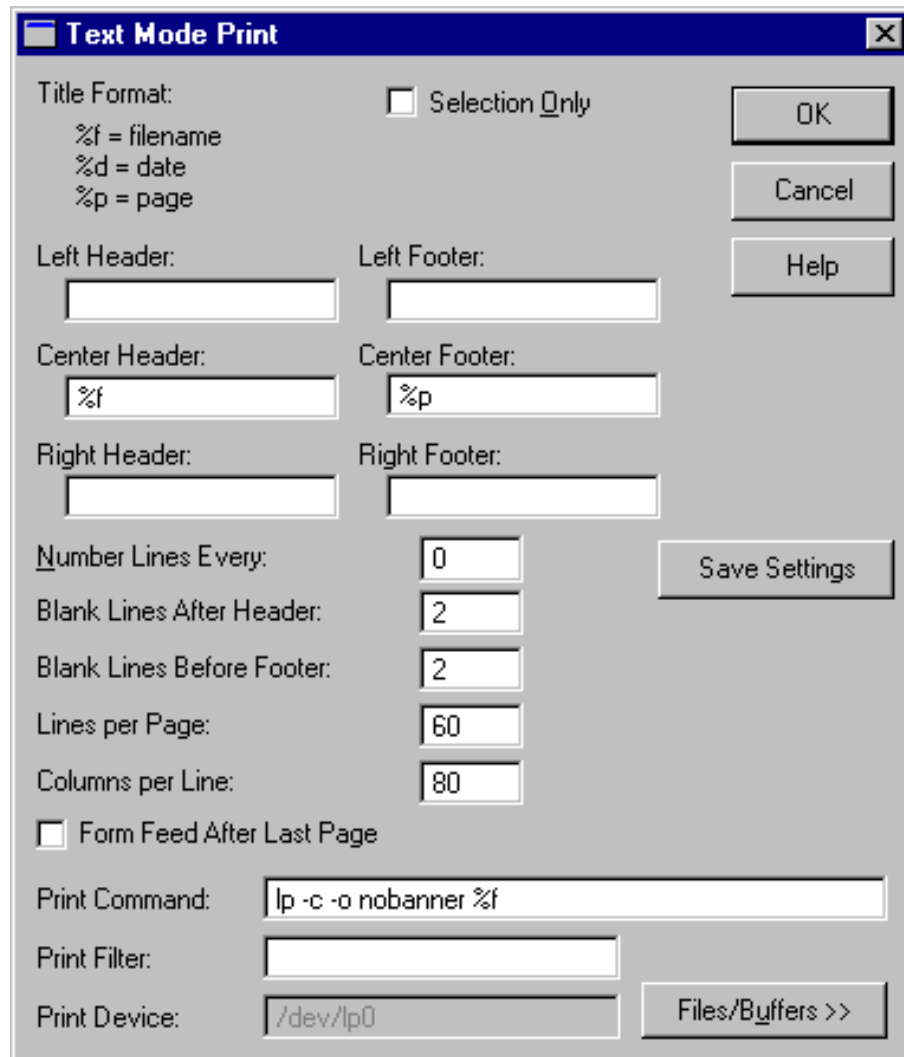
Deselects all buffers in the buffer list.

Text Mode Print Dialog Box

("File", "Print..." or **gui_print** command)

NOTE: The Text Mode Print dialog box is only available in UNIX versions of Visual SlickEdit. Graphical printing is not supported by the UNIX versions.

The Text Mode Print dialog box ("File", "Print...") is used to print the current buffer or selection. Below is a picture of the Text Mode Print dialog box:



Selection Only

When on, only the selection is printed.

Left, Center, and Right Header

The left, center, and right header text boxes specify the title that appears at the top left, center, and right of each page respectively. When one of the escape sequences %f, %d, or %p, is found, it is replaced with the file name, date, or page number respectively.

Left, Center, and Right Footer

The left, center, and right footer text boxes specify the title that appears at the bottom left, center, and right of each page respectively. When one of the escape sequences %f, %d, or %p, is found, it is replaced with the file name, date, or page number respectively.

Number Lines Every

When non-zero, lines at intervals of this value are printed with line numbers.

Blank Lines After Header

Specifies the number of blank lines between the header and the text on the page. This value is ignored if all header fields are blank.

Blank Lines Before Footer

Specifies the number of blank lines between the last line of text on the page and the footer.

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.

Form Feed After Last Page

Determines whether the built-in print text formatting prints a form feed after the last page.

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.

Print Filter

Specifies an external program to format text before it is printed and overrides the built-in print text formatting. The program specified here must accept input from standard in and writes its output to standard out. A typical print filter command is "**pr -1 -w76 -o0 -l50**".

Print Device

Specifies a printer device file name to write the formatted text to. This text box is ignored if a Print Command has been specified. The default print device is "**/dev/lp0**".

Save Settings

Allows you to save all the print options specified in this dialog box for your next edit session.

Improving the Readability of Your Code

Readability Overview

With Visual SlickEdit's many formatting options, support for color coding and code beautifiers, you can greatly improve the readability of your code.

Formatting Options

Margins and Tabs

Each buffer has its own margins, tabs, justification style, indent style, and word wrap style. When a buffer is loaded, these configuration options are set to the default values for that file extension or mode. You can access the default values for each file extension by ("Tools", "Configuration", "File Extension Setup...", select the Word Wrap tab). The Document drop-down menu is used to temporarily change these option values. You can have these default options set based on a format specification line at the first line of a file, described in "[-Format Line](#)" on page 136.

Tabs Dialog Box

("Document", "Tabs..." or **gui_tabs** command)

To change the default tab stops for specific file extensions, use the [Extension Options dialog box](#) ("Tools", "Configuration", "File Extension Setup...", select the "Indent" tab, select an extension, "Tabs"). To set tabs for the current buffer only, use the [Tabs dialog box](#) ("Document", "Tabs..."). You can set tabs in increments or at specific column positions. For example, to specify an increment of 3, enter +3 in the text box. To specify columns, you could enter **1 8 27 44** to specify tab stops that have absolute locations.

By default, the TAB key inserts enough spaces to move the text to the next tab stop. The SHIFT+TAB key combination deletes enough spaces to move the text to the previous tab stop. See "[Redefine Common Keys dialog box](#)" on page 227 for information on other TAB and SHIFT+TAB key bindings. Regardless of the TAB key binding, if "Indent with tabs" is on, a physical tab character is inserted (See "[Indent with Tabs](#)" on page 135).

tabs Command

If you prefer using the command line, the **tabs** command can be used to temporarily change the tab settings for the current buffer. The arguments to the **tabs** command are:

[+] t1 [+] t2 [+] t3 ... [+] tN

The '+' symbol is used to set tabs according to an increment value. Tab stops continue past the last tab stop by repeating the difference of the last two tab stop values.

Examples:

tabs 1 7 13 16	Creates tab stops 1 7 13 16 19 22 25 ...
tabs +3	Creates tab stops 1 4 7 ...

tabs 1 4	Creates tab stops 1 4 7 ...
tabs 1 49 +3	Creates tab stops 1 4 7 ... 43 46 49. This creates tab stops in increments of 3 between 1 and 49. The example below creates tab stops which increment by 3 starting at 49.
tabs 1 49 52	Creates tab stops 1 49 52 55 58 ...
tabs 1 49 +3 53 70 +4	Creates tab stops 1 4 7 ... 43 46 49 53 57 61 65 69
tabs 10 +3	Creates tab stops 10 13 16 ...

Indent with Tabs

Normally, when auto indent is turned on and you press ENTER, Visual SlickEdit will align the cursor under the first non-blank character of the previous line. For some column-oriented languages, such as Assembler, you might want to indent using tab characters. Indent with tabs is off by default. You can change the indent with tabs option for a particular file extension using the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup...", select the "Indent" tab, select an extension, "Indent with tabs"). To toggle indenting with tabs, use the Indent With Tabs menu item ("Document", "Indent with Tabs").

Margins Dialog Box

("Document", "Margins..." or **gui_margins** command)

Visual SlickEdit allows you to set the left, right, and new paragraph margins. To change the default margins for specific file extensions, use the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup...", select the "Word Wrap" tab, select an extension, "Margins"). To temporarily change the margins for the current buffer only, use the Margins dialog box ("Document", "Margins...").

After you set the margins, characters you type will be word wrapped within the left and right margin settings unless word wrap style is turned off. To change the default word wrap style for specific file extensions, use the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup...", select the "Word Wrap" tab, select an extension, "Word Wrap"). You can use the ("Document", "Justify...") menu item to temporarily change the word wrap style for the current buffer. After typing characters into a paragraph, there might be extra blanks or the text might not be justified correctly. If so, select ("Document", "Format paragraph") to reformat the current paragraph. The current paragraph is defined by a group of non-blank lines separated by a blank line. Alternatively, you can specify a selected area of text and reformat it by invoking ("Document", "Format Selection").

margins Command

If you like using the command line, you can use the **ma** or **margins** command ("Document", "Margins...") to temporarily change the margins for the current buffer. The **ma** command is an abbreviation of the **margins** command. The arguments to the **ma** or **margins** command are:

margins [*left* [*right* [*new-paragraph*]]]

Specifying no parameters displays the current values. When the *left* margin is specified and the *new-paragraph* margin is not, the *new-paragraph* margin is set to the same value as the *left* margin.

Examples:

ma 5 70 10	Set left, right, new-paragraph margins to 5, 70, 10.
ma 5 70	Set left, right, new-paragraph margins to 5, 70, 5.
ma	Display current values.

Justification Style

The Justification dialog box ("Document", "Justify..." or **gui_justify** command) changes the justification style for the current buffer. This affects word wrap as you type and reflow paragraph commands. To change the justification style for specific file extensions, use the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup...", select the "Word Wrap" tab, select an extension, "Justify Style"). If you prefer using the command line, you can use the **justify** command to change the justification style for the current buffer.

Visual SlickEdit provides three justification styles:

Justification style	Word wrap effects
Left and Respace	Left justification with space character reformatting. One space is placed between words except after the punctuation characters ".?!" which get two spaces. If you only want one space after the punctuation characters ".?!", turn on "1 space after period".
Left	Left justification with respect for space characters between words. This setting requires the save options to be set such that trailing spaces are not stripped when a buffer is saved. See " Save tab " on page 234.
Justified	Full justification. Left and right edges of text will align exactly at margins. Full justification does not occur as you type. Use the reflow_paragraph command ("Document", "Format Paragraph") when you are done typing to get full justification for the current paragraph.

Format Line

As described at the beginning of this section, you can set the default word wrap and indent options for each file extension and access them with the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup..."). However, if you want files of the same language to have a different default, you should use a format line for those files. To create a format line, make the first line of text in your file contain "@SLICK". The @SLICK keyword can be inside a comment. When you save a buffer with a format line, the format line is updated. Also, any time a word processing option is changed, such as the margins, the

format line will be updated. The next time you open this file, the word processing options will be set according to the format line.

Color Coding

Visual SlickEdit allows you to combine current line coloring, modified line coloring, and language-specific coloring. By default, “Fundamental” mode colors the current line. Languages with specific support already defined (i.e. keywords, comments, etc.) use language-specific coloring. When modified line coloring is on, the left edge (the space between the left edge of the window and the text) displays a different color depending on whether the line was inserted or modified. To change the colors, use the **Color Settings dialog box** (“Tools”, “Configuration”, “Colors...”).

When current line color coding is on, you do not see language-specific color coding for the current line. If you are looking for a better way to see the current line without losing language-specific color coding, try the “Draw box around current line” feature (“Tools”, “Configuration”, “General...”).

If you sometimes want to know what lines have been modified, bind the command **color_modified_toggle** to a key. It will toggle display of modified lines in a different color on/off. You might want to bind the **color_toggle** command to a key as well. This command toggles between current line, modified line, and language specific coloring individually. For help on changing your key bindings, see “**Changing Key Bindings**” on page 224.

Resetting Modified Lines on Save

Visual SlickEdit can clear the modified and inserted line color when you save a file. To enable this feature, turn on the “Reset line modify” check box on the **File Options dialog box** (“Tools”, “Configuration”, “Files Options...”, select Save tab).

Adding Keywords

Use the **Color Coding Setup dialog box** (“Tools”, “Configuration”, “Color Coding...”) to add color-coded keywords to an already supported language by following these steps:

1. Display the Color Coding Setup dialog box (“Tools”, “Configuration”, “Color Coding...”).
2. Select the “Tokens” tab.
3. Choose the language you want to modify from the “Lexer Name” combo box list.
4. Click “New...” (located to the right of the list box).
5. Enter the new keywords separated with a space character.
6. Click OK.
7. Click OK on the Color Coding Setup dialog box.

Creating Language-Specific Color Coding for a New Language

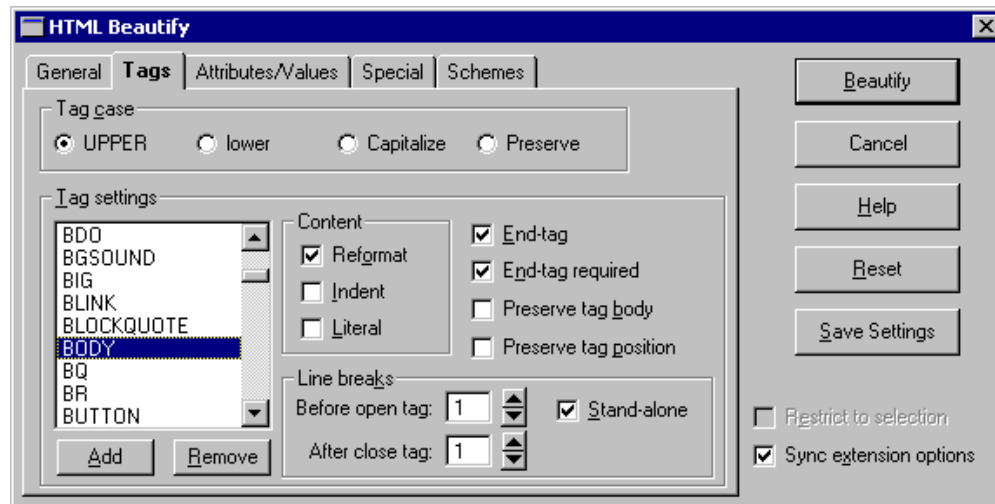
If Visual SlickEdit does not already have color coding support for your language, you can easily create it with the Color Coding Setup dialog box ("Tools", "Configuration", "Color Coding...") by following these steps:

1. Display the Color Coding Setup dialog box ("Tools", "Configuration", "Color Coding...").
2. Click "New..." (to the right of the Lexer name combo box).
3. Enter the new lexer name. Usually this is a language name such as C or Java.
4. 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 "Tokens" tab.
7. Add your keywords using the "New..." button to the right of the list box.
8. Select the "Comments..." tab. This lists the comments currently defined and allows you to define new multi-line and line comments. For each comment, click the appropriate "New" button to add a line or multi-line comment.
9. Select the "Numbers" tab to display various numeric style options. (See online help for more information on this tab.)
10. Select the "Strings" tab to display various string literal options. (See online help for more information on this tab.)
11. If you have not found all the options you need, click the "Language" tab. This displays some more advanced language-specific options. (See online help for more information on this tab.)
12. Click OK on the Color Coding Setup dialog box.
13. In order to see your color coding, fill in the "Lexer name" text box on the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup...") for the extension you want to use with this lexer. If your extension is not listed, click "New..." on the Extension Options dialog box to create a new extension.

Code Beautifiers

With Visual SlickEdit's code beautifier functionality, you can beautify selected lines or your entire buffer.

HTML Beautifier



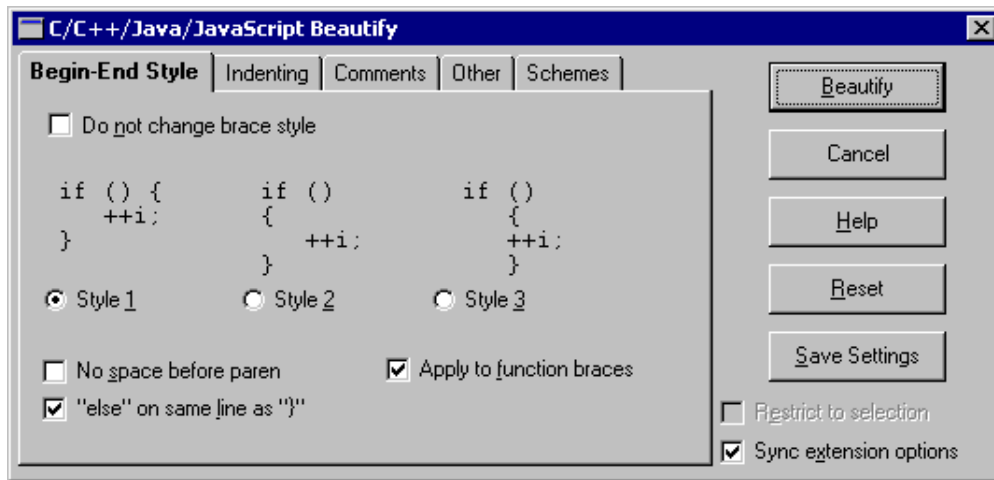
Use the **HTML Beautifier** dialog box ("Tools", "Beautify Source...") to make your HTML look the way you want. The HTML toolbar has a Beautify button to beautify the current source file or selection using your current settings.

XML Beautifier



Use the **XML Beautifier dialog box** ("Tools", "Beautify Source...") to make your XML look the way you want.

C/C++/Java/JavaScript/C# Beautifier



Use the **C/C++/Java/JavaScript/C# Beautifier dialog box** ("Tools", "Beautify Source...") to make your code look the way you want. The beautifier will change your brace style, correct indenting, correct spacing in conditional expressions, and more.

NOTE: SlickEdit has received many valuable comments and suggestions from users for enhancing the beautifier, and several of them have already been implemented. If you have any comments or suggestions about the beautifier, please contact SlickEdit Inc. (See **"Error! Reference source not found."** at the end of this Guide.)

vlx File Format

This section describes the format of the color coding vlx files. You might find the **Color Coding Setup dialog box** ("Tools", "Configuration", "Color Coding...") easier to use than editing this file directly.

The "vslick.vlx" file defines language-specific coloring support for the following languages:

- C++
- C
- Java
- Delphi
- Pascal
- AWK
- REXX

- Perl
- HTML
- Modula-2
- dBASE
- COBOL
- Python
- CICS
- Fortran
- Visual Basic
- VHDL
- CFScript
- Ada
- Slick-C
- Assembler

You can modify this file or create your own ".vix" file.

Perform the following steps to modify an existing language-specific coloring definition:

1. Edit "vslick.vix".
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, Ada, or Slick-C.
3. Modify the definition. See below for information on syntax of definitions.
4. Invoke the **load** command from the Visual SlickEdit command line. If the current buffer has a ".vix" extension, it will be loaded. Otherwise you will be prompted to specify a file name. Specify "vslick.vix" including path as the file name. Once the file is loaded you will see your changes immediately.

If you are creating a new lexer name (and thus a new section in the .vix file), perform all of the above steps and continue on to the following steps:

5. Display the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...").
6. Select the "Advanced" tab.
7. If this lexer definition is for a new extension, create the extension with the New button. Otherwise, choose the appropriate extension.
8. Set the "Lexer Name" for the new lexer definition you created.
9. Turn on the "Language Specific" check box.
10. Click "Update" to commit the changes.

Files with a ".vix" extension are text files which have a syntax similar to an ".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 which look like *name=value*.

The statements can be one of the following:

case-sensitive= [Y | N]

Defines the case sensitivity for the language. This statement must be the first or second statement within the section.

idchars=*start_id_chars after_id_chars*

Defines the characters which are the start of a valid identifier and additional valid characters which may follow. This statement must be the first or second statement within the section. You may use a '-' character to specify a range such as A-Z which specifies uppercase letters. To specify a '_' or '\' character as a valid word character, place a backslash before the character.

styles= [style] [style] ...

Defines zero or more styles. **style** can be one of the following:

linenum	Line numbers may be found as the first non-blank symbol of a line like BASIC.
dqbackslash	Color double-quoted strings. Characters following a backslash in a double quoted string are included in the string (like C).
dqbackslashml	Color double-quoted strings. If a double-quoted string ends in a backslash, it continues the string to the next line (like C).
dqmultiline	Color double-quoted strings. String may span multiple lines.
dqdouble	Color double-quoted strings. Two double quotes represent one double quote.
dqterminate	Do not color code a double-quoted string until the string is terminated. This style does not support dqmultiline or dqbackslashml.
dqlen1	Color double-quoted strings. Double-quoted strings contain exactly one character.
sqbackslash	Color single-quoted strings. Characters following a backslash in a single-quoted string are included in the string (like C).
sqbackslashml	Color single-quoted strings. If a double-quoted string ends in a backslash, it continues the string to the next line.
sqmultiline	Color single-quoted strings. String may span multiple lines.
sqdouble	Color single-quoted strings. Two consecutive single quotes represent one single quote (like Pascal).
sqterminate	Do not color code a single-quoted string until the string is terminated. This style does not support sqmultiline or sqbackslashml.
sqlien1	Single-quoted strings contain exactly one character (like Ada).
amphhex	Hexadecimal numbers are of the form &Hddd (like BASIC).
ampoct	Octal numbers are of the form &Oddd (like BASIC).
hexh	Hexadecimal numbers are of the form ddddH (like Intel Assembler).
octo	Octal numbers are of the form ddddO (like Intel Assembler).
octq	Octal numbers are of the form ddddQ (like Intel Assembler).

poundbase	Based numbers are of the form #base#number#exponent (like Ada).
underlineint	Numbers may have underlines between the numbers (like Ada).
xhex	Hexadecimal numbers are of the form 0xhhh (like C).
nonumbers	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.
rexxhex	Hexadecimal strings are followed by an upper or lowercase "x." For example, '414141'X or "414141"X are REXX-style hexadecimal strings which are both equivalent to the string "AAA".
packageimport	Language has Java syntax package and import statement where non-quoted file name follows package and import keyword.
idparenfunction	An identifier followed by an open parenthesis indicates a function (like C++ and Java).
html	Enables HTML syntax embedded languages and attribute coloring.
backslashescapechars	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.
heredocument	Enables support for Here Documents (UNIX Shell/Perl scripts e.g. print <<HTMLEOF HTMLEOF). Note that if you prefix your terminator with one of the lexer names you will get embedded language color coding.
perl	Adds support for Perl format statement and some other Perl specific changes.
tcl	Special support for TCL language color coding.
bquote	Perl and UNIX Shell style backquote (subshell).
model204	Special support for Model 204 language.
cics	Special support for CICS embedded in COBOL.
python	Special support for Python.

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 '-' character to specify a range such as A-Z which specifies uppercase letters. To specify a '-' or '\' 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 4 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 keywordcolor, numbercolor, stringcolor, commentcolor, ppkeywordcolor, linenumcolor, symbol1color, symbol2color, symbol3color, or symbol4color.

mlcomment= *start_symbol start_col* [**checkfirst**|**leading**] *end_symbol* [**lastchar**]

Defines a multi-line comment. This construct was designed to handle comments for the ATLAS language. *start_symbol* and *end_symbol* define strings which start and end the comment. *start_symbol* is only considered the start of a comment if it appears in column *start_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, *start_symbol* is limited to 1 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 2 multi-line comments may be defined.

mlkeywords= [*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= [*mlkeyword*] [*attribute*] [*attribute*] ...

Defines attributes for the *mlkeyword* 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 1 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.

keywords= [*keyword*] [*keyword*] ...

Defines words that should be displayed in keyword color. Keywords do not have to be valid identifiers.

cskeywords= [*keyword*] [*keyword*] ...

(case sensitive keywords) Defines words that should be displayed in keyword color only if found in the case specified. This statement should only be used for languages such as HTML which are case insensitive except for a few words. For other languages, use the **case-sensitive** and **keywords** statements. Keywords do not have to be valid identifiers.

ppkeywords= [*keyword*] [*keyword*] ...

Defines words that should be displayed in preprocessor color. The first character of a preprocessor keyword must not be a valid identifier. Preprocessing keywords must appear as the first non-blank symbol in the line.

symbol1= [*keyword*] [*keyword*] ...

Defines words that should be displayed in symbol1 color. Keywords do not have to be valid identifiers.

symbol2= [*keyword*] [*keyword*] ...

Defines words that should be displayed in symbol2 color. Keywords do not have to be valid identifiers.

symbol3= [*keyword*] [*keyword*] ...

Defines words that should be displayed in symbol3 color. Keywords do not have to be valid identifiers.

symbol4= [*keyword*] [*keyword*] ...

Defines words that should be displayed in symbol4 color. Keywords do not have to be valid identifiers.

Building and Debugging Your Code

Building and Debugging Overview

Visual SlickEdit provides several building and debugging tools.

Building from within the Editor

The Project menu contains all the commands related to compiling from within the editor.

Visual SlickEdit provides the commands **project_compile** and **project_build** for compiling. The **project_compile** command (SHIFT+F10 or "Build", "Compile") compiles the current buffer based on a Compile command you specify. Alternatively, the **project_build** command (CTRL+M or "Build", "Build") executes a Build command you specify. Both of these commands can be specified per-project or per-file extension.

Before you can execute the Build or Compile commands you must set the current project or define an extension-based project. To define an extension-based project command, use the **Extension Options dialog box** ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab) to select the extension, and then click "Extension Specific Project". You will probably want your Build command to be based on the current project and not the current extension. Use the **workspace_new** command ("Project", "New...") to create a workspace or project. See "**Creating Projects**" on page 32 for more information. If the current project has a Compile command defined, the extension-specific project Compile command will be ignored.

By default, the Compile or Build command is executed in Visual SlickEdit's concurrent process buffer (named ".process") which is displayed in the "Build" tab of the Output toolbar. This allows you to continue editing while your compiler runs. You can process the error messages as they appear in the concurrent process buffer instead of waiting until the compile(s) finish. You can use the **stop_process** command ("Project", "Break Process") to send a control break signal to stop your compiler running in the concurrent process buffer.

Viewing Build Errors

Execute the **next_error** command (CTRL+SHIFT+DOWN, or "Build", "Next Error") to place your cursor on the line of the file containing the next error. Click the "Build" tab on the Output toolbar (docked on the bottom by default) to view all of your compiler's error messages. You can use the **cursor_error** command (ALT+1, Double-click or "Build", "Go to Error/Include") to set the next error starting search position and go to a specific error.

You can also use the **cursor_error** and **next_error** commands to process messages from the **grep** (UNIX: **sgrep**) program provided with Visual SlickEdit. For example:

1. Activate the concurrent process buffer ("Project", "Activate Build")
2. Type "**grep main *.c**" (UNIX: "**sgrep main *.c**") and press ENTER.

3. Press CTRL+SHIFT+DOWN to cursor through the located occurrences.
4. Press PgUp to move the cursor off the concurrent process buffer command prompt so you can use the **cursor_error** command (ALT+1 or double-click) on a specific occurrence.

Adding Support for Other Error Message Formats

Visual SlickEdit supports many error message formats. If the error message format that is output from your compiler is not supported, you can add it if the compiler outputs at least the file name, line number, and the message on the same line.

Let's say your compiler error message format is as follows:

linenum, filename: message

Here are the steps to add support for the above error message format:

1. Define a SlickEdit regular expression (see "[SlickEdit Regular Expressions](#)" on page 111) which matches the above. The first character of the regular expression must be the '^' character, which matches the beginning of a line.

The regular expression "***^:i, :p\?:*\$***" will match this message.

2. Place curly braces around the file name, line number, column number, and message parts of the regular expression to signify the tagged expressions. In this example, there is no column number so nothing is required for that part.

The resulting regular expression is: "***^{:i}, {:p}\?:{?*\$}***"

3. Number the tagged expressions so that the filename is tagged expression 0, line number is tagged expression 1, column number is tagged expression 2, and the message is tagged expression 3.

The resulting regular expression is: "***^{#1:i}, {#0:p}\?:{#3?*\$}***"

4. Use the **gui_set_var** command ("Macro", "Set Macro Variable") to set the macro variable "def-error-re2" to the value of this regular expression. You can replace the current contents.
5. Load the "error.e" macro module ("Macro", "Load Module...") so the error searching macro can reinitialize.

Build Methods for GNU C/C++

The following build methods are provided:

- Build without a makefile (dependencies automatically checked)
- Build with a user-maintained makefile or custom build command
- Build with an auto-generated, auto-maintained makefile

With these build options you will not need to convert your current build methods to use the Visual SlickEdit GNU debugger. You can select one of these build methods when you create a new “GNU g++ Wizard” project.

Building without a makefile (dependencies automatically checked)

When you create a “GNU g++ Wizard” and select this build option, no makefile is ever generated. Instead, our “vsbuild” utility program determines what needs to be compiled dynamically. We recommend using this option when you are not worried about how the build gets done. Make sure your project include directories (“Project”, “Project Properties...”, select the Directories tab) are set up correctly so include files may be found.

Build with a user-maintained makefile or custom build command

When you create a “GNU g++ Wizard” and select this build option, no makefile is ever generated and by default the build command is set to “make”. You can change the build command to anything you want using the Project Properties dialog (“Project”, “Project Properties...”, select the “Tools” tab, select Build for the tool name). Choose this option when you already have your own method for building your source.

Build with an auto-generated, auto-maintained makefile

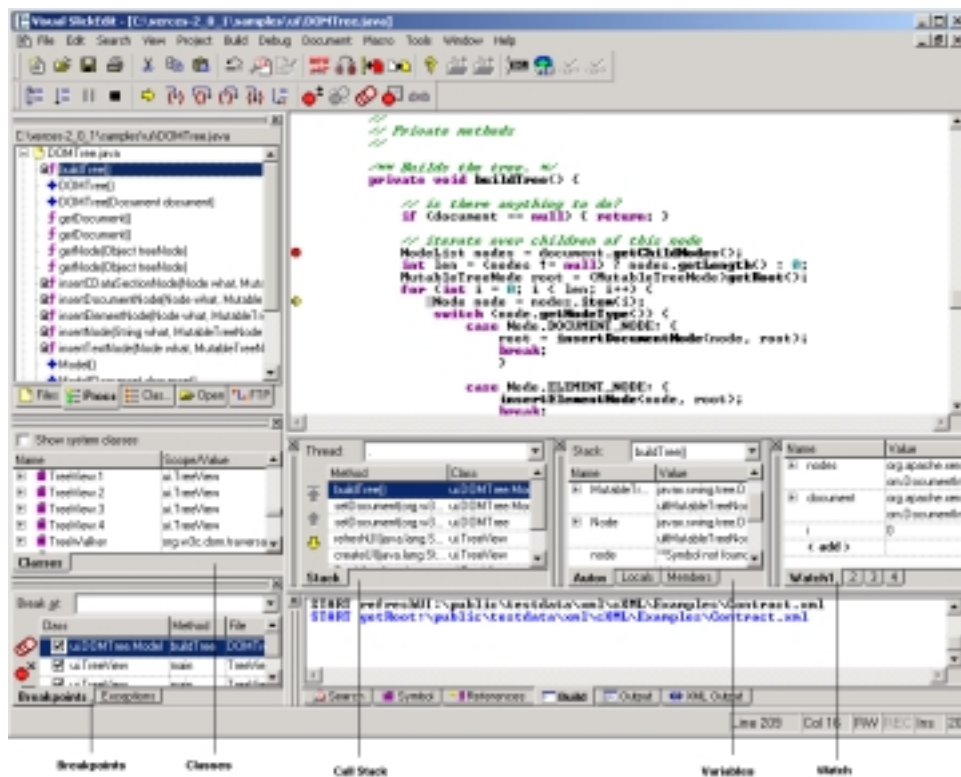
When you create a “GNU g++ Wizard” and select this build option, a makefile is automatically generated and updated when files are added to the project. We recommend using this option when you need a makefile and do not want to use our “vsbuild” utility. Make sure your project include directories (“Project”, “Project Properties...”, select the Directories tab) are set up correctly so include files may be found. To start a build from outside Visual SlickEdit, execute the following command:

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

Where *make* is the name of the make program, *Makefile* is the name of the makefile, and *ConfigName* is the name of the configuration.

Java and GNU C/C++ Debugger

The Java debugger supports SUN’s JDK 1.3.1 or higher. The GNU C/C++ debugger supports “gdb” version 5.1 or higher. A compatible “gdb” is included with Visual SlickEdit.



Starting the Debugger

Before you can use the debugger, you must create or open a Java or GNU C/C++ workspace. See [“Creating a Workspace and Project”](#) on page 33. Alternatively, you can open the Sample Java project (“Project”, “All Workspaces”, “Sample Workspaces”, “java.vpw”).

Choose “Start”, “Step Into”, “Run to Cursor” or “Restart” from the Debug menu to start the debugger. “Start” executes your program and will stop when a breakpoint is reached. “Step Into” places you on the first executable line of your program. “Restart” stops the current debugger session if necessary and then places you on the first executable line of your program (like “Step Into”). “Run to Cursor” will execute your program and will stop when the line under the cursor is reached.

Debug Key Bindings

Start/continue debugging	F5
Stop debugging	SHIFT+F5
Restart debugging	CTRL+SHIFT+F5
Toggle breakpoint	F9
Toggle breakpoint enable	CTRL+F9
Clear all breakpoints	CTRL+SHIFT+F9

Step over	F10
Step into	F11
Run to cursor	CTRL+F10
Show next statement	ALT+PAD-STAR
Activate breakpoints window	CTRL+ALT+B, ALT+F9
Activate watch window	ALT+3, CTRL+ALT+W
Activate variables window	ALT+4, CTRL+ALT+V
Activate call stack	ALT+7, CTRL+ALT+C
Activate threads window	CTRL+ALT+H

Setting Breakpoints

The quickest way to set or clear a breakpoint is to press F9. This toggles the breakpoint for the current line. You can also set a breakpoint using the appropriate button on the Debug toolbar. To display the Debug toolbar:

1. Bring up the Toolbar Customization dialog (“View”, “Toolbars...”).
2. Select the “Toolbars” tab.
3. Select the “Debug” toolbar in the list box.
4. Turn on the “Visible” check box.

Visual SlickEdit provides a Breakpoints toolbar which displays all of your breakpoints and lets you easily add, remove, and enable breakpoints. To display the Breakpoints toolbar, perform the steps above, but select “Breakpoints” instead of “Debug” for the toolbar.

Setting Conditional Breakpoints

To set a conditional breakpoint:

1. Set a breakpoint.
2. Display the Breakpoint toolbar (Bring up the Toolbar Customization dialog (“View”, “Toolbars...”), Select the “Toolbars” tab, select the Breakpoint toolbar in the list box, and turn on the “Visible” check box).
3. Select the “Breakpoints” tab on the Breakpoint toolbar.
4. Double-click on the breakpoint for which you want to set a conditional breakpoint.
5. Set the “Expression to be evaluated” or the “Number of times to skip before stopping”.
6. Click OK.

Avoiding Stepping Into Runtime Functions and Methods

The Runtime Filters tab on the Debugger Options dialog (“Debug”, “Debugger Options...”, select Runtime Filters tab) allows you to configure the Step Into command to skip certain runtime functions and methods.

For Java, use this tab to specify packages or specific classes or class patterns for the debugger to consider as “Runtime” classes. This will affect the behavior of “Step Into” and the Debug Classes dialog. “Step Into” will step over statements that call into runtime classes. The “Reset” button will reset the filters back to the defaults, which are “java.*”, “javax.*”, “com.sun.*”, and “sun.*”, which correspond to the defaults used by Sun's jdb debugger.

For GNU C/C++, use this tab to specify a function, or class and method patterns (ex. strcpy, str*, MyClass::*, MyClass::Insert*) for the debugger to consider as “Runtime” functions. This will affect the behavior of “Step Into”. “Step Into” will step over statements that call into runtime functions. The “Reset” button will reset the filters back to the defaults.

Setting Exception Breakpoints (Java Debugger Only)

To set a breakpoint when an exception occurs:

1. Display the Breakpoint toolbar (Bring up the Toolbar Customization dialog (“View”, “Toolbars...”), Select the “Toolbars” tab, select the Breakpoint toolbar in the list box, and turn on the “Visible” check box).
2. Select the “Exceptions” tab on the Breakpoint toolbar.
3. Type the name of the exception class in the “Exception” combo box or choose one from the combo box list.
4. Press ENTER to add the exception class to the “Exceptions to catch” list box.
5. Click OK.

Once an exception breakpoint is added, double-click on it to display the exception properties dialog. This dialog allows you to specify an expression, number of times to skip before stopping, and a specific thread.

Debugger Toolbars

Debug	Provides buttons for commonly used debugger commands including start, restart, step, toggle breakpoint, and add watch. This toolbar is also very useful when you are not in debug mode. If you use the Java debugger, you should make this toolbar visible when you are not in debug mode.
Breakpoints	Lists breakpoints (and exception breakpoints for Java) and allows you to modify them. You must use this toolbar if you want to set

	breakpoint properties. You can use this toolbar when you are not in debug mode.
Call Stack	Displays the call stack for any thread. This toolbar is only available when in debug mode.
Locals	Displays the locals for any stack context. You can modify the contents of a local variable by specifying a valid Java expression. You cannot add or remove entries from an array. This toolbar is only available when in debug mode.
Members	Displays the value of static and non-static members for any method context. You can modify the contents of a member variable by specifying a valid Java expression. You cannot add or remove entries from an array. This toolbar is only available when in debug mode.
Watches	Displays variable contents you specify. To add a watch variable type the name into the “Name” field, click on the debug toolbar “Add watch” button, or right click on the variable or a selected expression and select the “Add watch” menu item. This toolbar is only available when in debug mode.
Variables	Displays auto, local, and member variables. The “Autos” tab displays the contents of variables used before and after the current execution line. The Locals and Members tabs provide the same functionality as the Locals and Members toolbars. When support is provided for separately dockable toolbar tabs, duplicate toolbars will not be necessary. This toolbar is only available when in debug mode.
Threads	Displays the threads (and thread groups – Java only). Right click to display a context menu which allows you to perform various operations on threads.
Classes	(Java only). Displays the currently loaded classes. Double-click on a class to display class properties. Double-click on a member to go to the definition of the member. Right click on a method and select “Set breakpoint” to add a breakpoint to a method. Right click on a member variable and select “Add Watch” to add a watch on a static class member.

Attaching to a Remote VM (Java Debugger Only)

To attach to a remote VM:

1. Start the remote VM with command arguments similar to the following:

```
java -Xdebug -Xnoagent -Xrunjdwp:transport=dt_socket,server=y,  
suspend=y,address=8000 MainClass Arg1 Arg2
```

2. Choose “Attach to Remote VM...” from the Debug menu.

Attaching to Running Process (GNU C++ Debugger Only)

To attach to a running process:

1. Find the pid (process ID) using the UNIX PS command.
2. Choose “Attach to Running Process...”
3. Enter the pid and the path to the executable (to pick up debug symbols).

Managing Your Files

Managing Your Files Overview

To help you quickly and easily manage your files, Visual SlickEdit features an alias facility, an efficient way to access and edit remote files via a FTP connection, and a file manager with a rich set of file listing, selecting, and operating capabilities.

Editing Record Files

Visual SlickEdit is useful for editing text record files. To edit a record file, use the **Open dialog box** (CTRL+O or "File", "Open..."), and enter the record width in the Record Width text box.

UNIX:

Click on the "Advanced>>" button to view the Record Width text box. Currently, the width must be at least 2. The editor will use its standard display mechanism to display the file.

Alias Facility

Visual SlickEdit's alias facility has two major purposes:

- Directory aliases, which save you from having to type in long path names when you are prompted for a file name or directory. Directory aliases are expanded by pressing CTRL+SHIFT+O.
- Extension-specific aliases, which extend or modify syntax expansion. Templates are expanded when you press SPACEBAR.

Global directory aliases are stored in the file "alias.slk". Extension-specific aliases are stored in a file of your choice but typically have the extension ".als". SlickEdit standardized on the ".als" extension for alias files since ".slk" files are for miscellaneous Visual SlickEdit configuration files.

Defining Directory Aliases

The Directory Aliases feature is unique to Visual SlickEdit, and is particularly helpful if you are a touch typist. Directory aliases are typically 4 characters or fewer and consist of a short abbreviation of the last name in a long directory list. 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" (or "vinc" (MS VC++), "binc" (C++ Builder), "dinc" (Delphi) if you have multiple compilers).

By using a directory alias when opening a file you do not have to type in long directories or click the mouse repeatedly in the directory list box. You can also specify aliases in the

Change Directory dialog box. Directory aliases take advantage of the fact that most users are constantly opening files from a small number of directories throughout the day. Take the time now to identify the directories you access most frequently and define aliases for them. You'll immediately see how much it improves the way you work.

Defining Directory Aliases

To define directory aliases:

1. Invoke the Alias Editor ("Tools", "Configuration", "Aliases...") to edit global (non-extension specific) aliases stored in the "alias.slk" file.
2. Click "New" to add a new alias and type in its value.
3. Click OK.

Using Directory Aliases

Now that you have defined some directory aliases, here are the steps to try them out:

1. Display the Open dialog box ("File", "Open...").
2. Type in your alias in the "File Name" text box.
3. Press CTRL+SHIFT+O to expand the alias. This works in every text box in Visual SlickEdit, in the concurrent process buffer, and in any buffer.
4. Now you can either press ENTER to view the files in this directory or enter the rest of the file name to open the file.

Embedding Environment Variables

If you keep your 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 called "c:\version20\src\project2\", define a "p2" alias and give it a value such as "%VERSION%\src\project2\". Type the following command on the Visual SlickEdit command line to set or create the VERSION environment variable:

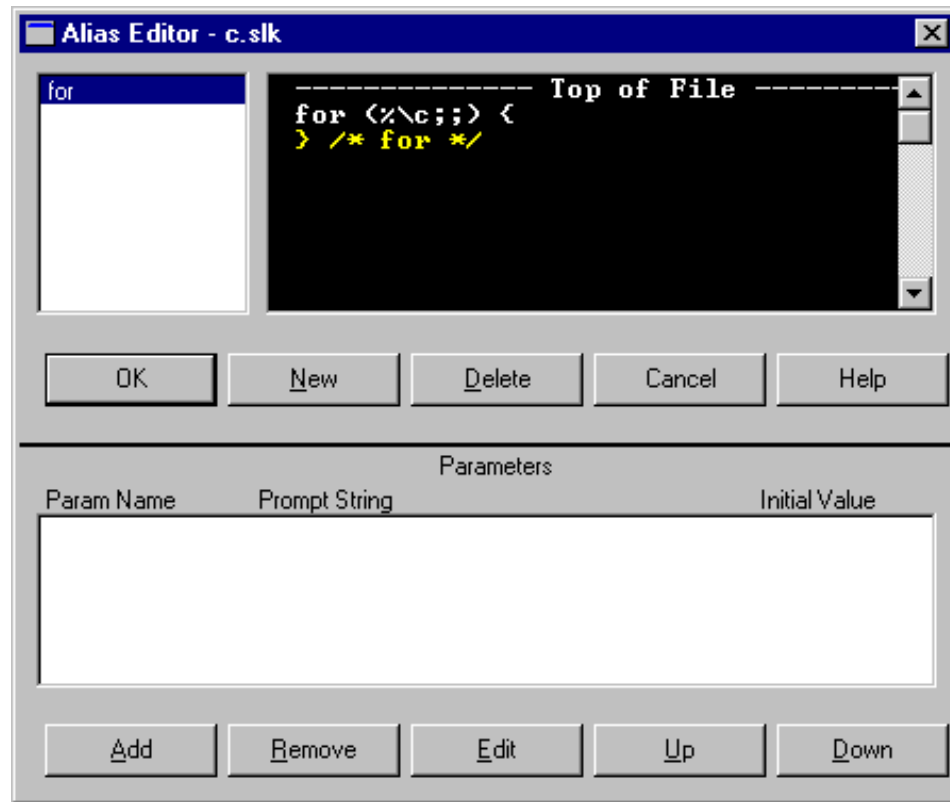
```
set VERSION=c:\version20
```

Alternatively, you can have environment variables automatically set when you open a project. See the "Open Command" tab on the [Project Properties dialog box](#) on page 41.

Extension-Specific Aliases

("Tools", "Configuration", "File Extension Setup...", select the "General" tab, "Aliases...")

The Alias Editor dialog box is displayed below:



Alias names are displayed in the list box at the top left of this dialog box. The value for the selected alias name is displayed in the editor control at the top right of this dialog box.

You can set up language-specific aliases which are expanded (completed) when you press SPACEBAR. These aliases can be additions or replacements to the existing syntax expansion. For example, to add a comment to the end of C **for**, **while**, **if**, and **switch** statements, perform the following steps:

Steps to Change Syntax Expansion:

1. Bring up the **Extension Options dialog box** for C ("Tools", "Configuration", "File Extension Setup...", select "c" in the extension combo box).
2. Select the "General" tab.

3. Click "Aliases..." to display the Alias Editor dialog box.
4. Click "New..." and type **for** as the alias name.
5. Type the following lines in the editor control (or edit box) to the right of the alias name:

```
for (%\c;;) {  
} /* for */
```

The "%\c" escape sequence above specifies the cursor placement after expansion is performed.

6. Repeat steps 4 and 5 for the **while**, **if**, and **switch** keywords.
7. Click OK to save your new aliases.

NOTE: The above steps replace the default syntax expansion for these keywords. The C brace style options will not affect aliases you define.

Alias Escape Sequences

The following escape sequences have special meaning when used within an alias:

%<env-var>%	Insert the value of the environment variable specified.
%\d	Insert the date.
%\t	Insert the time.
%\i	Indent.
%\b	Unindent.
%\c	Place cursor.
%\n	Current function name.
%\f	Current buffer name.
%%	Percent character.
%\ddd,	Go to column <i>ddd</i> .
%\ddd+	Increment column by <i>ddd</i> .
%\ddd-	Decrement column by <i>ddd</i> .
%(arg-name)	Argument replacement. See next section on "Aliases Which Prompt for Arguments".

Examples:

Alias Name	Value
comment	<pre>/* *****/ /* Date: %\d Time: %\t */ /* *****/</pre>
if	<pre>if(%\c){ %\i// Comment goes here }</pre>
wm	<pre>int APIENTRY WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow) { %\i%\c }</pre>

Aliases Which Prompt for Arguments

There are times when you want your own arguments inserted into an alias. The steps below illustrate defining a MessageBox alias for the C language which prompts for arguments.

Defining an Alias which Prompts for Arguments

1. Bring up the **Extension Options dialog box** for C ("Tools", "Configuration", "File Extension Setup...", select "c" in the extension combo box).
2. Select the "General" tab.
3. Click "Aliases..." to display the Alias Editor dialog box.
4. Click "New..." and type **mb** for the alias name.
5. Type the following line in the editor control to the right of the alias name:

```
MessageBox(HWND_DESKTOP, "%(message)", "Debug", MB_OK|MB_TASKMODAL);
```

The "%(message)" text will be replaced with the text you are prompted for when you expand the alias.

6. Click "Add...", type the text as shown below, and click OK.



7. Click OK to save your new aliases.
8. Now that you have completed writing your MessageBox alias for ".c" extension files, edit a C file and type **mb**<CTRL+SHIFT+O> to try out your alias.

MessageBox is one of the most commonly used functions when writing Microsoft Windows programs. The above example will save you some typing when using MessageBox.

Accessing And Editing FTP Files

FTP support includes a complete FTP client and the ability to easily open and edit FTP files.

Starting a Connection

Before you can access your FTP files, you need to start a connection. Use the FTP Profile Manager dialog ("File", "FTP", "Profile Manager...") to create a profile or start a connection.

FTP Client

After you have started a connection:

1. Invoke the menu item ("View", "Toolbars...").
2. Select the FTP Client toolbar.
3. Turn on the "Visible" check box. This will display the FTP client toolbar which is used for file management; that is, transferring files between your local machine and an FTP host.
4. Right click on the local or remote file listing to display a menu of operations you may choose from.

Opening FTP Files

Before you can open an FTP file you need to start a connection. See "Starting a Connection" above for more information. After you have started a connection:

1. Click on the "FTP Open" tab on the Project toolbar (docked on the left by default). You will see a listing of FTP files.
2. Double-click to open a file for editing or to change directory.
3. Right click on the file listing to display a menu of operations to choose from.

Visual SlickEdit File Manager

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

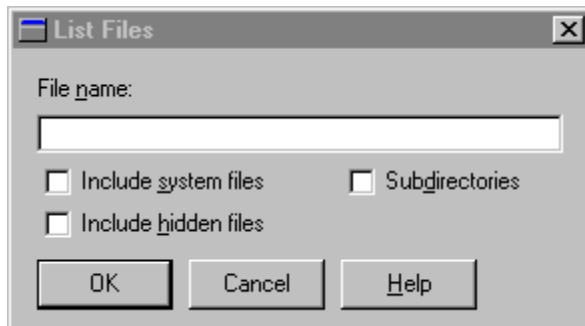
Windows:

The Visual SlickEdit file manager will handle multiple disk backups by requesting another disk and formatting it if necessary. All file management capabilities are driven by the Slick-C language. New Slick-C language commands may be written to be used by the file manager. You can use the file manager to quickly search your hard disk to find a file you wish to edit, or perform a search and replace on a specific set of files. The difference between the Visual SlickEdit file manager and other file managers is its extraordinary degree of programmability.

List Files Dialog Box

("File", "File Manager", "New File List..." or **fileman** command)

To start the Visual SlickEdit file manager, use the List Files dialog box ("File", "File Manager", "New File List..."). Below is a picture of the List Files dialog box:



File name

Specify in this text box one or more files separated by spaces. Each file specification may contain wildcard characters. For example, "***.c *.h**" will list all C and H files.

Include system files

When checked, files with the system attribute set are included in the list. This option is ignored on UNIX. This option is always turned on under Windows if the "Show all files" explorer option is set.

Include hidden files

When checked, files with the hidden attribute set are included in the list. This option is ignored on UNIX. This option is always turned on under Windows if the "Show all files" explorer option is set.

Subdirectories

When checked, all subdirectories below each file specified in the File Name text box are searched.

dir and list Commands

If you prefer using the command line, you can invoke the file manager with the commands **dir** or **list**. The **dir** command creates a buffer and inserts a list of the files within a directory with columns for name, extension, size, date, time, attributes, and optional path. The mode is changed to fileman mode. The syntax of the **dir** command is:

```
dir {[ [- | +] option_letters] [ [ @ ] filespec ] }
```

The **list** command is the same as the **dir** command except it lists files at or below a directory you specify. The **fileman_mode** command may be used to change the edit mode of the current buffer to fileman mode. The syntax of the **list** command is:

```
list {[ [- | +] option_letters] [ [ @ ] filespec ] }
```

option_letters can be any of the following:

H	Include hidden files. Defaults to OFF. This option is ignored by the UNIX version.
S	Include system files. Defaults to OFF. This option is ignored by the UNIX version.
D	Include directory files. Defaults to ON for dir command. Defaults to OFF for list command.
P	Append path. Defaults to ON.
T	Tree file list. Defaults to OFF for dir command. Defaults to ON for list command.

filespec may contain operating system wild cards such as '*' and '?'. If *filespec* is not specified, the current directory is used. The '@' sign prefix to *filespec* indicates that *filespec* is a file or buffer which contains a list of file names to be used as arguments to this command.

Examples:

dir	List file and directory entries of the current directory.
dir @backup.lst	List file and directory entries specified by each line of the file "backup.lst".
dir *.c	List files and directory entries of the current directory that have the C file extension.
list @backup.lst	List files specified by each line of the file "backup.lst".
list	List all files starting from the current directory.

Windows Examples:

dir c:\	List file and directory entries in the root directory of drive C.
dir +HS c:\	List file and directory entries in the root directory of drive C including hidden and system files.
list c:\	List all files on drive C.
list +HS c:\	List all files on drive C including hidden and system files.

UNIX Examples:

dir \$HOME	List file and directory entries in the HOME directory.
dir ~	Same as above command.
list \$HOME	List all files starting from HOME directory.

Fileman Mode Keys

The following key bindings change when Fileman mode is active:

ALT+SHIFT+A	Select all files.
ALT+SHIFT+B	Backup selected files.
ALT+SHIFT+C	Copy selected files.
ALT+SHIFT+D	Delete selected files.
ALT+SHIFT+E	Edit selected files.
ALT+SHIFT+G	Search and Replace on selected files.

ALT+SHIFT+M	Move select files.
ALT+SHIFT+N	Key in current line file name.
ALT+SHIFT+O	Fileman Sort dialog box.
ALT+SHIFT+R	Repeat Command on Selected dialog box.
ALT+SHIFT+T	Fileman Set Attributes dialog box.
ENTER	Edit current file or insert directory files.
SPACEBAR	Select current file toggle.
SHIFT+UP	Select and cursor up.
SHIFT+DOWN	Select and cursor down.
SHIFT+PGUP	Deselect and cursor up.
SHIFT+PGDN	Deselect and cursor down.

Selecting Files and Operating on Files

Selected files show the character '>' at the beginning of the line. The file management commands which copy files, move files, delete files, or change attributes of files will only operate on the selected files. Select files before executing a file management command. If you forget to select files, you will receive the message: "Select file(s) first. Press F1 for help."

Most of the commands which select or deselect files can be accessed from the Select menu (Right Button, "Select" or "File", "File Manager", "Select"). The following commands select files:

fileman_space	(SPACEBAR) Toggles the selection character on/off.
fileman_selection_up	(SHIFT+UP) Selects the current file and moves the cursor to the previous line.
fileman_selection_down	(SHIFT+DOWN) Selects the current file and moves the cursor to the next line.
fileman_deselect_up	(SHIFT+PGUP) Deselects the current file and moves the cursor to the previous line.
fileman_deselect_down	(SHIFT+PGDN) Deselects the current file and moves the cursor to the next line.
select_all	(ALT+SHIFT+A or "File", "File Manager.", "Select", "All") Selects all files.
deselect_all	("File", "File Manager", "Select", "Deselect All") Removes the selection character from all files.

select_reverse	("File", "File Manager", "Select", "Reverse Select") Toggles the selection character on or off for all lines.
select_attribute	("File", "File Manager", "Select", "Attribute") Selects files that contain a specific attribute search string you specify.
select_ext	("File", "File Manager", "Select", "Extension...") Selects all files with a given extension. Enter the extension without the '.' character.
select_mark	("File", "File Manager", "Select", "Highlight") Selects all files within a marked area.
deselect_mark	("File", "File Manager", "Select", "Deselect Highlight") Removes the selection character from files within the selected area.

The following commands operate on the selected files:

fileman_backup	(ALT+SHIFT+B or "File", "File Manager", "Backup...") Backup selected files. Files are copied to a directory of your choice. The path of the source file is appended to the destination directory to determine each destination file name.
fileman_copy	(ALT+SHIFT+C or "File", "File Manager", "Copy...") Copy selected files to a directory of your choice. If the destination directory does not exist it is created.
fileman_move	(ALT+SHIFT+M or "File", "File Manager", "Move...") Move selected files to a directory of your choice. If the destination directory does not exist it is created. The destination directory does not have to be on the same disk partition.
fileman_delete	(ALT+SHIFT+D or "File", "File Manager", "Delete...") Erase selected files. A confirmation message box is displayed before the selected files are erased.
fileman_replace	(ALT+SHIFT+G or "File", "File Manager", "Global Replace...") Search and replace on selected files.

fileman_edit	(ALT+SHIFT+E or "File", "File Manager", "Edit...") Edit the selected files.
fileman_attr	(ALT+SHIFT+T or "File", "File Manager", "Attribute...") Change file attributes. Windows: Displays menu for selecting read-only, hidden, system, or archive attribute. UNIX: Displays menu for setting file permissions.
for_select	(ALT+SHIFT+R or "File", "File Manager", "Order") Repeats a command you specify for each of the selected files.

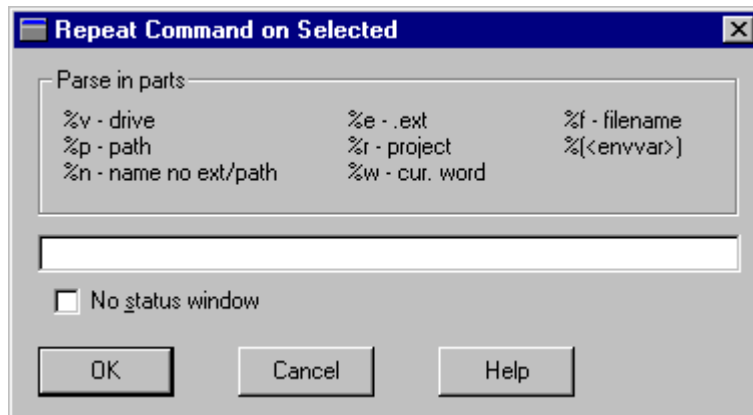
Windows:

You can place unformatted disks in the destination drive. The Visual SlickEdit file manager will detect that the disk is not formatted and prompt you to provide a format command to format the disk.

Repeat Command on Selected Dialog Box

("File", "File Manager", "Repeat Command..." or **for_select** command)

You can create a command string which executes an internal macro command or an external program. Parts of the file name can be inserted into the command string as well as a few other items. You can prepend the **fmsay** command to a file manager command to test if you have built a command string properly. The **fmsay** command does not touch your files in any way. It simply displays the command that is about to be executed in the fileman status window. Once you are satisfied with your command string, remove the **fmsay** command from your command string.



How Visual SlickEdit Finds External Files

There are several files Visual SlickEdit automatically looks 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" (UNIX: "vslick.stu"), and "vrestore.slk" is as follows:

- Configuration directory. The configuration directory is defined by the VSLICKCONFIG environment variable.

UNIX:

If VSLICKCONFIG is not defined, then "\$HOME/.vslick" is used.

Windows:

If VSLICKCONFIG is not defined, then the executable directory is used.

- **Windows:** Current directory.
- Paths specified in VSLICKPATH environment variable.
- Paths specified in PATH environment variable.

Search Order for Executable Files

The search order for executable files, batch macro programs, and miscellaneous files is as follows:

- **Windows:** Current directory.
- Configuration directory. The configuration directory is defined by the VSLICKCONFIG environment variable.

UNIX:

If VSLICKCONFIG is not defined, then "\$HOME/.vslick" is used.

Windows:

If VSLICKCONFIG is not defined, then the executable directory is used.

- Paths specified in VSLICKPATH environment variable.
- Paths specified in PATH environment variable.

Using Visual SlickEdit for Web Development

Web Overview

New, advanced support for developing global, Web and wireless applications has been added. Visual SlickEdit now includes many XML features and very rich Unicode support (see “Using Unicode” on page 170). In addition, the HTML support has been enhanced to include syntax indenting.

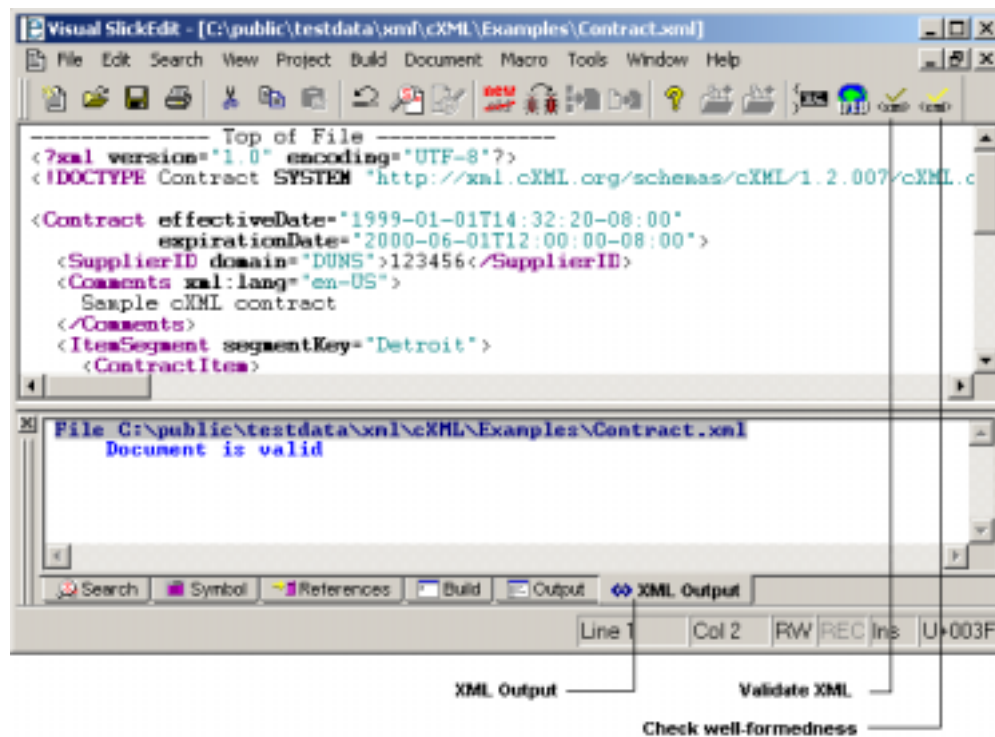
Using XML

XML Feature Overview

XML features now include Context Tagging, validation, well-formedness checking, beautifier, color coding, URL Mapping, syntax expansion, and syntax indenting for XML, XSLT, and Schemas (DTD or XSD).

Validating and Well-formedness Checking of XML Documents

An XML file can be validated against a DTD or schema by clicking the “Validate XML document” toolbar button. The results of the validation are placed in the “XML Output” tab on the Output toolbar. If there are errors during validation, you can double-click on the error line and Visual SlickEdit will open the appropriate file and move to the specified line, just like the Search tab works in the Output toolbar.



You can check to see if an XML document is well-formed by clicking the “Check well-formedness” button on the Standard toolbar.

Automatic Validating for XML Documents

By default, when you open an XML document, the document is validated. To turn this feature off, bring up the XML Options dialog (“Tools”, “Configuration”, “File Extension Setup...”, select the “xml” extension”, click the “Options” button), and turn off the “Auto validate on open” check box.

Beautifying XML

To beautify an XML document, open the document you wish to beautify. Then, bring up the XML Beautifier dialog (“Tools”, “Beautify Source”). Press F1 for help on the various options of this dialog.

DTD Caching

When you open an XML document which has a document type definition (!DOCTYPE) which refers to a remote external DTD, the DTD file is downloaded and cached locally so that no download is necessary the next time the file is referenced. The DTD is processed in order

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 which reference remote DTDs when you do not have an internet connection. If you want to force recaching of the DTD for the current XML document, right click to bring up the context menu and selection “Apply DTD changes”. Applying DTD changes is necessary after you create a new XML document and complete the document type definition (!DOCTYPE).

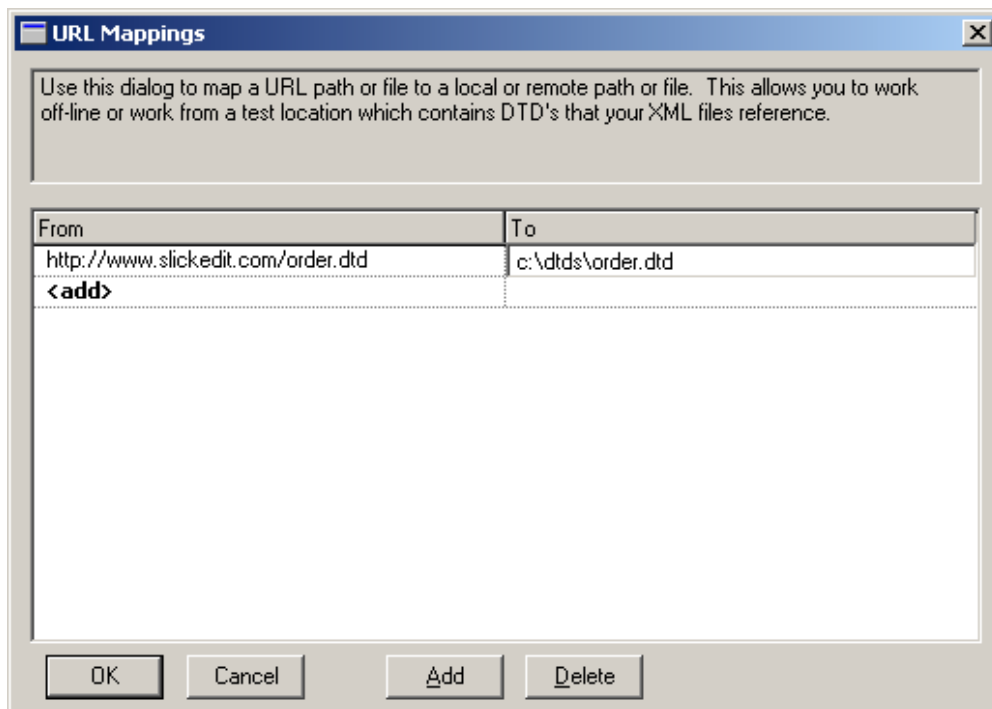
Opening DTD Files from XML in One Step

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

URL Mappings

Visual SlickEdit provides a way to map URLs to a different location. Whenever Visual SlickEdit needs to open a URL it first looks in the URL map to see if this URL is mapped to a different location. If the URL is mapped to a different location then Visual SlickEdit uses the mapped location.

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 documents DTD references, you can map the URL to the test DTDs location.



To map a URL:

1. Bring up the URL Mappings dialog (“Tools”, “Configuration”, “URL Mappings...”).
2. Click on the From cell that reads <add>.
3. Type in the URL that will be mapped to a different location.
4. Click on the To cell to the right of the From cell.
5. Type in the location to use for this URL.
6. Click on OK.

Toggling Between Begin and End XML Tags

Place your cursor anywhere on the begin or end tag and press CTRL+] to find the corresponding end or begin tag respectively.

Using Unicode

This feature requires Unicode support (OEMs typically turn this feature off).

NOTE: The UNIX version of Visual SlickEdit does not support Unicode yet. If you would like UNICODE support under UNIX, please send an e-mail message to support@slickedit.com indicating the UNIX platform you want it for and the reason you need it.

Unicode Overview

Visual SlickEdit's Unicode support includes the following:

- All Context Tagging™ features. See "Context Tagging™" on page 45 for more information.
- DIFFzilla. See "DIFFzilla™" on page 63 for more information.
- Color coding.
- Level 1 regular expressions as defined by the Unicode consortium. See "**Regular Expressions**" on page 107 for more information.
- 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" on page 173.
- Multiple clipboards. See "Moving Text Among Files With Clipboards" on page 83 for more information.
- Sorting.
- 3-Way Merge.
- Support for composite and surrogate characters.
- Support for storing up to 31-bit Unicode characters.
- SmartPaste™. See "SmartPaste™" on page 83 for more information.
- Syntax expansion and syntax indenting.

- Beautifier. (Note: the source files supported by our beautifier are not typically stored as Unicode.)
- Spell checking. (Note: the spell checker currently only supports English.)
- Support for almost all of Visual SlickEdit's SBCS/DBCS active code page features. See Unicode Limitations below.

In order to provide better support for editing Unicode and non-Unicode files, Visual SlickEdit has two modes of editing; Unicode, and SBCS/DBCS mode. By choosing the encoding, you can specify which mode Visual SlickEdit uses.

What are Encodings For?

Encodings tell Visual SlickEdit how 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 ".c#" source files are loaded as SBCS/DBCS active code page data.

Visual SlickEdit can be configured to convert all file data to Unicode UTF-8 data, but this would cause some problems. Loading files containing SBCS/DBCS data would take significantly longer, slowing down 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 (See Unicode Limitations below).

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 which are not XML and do not have signatures, configure Visual SlickEdit default options to get the best recognition possible. This is important because some features such as drag/drop files and DIFFzilla do not prompt you for the file encoding.

Each extension may have its own encoding specification. If the extension-specific encoding is set to "Default", then the global setting defined in the File Options dialog ("Tools", "Configuration, "File Options", select the Load tab) is used. Both the extension-specific and global setting are overridden if you previously specified an encoding in the Open dialog. Visual SlickEdit records the encoding used to override default encoding settings and reuses this setting the next time you open the same file. This provides you with per-file encoding support.

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). Visual SlickEdit does not support this and assumes that the file is SBCS/DBCS active code page data.

Opening Unicode Files

To open a Unicode file:

1. Use the Open dialog “File”, “Open...”.
2. Specify the encoding if necessary.
3. Press ENTER.

Surrogate Support

Surrogates are not much of a special case for Visual SlickEdit since Unicode data is stored as UTF-8 and not UTF-16. Since Visual SlickEdit uses Windows Win32 calls 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. SlickEdit relies on Windows for casing support and Windows casing features do not support surrogates.

Converting Unicode to UCN

Visual SlickEdit allows you to convert a selection from Unicode to UCN or vice versa. These conversion features are located on the (“Edit”, “Other”) menu. Visual SlickEdit’s Unicode to UCN conversion feature is most useful for specifying Unicode character strings in SBCS/DBCS active code page source files. For example, here are the steps to store some UCN in a Java source file:

1. Open the Unicode file containing the Unicode characters or create a new Unicode file and enter the characters you want to convert.
2. Select the Unicode characters you want to convert.
3. Execute the “Java/C# (\uHHHH)” menu item (“Edit”, “Other”, “Copy Unicode As”).
4. Open the Java source file and paste (“Edit”, “Paste”) the UCN data into your file.

Unicode Limitations

- Bold and italics color coding is not supported. Support for this will be added in a future version.
- Tab character operations are not fully supported. Tab display, the “Expand tabs to spaces” save option (“Tools”, “Configuration”, “File Options...”, select the Save tab), and save with tabs (“save +t”) only work correctly if all the characters are below 128. The “Expand tabs to spaces” load option (“Tools”, “Configuration”, “File Options...”, select the Load tab) is ignored.
- Column selections do not fully support Unicode. If all the characters are below 128 and the font is fixed then it works. Support for this will be added in a future version.
- Word wrap does not fully support Unicode. If all the characters are below 128 and the font is fixed then it works. Support for this will be added in a future version.
- The Unicode line end character 0x2048 is not yet supported..
- Hex editing is not supported. The current character (Composite character) is displayed on the status line. In addition, you can use the Open dialog with the “Binary, SBCS/DBCS mode” encoding to view a Unicode file in hexadecimal.

NOTE: If you would like hex editing support for Unicode files, please contact Product Support with your request and describe exactly what you are looking for.

- Casing features (upper case, lower case, ignore case) do not support surrogates. SlickEdit relies on Windows for casing support and Windows casing features do not support surrogates.
- “Vertical line column” (“Tools”, “Configuration”, “General...”, select the General tab) is not supported.
- “Truncation line length” is not supported
- “Record width” on the File Open dialog is not supported.
- DDE is not supported. Unicode DDE does not work with Internet Explorer or Netscape. You can view files with Unicode data in Internet Explorer; however, this feature will fail if the file name contains characters not in the active code page.
- Version control supports files containing Unicode data but does not support file names which contain characters not in the active code page.
- Special character display is not yet supported for Unicode buffers.

- The Visual SlickEdit "grep" 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 tag 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 tagging seek positions match the editor seek positions. You might need to rebuild your tag files if you change the default encoding options.
- The install (setup.exe), uninstall (uninstall.exe), and update (update.exe) programs are not Unicode applications so the installation directory must contain characters in the active code page.

Implementation Overview

Visual SlickEdit supports native Unicode and SBCS/DBCS editing modes. 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 which 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 Visual SlickEdit.
- Unicode editing modes cannot support all the features you were used to when editing SBCS/DBCS files (See "[Unicode Limitations](#)" on page 174).
- Macros can be written once to support both editing modes. This was very important to us because we wanted to reduce development time.
- Since Visual SlickEdit stores Unicode as UTF-8, only one set of binaries is required. Most products which support SBCS/DBCS and Unicode (UTF-16), use preprocessing. This requires two sets of binaries.

Using HTML

HTML Feature Overview

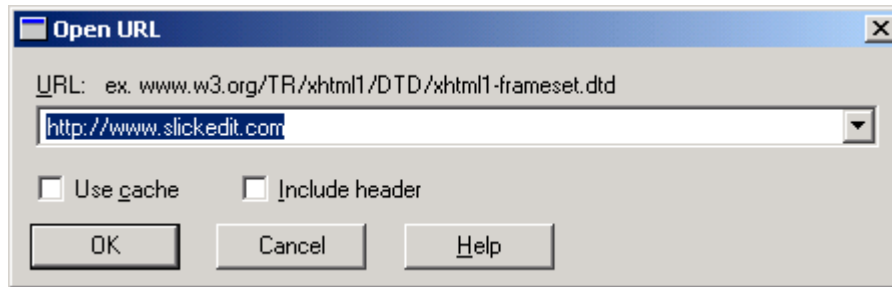
HTML support includes Context Tagging, beautifier, color coding, syntax expansion, and syntax indenting for HTML, JSP, and ASP. Many of Visual SlickEdit's language features are supported for languages embedded in HTML, including Context Tagging, color coding, SmartPaste, syntax expansion, and syntax indenting.

Beautifying HTML

To beautify an HTML document, open the document you want to beautify and bring up the HTML Beautifier dialog (“Tools”, “Beautify Source”). Press F1 for help on the various options of this dialog.

Open URL

Visual SlickEdit provides a way to open the contents of a URL into a Visual SlickEdit buffer. This feature can be used to open HTML, XML, DTD, and any other resource available from a URL.



To open a URL:

1. Bring up the Open URL dialog (“File”, “Open URL...”).
2. Type the URL OR select a URL from the history combo box.
3. Click OK.

You will not be able to save back to the URL. However, you can save the buffer to a local file.

When you open a URL, Visual SlickEdit puts the buffer in Fundamental mode. To select a different mode (e.g. HTML) for the buffer, choose “Document”, “Select Mode...” and select the mode from the list.

HTML Editing

Visual SlickEdit provides many tag and attribute options which will save you time entering HTML just the way you prefer. Bring up the HTML Options dialog to set one of the various options (“Tools”, “File Extension Setup...”, select the “html” extension, click on the Options button). An HTML toolbar will save you time entering typical HTML tags and attributes.

To display the HTML toolbar:

1. Bring up the Toolbar Customization dialog (“View”, “Toolbars...”).
2. Select the “Toolbars” tab.

3. Select the HTML toolbar in the list box.
4. Turn on the “Visible” check box.

Toggling Between Begin and End HTML Tags

Place your cursor anywhere on the begin or end tag and press CTRL+] to find the corresponding end or begin tag respectively.

Selecting a Block of Text

To select the current block of text, invoke the (“Edit”, “Select”, “Code Block”) menu item.

Hiding a Block of Text

To hide the current block of text, invoke the (“View”, “Hide Code Block”) menu item.

Using Visual SlickEdit with Visual C++ or Visual Studio .NET

Visual Studio Overview

Visual SlickEdit makes it easy to work in new programming environments – if you are using Visual Studio .NET no special setup is required to open your workspace files.

Opening Visual C++ or Visual Studio .NET Workspace Files

If you are using Visual C++ v5.0 or later, you should create your projects and workspaces in Visual C++ and then use the Open Workspace menu item ("Project", "Open Workspace...") to open your Visual C++ workspace. Once you have done this, Visual SlickEdit will build a workspace tag file, set up build/compile commands, and configure your browser databases if any exist.

When you open a Visual C++ workspace, you need to use Visual C++ to add files to your project or add projects to your workspace. After making a change to your project or workspace file in Visual C++, you need to invoke the "Save Workspace" menu item in Visual C++ or exit Visual C++ so that Visual SlickEdit will see the changes.

Compiling Your Project in Visual SlickEdit

For Visual C++ v5.x and v6.x, the default compile command uses the **"nmake"** program which requires a makefile (".mak" extension). Visual C++ v5.x and v6.x do not automatically create a makefile for you. Use the "Export Makefile.." menu item ("Project", "Export Makefile...") in Visual C++ to create or update your makefile. For Visual C++ v5.x or higher, the default build and rebuild commands do not need a makefile.

You can customize the compile, build, and rebuild commands from the "Tools" tab of the Project Properties dialog box ("Project...", "Properties").

If you get an error when you run **"nmake"**, you need to run the VCVARS32.BAT file (shipped with Visual C++) in a DOS box that you start the editor from. This will set the environment so that the editor can run these compiles.

Help For The Word at The Cursor

Press F1 to get help for the word at the cursor. This is helpful for getting help on run-time functions or methods. If you are unable to get help for the word at the cursor, make sure **"vckwds.hlp"** has been indexed. To check this, display the Configure Help Index File dialog ("Help", "Configure Index File..."). If vckwds.hlp is not displayed, click on the "Add.." button and add the **"vckwds.hlp"** file located in the root directory of your **Visual SlickEdit** installation (c:\vslick by default).

NOTE: Word help is not yet supported for Visual Studio.NET

Special Features on OS/390 and z/OS

OS/390 Features Overview

Any item in this guide or online help that refers to OS/390 also applies to z/OS.

If you are an OS/390 user, this section contains additional important information about how you can use Visual SlickEdit to access code that resides in data sets and to develop CICS, COBOL, Assembler, PL/I, JCL, and REXX applications. After reading this section, continue to the "[Quick Start](#)" section on page 13. If the source files reside in a USS HFS (UNIX System Services Hierarchical File System), you can turn to "[Quick Start](#)" now.

OS/390 Features

Some of the general Visual SlickEdit features are not relevant for the OS/390 platform. This section specifically describes features that make Visual SlickEdit powerful for developing OS/390 applications.

Edit Multiple Data Sets

You can simultaneously edit any number of sequential data sets or PDS members using one Visual SlickEdit session. You can open sequential data sets, PDS members and USS HFS files for editing or browsing harmoniously alongside each other.

Performance

The OS/390 Optimizations dialog ("Tools", "Configuration", "OS/390 Optimizations...") allows you to reduce CPU utilization and speed up start time. Optionally, you can disable mouse over detection, large toolbars, cursor blink (off by default), and key release events. You can also specify these options when you invoke Visual SlickEdit (see "[Invocation Options](#)" on page 23).

Language Type Determination

Visual SlickEdit supports language type determination in two ways:

- **Automatic Language Type Determination.** Like SPF Edit, Visual SlickEdit looks at the file contents and the data set name in order to select the language support.
- **Language Type Save/Restore per Data Set or Member.** In cases where the automatic language type determination is insufficient, you can manually select the language type ("Document", "Select Mode..."). For PDS members, you have the option of affecting all members or just the one member.

Visual SlickEdit automatically determines the language type as follows:

Assembler	Asterisk in column 1 or a recognized opcode of CSECT, DSECT, MACRO, TITLE, START or COPY NOTE: *PROCESS in column 1 is recognized as PL/I
PL/I	First string is % or /* or the first string is *PROCESS in column 1. See REXX, and C
Pascal	First string is (* or the first string is /* and the data set name ends in .PASCAL
COBOL	First non-blank is a * or / in column 7, or Columns 1 to 6 contains an integer and * or / in column 7, or First word is "CBL", or Columns 1 to 6 contain an integer and follows by "CBL", or Columns 1 to 6 contain an integer and first two words are "IDENTIFICATION DIVISION" or "ID DIVISION"
C	First string is # or first string is // and data set type is not .PROCLIB, .CNTL, .JCL, or .ISPCTLx or first string is /* and data set type is .C
REXX	First string is a /* comment containing REXX or first string is a /* comment and the data set type is .EXEC or .REXX
JCL	//anything in column 1 followed by a JOB, DD, PROC, EXEC, or MSG or /* in column 1 or // in column 1 and data set type is .PROCLIB, .CNTL, .JCL, or .ISPCTLx

Language Support

The languages listed below apply to mainframe development; Visual SlickEdit supports more languages than those listed.

- **Concurrent Compiling of Data Sets.** Supported foreground compilers are COBOL, PL/I, Assembler, C, C++, and Fortran.
- **Tagging.** Supported for COBOL, PL/I, JCL, REXX, OS/390 Assembler, C, C++, Ada, Java, Fortran, Pascal, HTML, Perl, JavaScript, and Tcl.
 - **List tags.** List tags on demand by pressing ALT+PERIOD.
 - **Tag navigation.** Press CTRL+PERIOD to go to the definition of a symbol under the cursor. To "pop" back to where you were, press CTRL+COMMA (or use "Search", "Pop Bookmark...").
 - **Tag preview.** The "Symbols" tab of the Output toolbar (docked on the bottom by default) displays the symbols for the word under the cursor and utilizes the context information. If only one tag is found, the source file containing the tag definition is displayed. Otherwise, the tag symbols are

listed. When a source file (such as a copy library) is displayed in the “Symbols” tab, you can double-click on it to edit the file. When symbols are listed, double-click on the symbol to go to the definition. Alternatively, you can enter a symbol to view in the "Symbol:" combo box of the “Symbols” tab. Right click inside the “Symbols” tab to show an option menu of tag types to show/hide.

- **Tag completion.** Press CTRL+SPACEBAR to complete (type the rest of) the current symbol you are entering. If a unique match is not found, a list is displayed allowing you to select the exact match.
- **Context Tagging™.** Supported for COBOL, OS/390 Assembler, PL/I, C, C++, Java, Pascal, and Ada. Context Tagging™ is a feature set which includes Auto List Members, Auto Parameter Info, context tag navigation, context tag preview, and context tag completion. The Context Tagging feature set performs expression type, scope, and inheritance analysis.

Visual SlickEdit is the first application to support Context Tagging for COBOL. Furthermore, Visual SlickEdit and Microsoft Visual C++ are the first products that support Context Tagging for C and C++ (both shipped in 1998). In addition, **Visual SlickEdit** is among the first three products to support Context Tagging for Java. (The others are non-OS/390 Microsoft Visual J++ and Symantec cafe.)

Below are descriptions of how context tagging helps you write COBOL programs. See “**Context Tagging(TM)**” on page 45 for more information.

- **Auto List Members.** When you type "in<SPACEBAR>" or "of<SPACE>", a list of group-level field names is automatically displayed. Press ALT+PERIOD to list accessible symbols on demand.

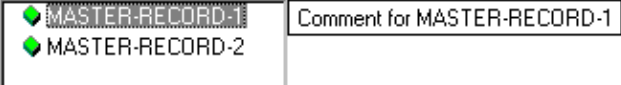
Here is a picture of Auto List Members:

```

FILE SECTION.
FD MASTER-FILE.
* Comment for MASTER-RECORD-1
01 MASTER-RECORD-1.
    03 GOODS-CODE      PIC X(4).
    03 GOODS-NAME      PIC X(38).
    03 GOODS-PRICE     PIC 9(4) BINARY.
01 MASTER-RECORD-2.
    03 GOODS-CODE      PIC X(4).
    03 GOODS-NAME      PIC X(48).
    03 GOODS-PRICE     PIC 9(4) BINARY.
WORKING-STORAGE SECTION.
01 FILE-NAME           PIC X(100).
01 NEW-FILE-NAME-SUF  PIC X(2) VALUE ".a".

PROCEDURE DIVISION.
  IF GOODS-CODE OF
    THEN
      MASTER-RECORD-1
      MASTER-RECORD-2

```



- **Auto Function Help.** When you type **using**<SPACEBAR>, help on subroutine arguments is automatically displayed with the current argument you are entering in boldface type. To access this feature on demand, place the cursor after the “using” statement and press ALT+COMMA.

NOTE: Place the text cursor anywhere on the statement and press ALT+COMMA to get statement syntax help for built-in constructs such as MOVE, DISPLAY, OPEN, ADD, SUBTRACT, READ, WRITE, and so on.

```

CALL SUMCALC USING "19991231" OUTPUT-DAYS|
SUMCALC::SUMCALC USING INPUT-DATE, OUTPUT-DAYS

```

Here is a picture of statement syntax help:


```

input-output section.
file-control.
  select input-file-1 assign to "test1.dat".
  select output-file-2 assign to "test2.dat".

```

```

procedure division.
  open output output-file-1 sharing no other.

```

```

OPEN { (INPUT or OUTPUT or I-O or EXTEND)
[SHARING WITH {ALL OTHER or NO OTHER or READ ONLY}]
[TIMEOUT {AFTER arithmetic-expression-1 SECONDS
or RETRY {arithmetic-expression-2 TIMES or NO LIMIT}}]
{file-name-1 [{REVERSED or WITH NO REWIND}}]... }...

```

☐ input-file-1
☒ output-file-2

Returns assign to "test2.dat"

- **Context tag navigation.** Press CTRL+PERIOD to go to the definition of a symbol under the cursor. To "pop" back to where you were, press CTRL+COMMA (or use "Search", "Pop Bookmark"). You can also right click on a symbol and select "Go to Definition..."
- **Context tag preview.** The "Symbols" tab of the Output toolbar (docked on the bottom by default) displays the symbols for the word under the cursor using context tagging information. If only one tag is found, the source file containing the tag definition is displayed. Otherwise, the tag symbols are listed. When a source file is displayed in the "Symbols" tab, you can double-click on it to edit the file. When symbols are listed, double-click on the symbol to go to the definition. Alternatively, you can enter a symbol to view in the "Symbol:" combo box of the "Symbols" tab. Right click inside the "Symbols" tab to show an option menu of tag types to show/hide.
- **Context tag completion.** Press CTRL+SPACEBAR to complete (type the rest of) the current symbol you are entering. If a unique match is not found, a list is displayed allowing you to select the exact match.
- **Symbol References and Uses.** Supported for COBOL, C, C++, and Java. Allows you to find all references to a specific symbol. For example, for COBOL you can find all source files which include a particular copy book. (Requires tag file.)
- **Class Browser.** Lists tags alphabetically or by category (procedures, variables, groups, files, etc.) Allows you to search for a symbol. See "Class Browser" on page 60.
- **Configurable Syntax Expansion.** Supported for COBOL, PL/I, JCL, REXX, C, C++, Ada, Java, Fortran, Pascal, Perl, JavaScript, Tcl, and more. For example, for COBOL, **pro**<SPACEBAR> inserts "procedure division" and places the cursor appropriately. See "Extension-Specific Aliases" on page 156 for more information.

- **Syntax Indenting.** Supported for COBOL, PL/I, JCL, REXX, C, C++, Ada, Java, Fortran, Pascal, Perl, JavaScript, Tcl, and more. The ENTER key intelligently indents to match the indentation of the surrounding code.
- **Color Coding.** Supported for COBOL, PL/I, JCL, REXX, OS/390 Assembler, DB2, C, C++, Ada, Java, Fortran, Pascal, HTML, Perl, JavaScript, Tcl, and more. Color coding definitions can be created for new languages or customized for specific variation of a language.
- **Embedded Language Support.** For example, Visual SlickEdit will color code SQL embedded in COBOL source. The embedded SQL support can be configured for DB2, PL/SQL, or SQL Server. In addition, Visual SlickEdit supports all HTML client-side and server-side embedded language syntax for JavaScript, Java, Perl, VBScript, and more.
- **SmartPaste™.** Supported for PL/I, C, C++, Java, PL/SQL, Slick-C, and Perl. When lines of text are pasted into a source file, the added lines are re-indented to intelligently match the surrounding code.
- **Selective Display of Procedures.** Supported for COBOL, PL/I, JCL, REXX, C, C++, Ada, Java, Fortran, Pascal, Perl, JavaScript, Tcl, and more. Shows just procedure headings and comments, and allows you to expand the heading to see the code.
- **C/C++/Java/JavaScript/HTML/C# Code Beautifier.** Beautifies your entire buffer or selected lines. See "HTML Beautifier" on page 139 or "C/C++/Java/JavaScript/C# Beautifier" on page 140.

Other Features

- **FTP Support.** Includes a complete FTP client and lets you easily open and edit remote files. The FTP client transparently and correctly performs ASCII/EBCDIC conversions. Visual SlickEdit FTP supports over 12 different FTP server types including OS/390, OS/400, VM, NT, UNIX. See "FTP Support" on page 159.
- **Customizable Dockable Toolbars.** Project toolbar with tabs for Class Browser, tags in current file, file open, project files, and FTP files. Output toolbar with tabs for tag symbol tracking, all occurrences search results, compilation output, symbol reference output, and JES job output. FTP Client toolbar for uploading and downloading remote files or entire directories.
- **Hex Editing.** View and edit files in hex. Supports inserting and deleting bytes.
- **3 Way Merge Editing.** Automatically merges two revision files with a base file. Collisions are color coded and you can interactively add changes to the output file while the files are being merged. See "3 Way Merge Editing" on page 69.

- **Code Block Selection.** Selects the lines in the current function, **for** loop, or **if** block, or other block structure with a single command. Then you can drag drop the code to another location.
- **Unlimited Undo and Redo.** Even undoes past saves.
- **No Limits.** Line length, file size, number of files, number of bookmarks, and number of windows only limited by system resources (i.e. disk space, memory).
- **Multi-file Search and Replace.** Search and replace across files. Also supports recursing subdirectories.
- **Multi-file Spell Checking.** Spell check comments and strings, the selection, the current word, or the whole document. Includes special HTML spell checking.
- **Line, Column, and Character Selections.** See "[Text Operations](#)" on page 76.
- **Command Line For the Seasoned Pros.** Supports argument retrieval and shelling commands just as you would type them at the operating system prompt. Invoke many editor commands from the command line. For more information see "Command Line and Text Box Features" on page 71.
- **Macro Recording.** Generates Slick-C source code you can edit, and supports dialog box interactions. Write Slick-C source code without understanding the language. See "Recording and Using Macros" on page 90.
- **Concurrent Process Buffer.** Provides a natural way to run command line compilers, linkers, and tools concurrently from within the editor. If you know how to use a shell command or operating system command line prompt to run a command line tool, just type in the name into the concurrent process buffer to run it. USS commands (such as `ls`, `cp`, `chmod`,...) can be executed from within the process buffer. See "Building from Within the Editor" on page 146.
- **Multiple Clipboards.** Paste from a stack of your most recent clipboards. See "Moving Text Among Files With Clipboards" on page 83.
- **Programmable File Manager.** Great for managing USS HFS files. See "[The Visual SlickEdit File Manager](#)" on page 160.
- **Project Management.** Organize your work into a neat package. In addition, this provides integration with OS/390 compilers. See "Projects" on page 32.
- **Alias Facility.** Defines aliases for directories which can be used by the Change Directory dialog box. Aliases can also be expanded with CTRL+SHIFT+O in any text box, combo box, or edit window. Even though aliases are typically used as shortcuts for long directories, they are also used to expand a short key sequence to any text string of any length. See "Directory Aliases" on page 20 and "Alias Facility" on page 154.

- **Fast Incremental Searching.** Quickly search for a string as you type. See "Incremental Searching" on page 95.
- **Advanced Text Box Editing.** Includes completion, aliases, multiple clipboards, cut line, cut word, uppercase word, and lowercase word. For more information, see "Command Line and Text Box Features" on page 71.
- **Flawless Emulations.** Emulates ISPF, BRIEF, GNU Emacs, VI, Epsilon, and Visual C++. Veteran OS/390 users will enjoy the ISPF emulation which includes the familiar prefix area and supports most SPF Edit line prefix commands and most primary commands.
- **Demand File Loading.** Edits bottom of a file, or anywhere in a file, without loading the entire file first.
- **Autosave.** Automatically backs up modified files at configurable intervals.
- **Document Math.** Just type some numbers into a file, select them, and use the **add** command ("Tools", "Add Selected Expr") to compute the sum. See "Calculating Mathematical Expressions" on page 88.
- **Multi-Platform Configuration.** Slick-C recorded macros, dialog boxes, and menus are source code compatible across all platforms. Slick-C macro Pcode is compatible across all platforms (except Visual SlickEdit OS/390 due to EBCDIC).

Data Sets

Visual SlickEdit allows you to edit sequential data sets and partitioned data set (PDS) members. You can open data sets using the Open dialog box, the command line, or the Data Set Utilities dialog.

Opening Data Sets

Perform the following steps to open a data set using the Open dialog:

1. Display the Open dialog (CTRL+O or "File", "Open...").
2. The syntax for specifying a sequential data set is **//dataset**. The syntax for specifying a PDS member is **//dataset/member**. For example, type the following into the "File Name" text box and press ENTER to edit the BPXPRM00 member of SYS1.PARMLIB:

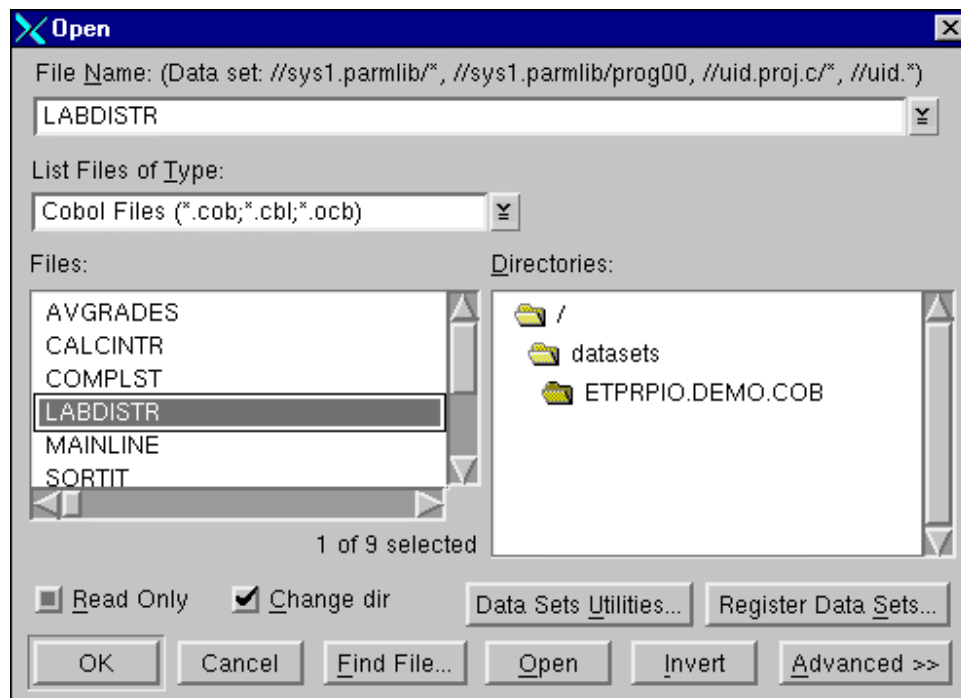
//SYS1.PARMLIB/BPXPRM00

Perform the following steps to open a data set using the Open dialog and the mouse:

1. Display the Open dialog (CTRL+O or "File", "Open...").

2. Double-click on the "/" directory in the "Directories" list box.
3. Double-click on the "datasets" directory in the "Directories" list box.
4. If the data set you are interested in does not appear in the Directories list box (a PDS) or in the File list box (a sequential data set), click on the "Register Data Sets..." button to add it.
5. To open a sequential data set, double-click on the data set name in the "Files" list box.
6. To open a PDS member, double-click on the PDS name in the "Directories" list box to display its members in the "Files" list box. Double-click on the member to open it.

Below is a picture of the Open dialog showing PDS ETPRPIO.DEMO.COB and its members. The dialog also indicates that member LABDISTR is about to be opened.



Perform the following steps to open a data set using the Visual SlickEdit command line:

1. Press ESC to activate the command line.
2. Use the **e** or **edit** command to open the file.

- The syntax for the **e** command is **e *filename*** where *filename* specifies a sequential data set or a PDS member.
- The syntax for specifying a sequential data set is **//dataset**.
- The syntax for specifying a PDS member is **//dataset/member**.

For example, to edit the BPXPRM00 member of SYS1.PARMLIB, type:
e //SYS1.PARMLIB/BPXPRM00 on the command line and press ENTER

Use the auto-completion keys (SPACEBAR and ?) to complete partial data set names and aliases (“Tools”, “Configuration”, “Aliases...”) to reduce key strokes.

Perform the following steps to open a data set using the Data Set Utilities dialog:

1. Display the Data Set Utilities dialog ("File", "Data Set Utilities..." or click the "OPENED MVS FOLDER" icon on the button bar, located below the editor's menu bar).
2. If the data set you are looking for is not in the data set list, type the prefix in the "Search:" text box and press ENTER for a list of matching data set names. (Be sure to put // in front of the prefix and end it with an * character.)

For example, for a listing of data sets that have the matching "SYS1.PARM" qualifiers type: **//SYS1.PARM***

3. To open a sequential data set, double-click on the desired data set, or select the data set and click "Open".
4. To open a PDS member, double-click on the PDS to show its member list then double-click on a member to edit. Or, select the desired member(s) and click "Open".

You can open multiple data sets or members by selecting multiple names and clicking "Open". Clicking "Browse" instead of "Open" opens a data set or PDS member in read-only mode.

Searching for Data Sets and PDS Members with the Open Dialog

Perform the following steps to search for a data set:

1. Display the Open dialog (CTRL+O or "File", "Open...").
2. Type // followed by a data set name containing a wild card character in the "File Name" text box and press ENTER.

The following characters have special meaning when entered as part of a file name:

- | | |
|---|-------------------------------|
| * | Matches 0 or more characters. |
| ? | Matches any character. |

[<i>char-set</i>]	Matches any one of the characters specified by <i>char-set</i> . A '-' character may be used to specify ASCII (not EBCDIC) ranges. The expression [A-Z] matches any uppercase letter. You can use '\' inside the square brackets to define literal characters or define ASCII characters. For example "\-" specifies a literal dash character.
--------------------------	--

File Name Wild Card Examples

//sys1.*	Matches data sets starting with "sys1."
//p*a.b.*	Matches data sets starting with "p" that have "a.b." in the middle.
//sys1.maclib/a*	Matches all members of SYS1.MACLIB that start with the letter "a".
//dataset/vs*xy	Matches all members starting with "vs" and ending with "xy".
//dataset/a?c	Matches member names that start with "a", end with "c", and are exactly 3 characters long. For example, this matches "abc" or "axc".
//dataset/a[a-c]c	Matches member names "aac", "abc", and "acc".

Searching for Data Sets and PDS Members with the Listds Command

Perform the following steps to search for a data set:

1. Press ESC to place the cursor on the command line.
2. Type **listds**<SPACEBAR> followed by a wild card data set name and press ENTER. A list of matches will be shown in the Data Set Utilities dialog.

The following characters have special meaning when entered as part of a data set name:

*	Matches 0 or more characters.
?	Matches any character.
[<i>char-set</i>]	Matches any one of the characters specified by <i>char-set</i> . Use a '-' character to specify ASCII (not EBCDIC) ranges. The expression [A-Z] matches any uppercase letter. Use '\' inside the square brackets to define literal characters or define ASCII characters. For example: "\-" specifies a literal dash character.

Wild Card Examples

listds //sys1.*	Matches data sets starting with "sys1."
listds //p*a.b.*	Matches data sets starting with "p" that have "a.b." in the middle.
listds //sys1.maclib/a*	Matches all members of SYS1.MACLIB that start with the letter "a".
listds //dataset/vs*xy	Matches all members starting with "vs" and ending with "xy"
listds //dataset/a?c	Matches member names that start with "a", end with "c", and are exactly 3 characters long. For example, this matches "abc" or "axc".
listds //dataset/a[a-c]c	Matches members names "aac", "abc", and "acc".

Searching for Data Sets with the Edit Command

The **e** or **edit** command supports a feature called “completion,” a mechanism which helps you type a command or argument to a command. When you press SPACEBAR on either the command line or in a dialog input field, Visual SlickEdit attempts to insert (complete) as much of the argument as possible. For example:

1. Press ESC to place the cursor on the command line
2. Type **e //sys1.parm**
3. Press SPACEBAR. You will see the word **//sys1.parm** expanded to **//sys1.parmlib**.

You can also press the question mark key “?” to display a list of matches to an argument you have partially typed, and also if you see the message “Word expanded as much as possible” after you press SPACEBAR while entering an argument. Completion is especially useful to list members of a PDS. For example:

1. Press ESC to place the cursor on the command line
2. Type **e //sys1.parmlib/?**
3. You should now see a list of the members of SYS1.PARMLIB
4. To open a member in the list, press the up and down arrow keys to select and press ENTER

With completion, you can quickly and correctly enter an argument without having to remember the argument's exact name or spelling. Other dialogs which support completion are as follows:

- Open
- Save As
- Go To Definition
- Change Directory
- Bind Command to Key

Multi-file Search and Replace

Both the Find and Replace dialogs ("Edit", "Find..." or "Edit", "Replace...") have a "Files/Buffers>>" button which expands the dialog to show advanced options. You can perform a search and/or replace operation through all members of a PDS by entering **//dataset/*** in the Files text box. The "*" is a wild card file name specification. The following characters have special meaning when entered as part of a file name:

*	Matches 0 or more characters.
?	Matches any character.
[<i>char-set</i>]	Matches any one of the characters specified by <i>char-set</i> . Use a '-' character to specify ASCII (not EBCDIC) ranges. The expression [A-Z] matches any uppercase letter. Use a '\' character inside the square brackets to define literal characters or define ASCII characters. For example: "\-" specifies a literal dash character.

File Name Wild Card Examples

//dataset/*	Matches all members.
//dataset/vs*	Matches all members starting with a "vs" prefix.
//dataset/*xy	Matches all members ending with a "xy" suffix.
//dataset/vs*xy	Matches all members starting with "vs" and ending with "xy".
//dataset/a?c	Matches member names that start with "a", end with "c", and are exactly 3 characters long. For example, this matches "abc" or "axc".
//dataset/a[a-c]c	Matches member names "aac", "abc", and "acc".

Data Set Utilities

The **Data Set Utilities dialog box** ("File", "Data Set Utilities..." or clicking the "OPENED MVS FOLDER" icon on the button bar, located right below the editor's menu bar) allows you to perform various data set management tasks, including:

- Open/Browse, copy, delete, rename, catalog, uncatalog, and free data sets
- Allocate new data sets
- Submit sequential data sets to JES
- Compress PDS
- Query detailed data set information
- Open/Browse, copy, delete, rename PDS members
- Submit PDS members to JES

To list data sets, type the data set name or name prefix in the "Search:" text combo box and press ENTER. You can also click on the combo box's "down-arrow" button and select from a list of previously entered data set names. Multiple "*" can be used. For example, type `//USER.PROJECT.*` to get a list of data sets with matching qualifiers "USER.PROJECT". (You can type `//*` to get a listing of all catalogued data sets on your system, although this is not recommended.)

For a PDS member list, double-click on the PDS name in the list, or select the PDS and click "Open". You can also list the members of a PDS by typing the PDS name followed by the "*" in the "Search:" text box. For example: type `//USER.PROJECT.COB/*` to get the members of PDS "USER.PROJECT.COB". From the PDS member listing, press ESC or "<<Back" to go back to the previous data set listing.

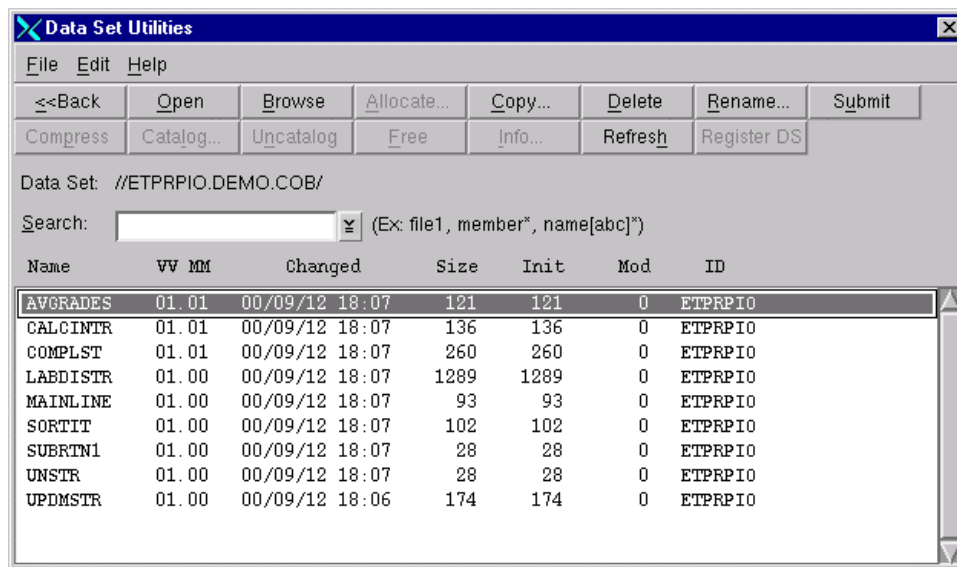
To open a sequential data set, double-click on the data set, or select the data set and click "Open". A PDS member can be opened in a similar manner.

To open multiple PDS members, select the desired members and click "Open". Press CTRL and click on a member to add it to or remove it from the existing selection. Press SHIFT and click a member to select a range from the member last selected.

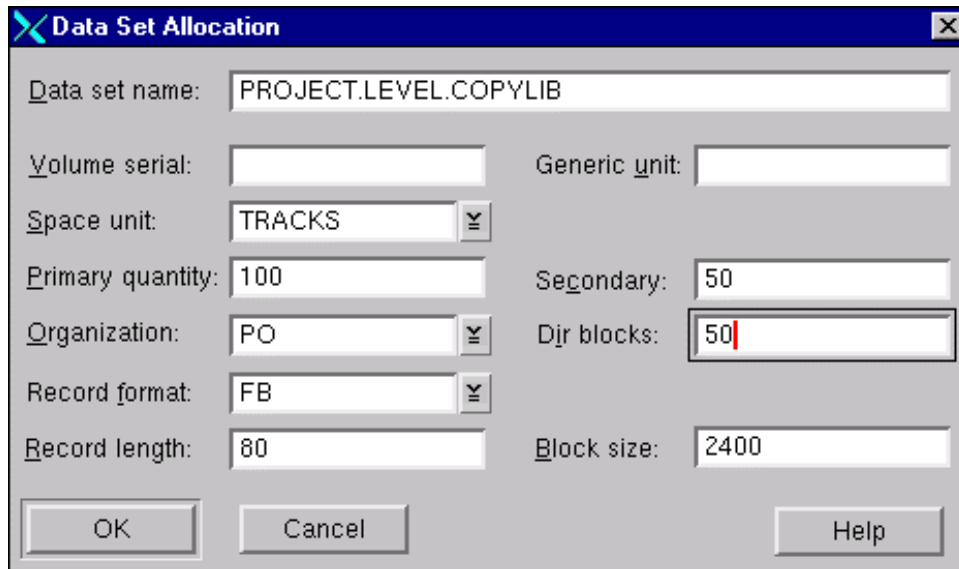
Below is a picture of the Data Set Utilities dialog showing the list of data sets with matching prefix “SYS1.AB”. In addition to the data set name, the volume serial, data set organization, record format, record length, and block size are also shown.

Data Set Name	Volume Serial	Organization	Record Format	Record Length	Block Size
//SYS1.ABDTCLIO	VDMVSD	P0	FB	80	27920
//SYS1.ABDTHELP	VDMVSD	P0	FB	80	27920
//SYS1.ABDTMAC	VDMVSD	P0	FB	80	27920
//SYS1.ABDTMOB	VDMVSD	P0	U	0	13030
//SYS1.ABDTMSG	VDMVSD	P0	FB	80	27920
//SYS1.ABDTPNL0	VDMVSD	P0	FB	80	27920
//SYS1.ABDTSAMP	VDMVSD	P0	FB	80	27920
//SYS1.ABLSCLIO	VDMVSB	P0	FB	80	28000
//SYS1.ABLSKEL0	VDMVSB	P0	FB	80	6160
//SYS1.ABLSMSG0	VDMVSB	P0	FB	80	6160
//SYS1.ABLSPNL0	VDMVSB	P0	FB	80	28000
//SYS1.ABLSTBL0	VDMVSB	P0	FB	80	6160
//SYS1.ABMFMOD0	VDMVSD	P0	U	0	6144
//SYS1.ABPXEKEC	VDMVSB	P0	FB	80	6160
//SYS1.ABPXMENU	VDMVSB	P0	FB	80	6160
//SYS1.ABPXMOD1	VDMVSB	P0	U	0	6144
//SYS1.ABPXPENU	VDMVSB	P0	FB	80	28000
//SYS1.ABPXSPC	VDMVSB	P0	U	0	6144
//SYS1.ABPXTENU	VDMVSB	P0	FB	80	6160

Below is a picture of the Data Set Utilities dialog showing members of ETPRPIO.DEMO.COB.



Below is a picture showing the new PDS PROJECT.LEVEL.COPYLIB being allocated.



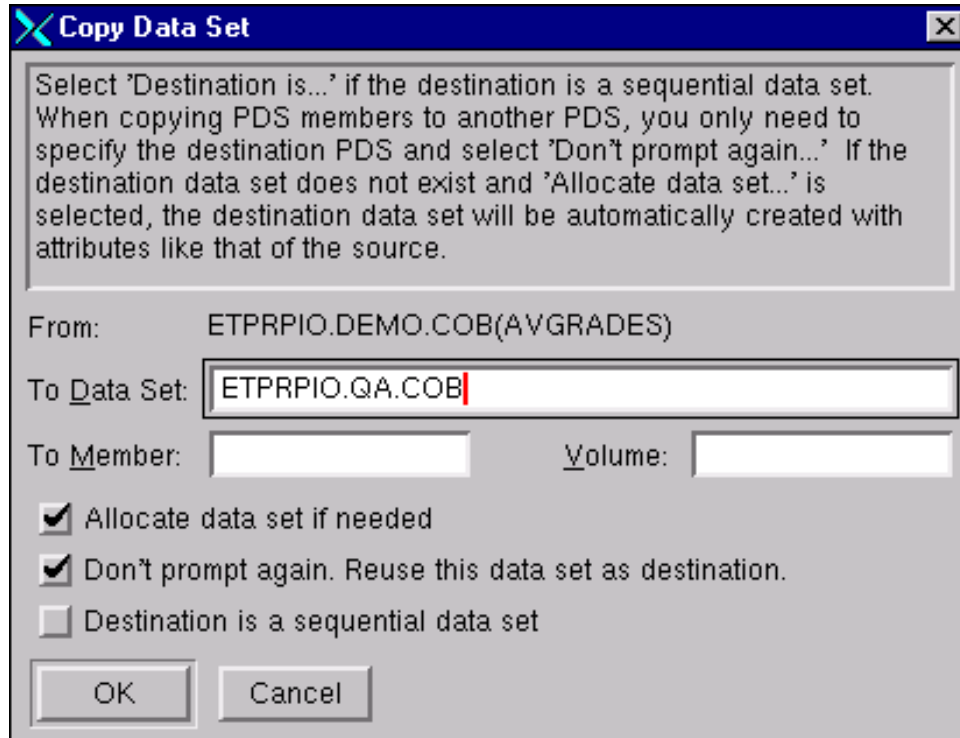
The image shows a 'Data Set Allocation' dialog box with the following fields and values:

Field	Value
Data set name:	PROJECT.LEVEL.COPYLIB
Volume serial:	
Generic unit:	
Space unit:	TRACKS
Primary quantity:	100
Secondary:	50
Organization:	PO
Djr blocks:	50
Record format:	FB
Record length:	80
Block size:	2400

Buttons: OK, Cancel, Help

Below is a picture showing member ETPRPIO.DEMO.COB(AVGRADES) being copied to another PDS ETPRPIO.QA.COB.

NOTE: If you copy multiple members the “Don’t prompt again. Reuse this data set as destination” check box will be automatically selected. If you want to copy members to a different destination data set deselect this box and you will be prompted for the “To Data Set:” for each member.



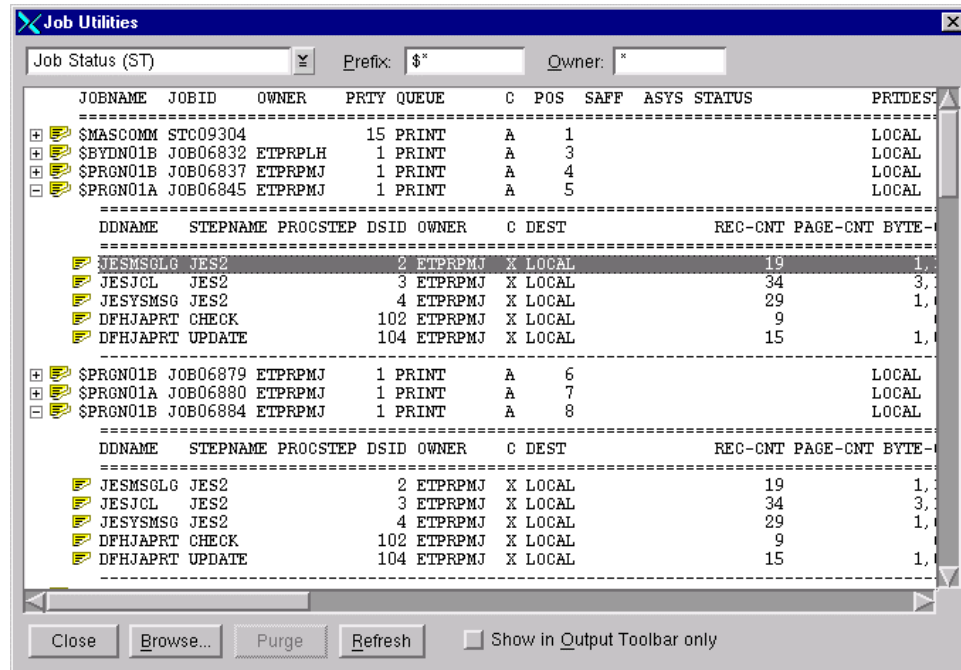
Job Utilities

The Job Utilities dialog lets you manage your jobs in the JES queues. With this dialog, you can perform the following actions:

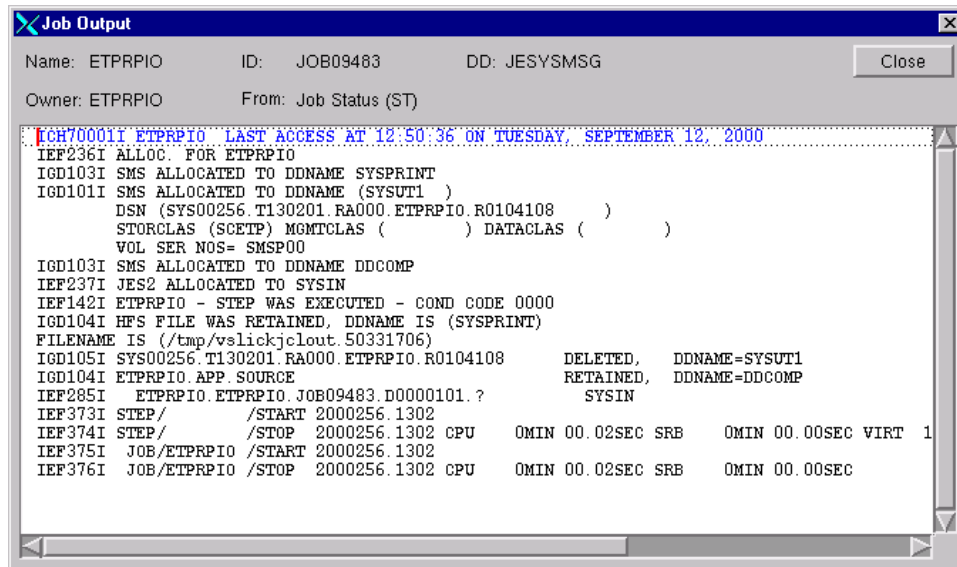
- List jobs from the input, output, and held output queues
- List active jobs
- List jobs status
- Simultaneously list SYSOUT DDNAMEs of multiple jobs
- Browse and search the full output of a job or a job’s SYSOUT DD
- Purge jobs

To bring up the Job Utilities dialog, go to “File” and select “Job Utilities...” or click the “Job Output” button on the button bar (located right below the editor’s menu bar).

Below is a picture of the Job Utilities dialog showing status for jobs with matching prefix “\$”. Note that SYSOUT DDNAMEs of multiple jobs can be listed at the same time.



Below is a picture showing the content of “JESYSMSG” DD from job JOB09483.



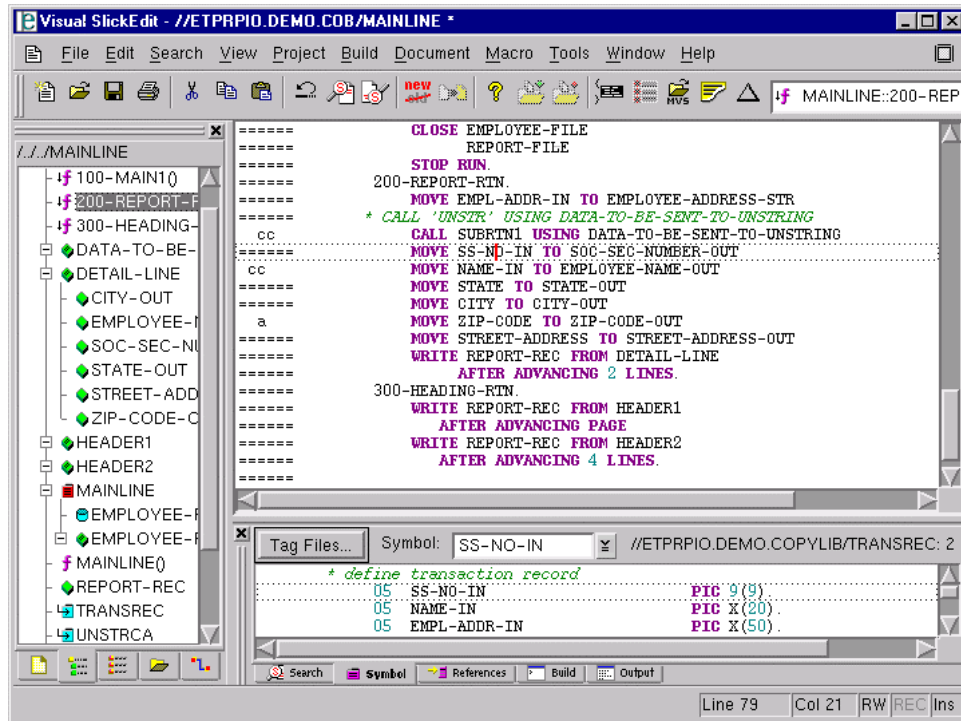
Invoking Visual SlickEdit from TSO

Visual SlickEdit provides a REXX script called "runvs.rexx" (installed in "/usr/lpp/vslicck/bin" by default) which you can use to start Visual SlickEdit from a TSO session. Read the comment at the top of this file for instructions on installing and running this REXX script. Visual SlickEdit performance may be better if you can increase the TSO region size using the SIZE parameter on the TSO/E Signon screen (where you enter your TSO password). Check with your system programmer before increasing this value.

Using Visual SlickEdit ISPF Emulation

For veteran System/390 users, Visual SlickEdit provides emulation for the standard IBM ISPF editor on OS/390. The emulation does not replace ISPF; rather, it mirrors the look and feel of many ISPF functions so that programmers who are familiar with this style of editing can immediately be productive with Visual SlickEdit.

Below is a picture showing Visual SlickEdit in ISPF emulation.



Compiling COBOL, Assembly, PL/I, and REXX Applications

Before you can compile your code, you need to create a project (see [“Creating Projects”](#) on page 32) and select a compiler package. You do not have to add source files to your project in order to compile your source files. However, adding files to a project lets you take full advantage of other features provided by projects such as Context Tagging™ (see [“Context Tagging\(TM\)”](#) on page 45) and the Class Browser (see [“Class Browser”](#) on page 60). A few dialogs take advantage of your project source files, such as Load Files (Project menu), Find, Replace, and Print.

See [“Creating Projects”](#) on page 32 for information on creating a project and adding files to a project.

The following compilers are already supported:

- IBM High Level Assembler
- IBM VS COBOL II
- IBM VS FORTRAN
- IBM PL/I OPTIMIZING COMPILER

- IBM C/370
- IBM OS/390 C/C++

Once you have created a project, you can compile your source file by opening the file and invoking the Compile menu item (SHIFT+F10 or "Build", "Compile"). This will activate the "Build" tab on the Output toolbar (docked at the bottom of the Visual SlickEdit screen by default). Compiles occur concurrently by default.

NOTE: For custom compile/build environments or one that involves multiple steps, Visual SlickEdit provides the "vssubmit.REXX" tool which allows direct compiles and builds from within Visual SlickEdit projects. Please contact SlickEdit Inc. for assistance on customizing and using vssubmit in your environment.

Testing Your Application

You can use the vssubmit tool to modify and submit batch test jobs while you are looking at or modifying an input data set for that job. For example, you could have a PDS setup with different run parameters in different members, then, while editing a member that contains weekly run parameters, you can invoke the vssubmit tool which will submit the batch job pointing to that member. Use the Job Utilities dialog (JES queue interface) to view the batch job results.

Differencing Data Sets

DIFFzilla™ allows you to easily view and merge differences in sequential data sets or PDS members.

To difference two sequential data sets or two PDS members:

1. Display the DIFFzilla™ dialog (CTRL+= or "Tools", "File Difference..." or click the "delta triangle" button on the button bar located right below the editor's menu bar).
2. Enter the first data set name in the "Path 1" combo box (e.g. `//sys1.maclib/open`).
3. Enter the second data set name in the "Path 2" combo box.
4. Click OK.

To difference all or some of the members of two PDSs:

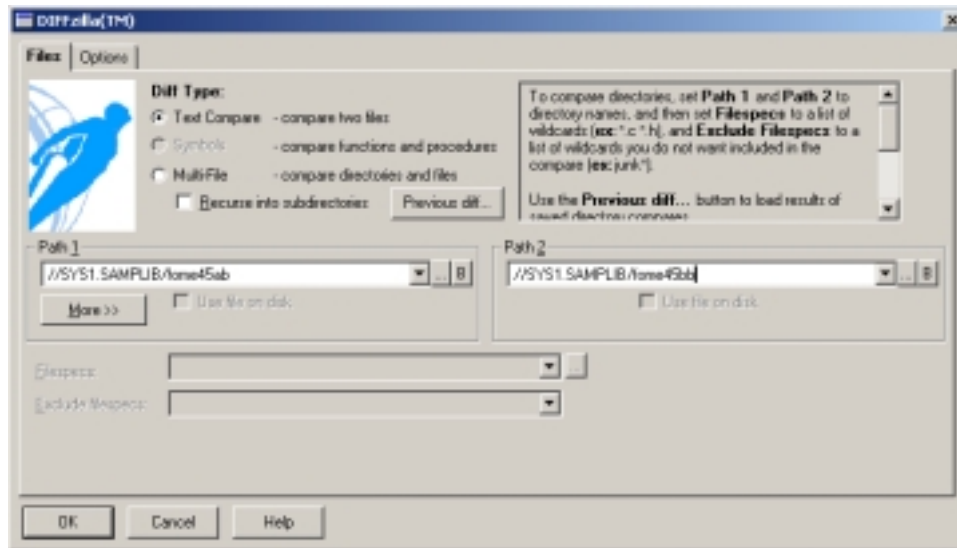
1. Display the DIFFzilla™ dialog (CTRL+= or "Tools", "File Difference..." or click the "delta triangle" button on the button bar located right below the editor's menu bar).
2. Enter the first PDS name in the "Path 1" combo box (for example: `//user.src1.cob`).

3. Enter the second PDS name in the "Path 2" combo box (for example: **//user.src2.cob**).
4. To compare all members, type "*" in the "Filespecs" combo box.

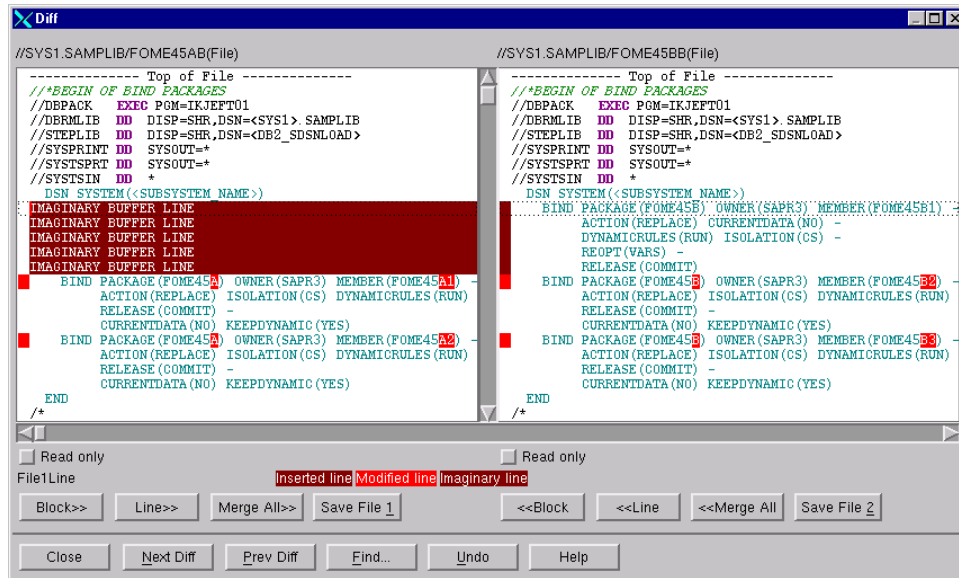
The "Filespecs" combo box supports multiple wild card member specifications. For example, **file1 file2 filea*** selects the files "file1" and "file2" and all members starting with the prefix "filea". One less obvious feature is that you can enter ***.cob**, ***.pl1**, ***.s**, ***.c**, ***.jcl**, or ***.rexx** to select members that Visual SlickEdit recognizes as COBOL, PL/I, Assembler, C/C++, JCL, or REXX language files respectively.

5. Click OK.

Below is a picture of the DIFFzilla™ dialog showing that PDS members SYS1.SAMPLIB(FOME45AB) and SYS1.SAMPLIB(FOME45BB) are about to be compared to each other.



Below is a picture of the side-by-side difference editor showing the differences between two JCL members SYS1.SAMPLIB(FOME45AB) and SYS1.SAMPLIB(FOME45BB). Each highlighted difference can be merged from one side to the other. When you type changes in either window, the difference editor updates the highlighting as you type. More powerful editing capabilities such as auto-list, auto-completion, aliases and more, are also available in the difference editor windows.



Preserving Column of Line Numbers

While other graphical editing tools that edit mainframe code do not prevent the line numbers (typically in columns 73-80) from changing column, all Visual SlickEdit features, including search and replace, preserve the line numbers column. You can use insert mode without shifting your line numbers. Use the Extension Options dialog ("Tools", "Configuration", "File Extension Setup...", select the General tab) to reconfigure the default truncation length for a particular language. To change the truncation length for the current file, use the "trunc" command and specify a column. The command "trunc 0" turns truncation off. Press ESC to get to the Visual SlickEdit command line.

OS/390 Version Control

Please contact SlickEdit Inc. about support for your version control system.

Configuring Visual SlickEdit

One of Visual SlickEdit's strong points is its degree of configurability. You can configure Visual SlickEdit the way you want to work.

Almost all of Visual SlickEdit's configuration options are available from the configuration menu ("Tools", "Configuration"). If multiple users are accessing Visual SlickEdit on a network, you will want each user to define a VSLICKCONFIG environment variable which refers to a local directory. This allows each user to have their own configuration. If a user wants to make modifications to "vslick.ini", he/she should make a local copy of this file and place it in the VSLICKCONFIG directory.

Please read "**Quick Start**" on page 13, which covers the most common configuration changes. This "Configuring Visual SlickEdit" section is larger and covers almost all configuration options.

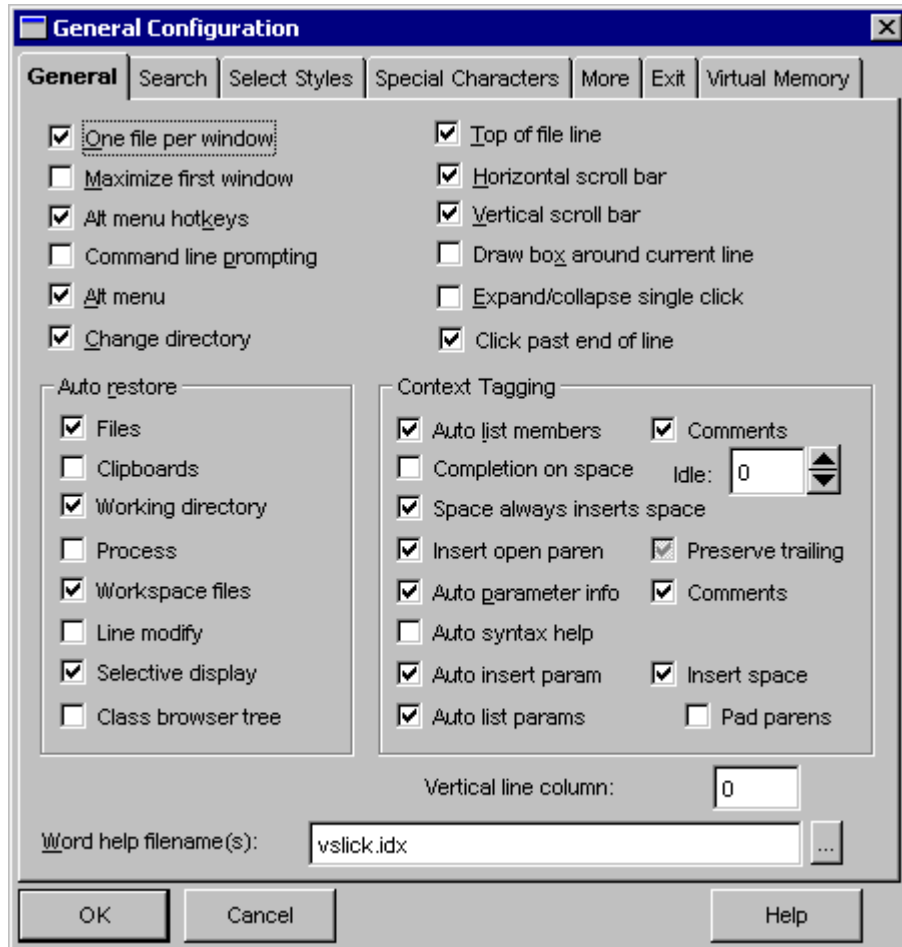
The following configuration items have already been covered in previous sections: **Selecting an Emulation** on page 13, **Color Coding** on page 137, and **Alias Facility** on page 154.

General Configuration Dialog Box

("Tools", "Configuration", "General...")

General Tab

Below is a picture of the General Configuration dialog box.



Select the various options you want and click OK to apply the configuration changes. Your configuration changes for this dialog box are NOT saved on disk until you exit the editor. By default, your configuration changes are automatically saved without prompting you. If you want to be prompted to save your configuration changes when you exit the editor, click the "Exit" tab and turn on "Always prompt before saving configuration".

One file per window

Affects whether the Open dialog box (CTRL+O) creates a window for each file opened. If you are used to editors such as SlickEdit, Brief, VI, ISPF, EMACS, and PE you will not want this feature checked.

Maximize first window

When checked, the first MDI child edit window is maximized when it is created. This option has no effect on **Auto Restore**.

Alt menu hotkeys

This option is grayed if you are using CUA emulation. If you want ALT and a selected menu bar character to display the corresponding drop-down menu, check the "Alt Menu Hotkeys" check box. If you do not check this box you can be more selective about key bindings because you are permitted to bind ALT keys you normally could not, such as ALT+F. If you choose to bind ALT keys that are normally menu keys, do not turn on this check box because you will lose those key bindings.

Command line prompting

Many of Visual SlickEdit's commands which display dialog boxes have equivalent commands which prompt for arguments on the command line. If you want faster prompting than the dialog boxes allow, check command line prompting. You can be more selective than this check box permits by changing your key bindings. For example, you might only want to be prompted on the command line when you open files. This can be done by binding the **edit** command to CTRL+O which is bound to the **gui_open** command by default.

The following is a partial list of graphical commands and their command line counterparts:

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

Alt menu

If you prefer not to be unintentionally placed on the menu bar when you press the ALT key without following it with another key, uncheck this box.

Change directory

This check box affects whether changing the directory in the directory list box and the drive combo box in the Open and Save As dialog boxes change the current directory. When checked, the editor will change the current directory.

Top of file line

When checked, each buffer displays a line 0 which contains the text "Top of File". This line is useful for inserting a line before the first line of a buffer when in CUA or SlickEdit

emulation. It is also useful in Selective Display because a '+' bitmap can be displayed on line 0.

Horizontal scroll bar

When checked, each edit window displays a horizontal scroll bar. This check box does not affect edit window controls on dialog boxes.

Vertical scroll bar

When checked, each edit window displays a vertical scroll bar. This check box does not affect edit window controls on dialog boxes.

Draw box around current line

When checked, a rectangle is drawn around the current line. This makes it easier for you to tell where the cursor is.

Expand/collapse single click

When checked, selective display plus/minus bitmaps can be expanded or collapsed with a single click. This makes selective display operate similar to the Windows Explorer. However, when this is on, you will not be able to select a line by clicking to the left of a text line which contains a selective display bitmap.

Click past end of line

When this box is checked, you can place the cursor with the mouse past the end of a line.

Auto list members

When on, members are listed when you type a member access operator ('.', '>', and '::' for C++, '.' for Java, "IN" and "OF" for COBOL). You can also access this feature on demand by pressing ALT+PERIOD. This helps you with identifiers when there is no member operator (list locals, global variables, current class members, etc.). See "[Context Tagging\(TM\)](#)" on page 45 for more information.

Comments

When on, comments are displayed for the currently selected symbol in the list.

Idle

Idle time in milliseconds before Auto List Members displays the list. If you do not want the Auto List Members list to appear when you are typing fast, set this to about 300 milliseconds. Then, if you finish typing the member before you stop typing for .3 seconds, the list will never appear.

Completion on space

When on, pressing SPACEBAR when list members is displayed performs completion. For example, if the list contains "FLAG_CHAR" and "FLAG_LONG", then typing **FL**<ALT+PERIOD><SPACEBAR> will complete up to "FLAG_". If this is not checked, you can always use CTRL+SPACE when list members is displayed to perform completion.

Space always inserts space

When on, pressing SPACEBAR when list members is displayed inserts the current item in the list and inserts a space after it.

Insert open paren

When on, selecting an item in the list inserts the current item in the list and any extra characters typically required by the symbol. For example, an open parenthesis is inserted after a function name for languages which require an open parenthesis after a function name. For C++, a less than ('<') character is inserted after a template class name.

Preserve trailing

When on, only identifier characters before the cursor are replaced with an item selected from a list members dialog. Identifier characters after the cursor are preserved. When off, identifier characters following the cursor are replaced with the item selected from a list members dialog. When grayed, trailing identifier characters are preserved for “auto” list members but not when listing members on demand (ALT+PERIOD).

Auto parameter info

When on, Visual SlickEdit automatically displays the prototype and comments for a function when a function operator ('(' for most languages) is typed and highlights the current argument within the displayed prototype. To access this feature on demand, press ALT+COMMA. See "[Context Tagging\(TM\)](#)" on page 45 for more information.

Comments

When on, comments are displayed for when **Parameter Info** appears.

Auto syntax help

When on, Visual SlickEdit automatically displays help for statement syntax. Currently this only affects COBOL. For COBOL, statement syntax is display when you press SPACEBAR. For example, when you type **read<space>** help on the READ statement is displayed.

Auto insert param

When on, Visual SlickEdit automatically inserts the name of a parameter of the appropriate type/class if one exists in the current scope. Currently, this only affects C, C++, Java, C#, and Slick-C.

Insert space

When on, a space is inserted after a ',' when a parameter name is automatically inserted (e.g. myfun(a, b, c)). Currently this only affects C, C++, Java, C#, and Slick-C.

Pad parens

When on, a space is inserted after an open parenthesis when a parameter name is automatically inserted. A space is also inserted before a closed parenthesis which is typed during auto parameter info. (e.g. myfun(a)). Currently this only affects C, C++, Java, C#, and Slick-C.

Auto list params

When on, Visual SlickEdit automatically lists compatible parameters when entering arguments to a function/method. Global (non-module) variables are not listed. Currently this only affects C, C++, and Java.

Word help filename

Windows only: Specifies the word help file names used by the **wh** command (CTRL+F1), **wh2** command (CTRL+F2), and **wh3** command. Specify the help files for the **wh** command (CTRL+F1) first, each separated with a plus sign. Then use a semicolon to indicate the start of help files for the **wh2** command (CTRL+F2) and specify its help files separated with a plus sign. You can use another semicolon to indicate the start of help files for the **wh3** command. However, this command is not bound to a key.

Windows: Help files have the extension .MVB, .HLP, or .IDX. Do not use a plus sign to specify more than one .IDX file or .MVB. See "[Windows Word Help](#)" on page 21 for help on building an .IDX file.

Windows Example:

```
CtrlF1a.hlp+CtrlF1b.hlp;CtrlF2.hlp
vslick.idx;win31wh.hlp+mscxx.hlp+errors.hlp+vslick.hlp
CtrlF1.idx;CtrlF2.idx
vslick.idx;c:\msdev\help\vcbks40.mvb
```

Files

When on, Visual SlickEdit will restore the files and windows that were opened in your last edit session. When this is off, your files and windows will not be restored when you start the editor even if you have "Project Files" checked on.

Clipboards

When on, clipboards are saved across edit sessions.

Working directory

When on, the working directory is saved across edit sessions.

Process

When on, the concurrent process buffer is saved across edit sessions.

Workspace files

When on, switching workspaces will restore the files and windows that were opened for that workspace when the workspace was closed. If you do not want any files and windows closed or opened when switching workspaces, turn this check box off.

Line modify

When on, the line modify flags are saved and restored when you save and open files. This option has little use unless you turn on "Modified Lines" color coding in the [Extension Options dialog box](#) ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab). Line modify information for the last 200 files is saved. Visual SlickEdit stores line modify information in temporary files placed in a "SelDisp" directory.

Selective Display

When on, Selective Display is saved and restored when you save and open files. Selective Display for the last 200 files is saved. Visual SlickEdit stores selective display information in temporary files placed in a "SelDisp" directory.

Class browser tree

When on, the class browser tree is restored across edit sessions. Note that the current position (displayed selected) is always restored regardless of this setting.

Vertical line column

Specifies the editor window text column which should display a vertical line. This text box is typically set to 80 to help you determine when you have typed text past column 80. Specify 0 if you do not want a vertical line displayed.

Search Tab

("Tools", "Configuration", "General...", select Search **tab**)

Match case

When checked, various search commands default to case-sensitive searches.

Regular expression

When checked, various search commands default to regular expression searching. See "[Regular Expressions](#)" on page 107.

Wrap at beginning/end

When checked, various search commands default to wrapping to the beginning or end of a buffer to complete a search.

Restore cursor after replace

When checked, the cursor is restored to its original position after a search and replace operation completes and is not canceled.

Leave selected

When checked, the last occurrence of a search string that was found is left selected. This also affects whether pressing ESC during a search and replace leaves the search string selected.

Regular expression syntax

When you choose a regular expression syntax, all search commands will use this syntax when regular expressions are turned on.

NOTE: The **/**, **find**, and **c** commands allow you to explicitly specify UNIX, Brief, or SlickEdit regular expressions from the command line.

Initialization options

Specifies how the Find dialog box "Search For:" combo box is initialized.

Expand dialog

When checked, the Find and Replace dialogs are displayed initially expanded.

Select Styles Tab

("Tools", "Configuration", "General...", select Select Styles tab)

You will find that Visual SlickEdit has almost every text selection style. The "Select Styles" tab provides the following options:

User defined

When this is on, it means that your current selection style does not exactly match CUA or the emulation default style. This is disabled if you are in Brief or EMACS emulation.

CUA (default)

In this style, selected text is deleted before a paste or character insert unless the selection is locked. Pressing BACKSPACE or DEL 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 Visual SlickEdit emulation commands **select_line**, **select_block**, or **select_char** to be executed. You can access these commands from the Select menu ("Edit", "Select...") in any emulation.

Emulation Default

This selection style is the emulation default. It is disabled when in CUA or Visual C++ emulation.

Shift+Cursor always char select

When this is off, SHIFT+<ARROW KEYS> will perform line or block selections, depending upon the direction you move the cursor. This is disabled if you are in Brief emulation.

Mouse selection creates clipboard

Turn on this check box if you like the way X windows programs let you use the left mouse button to create a clipboard and then use the middle mouse button to paste.

Extend selection as cursor moves

When on, the selection is extended to cursor position. This is disabled if you are in Brief or EMACS emulation.

Deselect after paste

Indicates whether pasted text is selected. This is disabled if you are in Brief or EMACS emulation.

Inclusive character selection

Indicates whether a character selection includes the character following the cursor. This is disabled if you are in Brief or EMACS emulation.

Delete selection before insert

Indicates whether a selection is deleted before new text is inserted. This is disabled if you are in Brief or EMACS emulation.

Auto deselect (i.e. when cursor moves)

Indicates whether a selection should be cleared when the cursor moves or one of a few other editor operations occurs. This is disabled if you are in Brief or EMACS emulation.

Special Characters Tab

("Tools", "Configuration", "General...", select Special Characters tab)

This tab lets you choose how special characters such as Carriage Return, Line Feed, Tab, Space, Virtual Space, and End of File characters are displayed. How they appear depends on the "Window Text" font. These characters are only displayed when the View menu has special viewing turned on for one or more of these special characters. Select the Special Chars menu item ("View", "Special Chars") to turn on viewing of all special characters.

If you want to be able to tell the differences between a DOS format text file and another format when View New Line Chars is turned on, select something other than space for the End-Of-Line (2) character. Under Windows, we recommend 13 for End-Of-Line (1) and 10 for End-Of-Line (2).

More Tab

("Tools", "Configuration", "General...", select More tab)

Start in insert mode

When checked, the insert mode is set to "insert" when the editor is invoked. Otherwise, the insert mode is set to "replace."

Throw away file lists

When on, modified file manager file lists can be modified and closed without being prompted to save.

Auto exit process

When on, the concurrent process buffer is automatically exited when the buffer is closed or when exiting the editor.

Edit "A B C" start on file A

When on, and you select more than one file to Open, the first file becomes the active buffer.

Short key names

When checked, key names in the MDI menu bar are condensed (non-CUA). For long CUA key names, turn this check box off.

Reflow next

When on, the **reflow_paragraph** command ("Document", "Format Paragraph") places the cursor on the next paragraph after it has reformatted the current paragraph. Otherwise, the cursor is kept at the same location within the current paragraph.

Protect read-only mode

When on, the editor will not let you modify a file that is in read only mode. The save command always prompts you for a different output file name if the file is in read only mode.

CUA Text Box

When this box is checked, the keys CTRL+X, CTRL+C, and CTRL+V perform **cut**, **copy**, and **paste** commands respectively for a text box other than the command line. In addition, the dialog manager takes over the keys ALT+A..ALT+Z for selecting controls. DO NOT check this box if you want all keys to operate the same in a text box as the command line and edit windows.

Paren Match Style

The **keyin_match_paren** command must be bound to one of the keys),], or } for the Paren match style to have an effect. This feature always uses fast brace matching.

Paren Match Style	Effects
Highlight	Typing a closing parenthesis temporarily block selects the text within the parenthesis pair.
Cursor to Begin Pair	Typing a closing parenthesis temporarily places the cursor on the matching begin parenthesis.
None	Typing a closing parenthesis just inserts the close parenthesis.

Scroll Style

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

NOTE: Commands that move the cursor more than one page of text, such as searching, always center scroll text into view.

Preserve column on top/bottom

When on, the **top_of_buffer** (CTRL+HOME) and **bottom_of_buffer** (CTRL+END) commands do not change the column position unless you are already at the top or bottom of the buffer.

Show tool tips

When checked, pop-up tool tip help messages are displayed when you hold your mouse pointer over a button bitmap.

Delay

Specifies delay in tenths of a second before tool tip messages are displayed.

CR w/o LF erases line in process buffer

If you find that lines are getting erased in the concurrent process buffer, turn this option off.

Key message delay

Specifies the delay before a prefix key is displayed in tenths of a second. The prefix key is not displayed if the next key is pressed before the delay specified in this text box.

Hide mouse pointer

Windows only: If checked, the mouse pointer is hidden when you start typing. The mouse pointer reappears when you move the mouse or a dialog box is displayed.

Allow drag drop text

When checked, selected text may be copied or moved by dragging and dropping the selected text with the left mouse button.

Windows 3.1 style open dialog

Windows only: When checked, a Windows 3.1 style open dialog box is used to open and save files. This dialog does not support all the features of the default open dialog (e.g. the new encoding options are not supported).

Line insert style

When the line insert style is “Before,” commands which insert lines of text are inserted before the current line. Otherwise, lines of text are inserted after the current line.

Next word style

When the next word style is “begin”, the **next_word** command (CTRL+RIGHT) places the cursor on the beginning of the next word. Otherwise, the cursor is placed at the end of the next word.

Smart next window

Windows only: When on and CTRL+TAB is bound to the **next_window** command (the default binding), CTRL+ TAB can be used to switch between two edit windows when you have more than one edit window. This is how CTRL+ TAB works in Visual C++ and how ALT+ESC works when switching applications in Windows.

Use block cursor

When on, a text mode style cursor is used instead of a vertical cursor.

Window left margin

Specifies the space between the left edge of the window and the editor text in inches. This value has no effect when there are bitmaps displayed in the left margin since more space is necessary.

Max clipboards

Specifies the maximum number of clipboards saved. By default, Visual SlickEdit keeps a stack of your last 15 clipboards, any one of which can be pasted with CTRL+SHIFT+V ("Edit", "List Clipboards...").

Exit Tab

("Tools", "Configuration", "General...", select Exit tab)

Always save configuration

(Default) When on, configuration changes are saved without prompting.

Always prompt before saving configuration

When on, you will be prompted whether to save configuration changes.

Exit confirmation prompt

When on, you will be prompted whether you really want to exit the editor.

Virtual Memory Tab

("Tools", "Configuration", "General...", select Virtual Memory tab)

Spill File Path

This text box specifies a directory where spill files and temporary files should be placed.

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.

UNIX: This defaults to the directory specified by the TMP environment variable.

Buffer cache size

Specifies the maximum amount of memory used to store text buffer data in kilobytes. A value less than 0 specifies all available memory. We recommend a value that is about one quarter the size of available memory for optimal performance. If you have 16 MB of memory, 4000 is a good choice.

IMPORTANT: If the operating system starts swapping before Visual SlickEdit's cache is full, performance will be about three times worse. Make sure this cache size is smaller than the amount of actual memory you have.

State cache size

Specifies the maximum amount of swappable state file data (parts of "vslick.sta" or "vslick.stu" under UNIX) in kilobytes to be kept in memory. -1 specifies no limit. 0 specifies that the editor preload all state file data. The smallest value is 200.

Tagging cache size

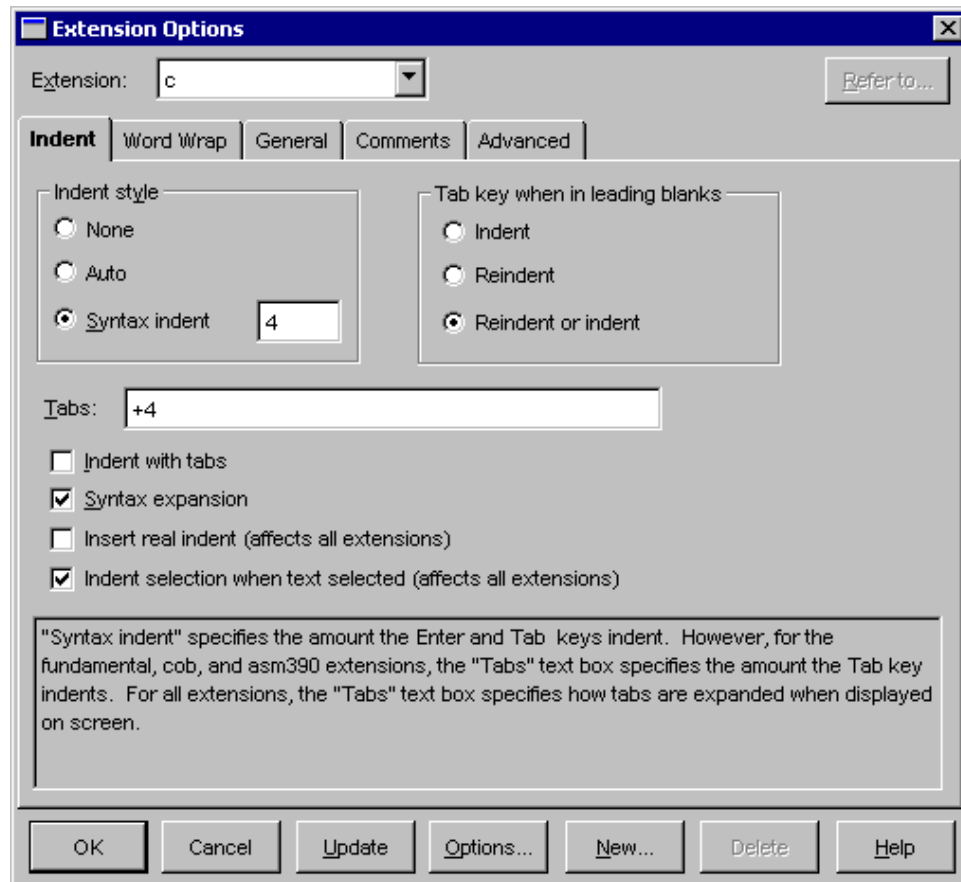
If you have a lot of memory and you have large tag files, you can improve tagging performance by increasing the tagging cache size. For example, if you have 16 MB of tag files (.vtg files) and 64 MB of memory, choosing 8000 (8 MB) for the tagging cache size will drastically improve tagging performance. Make sure you choose a small enough cache size so that the operating system does not perform memory swapping.

Extension Options Dialog Box

("Tools", "Configuration", "File Extension Setup..." or **setupext** command)

The Extension Options dialog box provides you with an easy way to set default indent, word wrap, and other options for specific file extensions.

The Extension Options dialog box is displayed below:



Extension

Select the extension you want to modify from the Extension combo box.

Refer To

The "Refer To" button (top of dialog) allows you to make the setup data for one extension refer to another. For example, the "h" and "cpp" extensions refer to the "c" file extension. When you modify the "h" or "cpp" extension setup, you are actually modifying the extension setup for all three. In addition, the "h" and "cpp" extensions will use the same tagging support as the "c" extension. This button is not enabled if other extensions already refer to this one.

Update

Updates the extension setup but does not close the dialog box. This allows you to modify more than one extension's options while the Extension Options dialog box is displayed.

Options

This button accesses options such as minimum keyword abbreviation and other language-specific options. Press this button to display the extension-specific options dialog box, and press F1 for help. This button is grayed when it is not supported.

New

Allows you to create a new extension and assign extension-specific options.

Delete

Deletes the extension setup information. This button is grayed when it is not allowed. The Fundamental extension setup information cannot be deleted. An extension such as C, which has other extensions (such as H, CPP, and CXX) that refer to it, cannot be deleted until all extensions that refer to it are deleted.

Extension Options Indent Tab

IMPORTANT: If you want your TABS for an extension to be different than syntax indent and indenting with tabs, be sure to check the "Hack Tabs Backspace" option on the Redefine Common Keys dialog box ("Tools", "Configuration", "Redefine Common Keys...").

Indent Style

Indent Style	Effects
None	ENTER key always goes to the beginning of the line.
Auto	ENTER key indents according to previous line.
Syntax Indent	ENTER key indents according to language syntax. The value in the text box specifies the amount to indent for each level. You will probably always want this value to match your tab stops.

Tab key when in leading blanks

Affects the behavior of the TAB key when pressed in the leading blanks of a line. This feature allows the TAB key to be used to “beautify” (reindent) the current line.

Option	Effects
Indent	Does not change the normal operation of the TAB key, which is to indent to the next tab stop.
Reindent	Reindents the current line.
Reindent or indent	Reindents the current line if the new column position is before the current column position. Otherwise, the TAB key indents to the next tab stop.

Tabs

You can set tabs in increments of a specific value or at specific column positions. To specify an increment of 3, enter **+3** in the text box. To specify columns, you could enter **1 8 27 44** to specify tab stops that are not an increment of a specific value.

Indent with tabs

Determines whether TAB key, ENTER key, and paragraph reformat commands indent with spaces or tabs.

Syntax Expansion

This option is sometimes grayed. When checked, templates are inserted when you press SPACEBAR. For example, when editing a C file, **if** <SPACEBAR> inserts an **if** template for the C language.

Insert real indent (affects all extensions)

When this box is checked, the ENTER key inserts real spaces or tabs representing your indent instead of virtual spaces. Many users like this because pressing END will place your cursor after the blank text where you want to start typing the new text.

Indent selection when text selected

By default, pressing TAB or SHIFT+TAB when there is a selection indents or unindents the selection, respectively. If you do not want your TAB or SHIFT+TAB keys to operate differently when there is a selection, turn this check box off.

Extension Options Word Wrap Tab

Margins

Sets the left, right, and new paragraph margins. See "**Margins dialog box**" on page 135 for more information.

Justify Style

Visual SlickEdit provides three justification styles:

Justification style	Word wrap effects
Justified	Full justification. Left and right edges of text will align exactly at margins.
Left and Respace	Left justification with space character reformatting. One space is placed between words except after the punctuation characters ".?!" which get two spaces. If you only want one space after the punctuation characters ".?!", turn on "1 space after period".
Left	Left justification with respect for space characters between words. This setting requires the save options to be set such that trailing spaces are not stripped when a buffer is saved. See " Save options " on page 234.

Word Wrap

When checked, the editor attempts to keep the cursor within the margins when entering text, moving the cursor, and deleting characters.

Extension Options General Tab**Mode Name**

Gives a more meaningful name for this extension setup. You need to define a mode name if you want the "Select Mode" ("Document", "Select Mode") menu item to work well.

Alias Filename

Each extension can have one alias file. This allows you to define aliases which do not affect other extensions. For example, you might have a comment header you use a lot or want to change or add to the syntax expansion. Click "Alias..." on this dialog box to define extension-specific aliases.

Aliases

This button displays the [Alias Editor dialog box](#) which gives you an easy way to define extension-specific aliases.

Encoding

Unicode support required: Each extension can have its own encoding specification. If the extension-specific encoding is set to “Default”, then the global setting defined in the File Options dialog (“Tools”, “Configuration”, “File Options”, select the Load tab) is used. Both the extension-specific and global setting are overridden if you previously specified an encoding in the Open dialog. Visual SlickEdit records the encoding used to override default encoding settings and reuses this setting the next time you open the same file. This provides you with per-file encoding support.

IMPORTANT: The “refer” to feature for extension options works differently for this option. If “h” refers to “c”, “h” might have a different encoding than “c”. SlickEdit takes advantage of this feature for supporting the new Microsoft project files (.vcproj, .csproj, .vbproj) which are XML files but which default to active code page encoding and not UTF-8 like XML.

In order to provide better support for editing Unicode and non-Unicode files, Visual SlickEdit has two modes of editing: Unicode and SBCS/DBCS mode. Files that contain Unicode, XML Files, or files that contain code page data not compatible with the active code page should be opened as Unicode files. See “Using Unicode” on page 170 for more information.

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.

The “**Auto XML**” 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. SlickEdit Inc. 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.

When the “**Auto Unicode**” 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.

When the “**Auto Unicode2**” 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 foolproof and might give incorrect results. Only UTF-16 is supported by this option.

When the “**Auto EBCDIC**” 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 foolproof and may give incorrect results. The option does attempt to support binary EBCDIC files.

The “**Auto EBCDIC and Unicode2**” encoding is a combination of the “Auto EBCDIC” and “Auto Unicode2” encodings described above.

See “Using Unicode” on page 170 for more information.

Truncation

Line truncation is typically used for mainframe languages such as COBOL, JCL, and OS/390 Assembler where there are line numbers (usually starting in column 73) which need to stay in the same column position. For these languages, the truncation line length is set to 72. When on, all editor operations do not allow the data to the right of the truncation line length to move or be modified. For example, search and replace operations will 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. If necessary, the replaced text is truncated to fit within the truncation line length.

Set this to "Auto" if you want the editor to determine the truncation line length based on the record format of the file (only OS/390 data sets can have a record format). For files which 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).

Auto Caps

When on, and a file is opened which contains no 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. Also see **caps** command.

Extension Options Comments Tab

Comment with border

These characters specify the sides and corners used to draw a comment box.

First line is top

When checked, the top side character is ignored and the first line of the selection is used as the top side.

Last line is bottom

When checked, the bottom side character is ignored and the first line of the selection is used as the bottom side.

Comment without border

These characters specify the text to insert at the left and right to create a line comment.

Start in column

Specifies which column to start the comment in for a line selection. This is useful for column-oriented languages such as COBOL.

Extension Options Advanced Tab

Color Coding

Control name	Description
Lexer Name	Name of the lexer. See " Color Coding " on page 137 for information on defining language-specific coloring elements for a new language.
Modified Lines	When checked, displays line prefix area of modified lines in user-specified color. Inserted lines have one color and changed lines have another.
Current Line	When checked, displays current line in current line color. You might want to try the "Draw box around current line" option on the General Configuration dialog box ("Tools", "Configuration", "General...", select General tab) instead of using this check box because it does not interface with other color coding.

The precedence of color from highest to lowest is current line, modified lines, and language specific. To change the colors, use the **Color Settings dialog box** ("Tools", "Configuration", "Colors...").

Color Coding

Displays **Color Coding Setup dialog box** which allows you to modify language-specific color coding for the current language. You can also add new color-coded keywords. This button also allows you create language-specific color coding for new languages.

Menu if no selection

This specifies the menu that is displayed when you right click in an edit window that does not have a selection.

Menu if selection

This specifies the menu that is displayed when you right click in an edit window that has a selection.

Select first (Affects all extensions)

When checked (default), a selection can be made with the right mouse button instead of displaying the extension-specific menu. When this is not checked, you can select menu items by clicking and dragging the mouse.

Begin/end Pairs

Languages such as Modula-2, Java, Delphi, Pascal, and Assembler which have begin/end structure pairs can take advantage of this feature. You specify the pairs in a format similar to a regular expression. When you place the cursor on the begin or end keyword of the pair, invoke the command **find_matching_paren** (CTRL+] or "Tools", "Match Paren") to find the match of the pair. This text box is disabled for languages which have special begin/end matching built in. The examples below illustrate the syntax for defining the begin/end pairs.

NOTE: Begin/end pair matching is case-sensitive by default. Append ";I" to ignore case.

Examples:

(begin),(case)|(end);I

The above begin/end pairs are for the Pascal language. The Pascal language requires a more sophisticated expression. This expression indicates the keywords BEGIN or CASE start a block and the keyword END terminates the block. The ';' is used to specify multiple begins or multiple ends. The '|' operator is used to separate begins from ends.

(#ifdef),(#ifndef),(#if)(#endif)

The above pairs are for the C language. The C language has the added complication that #if is a sub-string of #ifdef. Due to the implementation of begin/end matching, #ifdef must appear before #if.

Word Characters

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 '-' character as a valid word character, place a '-' at the beginning or end of the word character string.

Extension Specific Project...

This button displays the Project Properties dialog box. Each extension can have its own compile command, build command, include directories, and more. See ["Creating Projects"](#) on page 32 for more information.

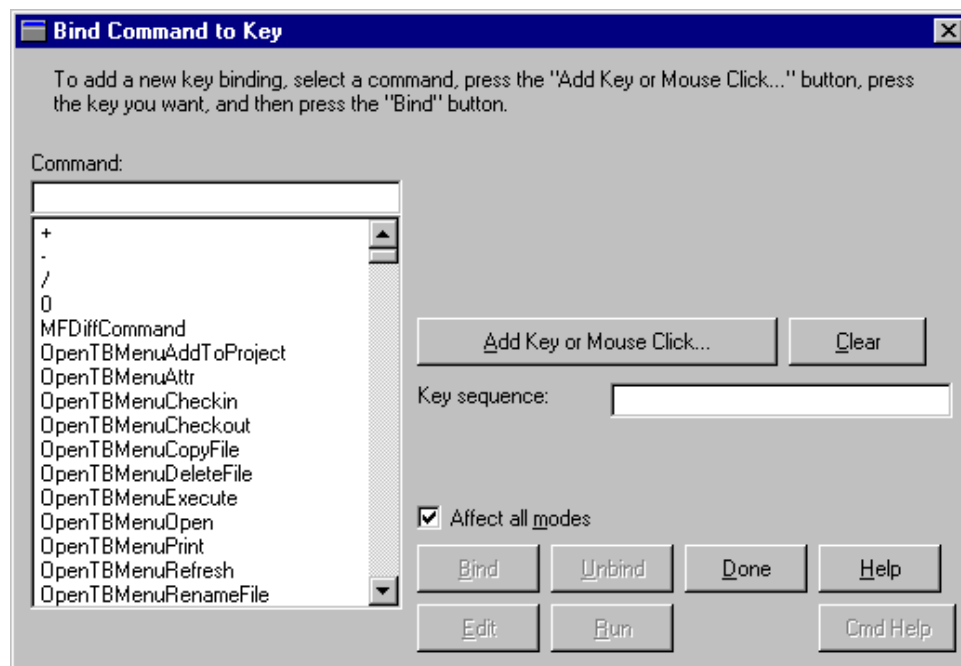
Changing Key Bindings

Bind Command to Key Dialog Box

("Tools", "Configuration", "Key Bindings...") or ("Macro", "List Macros...") or ([gui_bind_to_key](#) command)

Try using the [Redefine Common Keys dialog box](#) on page 227 ("Tools", "Configuration", "Redefine Common Keys...") to see if you can do what you want there. It is well organized for common key binding changes.

Below is a picture of the [Bind Command to Key dialog box](#):



The basic steps for binding a command to a key are as follows:

1. Choose a command from the Command combo box.
2. Click "Add Key or Mouse Click" and press the key to which you want the command bound.
3. Click "Bind".

The Command combo box supports completion of command names. If you do not know the name of the command but do know a key which invokes the command, click "Add Key or Mouse Click" and press the key for which you want to know the binding. Key binding information for the keys displayed in the Key Sequence text box is displayed above the Affect All Modes check box.

There are two ways to bind keys. You can edit Slick-C source code representing all your key bindings or you can use this dialog box. You should use this dialog box unless you are making many key binding changes. See "[Editing Key Binding Source](#)" on page 226 for information on the other method.

Command

Specifies the command you want to bind. Once entered, all key sequences the command is already bound to are displayed in the top right of this dialog box.

Add Key or Mouse Click

Once you press this button, you will be prompted to press a key or click the mouse. The key or mouse click you have selected will be added to the Key Sequence text box. If you make a mistake, click "Clear" to erase the contents of the Key Sequence text box.

Clear

Erases the contents of the Key Sequence text box.

Affect all modes

This check box is grayed if the current buffer does not have an extension-specific event table defined. You might want to uncheck this box if you want a particular key binding to only affect the current buffers mode.

Bind

The Bind button binds the command selected in the command combo box to the key sequence specified. This button is grayed until a valid command and key sequence is selected.

Unbind

Removes the key binding for the Key Sequence specified. This button is grayed if the Key Sequence text box is empty or is not bound to a command.

Quit

Closes the dialog box. Key bindings changes are not cancelled.

Edit

This button displays the Slick-C source for the command selected in the Command combo box. This button is grayed if the command in the Command combo box is not valid. You need to invoke the Bind Command to Key dialog from the ("Tools", "Configuration", "Key Bindings...") menu item for this button to be visible.

Run

This button executes the command selected in the Command combo box on the current buffer. You need to invoke the Bind Command to Key dialog from the ("Tools", "Configuration", "Key Bindings...") menu item for this button to be visible. This button is grayed if the command in the Command combo box is not valid.

Delete

Deletes the macro displayed in the Command combo box. This button is only visible when you use the List Macros menu item, and is grayed if the command in the Command combo box is not valid.

Cmd Help

Displays help on the command selected in the Command combo box. This button is grayed if the command in the Command combo box is not valid.

Editing 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 Bind Command to Key dialog. To create a Slick-C batch macro containing your current key bindings, enter the command **list_source** on the Visual SlickEdit command line. One of the files generated by this command is "vusrdefs.e". It is placed in your configuration directory if the VSLICKCONFIG environment variable is set (defaults to \$HOME/.vslick under UNIX) . Otherwise, it is placed in your macros directory. If you open this file (CTRL+O), the first part of the source code will be your key bindings, which look like the following:

```
defeventtab default_keys
def 'A-a'-'A-z'=
def 'A-F6'=
def 'F10'=
def 'C-A'= select_all
def 'C-B'= select_block
def 'C-C'= copy_to_clipboard
def 'C-D'= gui_cd
...
```

The default_keys are the key bindings that are active in fundamental mode. The other event tables defined by the **defeventtab** primitive are mode event tables containing key bindings which override the fundamental mode key bindings. Make changes to this buffer by adding or modifying the "def keyname= command" lines and then save the buffer by pressing CTRL+S. For complete information on the syntax for binding keys see "[Defining Events and Event Tables](#)" on page 289. The valid key names are listed in the online help under "Event Names". You can also list the key names of the keys through the online help by invoking the command

help Event Names. To run this Visual SlickEdit 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.

Unbinding a Key

To unbind a command from a key:

1. Invoke the Bind Command to Key dialog box ("Tools", "Configuration", "Key Bindings...")
2. Click "Add Key or Mouse Click..." to select the key.
3. Click "Unbind".

Determining the Key(s) a Command is Bound to

To determine what key(s) a command is bound to, invoke the **Bind Command to Key dialog box** ("Tools", "Configuration", "Key Bindings..."), and enter the command in the Command combo box. The key bindings are displayed in the top right of the dialog box.

Determining the Function of a Key

Use the **what_is** command ("Help", "What is Key...") to determine the function of a key. It will prompt you to press a key. If the key is bound to a command and the online help contains a help item for it, the help on the command invoked by the key will be displayed. Otherwise, the message line displays the command, if any, the key is bound to. This command will give the correct binding information regardless of whether or not you have changed the key bindings.

Redefine Common Keys Dialog Box

("Tools", "Configuration", "Redefine Common Keys...")

Because many users feel strongly about the function of the keys ENTER, BACKSPACE, DEL, HOME, BACKTAB, LEFT, RIGHT, UP, and DOWN, Visual SlickEdit provides a Redefine Common Keys dialog box specifically designed for changing the function of these keys ("Tools", "Configuration", "Redefine Common Keys..."). For changing the function of other keys, see "**Changing Key Bindings**" on page 224. Click "Tab Options" to view indent options which affect the TAB key.

ENTER Key Configurations

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 (default)

Splits the current line at the cursor. Enough blanks are inserted at the beginning of the new line to align it with the first non-blank character of the original line.

Maybe Split Insert Line

If insert state is on, 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 insert state is off, the cursor is moved to column one of the next line.

BACKSPACE Key Configurations

Rubout

Deletes the character to the left of the cursor. If word wrap is on, the cursor will wrap to the previous line when the left margin is reached. Otherwise the cursor is not wrapped to the previous line.

Linewrap Rubout (default)

Deletes the character to the left of the cursor. The cursor always wraps to the previous line when the left margin is reached. If you want line wrapping to occur when Column 1 is reached, set the "Line Wrap On Text" check box on ("Tools", "Configuration", "Redefine Common Keys...", "Line Wrap On Text...").

DEL Key Configurations

Delete Char

Deletes the character at the cursor. If word wrap is on and no more characters exist on the current line, the next line is joined to the current line.

Linewrap Delete Char (default)

Deletes the character at the cursor. If no more characters exist on the current line, the next line is joined to the current line.

HOME Key Configurations

Begin Line

Moves the cursor to Column 1.

Begin Line Text Toggle (default)

If the cursor is on the first non-blank character of the current line, the cursor is placed in column one. Otherwise, the cursor is placed on the first non-blank character of the current line.

Cursor wrap

Determines whether the cursor Left and Right commands wrap to the previous or next line respectively.

Up/Down on text

Determines whether the cursor up and down commands place the cursor in virtual space. By default, cursor Up and Down goes to the same column of the next or previous line, regardless of the length of the line.

Line wrap on text

The line wrap on text option affects whether line wrapping occurs when the left margin is reached or column one is reached for the LEFT and BACKSPACE key configurations. By default, when word wrap is on, wrapping occurs when the left margin is reached regardless of the LEFT or BACKSPACE key configurations. Check this box if you want wrapping to occur when Column 1 is reached.

Jump over tab characters

By default, moving the cursor over a tab character with the LEFT or RIGHT arrow key causes the cursor to jump across the virtual space. To allow the LEFT and RIGHT arrow keys to cursor into virtual space of tab characters, turn this check box off.

Pull chars backspace

By default, pressing BACKSPACE when in replace mode removes the previous character and moves the cursor left. If you just want the previous character to be replaced with a space character, turn this check box off.

Hack tabs backspace

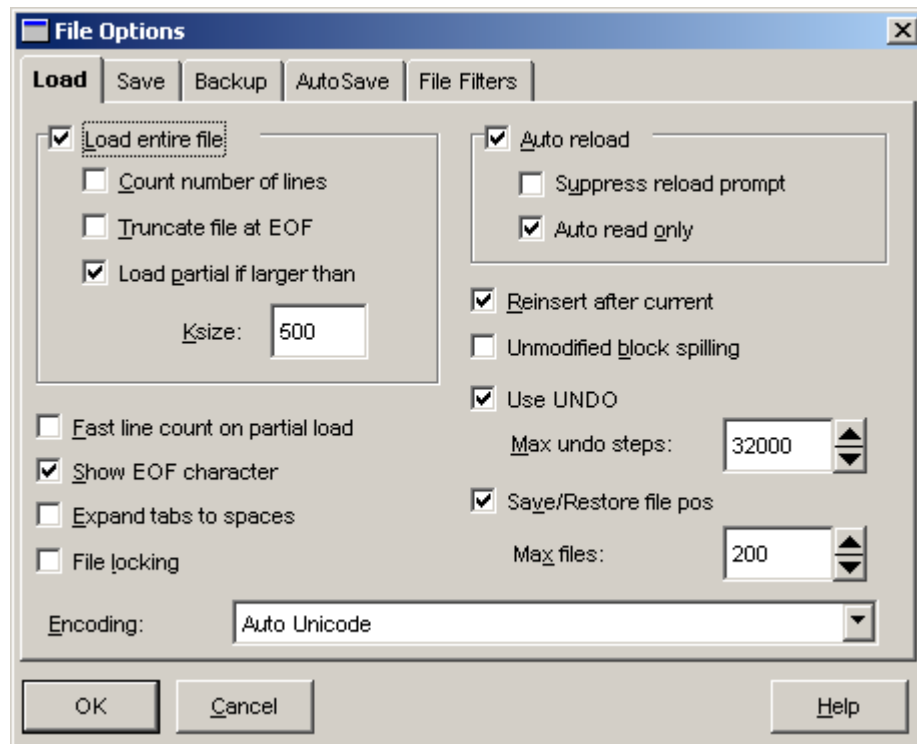
By default, pressing BACKSPACE when the previous character is a tab causes the rest of the line to be pulled to the previous tab stop. This is usually not what you want if you are in a mode that has a syntax indent for each level that is different from the Tab settings. If you want your BACKSPACE key to delete through tab characters one column at a time, check this box on.

File Options Dialog Box

("Tools", "Configuration", "File Options...")

The default load and save options are "safe" for editing binary files since all reformatting is turned off. Otherwise, to ensure that no load or save reformatting takes place when you open and save binary files, turn on the "Binary" check box when you open the file. It is not safe to edit binary files if any of the following conditions exist:

- "Show EOF Character" is off
- "Expand Tabs to Spaces" is on
- "Append EOF Character" is on
- "Remove EOF Character" is on
- "Strip Trailing Spaces" is on



Load Tab

("Tools", "Configuration", "File Options...", select Load tab)

The load options settings affect the **Open dialog box** and any other command which uses the **e** or **edit** commands to open a file. You can override many of the load options by checking the appropriate box in the advanced section of the **Open dialog box**.

Windows:

You can have different load options for different disk drives. Define an environment variable called `VSLICKLOAD` and specify each drive followed by the appropriate switch. For example, "`set VSLICKLOAD= a: +L b:+L`" specifies preloading files from floppy drives A and B.

When loading files via the command line (e and edit commands) the following options can be explicitly specified: '+' turns a switch on and '-' turns a switch off.

File Caching Notes

Visual SlickEdit always reads the entire file for files that are color coded. However, load partial still helps performance because the original file is used as a read-only spill file. You can determine when Visual SlickEdit has read your entire file by going to the bottom of the file. If the line number is displayed when at the bottom of the file, the entire file has been read, but this does not mean the entire file is in memory. By default, Visual SlickEdit uses 2 MB for the buffer cache. When the cache becomes full, modified blocks are written to the spill file. Blocks that are not modified and can be reread from the original file are discarded. Scrolling through a 2-gigabyte file requires no more than 2 MB of memory (default). A search and replace operation which hits every block will require almost all blocks to be spilled. Saving a 2-gigabyte file requires 4 gigabytes of disk space. Visual SlickEdit must spill the file data first before saving the file. Make sure you turn backups off before saving a large file since this will require additional disk space equal to the size of the file. The block size is 16k.

The options on the "Load" tab are as follows:

Load entire file

Switch +L. When on, the entire contents of opened files are read into memory. The "Line" status field may become blank if the file did not fit in the editors cache (defaults to 2 MB). When this check box is off, Auto Reload will not work until the file is saved.

Count number of lines

Switch +LC. When on, the entire contents of opened files is read into memory and the number of lines in the file are counted. The line number is always displayed in the "Line" status field. The Load Entire File check box will have the same effect as this check box when the entire file can fit in the editor's cache (defaults to 2 MB) and does not have to be spilled.

Truncate file at EOF

Switch +LZ. When on, the entire contents of opened files are read into memory and the number of lines are counted. In addition, DOS format files are truncated when an EOF character is found. The line number is always displayed in the "Line" status field. This option is useful for REXX ".cmd" files which can have pcode appended to them after the EOF character.

Load partial if larger than

When on, and the file you are loading is greater than the size specified in the "Ksize" text box, the entire file is not read into memory. Since the file handle remains open to your file, Auto Reload will not work until the file is saved. The "Line" status field may go blank unless "Fast Line Count on Partial Load" is on.

Fast line count on partial load

Switch +LF. When on, Visual SlickEdit will count the number of lines when files are opened. The line number is always displayed in the "Line" status field. 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. Since the file handle remains open to your file, Auto Reload will not work until the file is saved.

Show EOF character

Switch +LE. When on, Visual SlickEdit will not remove the EOF character when files are loaded.

Expand tabs to spaces

Switch +E. When on, the entire contents of files are read into memory and tabs are expanded into spaces. If your tabs settings for the file being loaded are of the form "+<increment>" (e.g. "+4"), then tabs are expanded in increments of *increment*. Otherwise, tabs are expanded in increments of 8. To set your tabs in a form "+<increment>", bring up the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup", select the extension, and set the "Tabs:" text box). For languages such as REXX and UNIX shell scripts which require the contents of the file to be analyzed before the file type is known, the fundamental mode tab settings are used.

File locking

Switch +N. When on, ensures that a file handle is kept open to the file for locking purposes. This enables Visual SlickEdit to detect when another user is editing the same file.

NOTE: This feature is not supported by the UNIX version.

Auto reload

When on, Visual SlickEdit will detect when files you are editing have been modified by other applications and prompt you or automatically replace the file with the new copy on disk. Under UNIX, this feature is turned off by default because many of the UNIX file systems do not give correct file time stamps. This causes the editor to prompt you to reload a file which has not been modified.

Suppress reload prompt

When on, files you are editing which have been modified by other applications will automatically be replaced with the new copy on disk.

Auto read only (Windows only)

When on, Visual SlickEdit detects when other applications change the read-only attribute of the file and turns Read Only mode on/off.

Reinsert after current

Switch +BP. When on, buffer and window rings are rearranged so that after an operation which switches the active buffer/window such as Open, Go To Definition, or Cursor Error, you can switch back to the previous buffer or window with the **prev_buffer** or **prev_window** command.

Unmodified block spilling

Switch +S. When on, unmodified blocks are spilled when the memory/buffer cache is full. When the spill file destination is faster than the disk the file resides on (such as a floppy drive), you should set this switch to +S for optimal performance.

Use UNDO, max undo steps

Switch +U:*max_undo_steps* (e.g. +U:32000 turns on undo and specifies 32000 step max). When the “Use UNDO” check box is on, modifications to buffers may be undone. The “Max Undo Steps” text box specifies the maximum number of steps that are stored. Cursor motion can be undone but is not counted as a step.

Save/Restore file pos, max files

If the “Save/Restore File Pos” check box is on when you open a file, your cursor position is restored to its previous location when the file was closed. The “Max Files” text box specifies the maximum number of cursor positions saved. The most recently closed file positions are stored.

Encoding

Unicode support required. Specifies the global (non-extension specific) file encoding. This setting is overridden if an extension-specific encoding is defined. Use the Extension Options dialog (“Tools”, “Configuration”, “File Extension Setup...”, select the General tab) to define an extension-specific encoding. Both the extension specific and global setting are overridden if you specify an encoding in the Open dialog. Visual SlickEdit records the encoding used to override default encoding settings and reuses this setting the next time you open the same file. This provides you with per-file encoding support.

In order to provide better support for editing Unicode and non-Unicode files, Visual SlickEdit has two modes of editing: Unicode and SBCS/DBCS mode. Files that contain Unicode, XML Files, or files that contain 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”
- “EBCDIC, SBCS/DBCS mode”

In addition, the following encodings put the editor into SBCS/DBCS editing mode when the file is determined not to be Unicode:

- “Auto Unicode”
- “Auto Unicode2”

- “Auto EBCDIC and Unicode”
- “Auto EBCDIC and Unicode2”

All other encodings put the editor in Unicode mode and required that the file data be converted to UTF-8.

The “**Auto XML**” 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.

When the “**Auto Unicode**” 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.

When the “**Auto Unicode2**” 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 foolproof and may give incorrect results. Only UTF-16 is supported by this option.

When the “**Auto EBCDIC**” 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 foolproof and may give incorrect results. The option does attempt to support binary EBCDIC files.

The “**Auto EBCDIC and Unicode2**” encoding is a combination of the “Auto EBCDIC” and “Auto Unicode2” encodings described above.

See “Using Unicode” on page 170 for more information.

Save Tab

("Tools", "Configuration", "File Options...", select Save tab)

The save and backup options affect save operations. The **save** command switches and check boxes are described below. A '+' turns a switch on and a '-' turns a switch off.

Windows: You can have different save and backup options for different disk drives. Define an environment variable called VSLICKSAVE and specify each drive followed by the options. For example, "**set VSLICKSAVE= a: +O b:+O**" specifies no backup for floppy drives A and B.

The options on the “Save” tab are as follows:

Append EOF character

Switch +Z. When on, an EOF character is appended to the end of DOS files when the buffer is saved. This switch has no effect on UNIX, Macintosh, or binary files.

Remove EOF character

Switch +ZR. When on, the EOF character is removed from the end of DOS files when the buffer is saved. This switch has no effect on UNIX, Macintosh, or binary files.

Expand tabs to spaces

Switch +E. When on, tabs are expanded to spaces according to the tab stops when the buffer is saved.

Strip trailing spaces

Switch +S. When on, trailing spaces at the end of each line are stripped when the buffer is saved.

Save files on loss of focus

When on, all modified files will be saved when you switch to another application.

Reset line modify

Switch +L. When on, line modify flags are reset when the buffer is saved.

File format

When "Automatic" is selected, your files are saved as is and no changes are made to the line end characters. If you want your line end characters translated when you save your files, select the DOS, MAC, or UNIX button to force a particular ASCII file format. Translating line end characters on save does make save operations a little slower. The Save As dialog lets you translate the line end characters for the current file.

Backup Tab

("Tools", "Configuration", "File Options...", select Backup tab)

Visual SlickEdit only makes a backup of a file the first time you save your file in a given edit session. Your backup file may not represent your last save. For more up-to-date backups, use the AutoSave feature. See "[AutoSave Tab](#)" on page 236 for more information.

The options on the "Backup" tab are as follows:

Make backup files

Switch +O (overwrite switch). Turns backup on/off. When the +O options is given to the **save** command, no backup file is created regardless of the backup option settings.

Backup dir

Specifies the backup directory used by "Global directory" or "Global nested directories" backup styles. The backup directory may contain a drive specifier. This sets the VSLICKBACKUP environment variable.

Same directory as *.BAK

Switch +DB. When on, backup files are placed in the same directory as the destination file but the extension is changed to ".bak". This option is useful on networks.

Global directory

Switch +D. When on, backup files are placed in a single directory. The default backup directory is "\vslick\backup\" (UNIX: "\$HOME/.vslick/backup"). The backup file gets the same name part as the destination file. For example, given the destination file "c:\project\test.c" (UNIX: "/project/test.c"), the backup file will be "c:\vslick\backup\test.c" (UNIX: "\$HOME/.vslick/backup/test.c").

Windows: For a network, you might need to create the backup directory with appropriate access rights manually before saving a file. See "**Backup Dir**" above.

Global nested directories

Switch -D. When on, backup file directories are derived from concatenating a backup directory with the path and name of the destination file. The default backup directory is "\vslick\backup\" (UNIX: "\$HOME/.vslick/backup"). For example, given the destination file "c:\project\test.c", the backup file will be "c:\vslick\backup\project\test.c" (UNIX: "\$HOME/.vslick/backup/project/test.c").

Windows: For a network, you might need to create the backup directory with appropriate access rights manually before saving a file. See "**Backup Dir**" above.

Use child directory

Switch +DK. When on, backup files are placed in a directory off the same directory as the destination file. For example, given the destination file "c:\project\test.c" (UNIX: "/project/test.c"), the backup file will be "c:\project\backup\test.c" (UNIX: "/project/backup/test.c"). This option is most convenient on networks because of file permission issues.

AutoSave Tab

("Tools", "Configuration", "File Options...", select AutoSave tab)

The "AutoSave" tab allows you to turn on AutoSave to prevent losing data when an abnormal editor exit occurs (possibly from loss of electricity to your computer). You must check the "AutoSave activated" check box in order to turn on AutoSave.

NOTE: AutoSave does not support OS/390 data sets.

After you have set the AutoSave options the way you like them, you might want to simulate an abnormal editor exit to get a better understanding of how AutoSave works. To simulate an abnormal editor exit:

1. Modify one or two files.
2. Wait for an AutoSave to occur.

3. Type **fexit** on the Visual SlickEdit command line.

The **fexit** command will EXIT the editor WITHOUT prompting you to save modified files or your configuration. By default, AutoSave temporary files are placed in a directory called "autosave" below your configuration directory.

NOTE: AutoSave temporary files are deleted when the editor exits normally. Once a file has been saved or closed, the AutoSave temporary file is deleted the next time AutoSave occurs. This allows you to stay in the editor all day without creating hundreds of autosave files. AutoSave temporary files are only needed for files that are modified.

The current implementation of AutoSave does not save files which have no name. In addition, AutoRestore does not restore files which do not yet exist on disk. You should save your file at least once to ensure that the file has a name and exists on disk.

The options on the "AutoSave" tab are as follows:

AutoSave activated

When checked, the editor will AutoSave files after the specified period of inactivity or after the period of time specified.

Save after period of inactivity

Specifies amount of idle time, in minutes or seconds, when modified files should be saved. Set this value to 0 if you want this option ignored.

Save after period of time

Specifies amount of time in minutes or seconds when modified files should be saved. Set this value to 0 if you want this option ignored.

AutoSave directory

Specifies the AutoSave directory if the "Save to different directory" option has been selected. If the AutoSave directory is blank, "<configuration directory>\autosave" is used. The configuration directory is defined by the VSLICKCONFIG environment variable.

Windows: If the VSLICKCONFIG environment variable is not set, the directory below the editor executable directory is used as the configuration directory.

UNIX: If the VSLICKCONFIG environment variable is not set, "\$HOME/.vslick" is used as the configuration directory.

Save to different directory

Specifies that all AutoSave temporary files be placed in the directory specified by the "AutoSave directory" text box. This option makes it easy for you to clean up or find all AutoSave files should an abnormal editor exit occur.

WARNING: If you are editing two files with the same name (i.e. "makefile") but in different directories, one AutoSave temporary file will be overwritten by the other.

Save to same name, different extension

Windows: Specifies that the AutoSave file be placed in the same directory as the file being auto saved but given a different extension. The third character of the extension is replaced with a ~ character. The length of the extension is padded with underscores if the length of the extension is less than 3 characters. For example, the AutoSave file for "test.c" is "test.c_~". The AutoSave file for "test.prg" is "test.pr~". If you are editing two files in the same directory which differ only by the third character, one AutoSave temporary file will be overwritten by the other.

UNIX: Specifies that the AutoSave file be placed in the same directory as the file being auto saved but a ~ character is appended to the name. For example, the AutoSave file for "test.c" is "test.c~". The AutoSave file for "test" is "test~".

Save to same name

Specifies that modified files be automatically saved. No AutoSave temporary files are created.

Largest file to AutoSave (K)

Files greater than this size will not be autosaved. Set this value to 0 if you want all files autosaved.

File Filters Tab

("Tools", "Configuration", "File Options...", select File Filters tab)

The "File Filters" tab allows you to change the file specifications that appear in the "List files of type" combo box at the bottom left of the Open dialog box. The first wildcard pattern(s) specified in the File Filters dialog box is used to initialize the **Open dialog box**. The default is to list all files. You might want a smaller list initially, which displays the source files you typically edit.

Color Settings Dialog Box

("Tools", "Configuration", "Color..." or **color** command)

Use the Color Settings dialog box ("Tools", "Configuration", "Color...") to change the default colors:

1. Select the screen element.
2. Set the foreground and background colors.
3. Click "Apply" or OK. ("Apply" updates the colors you have modified without closing the dialog box.)

Set embedded language color

When on, you can define the colors for source code (e.g. 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.

Perl example:

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

Screen Element

Select the screen element before changing the foreground and background colors. Most of the screen element items are obvious except for the following:

Cursor	The Cursor screen element is displayed in the active edit window when the cursor is placed on the Visual SlickEdit command line. It is not the color of the blinking cursor.
Window Text	This is the color of other text which is not a specific syntax element.
Attribute	HTML only. This is the color used for a recognized attribute of an HTML tag. For example, the "src" attribute of the "img" tag gets this color.

Style

For color coded elements, you may also choose whether the element is bold, italic, or underlined. For example, keywords are bold by default.

Apply

Updates all modified screen elements. The dialog box is not closed.

Reset

Restores all colors to the values they were when the editor was invoked.

Cancel

Restores all colors to the values they were when the colors dialog box was displayed.

Use system default

When checked, the operating system default colors are used. Currently, this check box is only enabled for the "Status" and "Message" fields. For UNIX, the "System" default colors are selected by the editor and not the operating system.

Enable bold and italic for fixed Unicode fonts

(Unicode support required.) When on 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 might 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).

Schemes>>

Expands the Color Settings dialog box so you can try different color schemes, or define your own.

To try a different color scheme, pick a color scheme and click "Apply". If you want to keep your changes, click OK. To define a new color scheme, set your colors for the various screen elements and click "Save Scheme...".

User-defined color schemes are stored in "uscheme.ini".

Font Configuration Dialog Box

("Tools", "Configuration", "Font...")

Visual SlickEdit allows you to change the fonts used by edit windows, the command line, and status text. Select the screen element you want to change and then select the font you want.

SBCS/DBCS Windows: Choose the "Terminal" font for the largest selection of attractive, fixed fonts. These are the fonts for a command prompt.

Unicode Source Windows: The default font for editing Unicode files is "Default Unicode Font". When this font is selected, the "Arial Unicode MS" font is used if it is installed. Otherwise, the "Ansi Fixed Font" is chosen which only supports the English character set. The "Arial Unicode MS" font is a fairly complete font which is included with Microsoft Office. Currently, no version of Windows ships with a fairly complete Unicode font. For more information on Unicode support, see "Using Unicode" on page 170.

Configuration Files

This section describes user configuration and system configuration files.

User Configuration Files

User configuration changes are stored in the following files:

vslick.ini	A text file which 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.
user.vlx	A text file which contains color coding changes (keywords, etc.). This file is updated when you close the color coding dialog box.
vslick.sta	A binary file which 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. The

installation or update program can safely replace this file because source representations of user configuration changes are stored in the Slick-C macro files `vusrdefs.e` (UNIX: `vunxdefs.e`), `vusrobs.e` (UNIX: `vunxobjs.e`), and `vusrsyNN.e` (UNIX: `vunxsyNN.e`). The first time a user runs the editor, configuration changes stored in the Slick-C macro files mentioned above are automatically transferred.

NOTE: The reason some configuration data is stored in binary in “`vslick.sta`” and some in Slick-C macro files is so the configuration data can be loaded faster.

<code>vusrdefs.e</code>	(UNIX: <code>vunxdefs.e</code>) A Slick-C text file which contains the emulation setting, key bindings, color settings, file extension setup information, and some other miscellaneous options.
<code>vusrobs.e</code>	(UNIX: <code>vunxobjs.e</code>) A text file which contains user-defined dialog boxes and menus in Slick-C syntax.
<code>vusrsyNN.e</code>	(UNIX: <code>vunxsyNN.e</code>) A text file which contains system modified dialog boxes and menus. These changes are NOT automatically transferred to Visual SlickEdit unless the version encoding matches (i.e. NN matches the version of the editor).
<code>*.als</code>	A text file which contains user-defined extension-specific aliases.
<code>ftp.ini</code>	(UNIX: <code>uftp.ini</code>) A text file which contains user-defined FTP configurations.
<code>uprint.ini</code>	A text file which contains user-defined printing schemes.
<code>uscheme.ini</code>	A text file which contains user defined color schemes.
<code>usrpacs.slk</code>	A text file which contains user defined project packages.
<code>ubox.ini</code>	A text file which contains user defined box and line comment styles.
<code>alias.slk</code>	A text file which contains user defined global aliases (directory aliases).
<code>projects.slk</code>	(UNIX: <code>uproject.slk</code>) A text file which contains user defined extension specific projects.
<code>uservc.slk</code>	A text file which contains user defined version control systems.
<code>uformat.ini</code>	A text file which contains user defined beautifier schemes.
<code>vrestore.slk</code>	A text file which contains auto-restore information. Note that the workspace files also contain auto-restore information, but only for the files/windows previously open.
<code>usercpp.h</code>	A text file which contains defines (default preprocessing) for tagging C++ and C code.

System Configuration Files

Typically, system configuration files are only modified by SlickEdit Inc. or OEM customers (except for those files which contain both user and system configuration data like “vslick.sta”). OEM customers might want to modify one of these files in order to ship a customized version of Visual SlickEdit. If you use one of the dialogs to make a change to a user configuration file such as “user.vlx”, you need to transfer your changes to the appropriate system configuration file (“vslick.vlx”).

CAUTION: When you get a new version of Visual SlickEdit, merge your “vslick.vlx” changes into the newly installed version.

vslick.ini	A text file which 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. Installation programs SHOULD NOT OVERWRITE this file unless the file does not exist.
vslick.vlx	A text file which contains system color coding lexer definitions.
vslick.sta	A binary file which 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. The installation or update program can safely replace this file because source representations of user configuration changes are stored in the Slick-C macro files <code>vusrdefs.e</code> (UNIX: <code>vunxdefs.e</code>), <code>vusrobs.e</code> (UNIX: <code>vunxobs.e</code>), and <code>vusrsyNN.e</code> (UNIX: <code>vunxsyNN.e</code>). The first time a user runs the editor, configuration changes stored in the Slick-C macro files mentioned above are automatically transferred.
print.ini	A text file which contains system printing schemes. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
vsscheme.ini	A text file which contains system color schemes. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
box.ini	A text file which contains system box and line comment styles. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
prjpacks.slk	(UNIX: <code>uprjpacks.slk</code>) A text file which contains system project packages. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.
vcssystem.slk	(UNIX: <code>uvcsys.slk</code>) A text file which contains system defined version control systems. This file is NOT modified by the dialogs and is not preserved when a new editor is installed.

format.ini	A text file which contains system-defined beautifier schemes.
alias.slk	A text file which contains user-defined global aliases (directory aliases). If this file does not exist, install programs should install the default "alias.slk" which contains two predefined aliases (filecomment and functioncomment).
syscpp.h	(UNIX: usyscpp.h) A text file which contains system defines (default preprocessing) for tagging C++ and C code.

Configuration Changes not on Configuration Menu

Printing options and spell checking options are not on the configuration menu.

For information on printing options, see "[Printing](#)" on page 127. For information on spell checking options, see "[Spell Checking](#)" on page 85.

Saving Disk Space with Tab Compression

Column-oriented operations are more convenient with files that do not have tab characters. However, more disk space is consumed by space characters than by the appropriate number of tab characters. Specific file extensions can be selected to be compressed with tab characters when a save is performed. By default, Visual SlickEdit does not compress any files. The save command takes about 15% longer to compress a file with tabs while writing the file. The amount of disk space saved by compressing a file with tabs varies from file to file. For some Assembler files, the file size will reduce by 30%. Tab compression of a line stops if there are quote characters. Lines that are part of a line continuation from the previous line (last character of line is '\') are not tabbed.

To select the file extensions compressed with tabs when saved, execute the **tab_ext** command on the command line with no parameters and add the extension you want compressed. Separate each extension with a space and prefix each extension with a period.

Configuration Macro Variables

The macro variables below offer some additional options. Use the **set_var** command to set a macro variable. Invoke the **set_var** command with just the macro variable name to view the current line.

def_preplace	Default is 1. If zero, the save command will NOT prompt you if you are inadvertently overwriting a file. For example, if you invoke the command "save xyz", 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.
--------------	---

def_buflist	Default is 3. This macro variable determines how the list_buffers commands displays the buffer list. By default, the buffer list is sorted and path information is in a separate column to the right of the name. This macro variable is composed with the following flags: <pre>SORT_BUFLIST_FLAG 1 SEPARATE_PATH_FLAG 2 SHOW_SYSTEM_BUFFERS_FLAG 4</pre> Add the flags together to select a configuration. Leaving out a flag removes the features. If the buffer list is not sorted, the list will be in the order of the buffer ring.
def_from_cursor	Default is 0. If non-zero the commands upcase_word, lowcase_word, and cap_word will start case change from the cursor position instead of the beginning of the current word.
def_modal_paste	Default is 0. If non-zero, commands which insert a BLOCK type clipboard will overwrite the destination text if the cursor is in replace mode.
def_binary_ext	Default is ".ex .obj .exe .lib". This variable is used by the "editflst.e" macro for BRIEF emulation. When the "editflst.e" macro is loaded, the extensions listed by this variable are filtered out by the edit command's completion.
def_save_macro	Default is 1. Set this variable to "0" if you do not want to be prompted with the Save Macro dialog box after ending macro recording. Examples: (Press ESC to place cursor on the command line) <pre>set-var def-save-macro 0 Do not prompt to save recording macro. set-var def-from-cursor View the current value of the "def-from-cursor" macro variable. The current value can be edited and set to a new value.</pre>

Setting Environment Variables

SlickEdit recommends that you set environment variables in "vslick.ini". However, DO NOT set VSLICKCONFIG in "vslick.ini".

VSLICKRESTORE	Directory to store Auto Restore files. See " Auto Restore " on page 20 for more information.
VSLICKCONFIG	Directory where Visual SlickEdit should store user's local configuration files. Used in multi-user environments. Under UNIX, this defaults to the HOME environment variable when VSLICKCONFIG is not set.

VSLICK	Specifies additional command line arguments to editor as if you were typing them in when invoking the editor.
VSLICKPATH	One or more directories separated with ';' (UNIX: ':') where Visual SlickEdit should look for batch macros or executable files.
VSLICKMACROS	One or more directories separated with ';' (UNIX: ':') which contain macro files (*.e). VSLICKPATH must also contain the directories listed here.
VSLICKBIN	One or more directories separated with ';' (UNIX: ':') which contain Visual SlickEdit binary files. VSLICKPATH must also contain the directories listed here.
VSLICKBITMAPS	One or more directories separated with ';' (UNIX: ':') which contain bitmap files (*.bmp). VSLICKPATH must also contain the directories listed here.
VSLICKMISC	One or more directories separated with ';' (UNIX: ':') which contain miscellaneous Visual SlickEdit files including *.vlx, *.slk, *.api, *.idx, vslick.sta (UNIX: vslick.stu), *.hlp, scommon.lst, main.dct, *.pif, *.ini (except for vslick.ini), and *.lst. VSLICKPATH must also contain the directories listed here.
VSLICKALIAS	One or more file names separated with ';' (UNIX: ':') which contain ALIAS definitions.
VSLICKTAGS	Specifies global tag files. One or more file names separated with ';' (UNIX: ':') which contain tags. Do not put this environment variable in "vslick.ini". See "Dynamic Tagging" on page 57.
VSLICKBACKUP	Directory to place Visual SlickEdit backup files. Affects +D (default) and -D backup configurations only. See "Backup tab" on page 235 for more information.
VSLICKSAVE	(Windows only.) Allows save options to be specified per drive. See "Save tab" on page 234 for more information.
VSLICKLOAD	(Windows only.) Allows load options to be specified per drive. See "Load tab" on page 230 for more information.
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	(Windows only.) If you have a license code (see "Help", "Programming Information...") in addition to a serial# , you are using the Visual SlickEdit license manager. The license manager handles system crashes better if each user sets 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.

Setting Environment Variables in vslick.ini

You should place configuration environment settings specific to Visual SlickEdit in the file "vslick.ini". However, do not set the VSLICKCONFIG environment variable in "vslick.ini" (VSLICKCONFIG determines where the editor looks for "vslick.ini"). For a complete listing of Visual SlickEdit configuration environment variables, see "[Configuring Visual SlickEdit](#)" on page 204. When the editor starts up, it sets the value of environment variables specified in "vslick.ini". Below is a sample "vslick.ini" file with an environment section.

Windows example:

[Environment]

```
VSCLICKPATH=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
VSLICKSAVE=a: +o b: +o
MYPROJECTVERSION=c:\myprog4.2
```

UNIX Example:

[Environment]

```
VSCLICKPATH=/usr/lib/vslick:/usr/lib/vslick/macros:/usr/lib/vslick/bitmaps
VSLICKALIAS=%HOME%/vmacros/alias.slk
VSLICKINCLUDE=/usr/lib/vslick/macros
MYPROJECTVERSION=/myprog4.2
```

When the editor starts, the following environment variables are created by the editor:

VSDRIVE	(Windows) Drive letter followed by colon where editor executable resides.
VSDIR	Directory of editor executable with trailing backslash (UNIX: forward slash).

Environment variables can be embedded in any line within a section by placing % characters around the environment variable (as shown in the UNIX example above).

set Command

You can change or view the environment while you are running Visual SlickEdit via the **set** command. The operation of the built-in **set** command is almost identical to the DOS SET command. Use the **set** command to temporarily change one of Visual SlickEdit's configuration environment variables or any other environment variable. For a complete

listing of Visual SlickEdit configuration environment variables, see "[Setting Environment Variables](#)" on page 244. 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. Specifying 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 Visual SlickEdit command line will give you a selection list of all of the environment variable names.

Slick-C Language

Overview

Some History

Why did Microsoft decide to use the BASIC language as the extension language for products such as Word and Excel? Why is Hypercard based on an English-like grammar with container variables? Why did database companies develop their own languages instead of using C++?

The answer is simple. The C and C++ languages are incomplete, too difficult, and too dangerous. While software products have all become easier to use, why hasn't the most popular object-oriented language, C++ followed suit?

When considering the alternatives for extending a graphical version of Visual SlickEdit, it was very clear that DLLs written in C/C++ should not be the primary means (see "[DLL interface](#)" on page 342). Extending the editor had to be easier to use, safe, and preferably more portable. This was the reason Visual SlickEdit created the Slick-C language.

A Recurring Theme

The easier to use programming languages always seem to add one or more of the features below to overcome problems in C++:

- Provide safe string processing which requires no memory management.
- Provide a language with more built-in primitives for building dialog boxes and building a graphical user interface.
- Remove requirement to allocate and keep track of memory.
- Reduce or remove the need for pointers.
- Reduce need for declarations.
- Provide container variables for more easily passing around different kinds of data.

To date some of the valiant attempts include Java, Delphi, Visual Basic, and Tcl/Tk.

Recognizing Problems in C++

In order to create something much better than C++, you must first understand its problems:

- Insufficient built-in types. No boolean, string, or container type.
- Too much memory management, especially for objects.
- No primitives for dialog boxes or building a graphical user interface.
- Overuse of pointers due to lack of language constructs.
- Lots of declaration data required even for the simplest program.
- Grammar is very large and ambiguous.

What Makes Slick-C the Best Attempt Yet

Slick-C has addressed all of the problems in C++. The strongest features in Slick-C are listed below:

- Language syntax is C++ with few major deviations. Unlike Java, you can pass any type by reference or value as you would expect in C++.
- Many built-in language and compiler/linker features for building graphical user interfaces. Language supports graphical resources such as dialog box templates, menu templates, and bitmaps. A control type is built-in so that compiler/linker detect references to controls which don't exist on a dialog box.
- Typed and untyped variables. A major problem with existing highly typed object oriented languages such as C++ and Java is that they can't provide users with a good container type variable. This makes these languages unsuitable for simple editor macros, database applications, or shell scripts which don't need the speed or rigidity of a highly typed language. How many times have you heard someone first learning C++ say that they want to define a container class? Then they find out it can't be done in any useful manner. Slick-C is highly typed, but also provides a very powerful container type. See container variable code samples below.
- Better built-in types. Arrays in Slick-C dynamically grow without requiring the user to allocate memory. Hash tables (sometimes called associative arrays) dynamically grow and can be pre-initialized similar to an array. A hash table is an array which can be indexed with a string. One array or hash table can be assigned to another as long as the types are compatible. The string type is much more integrated into the language. String constants are converted to numeric types and boolean (1 or 0) types. The **switch** statement supports string expressions. There are separate string operators so you can determine whether a string or numeric operation will be performed without knowing the variable types.
- Clipboard Inheritance® (US patent No. 5,710,926) provides inheritance specifically for dialog boxes. This feature allows you to copy parts of existing dialog boxes to the clipboard, paste them elsewhere, and the original code still runs. New code can be attached to the new controls without affecting the original controls. New code can be added to the original controls to affect both instances of the controls; this is, of course, 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\(R\)](#)" on page 306.

Work Still to be Done

Slick-C is not finished. What we have is the ground work for a language which will be the most powerful general purpose programming language. To complete that language, it will need the following:

- C++ style classes. Like Java, classes in Slick-C will not require a **delete** to free objects.
- Support for **enum**.
- Speed and size improvements for typed variables. Currently Slick-C has been optimized for typeless container variables and strings. String concatenation is 5 times faster than Java. Integer arithmetic is 3 times slower than Java. Floating point

performance is much worse but can easily be optimized. Some of the poor performance is due to the fact that Slick-C does 32 DIGIT integer and floating point arithmetic. 64 bit longs and smaller floating point numbers would really be big enough.

- Support for char and short types for conserving memory.
- Support for template classes. Container variables are sometimes a more powerful mechanism (See sample code below) for accomplishing much of what is done with template classes. However, users will likely want the speed and additional type checking of template classes and many of us are already used to template classes.
- **instanceof** operator like Java. This primitive is very important because it can tell if a variable is an instance of a particular class which may be in the inheritance chain. Currently, Slick-C has a **_varformat()** method, but it can't be enhanced to traverse an inheritance chain.
- C++ code generation. That is, the ability to convert Slick-C code into source code which can be compiled by any C++ compiler.

Container Variable Sample Code

These examples show what can be done in a language which truly supports typed variables and untyped container variables.

Here is how to define a set of stack routines in Slick-C which support any type of element.

```
void stacknew(typeless &stack)
{
    stack._makeempty(); // Destroy current contents of stack.
    stack[0]=0;         // Make an array and use first element as top count.
}
void stackpush(typeless &stack, typeless &value)
{
    stack[++stack[0]]=value;
}
typeless stackpop(typeless &stack)
{
    if (stack[0]<=0) return('');
    // Make a copy of the element.
    result=stack[stack[0]--];
    // Free space allocated by value and delete array element. _deleteel is a
    // built-in method which operates on arrays and hash tables.
    stack._deleteel(stack[0]+1);
    return(result);
}
defmain()
{
    // The above routines can handle variables of any type, including string
    constants.
    struct RECORD {
        int i;
        _str s;
    };
    // You can't make a limit on the number of elements in an array.
    // We will add support for initially allocating a specific number of elements.
    RECORD arecord[];
    arecord[0].i=4;arecord[0].s="element 0";
    RECORD symboltable[]; // Declare a hash table/associative array
    symboltable:["name1"].i=1;symboltable:["name1"].s="element 0";
    stacknew(stack);
    stackpush(stack,arecord); // Push an array onto the stack
}
```

```
    stackpush(stack,symboltable);    // Push a hash table/associative array onto the
same stack
    stackpush(stack,"string");        // Push a string constant onto the same stack
}
```

This example illustrates how a container variable can access structure members as an array. This cannot be implemented elegantly in any other language.

```
/*
  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 unable to come up with a function syntax
      // that we were satisfied with so with went with a REXX style syntax.
      // Place text up to but not included tab character into value variable.
      // Place tab character and rest of data in rest variable.
      parse rest with value "\t" +0 rest;
      if (substr(rest,1,1)=="\t") {
        rest=substr(rest,2);
      }
      p->[i]=value;
    }
  }
  _delete_temp_view(temp_view_id);
  activate_view(orig_view_id);
  return(0);
}

struct TABLE_ENTRY{
  _str name;
  int value;
};

// defmain is the main entry pointer for a Slick-C batch/script macro
defmain()
{
  TABLE_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);
}
```

Getting Started by Example

There are three basic ways to extend Visual SlickEdit using Slick-C. You may either write a **key bindable command**, an **event driven dialog box**, or a **batch macro**.

A Key Bindable Command

A key bindable command is the most common way to extend the editor. Command macros may be bound to keys or invoked from a menu. When you use macro recording, Slick-C source code is created for a key bindable command. You may edit the source code for a recorded macro by using the **list_macros** command ("Macro", "List Macros...", Select a macro and press the Edit button). See "**Recording Macros**" on page 90 for more information.

Open a new file called "test.e" (invoke the File Open menu item and type "test.e") and insert the following lines:

```
_command hello()
{
    message("Hello World");
}
```

Invoke the **load** command (F12 or "Macro", "Load Module...") to compile and load your macro. You have now defined a new command called **hello** which may be bound to a key or invoked from the Visual SlickEdit command line. Press Esc to place the cursor on the command line, type "**hello**" and press Enter. You should see the message "Hello World" in the message area.

Here's how you can bind your **hello** command to Alt+5:

- Invoke the **Bind Command to Key dialog box** ("Tools", "Configuration", "Key Bindings...").
- Type "hello" in the text box.
- Click the "Add Key or Mouse Click..." button
- Press Alt+5.
- Press Esc or click on the Quit button to close the dialog box. The editor will prompt you to save the configuration when you exit. Newly loaded macro modules require the state file to be saved.

An Event Driven Dialog Box

Event driven forms are ideal for creating dialog boxes. After creating the dialog box, the **show** command may be used to display the dialog box from the command line or from a menu item. To display a dialog box from a key binding you must write a key bindable command which uses the **show** command to display the dialog box.

Perform the following steps to create a simple event driven dialog box:

- Create a new form ("Macro", "New Form").

- Double click on the command button bitmap in the dialog editor **Properties dialog box** (above the check box bitmap which is a square box with an X in it).
- Double click on the text box bitmap in the dialog editor **Properties dialog box** (the one with the letters "ab" in it). If you can't see the dialog editor **Properties dialog box**, invoke the Properties menu item ("Window", "Properties") to make it visible.
- Move the command button or the text box so they do not overlap. Click on the object with the left mouse button, hold it, and drag to move the object.
- 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 should appear with **"lbutton_up"** displayed in the combo box. Press Enter to select the event.
- You should be prompted with an Open dialog box for a new file which is to contain the source code for this dialog box. Type **"form1.e"** and press Enter.

NOTE: If at any point you can't see the dialog editor **Properties dialog box**, invoke the Properties menu item ("Window", "Properties") to make it visible.

After the performing the above steps, you should see a file called "form1.e" with the following lines of code:

```
#include "slick.sh"

defeventtab form1
ctlcommand1.lbutton_up()
{
}
}
```

If you don't see the above code, probably because you already have a "form1.e" file. If this is the case, modify the "form1.e" file to contains the lines of code above.

Modify the above code to add the statement (ctltext1.p_text="Hello World"); as shown below.

Example (we have added some comments you don't need)

```
#include "slick.sh"

// Define an event table for the dialog box window.
defeventtab form1

// Since this is the first event handler defined for this control and
// the
// name of this control does not match the last defined event table,
// the Slick-C translator automatically defines the event table
// form1.ctlcommand1 and defines the lbutton_up event handler within
// this
// new event table
ctlcommand1.lbutton_up()
{
    // Set the p_text property of the text box control
    ctltext1.p_text="Hello World";
}
}
```


Invoke the **run_selected** command ("Macro", "Load and Run Form") to load the code, load the dialog box, and display the dialog box. Click the "ctlcommand1" button to display "Hello World" in the text box. To close the Form1 dialog box, double click on the system menu (not all UNIX window managers close windows when you double click on the system menu) or press Ctrl+Shift+Space (in the running version of form1; not the edited copy). You can press Ctrl+Shift+Space while any dialog box is running in order to edit it (including the **Properties dialog box**).

You can display this dialog box from the command line by typing "**show form1**". To display the dialog box modally enter "**show -modal form1**" on the command line. The **show** command provides many powerful options. For more information, see "**Displaying Dialog Boxes**" on page 301. The editor will prompt you to save the configuration when you exit. Dialog box templates and compiled macros are stored in the state file ("vslick.sta" or "vslick.stu" under UNIX).

For more information about building event driven dialog boxes see "**Creating Dialog Boxes with the Dialog Editor**" on page 294.

The example code below illustrates how to write a command which displays a dialog box. This is useful for binding a command to a key which displays a dialog box.

```
#include "slick.sh"
_command run_form1()
{
    // The -modal option disables other windows while the dialog box is
    // displayed.
    show("-modal form1");
}
```

A Batch Macro

Batch macros are useful for trying out Slick-C primitives and are easier to share between multiple users. Batch macros may not be bound to a key; however, you can execute a batch macro from the command line or a menu item.

Open a new file called "hello.e" (invoke the File Open menu item and type "**hello.e**") and insert the following lines:

```
defmain()
{
    message("Hello World");
}
```

You have now defined a new batch macro called **hello** which may be invoked from the Visual SlickEdit command line. Save your macro and press Esc to place the cursor on the command line. Type "**hello**" and press Enter. You should see the message "Hello World" in the message area. Batch programs must be saved before they are executed in order for the macro to get compiled. Visual SlickEdit automatically compiles batch programs if no corresponding pcode file exists (hello.ex in this example), or the date of the source file is newer than the pcode file.

Compiling and Loading Macros

The two commands **st** and **load** ("Macro", "Load Module...") are provided for compiling Slick-C modules from within the editor. The **st** command translates the module specified into binary code called pcode. When no module is specified, the current buffer is translated. The **load** command (F12 or "Macro", "Load Module...") will translate the module specified if necessary (MAKE), and load the resulting pcode. When no module is specified, the current buffer is saved if necessary, translated, and loaded. If a module is loaded which has already been loaded, it is replaced. Both the commands invoke the external program "vstw.exe" (UNIX: vstw) to translate the source module into pcode. DO NOT use the **load** command on batch programs. After doing so you will no longer be able to execute the batch program until you use the **unload** command ("Macro", "Unload Module...").

A module which was loaded with the **load** command may be unloaded with the **unload** command ("Macro", "Unload Module..."). However, Visual SlickEdit's symbol table (called the names table) will still contain 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 Visual SlickEdit command line to save the configuration at any time.

Language Constructs

Comments

Slick-C supports both C++ comment styles. Two forward slashes (//) declares the rest of the line to be a comment. The two characters '/*' open a comment and '*/' closes a comment. Unlike C++, comments may be nested.

Example:

```
i=1; // this is a comment

/* this is a /* nested */ comment */
```

String Literals

Strings may be in single or double quotes. Double quoted strings are identical to C++. A backslash followed by a character has special meaning as follows:

<code>\a</code>	Bell character (7)
<code>\b</code>	Backspace character (8)
<code>\f</code>	Form feed character (12)
<code>\n</code>	New line character (10)
<code>\r</code>	Carriage return (13)
<code>\t</code>	Tab character (9)
<code>\v</code>	Visual character (11)
<code>\?</code>	Question mark character
<code>\'</code>	Single quote character
<code>\"</code>	Double quote character
<code>\\</code>	Backslash character
<code>\xhh</code>	Hexadecimal character code
<code>\ooo</code>	Octal character code

If single quotes are used, two single quotes consecutively represents 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 a operator `"=="`. However, this operator will strip leading and trailing spaces and tabs from both operands (see "[Mathematical Operators](#)" on page 260 for a list of operators).

Example:

```
"abc":=='abc'
"Can't find file":=='Can't find file'
"\t":==_chr(9)
```

A backslash (not inside quotes) followed by a character or a number has the special meaning as shown below. However, using this feature is not recommended because it is not C++ compliant. This features remains only for backward compatibility.

<code>\a</code>	Bell character
<code>\b</code>	Backspace character (8)
<code>\f</code>	Form feed character(12)
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\v</code>	Visual character (11)
<code>\xdd</code>	Hexadecimal character code <i>dd</i>
<code>\ddd</code>	Decimal character code <i>ddd</i>

Quoted strings adjacent to escape sequences are read (lexed) as one string. No concatenation is performed.

Example:

```
\n:==\10
\r:==\xd
\r:==\x0d
"\r\n":==\r\n
"1"\50 + "1"\49:== "23"  // 12 + 11=="23", not 1:+(2+1):+1=="131"
```

Numeric Constants

The Slick-C language supports floating point numbers and hexadecimal numbers. The mantissa is limited to 32 digits and the exponent is limited to 9 digits. When precision is lost, the result is rounded. Overflow and underflow are detected. Floating point numbers have the following syntax:

[+|-] digits [.] [digits] [E[+|-] digits]

OR

[+|-] [.] [digits] [E[+|-] digits]

The leading sign may have blanks before and after.

Hexadecimal numbers are defined in the same way as the C language, as follows:

0x *hexdigits*

Example:

```
4
" + 4e2"
4e2
" -4E-2"
-4E-2
0xff
```

Defining Constants

Slick-C has full support for **#define** textual replacements like C/C++. **#define** can be used for defining constants or in-line functions.

This syntax for defining a textual replacement is:

```
#define name[(param1,param2,...)] value
```

Use a backslash at the end of a line to indicate that the *value* text continues to the next line. Any occurrence of *name* is replaced with the text *value* before the source is compiled.

WARNING: When *value* represents a constant expression, you should place parentheses around it to make sure there is not a problem with operator precedence.

Example:

```
#define MAXLINES 15
#define MAXLINESP1 (MAXLINES+1)
#define max(a,b)    ( (a) >= (b) ) ? (a) : (b) )
#define min(a,b)    ( (a) <= (b) ) \
                    ? (a) : (b) )

defmain()
{
    x=4;
    y=7;
    a=max(x,y);
}
```

Now that Slick-C has full support for **#define** replacement macros you should not use the current **const** primitive. The Slick-C **const** primitive is more like Pascal than C++ and will confuse C/C++ programmers.

The variable name is not prefixed with a type name, the expression must be a constant, and the comma is optional. The syntax for declaring a constant is:

```
const
    variable = constant_expression [,']
    variable = constant_expression [,']
....
```

The comma is used to declare more than one constant on the same line. Constants may be used in mathematical expressions.

Example:

```
const
    LONG_CONST = "line 1" :+
                "line 2" :+
```

```
        "line 3"  
DOS_INT  = 0x21
```

Mathematical Operators

Slick-C uses the operator precedence of C. An expression can have the following unary operators:

<code>!e1</code>	Logical NOT. Result is 1 if <i>e1</i> evaluates to 0. Otherwise result is 0.
<code>~e1</code>	Bitwise complement.
<code>-e1</code>	Negation.
<code>+e1</code>	No change.
<code>++v1</code>	Increments the variable <i>v1</i> and returns the result.
<code>v1++</code>	Returns the value of <i>v1</i> and then increments the variable <i>v1</i> .
<code>--v1</code>	Decrements the variable <i>v1</i> and returns the result.
<code>v1--</code>	Returns the value of <i>v1</i> and then decrements the variable <i>v1</i> .

The binary and ternary operators for the Slick-C language are listed below from lowest to highest precedence.

=	Assign right operand to left operand.
+=	Add left operand to right operand and assign to left operand.
-=	Subtract right operand from left operand and assign to left operand.
/=	Divide left operand by right operand and assign to left operand.
*=	Multiply left operand with right operand and assign to left operand.
=	Bitwise OR left operand with right operand and assign to left operand.
^=	Bitwise XOR left operand with right operand and assign to left operand.
&=	Bitwise AND left operand with right operand and assign to left operand.
<i>e1?e2:e3</i>	If expression <i>e1</i> is TRUE (not the string '0'), expression <i>e2</i> is returned. Otherwise, expression <i>e3</i> is returned.
&&	Logical AND. If left hand expression is false, right hand expression is not evaluated.
	Logical OR. If left hand expression is true, right hand expression is not evaluated.
	Bitwise OR.
^	Bitwise XOR.
&	Bitwise AND.
==	Equal. Performs a numeric or string comparison depending on the operands. This function is NOT identical to the C strcmp function (see := operator below).
NOTE:	If both operands are numbers, a numeric comparison is performed. Otherwise a string comparison is performed. In any case, leading and trailing spaces and tabs are stripped before the comparison is performed.
>	Greater than. Performs a numeric or string comparison depending on the operands. See note under == operator.
>=	Greater than or equal. Performs a numeric or string comparison depending on the operands. See note under == operator.
<	Less than. Performs a numeric or string comparison depending on the operands. See note under == operator.
<=	Less than or equal. Performs a numeric or string comparison depending on the operands. See note under == operator.
!=	Not equal. Performs a numeric or string comparison depending on the operands. See note under == operator.
:=	Exactly equal. Always performs string comparison. This is equivalent to the C expression (strcmp(a,b)==0).
!:=	Not exactly equal. Always performs string comparison.
<:=	Exactly less than or equal. Always performs string comparison.
<:	Exactly less than. Always performs string comparison.
>:=	Exactly greater than or equal. Always performs string comparison.
>:	Exactly greater than. Always performs string comparison.
:+	Concatenation.
<<	Bitwise shift left.
>>	Bitwise shift right.
+	Addition.
-	Subtraction.
/	Division with possible floating point result.
intdiv	Division with integer result.
*	Multiplication.
%	Remainder.

In addition to the operators above there is also implied concatenation. When there is no binary operator 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. In the future, bitwise operators will support 32 bit **int** and 64 bit **long** for all platforms. There are two sets of comparison operators. 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.

NOTE: Expressions may extend across line boundaries if the line ends in a binary operator or if the line ends with a backslash.

Example:

```
(1.0==1)           == 1
(1e2==100)         == 1
(1e2::=100)         == 0
(" abc "=="abc")   == 1
(" abc "':"abc")   == 0
(" abc "':"!abc")  == 1
(" 1 "=="1")       == 1
(" 1 "':"1")       == 0
("abc":<"def")     == 1
1 2                :=="12"
1 ( 2)             :=="12"
pow(4,2)           ==16
5%2                ==1
5/2                ==2
5/2.0              ==2.5
5 intdiv 2.0       ==2
5&2                ==0
5|2                ==7
(10<7)             ==0
(10:<7)            ==1
```

Assignment Operator

The simple assignment statement has the syntax:

variable=*expression*;

Example:

```
i=1;
i=i+1;
```

Assignment statements may be cascaded (x=y=z). Assignment statements within **if** statements are not allowed. The compiler flags assignments within **if** statements as an error.

Scoping and Declaring Variables

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_\$". The Slick-C language supports global, **static** (module), and local scope variables. Global variables may be accessed by any module. The scope of a

static variable is limited to the module it is defined in. The scope of a local variable is limited to the function it is defined in.

Variables are declared just like in C++; however, there are the following differences:

- Classes are not yet supported.
- **void** is only allowed as the returned type of a function.
- **char**, **short**, and **float** types are not yet available. All numeric types are signed. The current Slick-C numeric types are **int**, **long**, and **double**.
- Slick-C has a built-in string type **_str**.
- Arrays, while declared similarly, cannot have a size limit. Arrays elements are always dynamically allocated.
- Slick-C provides a **:[]** hash table operator which is similar to the array operator **[]** except that you index hash tables with a string type.
- There is no **sizeof** operator. Since the Slick-C interpreter currently stores all types as container variables, the **sizeof** operator has no meaning.
- Only global or static variables can be initialized with the "=" operator.
- **static** structure members are not yet supported.

Rather than show you the complex BNF syntax for declaring variables, we will break up declarations into categories. If you don't already know the C language, a book on the C language would help you learn more about Slick-C.

Simple Variables

"Simple variable" means a variable which is not an array, hash table, structure, or function pointer.

The syntax for defining a simple variable is:

```
[static] TypeName variable1[=expression1] , variable2[=expression2] ...;
```

Where *TypeName* may be any of the following:

boolean	Use this for variables which only take on the value 0 or 1 and are typically used with the boolean operators &&, , and !. You can also use the new keywords " true " and " false ".
int	32 bit signed integer.
long	32 DIGIT signed integer. In the future this will be a 64 bit signed integer. Same as older keyword bigint . Currently this type supports a 32 DIGIT number.
double	32 DIGIT mantissa and 32 bit signed exponent. Same as older keyword bigfloat .
_str	String. Same as older keyword bigstring .
typeless	Container variable. You can assign a typeless variable anything. Even an array of "whatever", or hash table of "whatever".
<i>StructName</i>	See " struct " below.
<i>UnionName</i>	See " union " below.

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 may declare variables within the scope of a function to ensure that the variable will be local even if the name has been declared elsewhere as a global or constant.

Examples:

```
// 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; // Declare a local string variable
    t=gi; // Copy gi into local container variable t.
    message("t="t);
}
```

Details About Variable Initializations

Global and static numeric variables, which include **boolean**, **int**, **long**, and **double**, are initialized to 0 when no user-specified value is given. 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 no user-specified initialization value is given.

Examples:

```
boolean  globalboolean=true;
int      globalint;
double   globaldouble;

defmain()
{
    // will see "globalboolean=1 globalint=0 globaldouble=0"
    message("boolean="globalboolean" "globalint" "globaldouble");
}
```

struct

Structures 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 structure to make easier for you to remember what data needs to be filled in for a complete record. Structures also can have the added effect of reducing the number of global variables. There is no **sizeof** operator like in C++. Since the Slick-C interpreter currently stores all types as container variables, the **sizeof** operator has no meaning.

The syntax for defining a structure is:

```
[static] struct [StructName ] {
    member-variable-decl1;
    member-variable-decl2;
    ...
} [variable1[={e1,e2, ...}], variable2[={e1,e2, ...}], ...];
```

The **struct** declaration gives you the choice of defining your own type called *StructName* and/or 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 yet supported.

The syntax for accessing a member of a **struct** variable is:

variable.member-name

Differences from C++

For a complete list of differences between Slick-C and C++, see "[Differences Between Slick-C and C++](#)" on page **Error! Bookmark not defined.**

- Space for structure elements is allocated when you access the element.
- Structure data is not continuous. This is obvious for string, array, and hash table member variables which contain variable size data. However, even other types are sometimes stored elsewhere. In the future, after a few changes are made, the way Slick-C lays out memory will be documented.
- There is not a **sizeof** function which tells you the size of a structure in bytes.

Examples:

```
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);
    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]);
}
```

union

Unions are typically used in place of a **struct** in the case where you have mutually exclusive member variables. In this case, a **union** will require less memory than a **struct**. Memory is only allocated for one member variable at a time.

The syntax for defining a structure is:

```
[static] union [UnionName ] {  
    member-variable-decl1;  
    member-variable-decl2;  
    ...  
} [variable1[={e1}], variable2[={e1}], ...];
```

The **union** declaration gives you the choice of defining your own type called *UnionName* and/or 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 yet supported.

The syntax for accessing a member of a **union** variable is:

variable.member-name

Examples:

```
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:  
        messageNwait("x.u.s="x.u.s);  
    }
```

```

        break;
    case KIND_DOUBLE:
        messageNwait("x.u.d="x.u.d);
        break;
    }
}

```

Anonymous Unions

An anonymous union is a union member variable which has no name. This saves you from having to type the union member variable name.

Example

```

defmain()
{
    struct {
        int kind;
        union {
            int i;
            _str s;
            double d;
        }; // No name for this union member variable
    } x;
    x.kind=1;x.i=1;
}

```

Arrays

Array variables are excellent for keeping track of a list of items. If you want to prompt a user to select from an array of strings, use the **`_sellist_form`** (see on-line help).

The syntax for defining an array variable is:

```
[static] TypeName variable1[][={e1,e2, ...}] , variable2[][={e1,e2, ...}] ...;
```

The first element of an array starts at 0. Use more than one set of `[]` for multi-dimensional arrays. Notice that you 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 may be placed in an array is about 2 billion. Your machine will likely run out of memory before this limit is ever reached.

Use the **`_length()`** method to determine the number of elements in an array. The syntax for using this method is *variable*.**`_length()`**.

The following statements will empty an array:

```

array._makeempty(); // Empty the array
array=null;         // Empty the array. Same as above.

```

Differences from C++

For a complete list of differences between Slick-C and C++, see "[Differences Between Slick-C and C++](#)" on page **Error! Bookmark not defined.**

- 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 not a **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.

Examples:

```
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
variable
    t[t._length()]=45; // Add another array element.
    for (i=0;i<t._length();++i ) {
        messageNwait("t["i"]="t[i]");
    }
    #if 0
        // This C style pointer loop is not supported.
        p= &array[0];pend= &array[2];
        for (;p<pend;++p){ // If p and pend were declared as pointers,
        } // compiler would not allow < or ++.
    #endif
}
```

Hash Tables

Slick-C provides a **:[]** hash table operator which is similar to the array operator **[]** except that you index hash tables with a string type. The syntax for defining a hash table variable is:

```
[static]      TypeName      variable1:[][={s1=>e1,s2=>e2,      ...]      ,
variable2:[][={s1=>e1,s2=>e2, ...] ...;
```

Examples

```
#include "slick.sh"
int      ghi:[]={"Symbol1"=>1,"Symbol2"=>2,"Symbol3"=>3};
int      ghai:[][]={"Symbol1"=>{1},"Symbol2"=>{1,2},"Symbol3"=>{1,2,3}};
_str     ghstring:[]={"Symbol1"=>"Value1", "Symbol1"=>"Value2"};

defmain()
{
    messageNwait("ghi:['Symbol1']="ghi:["Symbol1"]);

    // Copy all the hash table elements into a local container variable
```

```

t=ghi;
t:["Symbol4"]=4;    // Add another hash table element.

// The loop below illustrates how you can traverse the elements
// in a hash table without knowing the string indexes.
typeless i;
boolean doModify;
doModify=false;
for (i._makeempty();) {
    t._nextel(i);
    if (i._isempty()) break;
    if (doModify) {
        t._el(i)=4; // Modify element
    } else {
        messageNwait("i="i" value="t._el(i));
    }
}
}

```

Pointers Variables

Pointer variables in C++ are usually considered to be a bad thing. Due to limits in the C++ language, pointers sometimes get over used. However, pointers are what make C++ a "Systems" programming language. You can write your own memory management routines, interface with hardware, and get that extra ounce of speed. Currently you cannot use pointers in Slick-C to perform these types of low level functions. Unlike C++, Slick-C pointers variables do not support arithmetic operations such as '<', '>', '+', and '-'. Only the operators '==', '!=', and '!' are allowed. Pointers were added to Slick-C because we felt that there are a few programming tasks which are more easily solved by pointers. If you access an invalid pointer, Visual SlickEdit will not crash; however, your Slick-C macro will stop running.

WARNING: When a module is reloaded, **static** variable addresses change. Make sure you reinitialize global pointer variables which point to **static** (module scope) variables.

[static] *TypeName *variable1[=&v1], *variable2[=&v2] ...;*

The unary **&** operator is used to return the address of a variable. The unary ***** operator is used to pick up a pointer. However, you need to use the operator **->** (ex. *p->member-variable*) to access members of a pointer to a structure. See example below.

Differences from C++

For a complete list of differences between Slick-C and C++, see "[Differences Between Slick-C and C++](#)" on page **Error! Bookmark not defined.**

- Space for array elements is allocated when you index into the array.
- You can't use pointer variables to traverse array elements.
- You can't limit the number of elements that the array may contain.
- There is not a **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.

Example:

```

int      gi=1;
int      *gpi=&gi;

```

```
defmain()
{
    // Print out what gpi pointers to.
    messageNwait("gpi="(*gpi));
    *gpi=2;      // Change value of gi
    messageNwait("get 2 here gi="gi);

    struct {
        int x,y;
    } st,*pst;
    st.x=1;st.y=2;
    pst=&st;
    messageNwait("pst->x="pst->x" pst->y="pst->y);
    // This works too, but using -> is easier
    messageNwait("("pst).x="(*pst).x" (*pst).y="(*pst).y);
}
```

Pointers to Functions

Function pointer variables are useful for arguments to sort functions or for lookup tables which contain a particular function to call. If you access an invalid function pointer, Visual SlickEdit will not crash; however, your Slick-C macro will stop running.

WARNING: When a module is reloaded, **static** function addresses change. Make sure you reinitialize global function pointer variables which point to **static** (module scope) functions.

[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 "&" operator is used to specify call by reference parameters. Call by reference array and hash table parameters require parentheses around the **&** and *id*. See "[Argument Declarations](#)" on page 277.

The syntax for calling a pointer to function variable is:

(*pfn)(*e1, e2,...*)

Differences from C++

For a complete list of differences between Slick-C and C++, see "[Differences Between Slick-C and C++](#)" on page **Error! Bookmark not defined..**

- Only one syntax is currently supported for making a call with a pointer to a function variable. The *pfn(p1,p2,...)* syntax is not yet supported. This limitation is necessary for container variables because the compiler does not know the type of the variable. However, support will be added for the other syntax for typed pointers to functions.

Examples;

```
void proc(_str a)
{
    messageNwait("a="a);
}
defmain()
{
```



```

        // Assign a pointer to a function to the container variable "t"
        t=proc;
        (*t)("First call");
        // Declare a local pointer to function variable
        void (*gpfn)(_str a);
        gpfn=proc;
        (*gpfn)("Second call");
    }

```

Type Casting

Slick-C enforces strict type checking on typed non-container variables. There are times when you need to convert one type to another to prevent the compiler from “complaining”. Type casting helps tell the compiler you know what you are doing. In addition, type casting can change the value of an expression.

The syntax for converting one type to another is as follows:

(VarDeclWithoutName) expression

Where *VarDeclWithoutName* is identical to a variable declaration without the variable name except that the **static** keyword is not allowed.

Some casts are not allowed because they would require unrealistic assumptions by the compiler. For example, you can't cast a **struct** type to another **struct** type.

Differences from C++

For a complete list of differences between Slick-C and C++, see "[Differences Between Slick-C and C++](#)" on page **Error! Bookmark not defined.**

- Currently Slick-C only supports the less ambiguous C style type casting. Support for the new C++ type casting syntax *TypeName(variable)* may be added in the future. The argument is between having an elegant less ambiguous grammar or conforming to C++ so there is nothing new to learn.
- Because Slick-C does not allow low level manipulation of memory, you can't do things like type cast an "**int ***" to a "**long ***".

Examples;

```

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
    // IN THE FUTURE this will NOT be the case because it
    // will be physically impossible to store a float in a 32 bit int.

    boolean b;
    b= i!=0; // Can't use cast here.
    i=(int)b; // Need cast here.
}

```

}

How Typeless Container Variables Work

Container variables are declared with the **typeless** keyword or are declared by an assignment to an undefined variable. Slick-C has additional string operators so that the compiler always knows whether to perform a string or numeric operation. The '+' operator always means add two numbers, and the concatenation operator ":+:" always means concatenate two strings. The 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. In the future, there will be design changes to optimize binary types like integers. See "[Mathematical Operators](#)" on page 260 for a complete list of operators.

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.

Examples;

```
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 and 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]);  
}
```

The IF Statement

The syntax for an **if** statement is:

```
if (expression ) statement  
[else statement]
```

statement may be a C style statement block which contains multiple statements. See "[Statement Block](#)" on page 273 for more information.

IMPORTANT: 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.

Examples:

```
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;  
}
```

Statement Block

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 block. Syntax:

```
{  
    statement  
    statement  
    ...  
}
```

statement may declare local variables.

Example:

```
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

The syntax for valid Slick-C language loops are:

```
[label:] while (condition_exp) statement
[label:] for ([b4e1 ,b4e2 ... ,b4e3]; [condition_exp] ; [cont_e1 ,cont_e2 ... ,cont_e3])
statement
[label:] do statement while ( condition_exp );
```

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.

The C-style **for** loop is somewhat 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 as well. If *condition_exp* is **true** (not the value 0), the *statement* is executed. The *statement* will continue 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 **do** loop executes *statement* first and then evaluates *condition_exp*. If *expression* is **true** (not the value 0). The *statement* will continue to be executed until *expression* becomes **false** (0) or a **break** statement is reached.

Loops may be exited with the **break** primitive. The **continue** primitive may be used to skip to the top of a loop. Both of these primitives support an optional label argument (like Java). If specified, the label must match the *label* of one of the loops you are currently in.

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.

Example:

```
outerloop:
    for(i=1;i<3;++i){
        for (j=1;++j) {
            if (j==3) break outerloop; //Exit both loops
```

```

        messageNwait("i="i);
    }
}

for(i=1;i<10;++i){
    messageNwait("i="i);
}
// The above for loop is equivalent to the loop below.
i=1;
for (;i<10;) {
    messageNwait("i="i);
    ++i;
}

for(i=1;i<10;++i) messageNwait("i="i);

i=1;
while (i <10) ++i;

i=1;
do {
    messageNwait("i="i);
} while (i<10);

status=search(":", "@" /* No message option. */);
for(;;) {
    if ( status ) {
        break;
    }
    get_line(line);
    messageNwait("found match line="line);
    status=repeat_search();
}

```

Switch Statement

Slick-C supports the C **switch** statement. The Slick-C **switch** supports integers and string types. The syntax of the **switch** statement is as follows:

```

switch (expression) {
[ case expression:
    statements
]
[ case expression:
    statements
]
...

[ default:
    statements
]
}

```

The **switch** expression is evaluated and compared against all the case expressions. Once 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:

```
outeroop:
  for (i=1;;++i) {
    switch (i) {
      case 1:
      case 2:
        messageNwait("i=1 or i=2");
        break; // Done with these cases
      case 3:
        break outeroop;
    }
  }
```

Functions

We use the term “function” to mean something that can be called in some manner from the macro language. Slick-C has three kinds of functions: procedures, commands, and built-ins. The sections that follow describe each of these, and their differences.

Defining a Procedure

There are several differences between defining a procedure and defining a command with the **_command** primitive. Unlike command functions, the scope of a procedure may be limited to a module. Command functions may be invoked by typing the name on the Visual SlickEdit command line, from a menu item definition (see ["Changing or Adding Menus"](#) on page 327), by using the **execute** function, or by typing the name followed by arguments in parentheses in a Slick-C expression. Procedures may only be called by the latter method and may not be bound to keys. A procedure name must be a valid Slick-C identifier (same as C identifier). However, the name of a command may be a keyword (not recommended), or a string constant containing a single character such as "/" (Visual SlickEdit uses this to define a search command).

The syntax for defining a procedure is:

```
[static] [TypeName | void] id([ArgDecl1, ArgDecl2, ...])
{
    statement
    statement
    ...
}
```

Argument Declarations

The syntax for *ArgDecls* is the same as for declaring a variable except that the **static** keyword may not be used. In addition, an **&** before the *id* declares a call by references parameter. Call by reference array and hash table parameters require parentheses around the **&** and *id*. See example below.

The last argument in the declaration list may be an ellipsis ('...' 3 dots) 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. See ["TypeName"](#) on page 263. If *TypeName* or **void** is not specified, **return** statement checking is relaxed. 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** key word is used to limit the scope of a procedure to the module it is defined in. By default, procedures are global and may be accessed by any module. Procedures are called by specifying the name followed by comma delimited arguments, if any, in parentheses.

Example:

```
// 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.
//
// pfn is a reference to a pointer to a function.
boolean proc(int &p1,_str p2,_str (&list)[],int (*&pfn)(int))
{
    return(true);
}
```

Default Arguments

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 should always be specified in the function definition but not in the prototype. Unlike C++, default arguments in prototypes have no 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();           // Used defaults
    proc("param1");   // Used the second default value
    proc("param1",3);
    proc(,3); -->This is not allowed
}
```

Procedures may have up to 15 arguments defined. The procedure may be called with more arguments than defined by the procedure declaration. These "extra" arguments as well as the arguments defined in the procedure declaration can be retrieved by the function **arg**. Defining arguments with default values instead of using the **arg** function makes your code more understandable. Calling the **arg** function with no parameters returns the number of parameters the function was called with. The minimum number of arguments the procedure may be called with is defined by the procedure heading.

The OLD SYNTAX for defining a procedure is:

```
[static] [type-name] id([ var] id1, [var] id2, ... )
{
    statement
    statement
    ...
}
```

The above syntax has the same meaning as the following new syntax:

```
[static] [type-name] id(typeless [&] id1, typeless [&] id2, ...)
{
    statement
    statement
    ...
}
```

The keyword **var** placed before the parameter declares the parameter to be a call by reference parameter.

Defining a Command

The **_command** primitive is used to define a new command with argument completion. Commands must be translated into pcode and loaded by the **load** command ("Macro", "Load Module..."). A command may be invoked by typing its name on the Visual SlickEdit command line, selecting it from a menu item definition (see "[Changing or Adding Menus](#)" on page 327), pressing a key, quoting its name in a Slick-C function, or typing its name followed by arguments in parentheses in a Slick-C expression. Command procedures always have the scope global and may be bound to a key with the [Bind Command to Key dialog box](#) ("Tools", "Configuration", "Key Bindings...").

The syntax for defining a command is:

```
_command [TypeName | void] name1[,name2 [,name3... ]( [ArgDecl1, ArgDecl2, ...] )
    [name_info(const_exp)]
{
    statements
}
```

TypeName specifies the return type of the command. See "[TypeName](#)" on page 263. If *TypeName* or **void** is not specified, **return** statement checking is relaxed. 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. If your command uses the **arg** function to access arguments, specify an ellipsis ("...") for the last argument.

The syntax for *ArgDecls* is the same as for declaring a variable except that the **static** keyword may not be used. In addition, an **&** before the *id* declares a call by references parameter. Call

by reference array and hash table parameters require parentheses around the **&** and *id*. However, all typed or named arguments must have a default value.

The last argument in the declaration list may be an ellipsis ('...' 3 dots) to indicate that the function accepts more arguments of any type. Use the **arg** function to access these optional arguments.

The name of a command may be a valid Slick-C identifier, keyword (not recommended), or string constant of length one such as "/" (Visual SlickEdit uses this to define a search command).

Example

```
//Allow command in read only mode
//Use ellipsis because this accesses arguments
_command int goto_line(...)
    name_info('','VSARG2_READ_ONLY|VSARG2_REQUIRES_EDITORCTL')
{
    param=arg(1);
    if (param==" " || ! isinteger(param)) {
        message('Please specify line number');
        return(1);
    }
    p_line=param;
    return(0);
}
_command void mycommand(_str filename="") name_info(FILE_ARG)
{
    if (filename=="") {
        _message_box("No filename specified");
    }
    message("filename="+filename);
}
```

Commands get 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 on-line help for more information on **parse**). When another macro calls a command, more than one argument string may be passed. Calling the **arg** function with no parameters returns the number of parameters the command or procedure was called with.

name_info Attributes

The optional constant expression *const_exp* is used for argument completion, restricting when the command can be executed, and a few other options.

const_exp IS A SINGLE CONSTANT STRING 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 file "slick.sh" for constants that end in "**_ARG**". *const_exp* may contain one or

more of the **_ARG** constants. Separate each **_ARG** constant with a space. A 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 enabled/disabled, and a few other things. One or more of the following flags may be specified and ORed together with the bitwise OR (!) operator:

```
VSARG2_CMDLINE // Command supports the command line.
                // VSARG2_CMDLINE allows a fundamental mode
                // key binding to be inherited by the command line
VSARG2_MARK     // ON_SELECT event should pass control on
                // to this command and not deselect text first.
                // Ignored if command does not require an editor control
VSARG2_QUOTE    // Indicates that this command must be quoted when
                // called during macro recording. Needed only if
                // command name is an invalid identifier or
                // keyword.
VSARG2_LASTKEY  // Command requires last_event value to be set
                // when called during macro recording.
VSARG2_MACRO    // This is a recorded macro command. Used for completion.
VSARG2_TEXT_BOX // Command supports any text box control.
                // VSARG2_TEXT_BOX allows a fundamental mode
                // key binding to be inherited by a text box
VSARG2_NOEXIT_SCROLL // Do not exit scroll caused by using scroll bars.
                // Ignored if command does not require an editor
control
VSARG2_EDITORCTL // Command allowed in editor control.
                // VSARG2_EDITORCTL allows a fundamental mode
                // key binding to be inherited by a non-MDI editor control
VSARG2_NOUNDOS  // Do not automatically call _undo('s').
                // Require macro to call _undo('s') to
                // start a new level of undo.

VSARG2_READ_ONLY // Command allowed when editor control is in strict
                // read only mode. Ignored if command does not require an
                // editor control
VSARG2_ICON      // Command allowed when editor control window is iconized
                // Ignored if command does not require an editor control

VSARG2_REQUIRES_EDITORCTL // Command requires an editor control
VSARG2_REQUIRES_MDI_EDITORCTL // Command requires MDI editor control

VSARG2_REQUIRES_AB_SELECTION // Command requires selection in active
buffer
VSARG2_REQUIRES_BLOCK_SELECTION // Command requires block/column selection
// in any buffer
VSARG2_REQUIRES_CLIPBOARD // Command requires editorctl clipboard
VSARG2_REQUIRES_FILEMAN_MODE // Command requires active buffer to be
// in fileman mode
VSARG2_REQUIRES_TAGGING // Command requires <ext>_proc_search/find-tag
support

VSARG2_REQUIRES_SELECTION // Command requires a selection in any buffer
VSARG2_REQUIRES_MDI // Command requires MDI interface may be because
```

```
// it opens a new file or uses _mdi object.
// Commands with this attribute are removed
// from pop-up menus which the MDI interface is
// not available (editor control OEMs).
```

Example

```
#include "slick.sh"
// This command supports completion where the first argument
// is a filename and the second argument is an environment variable.
_command test1(...) name_info(FILE_ARG" "ENV_ARG)
{
    parse arg(1) with file_name env_name;
    message("file_name="file_name" env_name="env_name);
}
// This command is enabled only when the target is an editor control
// which has a selection.
_command void gui_enumerate()

name_info('','VSARG2_REQUIRES_EDITORCTL|VSARG2_REQUIRES_AB_SELECTION)
{
    ...
}
// This command supports completion on multiple filenames
_command                                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 Space Bar key. If a asterisk (*) is appended to the end of a completion constant, that command must support a space delimited list of strings. Double quotes are placed around arguments with embedded spaces.

The value of *const_exp* may be retrieved by the built-in function **name_info**.

Function Prototypes

Function prototypes are used to give the compiler 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 will recommend that you add a function prototype to your source so the compiler can find your error. You may 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 parenthesis of the parameter list. Unlike C++, default arguments in prototypes have no effect on the compiled code. No code or **name_info** is given.

Example:

```
int proc(_str s,_str list[]); // Function prototype
```

```
int (*pfn)(_str s,_str list[])=proc; //Pointer to function
_command void command1(...); // Function prototype
_command void command1(...) { //Must have ... here to match prototype
    // Use arg function here to get or set arguments.
}
```

What is a Built-in Function?

A built-in function is a function which was implemented in the interpreter and was not written in the Slick-C language.

Where Are the Functions?

There are over 1200 documented functions and 200 properties.

There are two ways to find the function you are looking for. First, you can use the menu item ("Help", "Macro Functions by Category...") which shows you smaller lists of these functions by category. Second, you can look at source code for existing commands we have already written which you think will call functions similar to ones you need. For example, let's say you want to write a macro which gets the word at the cursor and does something with it. Try to think of a command in Visual SlickEdit which operates on the current word. The **upcase_word** command (Ctrl+Shift+U) uppercases the word at the cursor. If you don't know the name of the command but you do know the key which invokes the command, use the **what_is** command ("Help", "What is Key...") to find out the name of the command that gets executed. Now that you have the name of the command, use the **find_proc** command ("Macro", "Find Slick-C Proc...") to bring up the macro source code. If you executed the command "**find_proc upcase_word**", you will quickly see a procedure called **cur_word** which looks like what you want. Place your cursor on any part of the **cur_word** procedure and press the "?" button on the button bar.

Differences Between Commands, Built-ins, and Procs

- 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 and built-ins always have the scope **global**. Procedures may have the scope **static** (module) or **global**.
- Commands may be bound to keys. Built-ins and procedures may not.
- Commands may be invoked from the command line or the **execute** function. Built-ins and procedures may not.
- Built-ins that do not return a value do not require parentheses when called (this will change in the future). For example, the **left** built-in may be called by with **left** or **left()**. You should always use parentheses so you don't have to remember which function requires parentheses.
- 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).

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 may be invoked by typing the name (extension not required) followed by arguments on the Visual SlickEdit command line, quoting the name in a macro, or by using the **execute** function. If the batch macro needs to be recompiled (MAKE), 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** will not be invoked and the module will remain in memory. If you accidentally load a batch program, use the **unload** command to unload it. When a batch program is executed, the **defmain** procedure is called after the procedure **definit** is called (see "**Module Initializations**" on page 292).

The syntax of the **defmain** function is:

```
[TypeName | void] defmain()
{
    statement
    statement
    ...
}
```

TypeName specifies the return type of the function. See "*TypeName*" on page 263. If *TypeName* or **void** is not specified, **return** statement checking is relaxed. 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. Beware, the execute function only supports returning an **int** type. You will need to 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 command line arguments will be in **arg(1)**. Use the **parse** statement to easily parse multiple space delimited arguments.

Example:

```
defmain()  
{  
    messageNwait("Arguments given: "arg(1));  
    parse arg(1) with word1 word2 .;  
    messageNwait("word1="word1" word2="word2");  
    return(0)  
}
```

The example above displays the arguments given to the macro on the Visual SlickEdit message line. If you define a procedure in a batch program, use the **static** keyword when possible to conserve memory. Visual SlickEdit stores the names of global procedures and variables in a names table.

Extending the editor with a batch macro has the advantage of conserving memory and reducing the size of the state file. In addition, batch macros can more easily be 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:

1. Enter the name of the module (extension not necessary) followed by arguments on the Visual SlickEdit command line.
2. Enter "**vs -p *program***" at the shell prompt, where *program* is the name of the batch program and "vs" is the name of the editor. The "-p" option tells Visual SlickEdit to terminate when the batch program completes. Alternatively, you may use the "-r" option to have Visual SlickEdit remain resident after the batch program completes.

For any of the above methods, Visual SlickEdit will invoke the translator to compile the source code file if the source code file exists and its date is later than the date of the pcode (.ex) file.

Preprocessing

Preprocessing in Slick-C is identical to C/C++ except that a **#error** directive has been added. Preprocessing allows you to conditionally compile source code or define textual replacements

The syntax of the Slick-C language conditional **if** block is:

```
#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 allowed. Unlike in C, the expression specified **MUST** be valid.

To display an error message and abort the compile, use the **#error** directive as shown below.

```
#error expression
```

Typically, preprocessing is used to help write macros which operate on multiple operating systems or environments. The following constants are automatically defined by the Slick-C translator.

<code>__PCDOS__</code>	Non-zero if current operating system is Windows. Use machine() built-in function to determine at run-time which of these operating systems you are running.
<code>__UNIX__</code>	Non-zero if current operating system is UNIX compatible.
<code>__NT__</code>	Non-zero if the current operating system is Microsoft NT compatible.
<code>__VERSION__</code>	Version number of Visual SlickEdit.

The above identifiers have two underscore characters, one before and one after the alphabetic name part.

In addition, you can use the Slick-C translator **-d** option to define a constant for use by preprocessing. To test whether a constant has been defined, use the **defined()** function.

Example:

```
#if !defined(my_constant)
    #define my_constant "default value"
#endif
#if __PCDOS__
    name="c:\util\myprog"
#elif __UNIX__
    name="/usr/bin/myprog"
#else
    #error "Don't know what to do for this OS"
#endif
```

Pragma

#pragma is used to change various options during the course of a compile. Slick-C offers the following options:

#pragma option(auto decl [, (on | off)])

Turns **auto declvars** and **auto declctls** on/off. If the second argument is not given, value is restored to command line invocation value. Default is on.

See **auto declvars** and **auto declctls** below

#pragma option(auto declvars [, (on | off)])

Turns **auto declvars** on/off. If the second argument is not given, value is restored to command line invocation value. Default is on.

When on, compiler will attempt to automatically declare typeless variables when an assignment is made.

#pragma option(auto declctls [, (on | off)])

Turns **auto declctls** on/off. If the second argument is not given, value is restored to command line invocation value. Default is on.

When on, compiler will attempt to automatically declare control variables.

#pragma option(re declvars [, (on | off)])

Turns **re declvars** on/off. If the second argument is not given, value is restored to command line invocation value. Default is off.

When on, any variable may be redeclared as a **typeless** variable. This is used to generate code for variables without having the type information.

#pragma option(strict return [, (on | off)])

Turns **strict return** on/off. If the second argument is not given, value is restored to command line invocation value. Default is on.

When on, and an explicit return type is given to a function, the compiler will flag an error if a return statement is potentially missing.

#pragma option(strict ellipsis [, (on | off)])

Turns **strict ellipsis** on/off. If the second argument is not given, value is restored to command line invocation value. Default is on.

When on, an ellipsis (... 3 dots) must be given as the last argument to a function or prototype for type checking to succeed when calling function with extra arguments.

NOTE: All **#pragma** options may be specified by command line compiler options. Run **vst.exe** (UNIX: **vst**) with no arguments to view compiler options. You may use the **VST** environment variable to specify compiler options for all your macros.

Including Header Files

The syntax of the include statement is:

```
#include string_constant
```

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 will look for the include file in the directories specified by the **VSLICKINCLUDE** and **VSLICKPATH** environment variables. Include files may be nested.

Defining Controls

Typically, you do not need to tell the compiler about a control you are referring to; however there are a couple of cases in which you must declare a control. This can happen when the compiler can't safely assume that you are referring to a control, or when the compiler is confused about what dialog box the control you are accessing is on. The compiler needs to tell the linker which dialog box is supposed to contain your control.

The syntax for declaring a control variable is:

```
[_nocheck] ObjectName ControlName;  
OR  
[_nocheck] _control ControlName;
```

Where *ObjectName* may be one of the following: **_text_box**, **_combo_box**, **_image**, **_picture_box**, **_command_button**, **_radio_button**, **_check_box**, **_label**, **_list_box**, **_gauge**, **_spin**, **_vscroll**, or **_hscroll**.

The **_nocheck** keyword tells the compiler not to check if the control exists on the current dialog box.

Currently the "[**_nocheck**] *ObjectName ControlName*;" declaration is only allowed outside the scope of a function. Support for local procedure scope will be added in the future. 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. 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;
}

```

Defining Events and Event Tables

Event tables provide a mechanism for describing event/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 no **defeventtab** declaration exists 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. See "[defmain. Writing Slick-C Batch Files](#)" on page 284 for information on batch programs. Even though executing the module as a batch program unloads the module when the **defmain** function terminates, the event table changes remain present. This is because the command or procedure names and the event tables are both maintained separately from the modules. The **def** primitive adds or replaces the binding of an event table already in memory and may be used to unbind a key.

The syntax for defining a key is:

```
def {prefix_key} event [- event] [, event [- event]] ... = [command]
```

Command may 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* above may be any valid event name (see "[Event Names](#)" on page **Error! Bookmark not defined.**). Some event names do not need to be enclosed in quotes. However, you may want to always enclose them in quotes so you don't have to remember which ones require quotes.

Example:

```

def "A-x"=safe_exit // Note that "A-a" is different than "A-A"
                        // which requires the Alt and Shift keys to
                        // be pressed.
def "A-?"=help
def "C-X" "b"=list_buffers
def \0 - \255= nothing

```

The **defeventtab** primitive is used to define a new event table. The syntax for defining a an event table is:

defeventtab *name*

name may contain a period ('.') character. The period is used to separate the form name from the control name. The **def** primitive changes the binding of events of the last event table defined. If no event table is defined, the **default_keys** event table is used.

Example:

```
defeventtab c_keys
def " "=c_space
def "ENTER"=c_enter
```

Event tables are global in scope. When an event table is loaded by the **load** command or by executing the module as a batch program, the new bindings will replace the event bindings of the existing event table. If the event table specified by **defeventtab** does not exist, a new one is created.

Event Driven Dialog Boxes

Event tables are also used to create event driven dialog boxes and inheritance. Much of the event table definition code is automatically inserted by the dialog editor. However, once you are more familiar with event tables, you will probably type in some event table definition code yourself. If you are not familiar with some of the terminology used, you may want to read the section **Creating Dialog Boxes with the Dialog Editor** first. 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 plan on specifying inheritance for an event table attached to a control.

```
defeventtab form_name[.control_name] [_inherit [etab_name]]
```

The **_inherit** primitive lets you link one event table to another. This makes it possible to perform Clipboard Inheritance® (See "**Clipboard Inheritance(R)**" on page 306). If no name follows the **_inherit** keyword, the inheritance is unlinked. You can add event handlers by using the **def** primitive or by defining an event handler function as follows:

```
[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* above may be any valid event name (see "**Event Names**" on page **Error! Bookmark not defined.**). Some event names do not need to be enclosed in quotes. However, you may want to always enclose the event names in quotes so you don't have to remember which ones require quotes.

The syntax for *ArgDecls* is the same as for declaring a variable except that the **static** keyword may not be used. In addition, an **&** before the *id* declares a call by references parameter. Call by reference array and hash table parameters require parenthesis around the **&** and *id*.

Example (form with text box and OK button):

```
#include "slick.sh"

// Define an event table for the dialog box window.
defeventtab form1

// Since this is the first event handler defined for this control and
// the name of this control does not match the last defined event 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 form **form1**. A **form1** event table is not created because no event handler for this event table was defined. When you save the configuration ("Tools", "Configuration", "Save Configuration..."), event tables that are not used are deleted.

Module Initializations

The Slick-C language provides two module initialization functions called **definit** and **defload**. If present, the procedures **definit** and **defload** are called when a module is loaded. **definit** is called before **defload**. Once the module is saved by the **write_state** command, the **definit** procedure is invoked each time the editor is invoked. This gives your modules a chance to perform initializations such as creating a temporary file, or allocating a selection, or bookmark.

The syntax for defining the special functions **definit** and **defload** are:

```
definit()
{
    statements
}
defload()
{
    statements
}
```

The return value of these functions is always **void**. Therefore, you cannot specify an argument to the **return** statement. For better performance, use the **defload** primitive instead of the **definit** primitive when possible since 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 ("").

The remainder of this section will present two useful examples of using **definit**.

Example:

```
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 will be 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 the example below as a template for creating a temporary buffer in the hidden window:

```
#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_QUIT;
    }
    // Note: ELSE case cannot empty bookmark buffer unless mark ids
    // are freed. Might as well leave them.
    get_view_id(bookmark_view_id);
    activate_view(view_id);
}
```

Debugging Macros

A debugger will be added in the future. Fortunately, the Slick-C translator (**vstw.exe** UNIX: **vstw**) is fast, so that debug messages may be inserted into your code and compiled quickly. Use the **messageNwait** (pronounced as "message and wait") function to display a message and wait until a key is pressed. The **_message_box** function may be used to display a dialog box with a message and wait until you press Enter. Two helpful features are described in the following sections that will help you debug and work on Slick-C macros.

These ".e" extension aliases can come in handy:

```
m          messageNwait("%\n: %\c");
mb         _message_box("%\n: %\c");
```

Finding Procedures

The **find_proc** command ("Macro", "Find Slick-C Proc...") finds Slick-C source code or help for a Slick-C tag name you specify. This is especially useful if you are browsing a macro and you want to find out what a function does. You can find the procedure at the cursor by pressing the Ctrl+Period key.

The syntax of the **find_proc** command is:

find_proc *proc_name*

Command Line Examples:

find_proc find_proc	Finds the source code for find_proc .
find_proc cursor_up	Finds the source code for cursor_up .
find_proc substr	Displays help on substr built-in.

Finding Runtime Errors

When a Slick-C error occurs, a message box with a "Slick-C Error" title is displayed. Usually the Slick-C Stack toolbar is displayed listing the call stack at the time of the error. Double click in this toolbar to view source for a call stack entry. The **find_error** command ("Macro", "Find Slick-C Error") finds the last Slick-C interpreter run time error. The module with the error is loaded and the cursor is placed on the line causing the error.

Creating Dialog Boxes with the Dialog Editor

Dialog Box Creation Overview

Creating event driven dialog boxes in Slick-C is similar to Microsoft Visual Basic™ except that the language has C++ style syntax and is more object oriented.

Here are some key differences between Slick-C and 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 Visual Basic. If a button control receives an event and executes a statement like this, **p_caption="New button caption"**, the button's caption is changed and NOT the caption for the dialog box, which is what Visual Basic would do.
- Built-in properties all start with the prefix "p_" to avoid these keywords from conflicting with your own identifiers.
- A more general method of object instance referencing is used. See "**Instance Expressions**" on page 311 for more information.
- Almost all properties that can be accessed at design-time may 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. See "[Dialog Box Inheritance Order](#)" on page 309 for more information.
- 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. See "[Clipboard Inheritance\(R\)](#)" on page 306 for more information.
- Functions may be used as methods which operate on an instance of an object. See "[Using Functions as Methods](#)" on page 312 for more information.

Dialog Editor Summary

Press Ctrl+Shift+Space to edit a dialog box that is being run. If you accidentally press Ctrl+Shift+Space when the [Properties dialog box](#) is active, 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.

Creating Controls

- Double click left mouse button on [Properties dialog box](#) control bitmap to create a control in the center of the form.
- Click left mouse button on [Properties dialog box](#) control bitmap. Click left mouse button on form, drag, and lift mouse button to create control the size of the dotted rectangle.

Selecting Controls in Form

LButton	If the control is not already selected, this deselects all selected controls and selects the current control.
Shift+LButton	Toggle selection of a control on/off.
LButton-ClickNDrag	When you click and drag on the form, a dotted rectangle is displayed. When you lift the left mouse button, the controls within the rectangle are selected.

Deleting Controls

- Select control(s). Press Backspace or Del key to delete the selected controls.

Setting Properties

- Select control(s). Click left mouse button on property in Properties list box. Type new value in Properties combo box. Use combo box arrow or dots to assist in entering a value. Press Enter when Properties list box is active, this sets the property.
- Select control(s). (some properties) 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 you want to align the other controls with. 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 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 arrow to move lower right corner of the selected controls by one pixel.

Moving Controls

- Select control(s). Click and drag with the left mouse button on any of the selected controls.
- Select control(s). Set the x or y property.
- Select control(s). Press Left, Right, Up, or Down arrow to move selected controls by one pixel.

Miscellaneous Assignments While Form is Active

RButton	Displays menu with various dialog editor commands.
Ctrl+Shift+Space	Loads form and Slick-C code. Runs dialog box. If you accidentally press Ctrl+Shift+Space when in the Properties dialog box , you will be editing the Properties dialog box . Double Click on the system menu to close the edited Properties form. Some UNIX window managers do not close windows when you double click on the system menu.
Ctrl+S	Loads form and saves into state file. Under UNIX, this may just list source for the form that can be executed the next time you invoke Visual SlickEdit.
Ctrl+L	Loads form
Ctrl+C	Copies selected controls.
Ctrl+V	Pastes controls from the clipboard.
Ctrl+X	Cuts selected controls.
Ctrl+A	Selects all controls with same parent as the already selected control(s).
Tab	Deselects all controls and selects next control in tab order (p_tab_index).
Shift+Tab	Deselects all controls and selects previous control in tab order.
LButton-Double-Click	(on control) Displays Select an Event Function dialog box for adding or modifying event handlers.

Miscellaneous Menu Items

System Box of form, Show Properties	Display Properties dialog box .
Window menu, Properties	Display Properties dialog box .
Window menu, Selected Form	Display selected form (form being edited).
Macro menu, New Form	Creates a new dialog box with a default name.

Macro menu, Open Form	Open existing dialog box or create new dialog box.
Macro menu, Grid	Sets the distances between the dots on edited form.

The remainder of this section discussing various pieces of building a dialog box.

Creating a Form

The term **form** refers to the outer window of a dialog box. The objects within the dialog box are called "controls". The term **form** also refers to the entire dialog box as well. A new form can be created in one of two ways:

1. Use the New Form menu item ("Macro", "New Form").

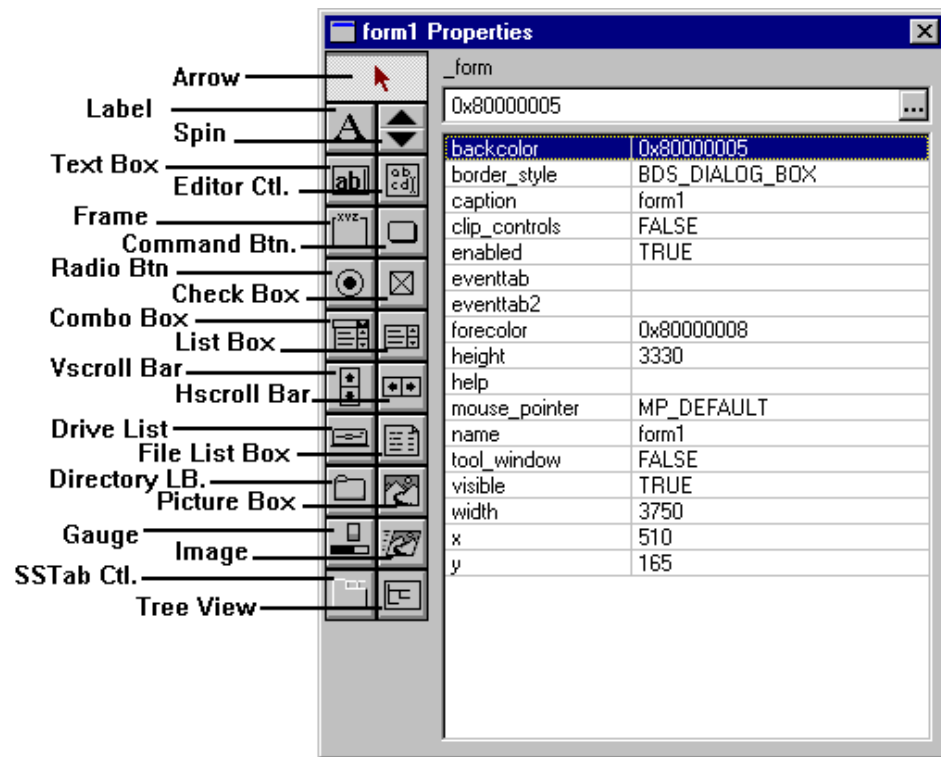
OR

2. Use the Open Form menu item ("Macro", "Open Form...") and specify the name of a form that does not already exist.

Adding Controls

("Window", "Properties..." or **show_properties** command)

Below is a picture of the Properties dialog box:



The bitmaps on the left of the Properties dialog box are used to create controls. There are two ways to create a control. You can double click the left mouse button on the bitmap of the control you want to create. This will place a new control in the middle of the selected form (form1 for this sample session). However, with this method, you cannot create a control and place it within another control. The Picture Box and Frame controls allow you to place controls inside of them.

Here's another way to create a control. Perform these steps now to create a text box control on form1:

- Single click on the text box bitmap so that it appears pressed in (like the arrow bitmap above).
- Move your mouse so that it appears on top of the form, form1, you are editing. If you can no longer see the form you are editing, use the Selected Form menu item ("Window", "Selected Form") to display it. Notice that the mouse pointer does not

look like a pointer any more. It should look like a large, thin plus sign. This indicates that you are in creation mode.

- 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 will appear within the rectangle.

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 will be referenced in the code. You should prefix your control names with **"ctl"** so you can easily recognize them. To add an event handler, perform the steps below:

- 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 should appear.
- Choose 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. Type a unique filename. Usually this filename is derived from the name of the dialog box you are creating (like **"form1.e"**).

After performing the above steps, the dialog editor will insert an event function definition into your source file and place your cursor in the function.

Loading and Running the Form

Invoke the **run_selected** command ("Macro", "Load and Run Form") to "run" the current dialog box you are editing. This will load the code, load the dialog box, and run the dialog box. The edited dialog box is not closed when you run it. 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). You can press Ctrl+Shift+Space when any dialog box is running to edit it (this includes the **Properties dialog box**).

You can display your dialog box from the command line by typing **"show <FormName>"**. To display the dialog box modally enter **"show -modal <FormName>"** on the command line. The **show** command provides many powerful options. For more information, see **"Displaying Dialog Boxes"** on page 301. Dialog box templates and compiled macros are stored in the state file ("vslick.sta" or "vslick.stu" under UNIX).

The example code below illustrates how to write a command which displays a dialog box. This is useful for binding a command to a key which displays a dialog box.

```
#include "slick.sh"
_command void run_form1()
{
    // The -modal option displays other windows while the dialog box
    // is displayed.
    show("-modal form1");
}
```

Saving a Form

Click on the form being edited and press Ctrl+S.

Adding a Cancel Button

Double click the left mouse button on the command button control. Set the "**caption**" property to "**Cancel**", set the "**cancel**" property to TRUE, and set the "**name**" property to "". You can set the **cancel** property to TRUE by double clicking the left mouse button on the **cancel** property in the Properties list box. As a rule of thumb, you should set the name property of a control (never the form) to "" if you are not going to reference the control by name because it wastes code space. Pressing the Cancel button (the command button with the **p_cancel** property set to TRUE) 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 will need to close the dialog box yourself (**p_active_form._delete_window()**).

Adding an OK Button and Closing a Dialog Box

- Create a command button control by double clicking on the command button control in the Properties dialog box.
- Set the "**caption**" property to "**OK**", set the "**default**" property to TRUE, and set the "**name**" property to "**ctlok**".
- 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 should appear.
- Choose the **lbutton_up** 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. Type a unique filename. Usually this filename is derived from the name of the dialog box you are creating (like "**form1.e**").

After performing the above steps, the dialog editor will insert an event function definition into your source file and place your cursor in the function. Add the code as shown below:

```
#include "slick.sh"

defeventtab form1
// Code for OK button.
ctlok.lbutton_up()
{
    // Close the dialog box and return a value. The _delete_window
    // function allows modal dialog boxes to return a value. For
    // more information, see
    // "Displaying Dialog Boxes" below. Each object
    // in the dialog box will receive an on_destroy event.
    // NOTE: If "" is a valid return value. Return 1 here and store
    // your results in the global _param1 variable.
    p_active_form._delete_window("return value");

    // Statements after closing a dialog box are executed.
}

```

There are some subtle points about closing a dialog box. First, if a modal dialog box returns a value, the value "" (zero length string) MUST be returned to indicate 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. Second, you will need to 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 most common way to display a dialog box is with the **show** command. The **show** command can be called in function style syntax from within a macro. It can also be invoked from the command line or a menu item (see "Changing or Adding Menus" on page 327 for more information).

Command line call syntax:

show *cmdline*

Function call syntax:

show(*cmdline* [,*arg1* [,*arg2* ... [*argN*]]])

cmdline is a string in the format:

[*option*] *form_name*

option may be one of following:

- mdi Keep the form on top of the MDI Window.
- app Keep the form on top of the Visual SlickEdit Application Window.
This allows the MDI Window to be displayed on top of the form.
- xy Restore the previous x, y position of the dialog box. If the old position cannot be found, the dialog box is centered. When the dialog box is

	closed, the x, y position is automatically saved (the dialog manager calls _save_form_xy).
-hidden	Do not make the form visible.
-modal	Run the form modally. All other forms are disabled. Control returns to the caller when the form window is deleted with _delete_window .
-nocenter	Do not center the form.
-new	Normally, when a form is already displayed, the existing form is given focus. This option allows for multiple instances of a form to be displayed.
-reinit	UNIX only. Ignored by other platforms. Causes _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 <code>form_parent()</code> function will not work because the next time the form is used, the parent is not changed to the new parent specified.
-hideondel	UNIX only. Same as -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

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**.

If the "-modal" option is not given, the form window id is returned if successful. Otherwise, a negative error code is returned.

Example:

```
// This example requires that you create a form called form1 with a
// command button and load this file.
#include "slick.sh"
_command mytest()
{
```



```

        result=show("-modal form1");
        if (result=="") {
            return(COMMAND_CANCELLED_RC);
        }
        message("_param1="_param1" _param2="_param2);
    }

defeventtab form1
ctlcommand1.on_create()
{
    // Global variable _param1.._param10 are defined in "slick.sh" to
    // allow for multiple strings
    // to be returned in separate variables.  Alternatively, if the
    // return strings do not contain spaces, you could concatenate
    // them together with a space and use the parse built-in to easily
    // separate them.
    _param1="string1";
    _param2="string2";
    // Close the dialog box and indicate that the dialog box was not
    // canceled. Each object in the dialog box will receive an
    // on_destroy event.
    p_active_form._delete_window(1);
}

```

Example:

```

// This example requires that you create a form called form1 with a
// command button and load this file.
#include "slick.sh"
_command void mytest()
{
    show("-modal form1","param1 to on_create", "param2 to on_create");
}

defeventtab form1
ctlcommand1.on_create(_str arg1="", _str arg2="")
{
    //arg1=arg(1); Could get the arguments with the arg built-in
    //arg2=arg(2);
    messageNwait("arg1="arg1" arg2="arg2);
}

```

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) {
        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);
}
```

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. 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 example above).

Modeless Example:

```
#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);
}
```

Sometimes you need to display a status dialog box while doing some sort of processing. This requires a modeless dialog box so control is returned to you. However, you still probably want to disable all other dialog boxes including the MDI window while you do your processing.

Advanced Modeless Example

```
#include "slick.sh"
static typeless gcancel;
_command void test()
{
    // Show the form modeless so there is no modal wait
    form1_wid=show("form1");
    // Disable all forms by the one with p_window_id==form_wid. A
    // space delimited string of disabled form window ids is returned.
    disabled_wid_list=_enable_non_modal_forms(0,form_wid);
    gcancel=0;
    for (;;) {
        // Read mouse, key, and all other events until non 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 will always remain on top of the parent window. Use the **show** command and specify the "-app" option if you want to allow a modeless dialog box to appear 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 Find, Replace, and Spelling dialog boxes to determine which buffer to operate on. This allows these 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. This does mean that these dialog boxes only support certain parent windows. For example, the Find dialog box will not run correctly if the "-app" option of the **show** command is used.

Remembering a Dialog Box Previous Position

By default, the **show** command centers the dialog box to the current form or MDI window. Usually this is fine. However, sometimes it is helpful for a dialog box to reappear in the same position 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(R)

Clipboard Inheritance® (US patent No. 5,710,926) is a mechanism by which objects can be transferred from one place to another via the clipboard in order to create new instances which inherit the code of the original objects. Code for the new instances can be added, affecting only the new instances, and code of the original instances can be modified, affecting both instances. This is just another way to describe inheritance. The difference is that, with Slick-C, the inheritance links are built for you in a very intuitive, natural way.

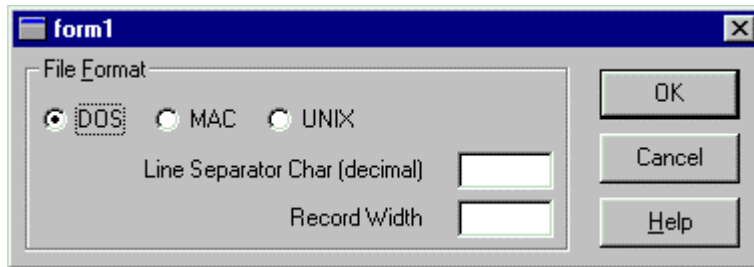
The most obvious flaw in Microsoft Visual Basic™, Hypercard, and other high level user interface building systems, is the loss of event handlers when you copy controls or dialog boxes from one place to another via the clipboard.

A sample problem with some heavily commented source code will best demonstrate how the Slick-C language works.

Let's say you want to create a group of controls which are needed by Visual SlickEdit's File Open dialog to allow the user to specify the various supported file formats. The file formats Visual SlickEdit supports are:

- 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.
- A single user specified line separator character.

Here is a partial dialog box which can handle Visual SlickEdit's file formats:



And here is the code which drives this dialog box:

```

// 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 users modifies the text in the text box. The event table
// automatically created here is form1.ctlopenlinesep.
ctlopenlinesep.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!='') {
        ctlopenlinesep.p_text=''; // Clear out the other text boxes text.
        zap_radio_buttons();    // Turn off all the radio buttons
    }
}

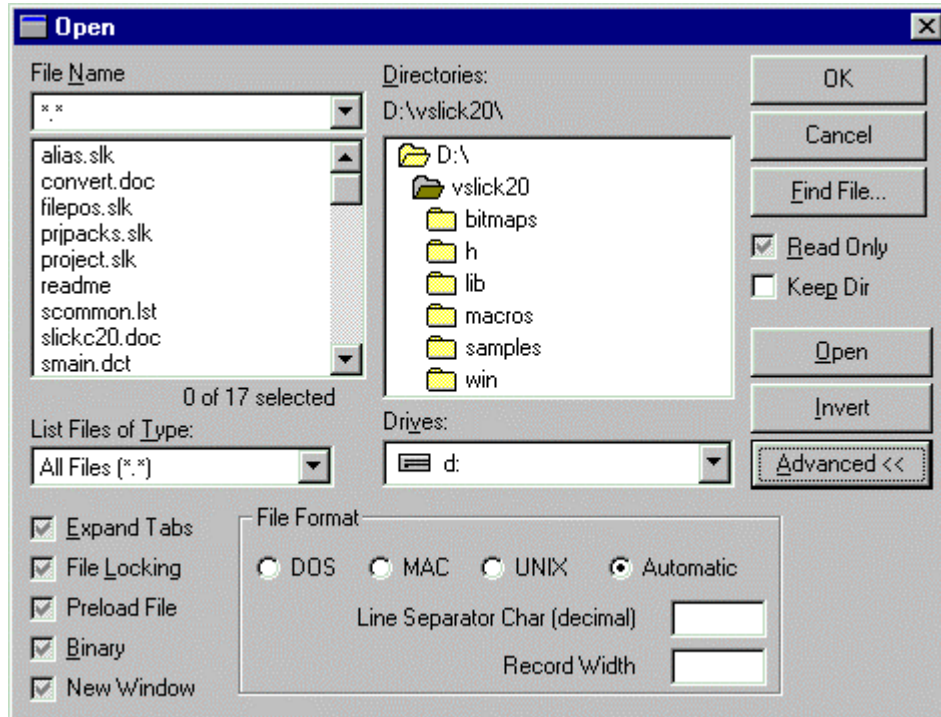
```

Notice that only the first radio button **ctlopendos** has an event handler defined. The other radio buttons just need to use the **form1.ctlopendos** event table. This can be accomplished in the dialog editor in one of two ways. You can use Clipboard Inheritance® or, if the radio buttons are already created, you can just set the **p_eventtab** property of the other radio buttons to **form1.ctlopendos**. To use Clipboard Inheritance® you would simply write the **lbutton_up** event code for the DOS radio button, copy the DOS radio button to the clipboard, paste it back onto the dialog box within the frame, and set the **p_caption** property for the new radio button to MAC. Either of these two methods can be used to propagate an event table. When **ctlopendos.lbutton_up()** function gets called, it simply gets and sets the properties of controls which obviously exist on this dialog box.

How Visual SlickEdit's File Open Dialog Box was Built

Clipboard Inheritance® is a very powerful concept. If you take a close look at some of Visual SlickEdit's complex dialog boxes, you will find out that there are massive amounts of inheritance going on which was simply created by copying some controls to the clipboard, and pasting them.

Below is a picture of Visual SlickEdit's ("fast" Windows 3.1 style) Open File dialog box:



The Open File dialog box has the form name `_edit_form`. This dialog box was actually 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 the same advanced controls. The advanced controls were taken from the Open File dialog box ("`_edit_form`") and all its related controls, and copied them 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.

Visual SlickEdit's Find and Replace dialog boxes both allow searching across multiple files and/or buffers. The controls for selecting files and buffers on these dialog boxes share the

same code. You might want to copy these controls onto your own dialog box to provide multiple file and buffer processing for some other operation. In Visual SlickEdit 2.0, these controls were pasted onto the Print dialog box.

It is important to note that while these examples did not require linking one event table to another, the dialog editor recognizes when inheritance is required and will automatically link event tables together. The statement below illustrates the syntax for linking one event table to another.

```
defeventtab dlgbox2.textbox1 _inherit dlgbox1.textbox1
```

Dialog Box Inheritance Order

Each control in Slick-C has two properties called **p_eventtab** and **p_eventtab2**. The **p_eventtab** property defines the user level 1 inheritance. User level 1 inheritance is intended to allow you to modify the event handlers for one specific instance of a control without affecting any other (except when Clipboard Inheritance® is used). Like Microsoft Visual Basic™, you simply double click on the combo box when you want to add an event handler. The dialog editor will automatically insert the necessary function declaration code so you need only add statements within the function. Once you have written the event handler and loaded the new code, the **p_eventtab** property displayed in the Properties list box will be updated to reflect that you have defined a user level 1 event table. The **p_eventtab2** property defines the user level 2 inheritance. User level 2 inheritance is typically used to affect all controls of a specific type. Normally, the dialog editor sets these properties for you when a control is created. For example, when you create a combo box control with the dialog editor, the **p_eventtab2** property is automatically set to **_ul2_combobox**. The **_ul2_combobox** event table defines the default processing used by every combo box. The user level 1 event handler receives an **on_change** event (sent from the user level 2 code) when the text in the combo box changes.

Visual SlickEdit uses a pre-defined inheritance order called the 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 we are searching is a key event, check the dialog box frame's user level 1 inheritance.
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 dialog box frame's user level 1 inheritance.
6. Check dialog box frame's 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.

Through trial and error, we found the above Dialog Box Inheritance Order to be the most powerful. It allowed the dialog manager and all the default control event handling to be written in Slick-C.

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 the image control, have a window id but DO NOT allocate an operating system resource known as a window.

The 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 another object's window id. Accessing a property without specifying a control name or instance handle accesses the property of the active object and not the active form. This is significantly different than Microsoft Visual Basic™. For example, if a button control receives an event and executes a statement like this, **p_caption="New button caption"**, the button's caption is changed and NOT the caption for the dialog box, which is what Visual Basic would do.

IMPORTANT: Changing the active object does NOT change the focus. Use the **_set_focus** method to change the focus.

The Active Form

Slick-C has a **p_active_form** property which 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 can be found rather quickly by traversing through the parents (p_parent) of the active object until a form is reached.

Instance Expressions

The following examples illustrate common instance expressions:

```

ctltext1.p_text="test";    // Assuming ctltext1 has been declared
globally
                           // or locally, lookup the
                           // ctltext1 control of the active form to get
                           // the window id, and set the
                           // p_text property.
x=_control ctltext1;      // Put the window id of the "ctltext1" control
of
                           // the active form.
                           // in the variable x. The variable x does not
                           // have to be declared.
                           // There are 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 spaces. This shows that multiple
                                // periods ('.') may be used in an instance
expressions.
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 were 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 may be called as a method without any additional declaration data. The sample Slick-C source below illustrates this feature:

```
#include "slick.sh"
defmain()
{
    // Call the tbupcase function as a method to operate on Visual
    // Visual SlickEdit's 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.
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);
}
```

Notice that **tbupcase** is not defined to be a method of a particular class. There are a significant number of functions which can operate on any Visual SlickEdit object. This feature also allows macros written in text mode Visual SlickEdit to be converted into Visual SlickEdit macros and used as methods. Also, most functions in Visual SlickEdit are written to operate on the current object which means you have access to a lot of methods. Using functions as methods is very useful when writing dialog box event handlers. The on-line help indicates which functions can be used as methods for specific objects. 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") may then be used to locate the source of the error. While using a function as a method is useful, it lacks the ability to define two functions with the same name which operate

on different objects. This can be solved by allowing a class name (possibly one of the built-in class names: `_text_box`, `_combo_box` etc.) when defining the function, like in C++. This will be added to a future version of Visual SlickEdit.

Built-in Controls

Label Control

The label control is used to display text in any font. Labels optionally may be aligned left, aligned right, centered horizontally, centered vertically, or centered horizontally and vertically. If you don't need to align the label, we recommend you should the `p_auto_size` property to `TRUE` to ensure that the text inside the window fits. 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. Select the Label control and use the up, down, left, and right arrow keys to help you center the label to the text box.

For a complete list of label control properties, methods, and events, use the menu item ("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 (the text box usually appears to the left of the spin control). This can be performed WITHOUT writing any code, simply by making the `"tab_index"` property of the text box one less than the `"tab_index"` property of the spin control. You will get an error 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. The easiest way to do this is to create the text box and then create the spin control. You will want to turn off the `"auto_size"` property of the text box so you can make the height of the text box larger than the font. Once you get the Spin control close to the right edge of the text box, select the spin control and use the up, down, left, and right arrow keys to help you center the Label to the spin control. The default increment is 1. You can also use the spin control to increment/decrement the value in a gauge or scroll bar control or increment/decrement a hexadecimal number displayed in a text box. In this case, however, you must set the `"increment"` property of the spin control to zero and process the **`on_spin_up`** and **`on_spin_down`** events yourself. 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 don't care.

Example:

```
#include "slick.sh"

// This example requires form name form1 with a text box and spin
// control. The spin control should be named ctlspin1 and the
// "increment" property should be zero. The tab index of the text
// box MUST be one less than the spin control. This code does not
// reference the name of the text box so that you can use Clipboard
// Inheritance(R) to create multiple working copies of a spin control
// capable of incrementing/decrementing the value in a text box control
```

```
// without writing any new code.
defeventtab form1
ctlspin1.on_change(reason)
{
    if (reason==CHANGE_NEW_FOCUS) {
        return(p_prev);
    }
}
ctlspin1.on_spin_up()
{
    new_dec_value=hex2dec(p_prev.p_text)+1;
    p_prev.p_text=dec2hex(new_dec_value);
}
ctlspin1.on_spin_down()
{
    new_dec_value=hex2dec(p_prev.p_text)-1;
    p_prev.p_text=dec2hex(new_dec_value);
}
```

For a complete list of spin control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

Text Box Control

The text box control allows the user to enter a single line of text. Use the editor control to allow any number of lines to be entered. Text boxes support completion with the Space Bar and question mark keys. Set the "completion" property of the text box if there is an existing completion type which suits your needs. The FILE_ARG completion type is the most common. It provides completion on file names. You can write new commands which operate in all text boxes, edit windows, and editor controls.

Example:

```
#include "slick.sh"
_command 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 above to Alt+F12. Notice that this command works in all text boxes, edit windows, and editor controls. The key binding may not work in a text box if you bind upcase_line to one of the CUA keys Alt+A..Alt+Z, Ctrl+X, Ctrl+C, or Ctrl+V. Use the **Redefine Common Keys dialog box** ("Tools", "Configuration", "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, use the menu item ("Help", "Macro Functions by Category").

Editor Control

The editor control allows a user to enter multiple lines. The editor control is used to view clipboards, for the calculator, and for version control comments. Almost all your key bindings for an MDI edit window will work in an editor control even if you have changed your emulation. You can use macro recording (see "[Recording Macros](#)" on page 90) to write a new command which works in an edit window and editor control. Just check the "Allow in non-MDI editor control" check box when you finish recording your macro.

For a complete list of editor control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

Frame Control

The frame control is used to group a set of related controls. Often, radio buttons are placed inside of a frame control to tell 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 a frame control:

- Click the left mouse button on the bitmap in the [Properties dialog box](#) of the control 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, use the menu item ("Help", "Macro Functions by Category").

Command Button Control

The command button control is most typically used to create an OK, Cancel, or Help button. See "[Adding a Cancel Button](#)" on page 300 and section "[Adding an OK button and Closing a Dialog Box](#)" on page 300 for more information.

For a complete list of command button control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

Radio Button Control

Radio buttons must be grouped. Unlike check boxes, when one radio button is on, the other radio buttons in the same group are off. 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 give the appearance that the picture box control does not exist.

Use one of the following methods to place a radio button inside a frame or picture box:

- Click the left mouse button on the radio button bitmap in the **Properties dialog box**. Click and drag with the left mouse button inside the frame or picture box control to create the control with the size of the rectangle displayed.
- Copy or cut the radio button(s) you want to place inside the frame or picture box to the clipboard. Select the frame or picture box and press Ctrl+V to paste the control(s) inside it.

For a complete list of radio button control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

Check Box Control

A check box is used for a true/false option. Check boxes may have the box part displayed to the left or right of the caption.

For a complete list of check box control properties, methods, and events, use the menu item ("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 allowed, or when a common set of responses are known and a different response may be typed in. Combo box retrieval is a mechanism by which the combo list box displays the previous responses entered in the text box of the combo box. The combo box has three styles ("style" property). Use the style which best suits your needs. The `PSCBO_NOEDIT` style is used when only a fixed set of responses are allowed. Combo boxes support completion with the Space Bar and question mark keys. Set the "completion" property of the combo box if there is an existing completion type which suits your needs. The `FILE_ARG` completion type is the most common. It provides completion on file names.

Example:

```
// This example illustrates combo box retrieval.
// This example requires a form named "form1", an OK button named
// "ok", and combo box named "ctlcombo1".
defeventtab form1
ctlok.lbutton_up()
{
    // When the OK button is pressed, you will want 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 4 controls: the root window, text box, picture box, and list box. The properties and methods of the sub-controls may be accessed individually with the **p_cb**, **p_cb_text_box**, **p_cb_picture**, **p_cb_list_box** instance handle properties. The **p_cb_picture** property is only available when the control is displayed.

Example:

```
defeventtab form1
ctlcombo1.on_create()
{
    // To make the loop a little more efficient, activate the list
    // box of the combo box control
    p_window_id=p_cb_list_box;
    for (i=1;i<=100;++i){
        // Add an item to the active list box.
        _lbadd_item("line="i);
    }
    // Activate the root window of the combo box.
    p_window_id=p_cb;
}
```

Example:

```
#include "slick.sh"
defeventtab form1
ctlcombo1.on_create()
{
    // Show a picture which indicates that clicking on the picture
    // box button displays a dialog box. _pic_cbdots is a global
    // variable defined in "slick.sh" which is a handle to a picture.
    p_cb_picture.p_picture=_pic_cbdots;
}
ctlcombo1.lbutton_down()
{
    // Check if the left mouse button was pressed 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);
}
```

Example:

```
// This example requires a form named "form1", command button named
// "ok", a combo box named "ctlcombo1", and another command button
// named "ctlcommand1"
#include "slick.sh"
defeventtab 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, to using the mouse to
    // select an item out of a list box.
    status=ctlcombo1._cbl_search("", "$");
    if (status) {
        _message_box("Combo box contains invalid input");
        return("");
    }
    // have valid input
```

```
}
ctlcommand1.lbutton_up()
{
    // Add some items to the combo box list
    ctlcombo1.p_cb_list_box._lbadd_item("Hello");
    ctlcombo1.p_cb_list_box._lbadd_item("Open");
    ctlcombo1.p_cb_list_box._lbadd_item("New");
    // Make the correct item in the combo box list current so combo
    // box retrieval works better. _cbi_search searches for p_text
    // in the combo list box. The "$" specifies that an exact match
    // should be found and not a prefix match.
    status=ctlcombo1._cbi_search("", "$");
    if (!status) {
        messageNwait("Found it!");
        // Select the line in the combo box so that an up or down arrow
        // selects the line above or below and not the current line.
        ctlcombo1.p_cb_list_box._lbselect_line();
    }
}
```

A combo box receives an **on_change** event with a *reason* argument under the following circumstances:

<i>reason</i>	<i>Description</i>
CHANGE_OTHER	p_text property changed, probably because of typing.
CHANGE_CLINE	p_text property changed because selected line in list box changed and the list was visible.
CHANGE_CLINE_NOTVIS	p_text property changed because a key was pressed which scrolls the list (up, down, pgup, pgdown) while the list was invisible.
CHANGE_CLINE_NOTVIS2	Same as CHANGE_CLINE_NOTVIS. Sent to user level 2 inheritance only. User level 2 inheritance will receive the CHANGE_CLINE_NOTVIS reason as well if the user level 1 inheritance does not catch the on_change event.

The **on_drop_down** event is sent to a combo box with a *reason* argument. The *reason* argument specifies one of the following conditions:

DROP_UP	After combo list box is made invisible.
DROP_DOWN	Before combo list box is made visible.
DROP_INIT	Before retrieve next/previous. Used to initialize list box before it is accessed.
DROP_UP_SELECTED	Mouse released while on valid selection in list box and list is visible

Example:

```
#include "slick.sh"
defeventtab form1
ctlcombo1.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, use the menu item ("Help", "Macro Functions by Category").

List Box Control

A list box is used to allow the user to choose from a fixed set of items. Multiple items from the list may be selected at once 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 *_lb???* functions cause an **on_change** event. Use the ***_find_longest_line()*** function to find the longest line in a list box.

Example:

```
// This example requires a form named "form1", a command button
// named "ok", and a list box named "ctl1list1".
#include "slick.sh"
defeventtab form1
ctl1list1.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
        }
    }
}
```

Example:

```
//This example 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_??? 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;
}
```

Example:

```
// This example illustrates how to resize a dialog box based on the
// longest item in a list box.
#include "slick.sh"
defeventtab form1
ctllist1.on_create()
{
    _lbadd_item("Line1");
    _lbadd_item("This is a longer line2");
    _lbadd_item("This is the longest item in the list box");
    longest=_find_longest_line();

    // Add on a little to account for the left and right borders of the
    // list box. Have to convert client width because its in pixels.
    list_width=longest+ p_width-_dx2lx(p_xyscale_mode,p_client_width);

    form_wid=p_active_form;

    // Again we have to account for the left and right borders.
    // Multiply p_x of list box by two to show equal amounts of
    // spacing on each side of the list box.
    form_width=2*p_x+ list_width+ form_wid.p_width-
        _dx2lx(form_wid.p_xyscale_mode,form_wid.p_client_width);

    p_width=list_width;
    form_wid.p_width=form_width;

    // Now make sure the whole dialog box can be seen on screen
    form_wid._show_entire_form();
}
```

Example:

```
// This example illustrates how to disable a list box and make the
```

```
// items in the list box appear grayed.
#include "slick.sh"
defeventtab form1
ctl1list1.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, use the menu item ("Help", "Macro Functions by Category").

Vscroll Bar Control

A scroll bar control is typically used to allow the user to choose an integer that has a fixed range or to display 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 which occurs when arrows are pressed, and increment/decrement which occurs when you click left button between arrow and thumb box respectively.

The **on_change** event is sent after dragging the thumb box is completed. The **p_value** property contains the new scroll position and will be in the range **p_min..p_max**.

The **on_scroll** event is sent while you click and drag the thumb box of a scroll bar.

Example:

```
#include "slick.sh"
defeventtab form1
ctlvscroll1.on_scroll()
{
    message("on_scroll p_value="p_value);
}
ctlvscroll1.on_change()
{
    message("on_change p_value="p_value);
}
```

For a complete list of scroll bar control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

Hscroll Bar Control

See **Vscroll Bar control** above.

Drive List Control

The drive list is a combo box which allows selection of different disk drives. The **Open dialog box** uses this control.

The drive list control receives an **on_change** event with a *reason* argument of CHANGE_DRIVE when the drive is changed by selecting a different drive from the combo list box.

Example:

```
#include "slick.sh"
defeventtab form1
ctlcombo1.on_change(reason)
{
    if (reason==CHANGE_DRIVE) {
        message("Item selected from list. Current drive is now
        "_drvdrive());
    }
}
```

For a complete list of drive list control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

File List Box Control

The file list box control displays a list of files. Multiple files may 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 following circumstances:

<i>reason</i>	<i>Description</i>
CHANGE_SELECTED	Occurs when items are selected or deselected because of a key press or mouse event. None of the <i>_lb???</i> functions cause an on_change event.
CHANGE_FILENAME	<i>_flfilename()</i> function was called which changed the file names listed.

Example

```
#include "slick.sh"
defeventtab form1
ctlcommand1.lbutton_up()
{
    ctllist1._flfilename("*.bat","c:\\");
}
ctllist1.on_change(reason)
{
    if (reason==CHANGE_FILENAME) {
        message("File list display directory "_flfilename());
    }
}
```

For a complete list of file list box control properties, methods, and events, use the menu item ("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 a *reason* argument under the following circumstances:

<i>reason</i>	<i>Description</i>
CHANGE_SELECTED	Occurs when items are selected or deselected because of a key press or mouse event. None of the <code>_lb???</code> functions cause an on_change event.
CHANGE_PATH	<code>_dlfilename()</code> function was called which changed the file names listed, the user double clicked the left mouse button, or the user pressed the Enter key.

Example:

```
// This example requires a form named "form1", a text box named
// "ctltext1", and a directory list box named "ctlldlist1".
#include "slick.sh"
defeventtab form1
ctlldlist1.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, use the menu item ("Help", "Macro Functions by Category").

Picture Box Control

The picture box is typically 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. However, the image control has better performance than the picture box because it does not require a system resource known as a window. You should use the image control (see "**Image Control**" on page 324) for displaying bitmaps and bitmap buttons.

For a complete list of picture box control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

Gauge Control

The gauge control is typically used to indicate the completion status of a process.

For a complete list of gauge control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

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. 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;
}
```

Image Control

The image control performs a subset of the features of the picture box control. However, it has better performance because an operating system resource known as a window is not required. The image control is best used for creating bitmap buttons for button bars or tool bars.

Adding a Bitmap Command Button or Check Box

Perform the following steps to add a bitmap button to a dialog box:

1. Create a new form for editing ("Macro", "New").
2. Create an image control. Double click on the image control (Looks like slanted rectangle) in the dialog editor.

3. Set the **p_picture** property to **bbfind.bmp**. Make sure you specify the full path (c:\vslick\bitmaps UNIX: /usr/lib/vslick/bitmaps) is the default path used by installation program).
4. Set the **p_command** property to **gui_find**. Notice that 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**.

In step 3, you enter the "**bbfind.bmp**" bitmap as an example. The "bb" prefix indicates that this is a bitmap that can be used by a toolbar. You can edit the "**bbfind.bmp**" file with paint brush (pbrush.exe). Use "**bbblank.bmp**" as a template for creating your own bitmap buttons.

Example:

```
// This example illustrates how to load your own picture like
// a toolbar button.
#include "slick.sh"
defeventtab form1
ctlimage1.on_create()
{
    index=_update_picture(-1,bitmap_path_search("bbfind.bmp"));
    if (index<0) {
        if (index==FILE_NOT_FOUND_RC) {
            _message_box("Picture bbfind.bmp was not found");
        } else {
            _message_box("Error loading picture
bbfind.bmp\n\n"get_message(index));
        }
        return("");
    }
    p_picture=index;
    p_command="gui_find";
    p_message="Searches for a string you specify";
    p_style=PSPIC_FLAT_BUTTON;
}
```

Example:

```
// This example illustrate how to give the appearance of a button
// being pushed in. While you can do this by setting styles, here
// you can see how some other functions accomplish this task.
// For this example, create a form named form1 and an image control
// named ctlimage1.
#include "slick.sh"
defeventtab form1
ctlimage1.on_create()
{
    index=_update_picture(-1,bitmap_path_search("bbfind.bmp"));
    if (index<0) {
        if (index==FILE_NOT_FOUND_RC) {
            _message_box("Picture bbfind.bmp was not found");
        } else {
            _message_box("Error loading picture
bbfind.bmp\n\n"get_message(index));
        }
        return("");
    }
}
```

```
    }
    p_picture=index;
    p_command="gui_find";
    p_message="Searches for a string you specify";
    p_style=PSPIC_BUTTON;
}
ctlimage1.lbutton_down()
{
    // Reset the button counter so we don't get
    // double and triple click events.
    get_event('B');
    mou_mode(1)
    mou_capture();
    done=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) {
                    // Show the button pushed in.
                    p_value=1;
                }
            } else {
                if (p_value) {
                    // Show the button up.
                    p_value=0;
                }
            }
            break;
        case LBUTTON_UP:
        case ESC:
            p_value=0; // Restore the button state
            done=1;
        }
        if (done) break;
        event=get_event();
    }
    mou_mode(0);
    mou_release();
    say('out');
    return("");
}
```

For a complete list of image control properties, methods, and events, use the menu item ("Help", "Macro Functions by Category").

Adding Dialog Box Retrieval

Dialog box retrieval is a mechanism which allows previous responses for check boxes, radio buttons, text boxes, and combo boxes to be retrieved. Press F7 and F8 to retrieve the previous or next response. For example, if you want to repeat your last Search and Replace again, display the **Replace dialog box** ("Search", "Replace") and press Enter. The Replace dialog box displays your last response including all check boxes. Try two search and replace

commands with this dialog box and then try pressing F7 or F8 when the Replace dialog box is displayed. Your responses to dialog boxes are saved for your next session when you exit the editor and auto restore is on ("Tools", "Configuration", "More Options...").

Example

```
// This example illustrates how to add dialog box retrieval to your own
// dialog boxes. Create a form named "form1", a text box (any name), a
// check box (any name), and a command
// button named "ok".
#include "slick.sh"
defeventtab form1
ctlok.on_create()
{
    // Retrieve the previous response to this dialog box.
    _retrieve_prev_form();
}
ctlok.lbutton_up()
{
    _save_form_response();
    p_active_form._delete_window(1);
}
```

Changing or Adding Menus

You may change or add menu items by using the Menu Editor dialog box ("Macro", "Menus...", select a menu). To create a new menu, use the Open Menu dialog box ("Macro", "Menus...") and press the New button. 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 Bind Command to Key dialog box is displayed which allows you to bind your new macro to a mouse click.

Menu Macro Programming

This section describes macro programming details about menus. You do not need to read this section unless you are attempting more advanced menu item enabling or you are writing macros which manage menus. See on-line help on [Menu Editor dialog box](#) for information on developing menus without macro programming.

Creating and Editing Menu Resources

Modified menus are stored in the state file "vslick.sta" (UNIX: "vslick.stu") The easiest way to create or change a menu is to use the [Open Menu dialog box](#) ("Macro", "Menus..."). After you have selected the menu, the [Menu Editor dialog box](#) is displayed which allows you to easily edit a menu resource. Once the menu is created, you may use the [show](#), [mou_show_menu](#), or [_menu_show](#) function to run the menu by displaying it as a pop-up. 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.

Defining a Menu Resource

Use the **insert_object** command to insert macro source code for a menu into the current buffer. You can edit the resource properties and then run the macro to apply the resource changes. You can ignore the message "No main entry point". Changing a menu resource WILL NOT change any menu bars. Menu bars represent menu resources that have been loaded. See **_menu_load** function in the on-line help for information on loading a menu resource.

A menu definition has the following format:

```
_menu menu_name {  
    submenu menu_item, help_command, help_message, categories  
        menu_item, command, categories, help_command, help_message  
    ...  
    endsubmenu  
    submenu  
        ...  
    endsubmenu  
    ...  
}
```

The items in italics have the following meaning:

<i>menu_item</i>	Menu item name in double quotes. Use '&' to choose selection character.
<i>command</i>	Any editor command. Place cursor on Visual SlickEdit command line and press '?' to list all editor commands.
<i>help_command</i>	Command to be executed when F1 is pressed. Usually it is a help or popup_imessage command.
<i>categories</i>	Specifies zero or more help categories in double quotes. Multiple help categories are separated with ' '.
<i>help_message</i>	A single line message in double quotes displayed on message line.

Sample Menu Definition:

```
_menu mymenu {  
    submenu "&File", "Help file menu", "Displays File drop-down menu", "ncw"  
        "&New", "new", "", "help new", "Creates a new file to edit"  
        "&Open\tCtrl+O", "gui_open", "", "help gui_open", "Open a file"  
    endsubmenu  
    submenu "&Edit", "Help edit menu", "Displays Edit drop-down menu", "ncw"  
        "Cu&t", "cut", "sel|nrndonly", "help cut", "Deletes the selection and copies it to the  
        clipboard"  
}
```

Predefined Attributes For Auto-Enabling Commands

For convenience, Visual SlickEdit has some predefined enable/disable attributes which you can specify for any command. You 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.

Example

```
#include "slick.sh"
_command void top_of_buffer()
    name_info(' ' VSARG2_READ_ONLY| VSARG2_REQUIRES_EDITORCTL)
{
    ...
}
```

The above command will be disabled when there is no editor control to operate on. For more examples, and a complete list of attributes see "[name_info Attributes](#)" on page 280.

Macro Callbacks for Enabling Commands

If the auto-enable attributes do not give you what you need, you can define a enable/disable callback for your command. The name of the callback function you define is based on the name of your command as shown below:

```
#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(TM), AutoSave, and some other features may be
    automatically terminated. Exit and restart the editor to restart
    this timer. Use the "say" function to debug your _OnUpdate
    callback
*/
int _OnUpdate_mycommand(CMDUI &cmdui,int target_wid,_str command)
{
    //say('h1');
    // Lets assume this command requires the target to be an
    // editor control with a selection.
    // IF the target is not an editor control
    if ( !target_wid || !target_wid._isEditorCtl() ) {
        //say('disabled at h2');
        return(MF_GRAYED);
    }
    //say('h3');
    // IF the editor control does not have a selection
    if (!target_wid.select_active2()) {
        //say('disabled at h4');
        return(MF_GRAYED);
    }
    //say('h5');
    if (gSomeOtherState) {
        //say('disabled at h6');
        return(MF_GRAYED);
    }
    //say('enabled at h7')
    return(MF_ENABLED);
}
```

```
}
_command void mycommand() name_info('VSARG2_REQUIRES_EDITORCTL)
{
    // Some code here...
}
// This command effects the enable/disable of mycommand
_command void mycommand2(_str argument="0")
{
    gSomeOtherState=(argument)?1:0;
    // Indicate that the enable state of the toolbar buttons must be
    // updated. The _tbSetRefreshBy function is very fast.
    // Toolbars will be updated after the macro terminates and
    // the user stops typing fast.
    _tbSetRefreshBy(VSTBREFRESHBY_USER);
}
```

Putting a Menu Bar on a Form

The following sample code illustrates how to place a menu on a form as a menu bar:

```
#include "slick.sh"
// Create a form called form1 and set the border style to anything BUT
// BDS_DIALOG BOX. Windows does not allow forms with a dialog
// box style border to have menu bars.
defeventtab form1
form1.on_load()
{
    // Find index of Visual SlickEdit 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 which are not allowed when there no
    child windows.
    _menu_set_state(p_menu_handle,"!ncw",MF_GRAYED,"C");
}
```

Displaying a Menu as a Pop-up

If the **show** or **mou_show_menu** function suits your needs, you should use one of them. The following sample code illustrates how to display a menu as a pop-up:

```
#include "slick.sh"
defmain()
{
    // Low-level code to display Visual SlickEdit 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 Dialog Boxes

The following are several important dialog box forms and functions you may want to use in your own macros. See on-line help for examples and detailed information. Use the menu item ("Help", "Search...") to get detailed on-line help for these items.

General purpose forms and dialog box functions:

<code>_textbox_form</code>	General purpose form which displays a variable number of text boxes or combo boxes.
<code>_sellist_form</code>	General purpose form which displays a list box, an optional combo box, and a variable number of command buttons.
<code>_open_form</code>	General purpose form used to open and save files which does not have the advanced controls.
<code>_edit_form</code>	General purpose form used to open and save files which has the advanced controls used for the ("File", "Open...") dialog box of Visual SlickEdit.
<code>_font_form</code>	General purpose form used to prompt for a font.
<code>_choose_font</code>	non-UNIX platforms only. General purpose dialog box built-in to operating system used to prompt for a font.
<code>_printer_setup</code>	non-UNIX platforms only. General purpose dialog box built-in to operating system used for printer setup.

There are a lot of commands which display dialog boxes. To find out what command a particular menu item invokes, place your cursor on the menu item and press F1 to get help on the command. If a key displays a dialog box, you can find out the command the key executes by using the **Bind Command to Key dialog box** ("Tools", "Configuration", "Key Bindings...").

String Functions Overview

The following are commonly used string functions. See on-line help on **String Functions** for a complete list.

<code>_str_center</code> (<i>_str string</i> , <i>int width</i> [<i>,_str pad_ch</i>])	Returns <i>string</i> padded evenly on left and right with spaces or a character you choose with the optional argument <i>pad_ch</i> .
<code>_str_dec2hex</code> (<i>long number</i> [<i>,int base</i>])	Returns <i>number</i> converted to <i>base</i> specified.
<code>_str_expand_tabs</code> (<i>_str string</i> [<i>,int start</i> [<i>,int count</i> [<i>,_str option</i>]]])	

	Very similar to substr function except that this function supports tab characters very well.
<code>_str field(_str string,int width)</code>	Returns <i>string</i> padded with trailing spaces to <i>width</i> characters
<code>long hex2dec(_str number [,int base])</code>	Returns <i>number</i> converted to <i>base</i> specified.
<code>_str indent_string(int width)</code>	If indent with tabs is on, a string of tabs of length <i>width</i> is returned. Otherwise, a string of spaces of length <i>width</i> is returned.
<code>boolean isalnum(_str ch)</code>	Returns non-zero value if <i>ch</i> is a numeric or alphabetic character.
<code>boolean isalpha(_str ch)</code>	Returns non-zero value if <i>ch</i> is an alphabetic character.
<code>boolean isdigit(_str ch)</code>	Returns non-zero value if <i>ch</i> is a numeric character.
<code>boolean isinteger(_str string)</code>	Returns non-zero value if <i>string</i> is a valid int . If <i>string</i> is floating point number, 0 is returned.
<code>boolean isnumber(_str string)</code>	Returns non-zero value if <i>string</i> is a valid double (floating pointer number).
<code>_str last_char(_str string)</code>	Returns last character of <i>string</i> . If <i>string</i> is null, the space character is returned.
<code>int lastpos(_str needle [, _str haystack [,int start [, _str options]]])</code>	Returns the position (1..length(<i>haystack</i>)) of the last occurrence of <i>needle</i> in <i>haystack</i> . If <i>needle</i> is not found, 0 is returned. Regular expressions are supported.
<code>int length(_str string)</code>	Returns the number of characters in <i>string</i> .
<code>_str lowercase(_str string)</code>	Returns <i>string</i> converted to lowercase.
<code>_str number2onoff(_str number)</code>	Returns "off" if <i>number</i> ==0. Otherwise "on" is returned.
<code>_str number2yesno(_str number)</code>	Returns "N" if <i>number</i> ==0. Otherwise "Y" is returned.
<code>parse expr with template</code>	Breaks apart the expression <i>expr</i> given into variables that appear in <i>template</i> , and much more.
<code>boolean parseoption(_str &cmdline, _str option_ch)</code>	Strips "+" or "-" option from <i>cmdline</i> . Returns non-zero number if <i>option_ch</i> was found.
<code>int pos(_str needle [, _str haystack [,int start [, _str options]]])</code>	Returns the position (1..length(<i>haystack</i>)) of the first occurrence of <i>needle</i> in <i>haystack</i> . If <i>needle</i> is not found, 0 is returned. Regular expressions are supported.
<code>boolean setonoff(_str &name,_str value)</code>	Sets <i>name</i> to 1 or 0 corresponding to <i>value</i> ="on" or <i>value</i> ="off". Returns 0 if input value is valid. Displays message if <i>value</i> is not "on" or "off".
<code>boolean setyesno(int &name,_str value)</code>	Sets <i>name</i> to 1 or 0 corresponding to <i>value</i> ="Y","Yes" or <i>value</i> ="N","No". Returns 0 if input value is valid. Displays message if <i>value</i> is not "Y" or "Yes", "N" or "No".
<code>_str strtranslate(_str string, _str replace_string, _str search_string, _str search_options)</code>	

	Returns <i>string</i> with all occurrences of <i>search_string</i> replaced with <i>replace_string</i> .
<code>_str stricq(_str string1 ,_str string2)</code>	Returns true if <i>string1</i> matches <i>string2</i> when case is ignored.
<code>_str strip(_str string ,_str ltb [, _str strip_char])</code>	Returns <i>string</i> stripped of leading and/or trailing <i>strip_char</i> .
<code>_str strip_filename(_str filename, 'P' 'D' 'E' 'N')</code>	Returns <i>filename</i> with part stripped. P=Path, D=Drive, E=Extension, N=Name.
<code>_str strip_last_word(_str &line)</code>	Returns the last space delimited word in <i>line</i> . The last word and trailing spaces are deleted from <i>line</i> .
<code>_str strip_options(_str cmdline ,_str &options)</code>	Returns <i>cmdline</i> without words that start with the characters "-", "+", or "[". <i>options</i> variable is set to stripped option words.
<code>_str substr(_str string ,int start [,int len [, _str pad]])</code>	Returns <i>length</i> characters of <i>string</i> beginning at <i>start</i> . By default, <i>length</i> defaults to rest of <i>string</i> . If <i>length</i> is greater than length of <i>string</i> , the return string is padded with blanks or <i>pad</i> character if specified.
<code>_str translate(_str string [, _str output_table [, _str input_table [, _str pad]])</code>	Returns <i>string</i> with characters translated according to arguments.
<code>_str upcase(_str string)</code>	Returns <i>string</i> converted to uppercase.
<code>int verify(_str string, _str reference [, "M" ""] [,int start])</code>	Returns the position (1..length(<i>string</i>)) of first character not matching or matching a character in <i>reference</i> . 0 is returned on failure.
<code>_str word(_str string,int Nth)</code>	Returns the <i>Nth</i> space or tab delimited word in <i>string</i> . "" is returned if the <i>Nth</i> word does not exist.

Use the menu item ("Help", "Search...") to get detailed on-line help and examples for these functions. For a list of string related functions, use the help item ("Help", "Macro Functions by Category").

Search Functions Overview

Visual SlickEdit has two levels of search functions. The high level functions provide a few extra features (user interfacing and multiple file searching) not provided by the lower level built-in functions. For best speed, use the built-in functions. The built-in functions can be used without affecting the high level search commands such as the **find_next** command. In addition, the built-in functions are not affected by the global editor search options. For a list of search related functions, use the help item ("Help", "Macro Functions by Category"). For a complete list of search related functions, see on-line help on **Search Functions**.

Use the menu item ("Help", "Search...") to get detailed on-line help and examples for these functions.

High Level Search Functions:

gui_find	Displays Find dialog box and performs search using the find or _mffind functions.
gui_replace	Displays Replace dialog box and performs search using gui_replace2 or _mfreplace functions.
gui_replace2	Performs a search and replace based on arguments given. This function is very similar to the replace function, except that this function uses a dialog box to prompt the user where to replace.
find_next	Searches for next occurrence of search string used by any of these high level search functions. This function is not affected by previous searches done with low-level built-in functions.
find	Performs search based on arguments given.
replace	Performs a replace based on arguments given. The user is prompted where to replace via the message line.
_mffind	Performs a multiple file and/or buffer search based on the arguments given.
_mfreplace	Performs a multiple file and/or buffer search based on the arguments given.

Lower Level Built-in Functions:

search	Performs search, or search and replace, based on arguments given. Does not support wrapping to top/bottom of file. When performing a replace, the user is not prompted at all.
repeat_search	Searches for the next occurrence of search string used by last call to the search built-in.

Example:

```
// This example searches for lines which contain a particular search
// string and places the lines in another window and buffer.
defmain()
{
    orig_wid=p_window_id;
    // The +w option forces a new window to be created. The +t options
    // force a new buffer to be created.
    status=edit("+w +t");
    if (status) {
        _message_box("Unable to create temp window and buffer\n\n":+
            get_message(status));
    }
    _delete_line(); // Delete the blank line
    output_wid=p_window_id;

    p_window_id=orig_wid;
    top(); // Place the cursor at the top in column 1.
    status=search("if","w@");// Case insensitive word search for "if"
                                // "@" specifies no string not found
    message.
    for (;;) {
        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;
}

```

Example:

```

// This 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("");
    }

    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);
        activate_view(orig_view_id);
        status=repeat_search();
    }

    // Display the buffer in a list box.
    // The _sellist_form dialog box will delete the temp view and buffer
    // The original view must be activated before showing the
_sellist_form or
    // the dialog box will operate strangely
    activate_view(orig_view_id);
    result=show("_sellist_form -mdi -modal",
        "Sample Selection List",
        SL_VIEWID|SL_SELECTCLINE, // Indicate next argument is
view_id
        temp_view_id,
        "OK",
        "", // Help item

```

```
        "", // Use default font
        "", // Call back function
    );
    if (result) {
        message("Selection list cancelled");
    } else {
        message("Item selected is "result);
    }
}
```

Buffer Input/Output Functions

The following are commonly used Input/Output Functions which operate on buffers and do not require a selection:

<code>void _insert_text(_str string[, int Binary, _str NLChars])</code>	Inserts <i>string</i> at cursor position.
<code>void _delete_text(int len [, "C"])</code>	Delete <i>len</i> bytes starting from the cursor position.
<code>_str _expand_tabs(int start, int count [, "E" "S"])</code>	Returns data from current line starting at column (not byte position) <i>start</i> . <i>count</i> columns of text are returned.
<code>void get_line(_str &line)</code>	Retrieves current line.
<code>void replace_line(_str line)</code>	Replaces current line.
<code>void insert_line(_str line)</code>	Inserts line after current line.
<code>int delete_line()</code>	Deletes the current line.
<code>_str get_text([int count [,int seek_pos])</code>	Gets a stream of text starting at cursor.
<code>_str cur_word(int &start_col [, _str from_cursor, _str end_prev_word])</code>	Gets the current word.
<code>int _buf_transfer(int src_buf_id [,start_linenum [,end_linenum [,start_col [,end_col]]]])</code>	Transfers data from <i>src_buf_id</i> to cursor.
<code>void keyin(string)</code>	Inserts <i>string</i> of characters as if typed from the keyboard. This function is SLOW because it performs word wrap.

Use the menu item ("Help", "Search...") to get detailed on-line help and examples for these functions.

Using selection functions is also an excellent way to transfer buffer data. For more information, see "[Selection Functions Overview](#)" on page 337.

Selection Functions Overview

Visual SlickEdit supports multiple selections. However, only one selection may be active or visible. Selections are specified by handles. The **_alloc_selection** built-in returns a handle to a selection. The function **_free_selection** frees a selection associated with the selection handle given. Most selection functions accept a selection handle. A handle of " specifies the active selection or selection showing, which is always available. The **_free_selection** function cannot free the active selection. To free the active selection, you must use the **_show_selection** function first to make another selection the active selection. The actual handle number of the active selection is returned by the expression **_duplicate_selection("")**. For a list of selection related functions, use the help item ("Help", "Macro Functions by Category").

Example:

```
// Duplicate the current line.
mark_id=_alloc_selection();
//
if (mark_id<0) {
    message(get_message(mark_id));
    return(rc);
}
_select_line(mark_id);
_copy_to_cursor(mark_id);
// This selection can be freed because it is not the active selection.
_free_selection(mark_id);

// This code copies selected text and keeps the resulting selection on
// the source text instead of the destination text.
if (_select_type()==""){
    message(get_message(TEXT_NOT_SELECTED_RC));
    return(1);
}
mark_id=_duplicate_selection() // Make a copy of the active selection.
_copy_to_cursor();
old_active_mark_id= _duplicate_selection("") // Save the active
selection id.
// 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);
```

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.

Example:

```
// This batch program converts the marked text into hexadecimal ASCII
// codes. Each hexadecimal ASCII code is separated by a comma. One
// possible use of this function could be to convert a binary font
// file into hexadecimal ASCII codes to be compiled into a C program.
#include "slick.sh"
defmain()
```

```
{
  if (_select_type()=="") {
    message(get_message(TEXT_NOT_SELECTED_RC));
    return(TEXT_NOT_SELECTED_RC);
  }
  // Under scores 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);
}
```

Unicode and SBCS/DBCS Macro Programming

If the Visual SlickEdit you are running does not support Unicode, you do not need to read this section. The UNIX version does not yet support Unicode and some Windows OEM's don't start Visual SlickEdit in UTF-8 mode (Unicode support mode).

When Visual SlickEdit 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. In order 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 which use or set column positions often don't work correctly for both buffer data formats. The solution is to call "raw" functions.

Example:

```
// This won't work if the current buffer is an SBCS/DBCS buffer,
// word is a UTF-8 string (which this example assumes), and word
// contains characters above 127.
p_col=p_col+length(word);
// This will work
p_col=p_col+_rawLength(word);
// This works too
word=_rawText(word);
p_col=p_col+length(word);
```

Example:

```
// This won't 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 will tell you whether Visual SlickEdit is running 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. You only have to worry about this if you enter characters above 127.

All macro functions and properties accept and return UTF-8 except the following:

The following Slick-C functions DO NOT accept or return UTF-8 data:

`_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()`

For performance reasons all seeking is done on raw data. Since our tag database stores seek positions, buffers need to be loaded in the same raw format so that seeks work.

All `_rawXXX()` or `XXX_raw()` functions

(Unlike the C API) The Slick-C functions `get_text()`, and `_expand_tabsc()` return UTF-8 data.

The following Slick-C properties DO NOT accept or return UTF-8 data:

`p_display_xlat`

The following are Slick-C new raw functions:

```
_expand_tabsc_raw()  
get_line_raw()  
get_text_raw()  
insert_line_raw()  
_insert_text_raw()  
replace_line_raw()  
_rawLength()  
_rawSubstr()  
_rawText()
```

The following are Slick-C new that optional support raw data:

pos() – When **p_rawpos** appended to *options* argument.
lastpos() – When **p_rawpos** appended to *options* argument.
uppercase() – When **p_UTF8** property given as second argument.
lowercase() – When **p_UTF8** property given as second argument.
parse - When **p_rawpos** appended to *options* of search argument.

The following are Slick-C new UTF-8 functions:

```
_MultiByteToUTF8()  
_UTF8()  
_UTF8Asc()  
_UTF8Chr()  
_UTF8ToMultiByte()
```

The following C API functions DO NOT accept or return UTF-8 data:

```
vsGetText(),    vsGetRText(),    vsExpandTabsC(),    vsQSelectedTextLength(),  
vsGetSelectedText()  
These function 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

For performance reasons all seeking is done on raw data. Since our tag database stores seek positions, buffers need to be loaded in the same raw format so that seeks works.

All vsXXXXRaw() functions

DLL Interface

Visual SlickEdit has a DLL interface for Windows. We recommend using the Slick-C macro language instead of the DLL interface except when you need to interface to another program's DLL, when you need better speed, or when the Slick-C macro language is missing a function you need. Once a DLL function has been added, it can be called from a Slick-C macro just like any other Slick-C function. In fact, DLL functions can be used for timer call backs and anywhere a Slick-C function is used. To get started using the DLL interface, edit the file "**simple.c**" in the <vslick>\samples\simple directory and read its comments for more information. Currently, all VSAPI functions have the prefix "vs". Use the ("Help", "Search") menu item and type "vs" for a list of VSAPI functions.

Shelling Programs from a Slick-C Macro

To execute another program from a Slick-C macro you can use the **shell** built-in, the **dos** command, or the **execute** built-in. The latter method is similar to executing a command on Visual SlickEdit's command line, and allows you to create expressions which executes 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 described in the on-line help.

Example:

```
// Capture the output of Slick GREP and process the error messages.
dos("-e grep DEBUG *.c");
// Redirect the output of grep to a file.
shell("grep 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");
```


Hooking Exit and Other Events

Invoking a Macro When Visual SlickEdit is Invoked

To invoke any macro command defined by **_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 execute without changing any invocation options is to create a module with a **definit** entry point. See **"Module Initializations"** on page 292 for more information.

Invoking a Macro Before Visual SlickEdit Terminates

If you want a function to be invoked when the editor exits, create a macro procedure with a name that has the prefix **"_exit_"**. For example, the following procedure will automatically get invoked when Visual SlickEdit exits:

```
_exit_cleanup_stuff()
{
    messageNwait("Got here");
}
```

Command Line Interfaces

This section describes how to write macros with command line interfaces.

Command Line Arguments

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. Alternatively, the command can declare a named argument whose value is the same as **arg(1)**. For example, invoking the edit command **"e file1 file2"** calls the **e** command with **"file1 file2"** in **arg(1)**. The **parse** built-in is an excellent function for parsing a command line string (See on-line help for more information on parse). When another macro calls a command, more than one argument string may be passed. Calling the **arg** function with no parameters returns the number of parameters the command or procedure was called with.

Example:

```
#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);
}
// This command supports completion on a filename followed by an
// environment variable argument
```

```
_command test1(_str cmdline="") name_info(FILE_ARG "ENV_ARG")
{
    parse cmdline with file_name env_name;
    message("file_name="file_name" env_name="env_name");
}
```

The string constant expression given to the **name_info** keyword is used for argument completion, restricting when the command can be executed, and a few other options. See ["name_info Attributes"](#) on page 280 for more information.

get_string Procedure

The **get_string** procedure reads a single argument from the user. See **get_string** procedure in the on-line help for a complete description.

Example:

```
#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

The **prompt** procedure is an easy and efficient way to write a command which 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, execution will not continue. See **prompt** procedure in the on-line help for a complete description.

Example:

```
// This command supports completion on an environment variable argument
#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);
}
```

State File Caching

By default, Visual SlickEdit does not load a module, event, dialog box template, or picture from "vslick.sta"(UNIX: "vslick.stu") until it is referenced. Using the **definit** primitive (see "**Module Initializations**" on page 292) forces a module to be loaded when the editor is invoked. The default state file cache is about 200k. You may set this size with the "-st" invocation option or with the **General Configuration dialog box** ("Tools", "Configuration", "General...", select Virtual Memory Tab). 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 may have critical modules which you want permanently stored in memory. Place the **no_code_swapping** key word somewhere in your module (usually the top) to force a module to be loaded and permanently stored in memory when Visual SlickEdit is invoked; then, should a critical disk failure occur reading the state file, Visual SlickEdit will not be crippled. A few of Visual SlickEdit's modules which provide basic editing capabilities remain permanently in memory.

Error Handling and the RC Variable

The **rc** variable is a predefined global (meaning accessible from all loaded modules) variable. In the future, the Slick-C language will not need to use the **rc** global variable for error handling. Most functions which set the **rc** variable, also return the error code so you don't have to use **rc**. The following functions require you to use the **rc** variable for error handling: **buf_match**, and **get_env**.

By convention, functions which use integer error codes return negative error codes which 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 Visual SlickEdit message line. Use the **clear_message** function to clear the message.

Example:

```
// Cause a message
_deselect();_copy_to_cursor();
// Clear the message
clear_message();
```

Typical Mistakes

Common programming mistakes occur when writing Slick-C language programs. One mistake is to use the **rc** variable in an IF statement when the **rc** variable has changed. The following example illustrates the mistake:

```
// Don't write code like this.
search("xyz"); // rc is set to 0 if "xyz" is found.
mark=_alloc_selection();
if (rc) { // Was "xyz" found?
    ...
}
```

The above sequence of statements does NOT work because the **_alloc_selection** function sets the **rc** variable. If you execute a statement before the IF statement, save the value of **rc** in a temporary variable, as shown below:

```
// Write code like this.
// rc is set to 0 if "xyz" is found and status is assigned rc.
status=search("xyz");
mark=_alloc_selection();
if (status ) {    // Was "xyz" found?
...

```

Another common programming mistake is to rely on **rc** for all error conditions. Most functions do not set **rc**. It is better to assign the return value to a variable, as shown below:

```
// Don't write code like this.
search("xyz");    // RC is set to 0 if "xyz" is found.
if (rc) {    // Was "xyz" found?
...
// Write above code like this.
// rc is set to 0 if "xyz" is found and status is assigned rc.
status=search("xyz");
if (status ) {    // Was "xyz" found?
...

```

Interpreter Error Handling

The two primitives **_suspend** and **_resume** may be used to process errors which would normally halt the interpreter.

The **_suspend** primitive pushes a new level of error handling. After a **_suspend** has been executed, errors which halt the interpreter will set the variable **rc** to the error code and resume execution at the first statement after **_suspend**. The **_resume** function restores the interpreter stack and code position to the state it was in after the last **_suspend** was executed. Be sure to execute **_resume** to restore the suspend/resume stack.

Example:

```
defmain()
{
    status=_suspend();
    if (status) {
        if (status<0) {
            _message_box(get_message(status));
            return(status);
        }
        return(0);
    }
    i=arg(1)+1;    /* Handle invalid number argument */
    message("i="i);
    rc=1;    // rc is the return value of the _suspend function
    resume();
}
```

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 length one or more. See "[String Literals](#)" on page 257 for complete information on defining strings. An event name string of 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 '_' character is used instead of the '-' character. The acceptable event constants are listed below:

ASCII characters

\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 keys "C-A", "C-B".."C-Z", "C-[","C-\","C-]", "C-^", and "C_". The ASCII keys \129..\255 are the same key binding as \128.

Function keys

"F1", "F2"... "F12"

Extended keys

"ENTER", "TAB", "ESC", "BACKSPACE", "PAD_STAR", "PAD_PLUS", "PAD_MINUS", "HOME", "END", "LEFT", "RIGHT", "UP", "DOWN", "PAD_5", "PGUP", "PGDN", "DEL", "INS", "PAD_SLASH"

Mouse events

"LBUTTON-DOWN", "LBUTTON-DOUBLE-CLICK", "LBUTTON-TRIPLE-CLICK", "RBUTTON-DOWN", "RBUTTON-DOUBLE-CLICK", "RBUTTON-TRIPLE-CLICK", "MBUTTON-DOWN", "MBUTTON-DOUBLE-CLICK", "MBUTTON-TRIPLE-CLICK", "LBUTTON-UP", "RBUTTON-UP", "MBUTTON-UP"

ON events

"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-SCROLL-LOCK", "ON-SPIN-DOWN", "ON-SPIN-UP", "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"

Ctrl/Shift/Alt combinations

"A-<Extended key>", "C-<Extended key>", "S-<Extended key>", "S-C-<Extended key>"

For example, "A-ENTER", or "A-TAB" or "S-C-END"

"A-<Function key>", "C-<Function key>", "S-<Function key>"

For example, "A-F1".

"A-<Mouse event>", "C-<Mouse event>", "S-<Mouse event>"

For example, "A-LBUTTON-DOWN"

"C-S-<Extended key>", "C-<Extended key>", "S-<Extended key>"

For example, "A-ENTER" or "A-TAB"

"A-<ASCII character 0..128>"

Note that "A-a" is a different key than "A-A" which requires Alt and Shift keys to be pressed. Also note that "A-\1" is equivalent to "A-C-A".

"C-A", "C-B"... "C-Z", "C-[", "C-\", "C-]", "C=^", "C-_", "C-,", "C-."

The keys \1..\29 are also represented by these keys.

"C-S-A", "C-S-B"... "C-S-Z", "C-S-1", "C-S-2"... "C-S-0"

"C-A-A".. "C-A-Z"

Note that "A-\1" is equivalent to "A-C-A".

Miscellaneous keys

"C-A-ENTER", "C-A-TAB", "C-A-ESC", "C-A-BACKSPACE"

"C-PRTSC", "C-CTRL", "A-ALT"

Miscellaneous events

"ON-SELECT"

Here are examples of uses for key names in the Slick-C language:

```
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") ) {
            ...
        }
    }
}

```

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 which contain variable size data. However, even other types are some times stored else where. In the future, after a few changes are made, the way Slick-C lays out memory will be documented.
- There is not a **sizeof** function which tells you the size of a structure in bytes.

Arrays

- Space for array elements is allocated when you index into the array.
- You can't use pointer variables to traverse array elements.
- You can't 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 not a **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.

Example

```

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 which is similar to the array operator `[]` except that you index hash tables with a string type. See "[Hash Tables](#)" for more information.

Assignment Statement

Assignment statements in Slick-C are not as shallow as C++. Array, hash table, and structure types are recursively traversed. Pointers are not traversed.

Example:

```
struct {
    int a[];
} s1, s2;
s1.a[0]=1;
s2=s1; // Copy structure and all elements of array.
```

Comparison Operator

The `==` and `!=` operators support comparing container types, arrays, hash tables and structures. Complex types are traversed recursively like the assignment statement. Strings within an array, hash table, or struct must match exactly (spaces matter).

Preprocessing

Preprocessing expressions may use string and floating point expressions.

switch Statement

The switch statement supports string expressions in addition to integer expressions.

Labeled Loops

break and **continue** statements take an optional label parameter so you can break a specific loop (like Java).

Example

```
outerlabel:
    for (;;) {
        for (;;) {
            if (...) break outerlabel;
            if (...) continue outerlabel;
        }
    }
```

Variable Argument Functions

An **arg** function allows you to define functions which accept a variable number of arguments. The **arg** function may be used on the left side of an assignment statement.

```
defmain()
{
    p("Param1", 2, x);
}
void p()
{
    messageNwait("Called with "arg() arguments");
    for (i=1; i<=arg(); ++i) {
        messageNwait("arg("i")="arg(i));
    }
}
```



```
//All undeclared variable parameters are passed by reference.  
//so when a variable is passed, we can change the contents of  
//the callers variable.  
arg(3)="New value for x";  
}
```

Built-in Graphics Primitives

You can define dialog box resources and menu resources. There are primitives for defining event handlers for dialog boxes and declaring control types. This allows the Slick-C linker to detect a reference to a control which does not exist on a dialog box before you execute your code. See "[An Event Driven Dialog Box](#)" on page 253 for a simple example.

Clipboard Inheritance (R)

Clipboard Inheritance® (US patent No. 5,710,926) provides inheritance specifically for dialog boxes. This feature allows you to copy parts of existing dialog boxes to the clipboard, paste them elsewhere, and the original code still runs. New code can be attached to the new controls without affecting the original controls. New code can be added to the original controls 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\(R\)](#)" on page 306.

End of Statement. Semicolon

Currently Slick-C assumes 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. In addition, expressions in parentheses may extend across line boundaries. In the future, Slick-C will enforce strict semicolons to conform with C++.

Type Checking

For the most part, 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.

Things Not Supported Yet

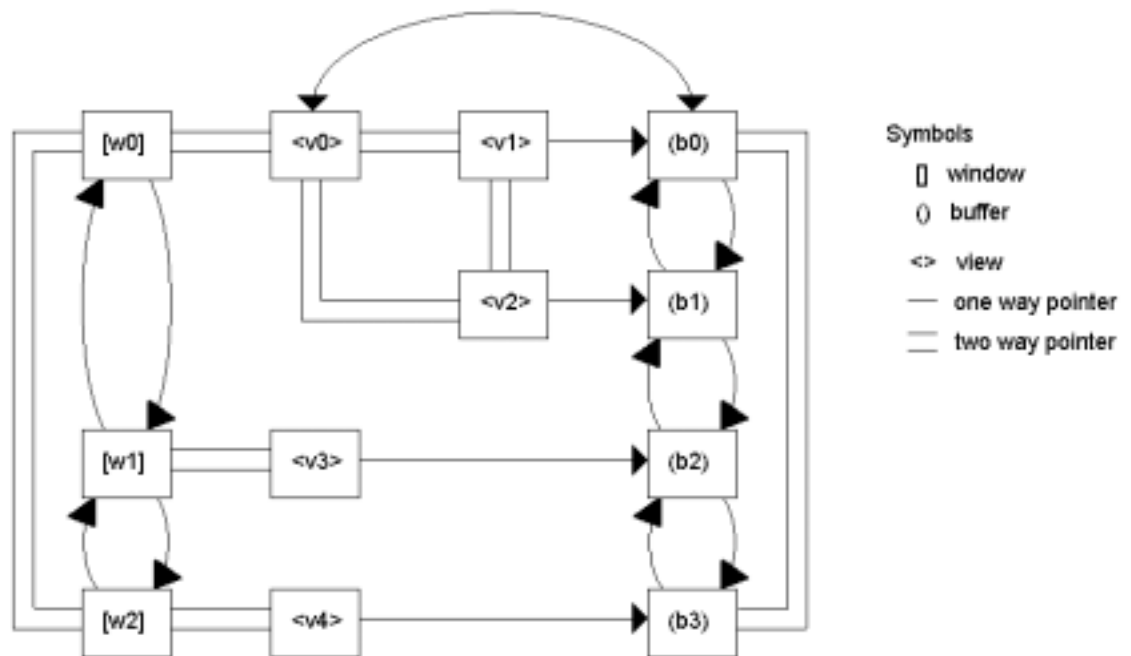
- You can't define your own classes, methods, or inheritance. The important **class**, **public**, **private**, and **new** keywords are not supported yet. Like Java, we have decided that classes in Slick-C will not require a **delete** to free objects.
- Only one syntax is currently supported for making a call with a pointer to a function variable. The `pfn(p1,p2,...)` syntax is not yet supported. This limitation is necessary for container variables because the compiler does not know the type of the variable. However, support will be added for the other syntax for typed pointers to functions.
- **char** and **short** types are not yet available.
- Template classes are not yet supported. Container variables are sometimes a more powerful mechanism for accomplishing much of what is done with template classes. However, users will likely want the speed and additional type checking of template classes and many programmers are already used to template classes.
- Function overloading is not yet supported.

- **enum** is not yet supported.
- Currently Slick-C only supports the less ambiguous C style type casting. Support for the new C++ type casting syntax *TypeName(expression)* may be added in the future. The argument is between having an elegant less ambiguous grammar or conforming to C++ so there is nothing new to learn. If you have an opinion on this topic, let us know.
- Because Slick-C does not allow low level manipulation of memory, you can't 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 filenames or regular expressions which require the use of backslashes. This syntax may be changed in the future so that single quotes may be used for character constants.
- **goto** is not supported (It may never be supported since Slick-C supports labeled loops).

Windows Structure

This section describes the internal data structure the editor uses to keep track of windows, buffer positions (views), and buffer data. You need not read this unless you want to write a Slick-C macro which manipulates windows, views, and buffers.

The picture below represents invoking Visual SlickEdit 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". To reduce the size of the drawing we have not given the buffers names. Visual SlickEdit has one hidden window which cannot be displayed. If you to attempt to leave the hidden window active, another window will be 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.

Visual SlickEdit maintains a ring of buffers and a ring of windows, 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 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. **_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. You cannot delete the hidden window.

A view holds the information necessary for the editor to remember your 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 don't use an option which overrides this) the active view information is saved in the active buffer's non-active view, and the new buffer's non-active view information is copied into the active view.

Below is a description of the contents of each structure.

Some window properties are:

p_window_x, p_window_y	Top left coordinates of window.
p_window_height, p_window_width	Height and width of window.
p_tile_id	Indicates which windows are part of a tile window group and whether a window is zoomed. Windows of a tiled window group have the same tile_id. A zoomed window has a negative tile_id.
p_x	The top left x position of window.
p_y	The top left y position of window.
p_height	Window height.
p_width	Window width.
p_view_id	Pointer to active view.
p_next	Window id of next window. (_next_window)
p_prev	Window id of previous window. (_prev_window)
p_child	Window id of child window.

Some view properties are:

block,line within block	Accessible via point and goto_point .
p_line	Line number of current line.
p_col	Column position within current line. 1..2 billion
p_left_edge	Column scroll position.
p_cursor_x	Text cursor x position.
p_cursor_y	Text cursor y position.
p_window_id	Window id.
p_buf_id	Buffer id.

Some buffer properties are:

p_buf_name	Name of buffer.
p_buf_flags	Indicates whether a buffer is hidden and may specify other buffer options. See on-line help on "internal-variables".
p_Noflines	Number of lines in file.
p_modify	Non zero indicates buffer has been modified.

p_margins	String containing left, right, and new paragraph margins (1..2 billion).
p_tabs	String containing up to 2 billion tab stops.
p_mode_name	Name of current mode.

Converting SlickEdit 2.3/2.4 Macros and Configuration

Perform the following steps to convert your SlickEdit configuration changes to Visual SlickEdit:

1. Install Visual SlickEdit into a different directory than SlickEdit as described below.
2. Run the **list_source** command from the Configuration menu in SlickEdit to list the macro source for changes you have made. This will create a file called "userdefs.e" (UNIX: "unixdefs.e").
3. **Windows:** Copy the "userdefs.e" file to your local configuration directory (determined by the VSLICKCONFIG environment variable if you have one) or to the macros directory (defaults to "c:\vslick\macros") and change its name to "vusrdefs.e".
UNIX: Copy the "unixdefs.e" file to your local configuration directory "\$HOME/.vslick" and change its name to "vunxdefs.e".
4. Run Visual SlickEdit and type **cleanup** on the command line. All options that are still present in SlickEdit will be converted.
5. Save the resulting file created by the "cleanup" command and type **vusrdefs** on the command line to apply the changes to Visual SlickEdit.

To convert a macro source file you have written, perform the following steps:

1. Install Visual SlickEdit into a different directory than SlickEdit as described below if you have not already done so.
2. **Windows:** Copy the macro source files you want to convert to your local configuration directory (determined by the VSLICKCONFIG environment variable if you have one) or to the macros directory (defaults to "c:\vslick\macros").
UNIX: Copy the macro source files you want to convert to your local configuration directory "\$HOME/.vslick".
3. Run Visual SlickEdit.
4. Edit the macro source file in the Visual SlickEdit directory you want to convert.

5. Type **tconvert** on the command line.

The converter can convert about 95% of your macro source. Typical conversion problems are documented in the text file "convert.doc" which is located in the configuration directory (defaults to "c:\vslick" or "/usr/lib/vslick" for UNIX).

Frequently Asked Questions

This section covers common questions and answers about Visual SlickEdit:

Q: How do I transfer my configuration?

A: Each user's configuration is automatically transferred to a newly installed version of Visual SlickEdit when they run the new version. Some configuration changes will not be transferred such as changes to the menus, and dialog boxes. Changes to the toolbars are preserved only if you are upgrading from 3.0b or after.

Windows:

For single-user configurations you need to install the new version of Visual SlickEdit on top of the previous version in order for your configuration to be transferred to the new version.

Single-User Windows Platforms:

To manually transfer your configuration changes from the previous version of Visual SlickEdit, perform these steps:

1. If you are currently running the new version of Visual SlickEdit, close the editor now.
2. If you have made color coding changes, copy your "vslick\user.vlx" file into the "vslick" directory of the new version.
3. If you have recorded macros, copy your "vslick\macros\vusrmacros.e" into the "vslick\macros" directory of the new version.
4. To transfer other configuration changes, bring up the previous version of Visual SlickEdit, activate the Visual SlickEdit command line at the bottom of the editor (ESC for most emulations), type **list-source** and press ENTER. This will create the macro file "vusrdefs.e", and possibly another macro file called "vusrobjs.e" in your "vslick\macros" directory. Copy "vusrdefs.e" and (optionally) "vusrobjs.e", into the "vslick\macros" directory of the new version.
5. Run the new version of Visual SlickEdit.
6. If you ran the "list-source" command, type **vusrmods** and press ENTER on the Visual SlickEdit command line. This will switch emulations if necessary, and run "vusrdefs.e" and "vusrobjs.e" (if present).
7. If you copied your recorded macros file ("vslick\macros\vusrmacros.e"), load this macro now with the Load Module dialog box ("Macro...", "Load Module...").

Multi-User Windows Platforms:

To manually transfer your configuration changes from the previous version of Visual SlickEdit, perform these steps:

1. If you have made color coding changes, copy your "user.vlx" file (located in the directory specified by your VSLICKCONFIG environment variable) into your VSLICKCONFIG directory of the new version.
2. If you have recorded macros, copy your "vusrmacs.e" file located in the directory specified by your VSLICKCONFIG environment variable into the VSLICKCONFIG directory of the new version.
3. To transfer other configuration changes, bring up the previous version of Visual SlickEdit, activate the Visual SlickEdit command line at the bottom of the editor (ESC for most emulations), type **list-source** and press ENTER. This will create the macro file "vusrdefs.e", and possibly another macro file called "vusrobs.e" in the directory specified by your VSLICKCONFIG environment variable. Copy "vusrdefs.e" and (optionally) "vusrobs.e", into the VSLICKCONFIG directory of the new version.
4. Run the new version of Visual SlickEdit.
5. If you ran the "list-source" command, type **vusrmods** and press ENTER on the Visual SlickEdit command line. This will switch emulations if necessary, and run "vusrdefs.e" and "vusrobs.e" (if present).
6. If you copied your recorded macros file ("vusrmacs.e"), load this macro now with the Load Module dialog box ("Macro...", "Load Module...").

UNIX Platforms:

To manually transfer your configuration changes from the previous version of Visual SlickEdit, perform these steps:

1. If you have made color coding changes, copy your "user.vlx" file (located in your "\$HOME/.vslick" directory) into your "\$HOME/.vslick" directory of the new version.
2. If you have recorded macros, copy your "vusrmacs.e" file (located in your "\$HOME/.vslick" directory) into your "\$HOME/.vslick" directory of the new version.
3. To transfer other configuration changes, bring up the previous version of Visual SlickEdit, activate the Visual SlickEdit command line at the bottom of the editor (ESC for most emulations), type **list-source** and press ENTER. This will create the macro file "vusrdefs.e", and possibly another macro file called "vusrobs.e" in your "\$HOME/.vslick" directory. Copy "vusrdefs.e" and (optionally) "vusrobs.e", into the "\$HOME/.vslick" directory of the new version.
4. Run the new version of Visual SlickEdit.

5. If you ran the "list-source" command, type **vusrmods** and press ENTER on the Visual SlickEdit command line. This will switch emulations if necessary, and run "vusrdefs.e" and "vusrobjs.e" (if present).
6. If you copied your recorded macros file ("vusrmacs.e"), load this macro now with the Load Module dialog box ("Macro...", "Load Module...").

Q: How do I backup my configuration?

A: Visual SlickEdit version >=7.0 stores all configuration files in a configuration directory. To determine the configuration directory, invoke the Program Information menu item ("Help", "Program Information...") and look for the configuration directory setting. Simply backup all the files in this directory. Use the **"-sc Directory"** invocation option to change the configuration directory.

Q: Frequently asked license manager questions.

See **License Manager** in the Visual SlickEdit Installation Guide.

Q: Why doesn't Auto List Members work for me?

A: Make sure you have created tag files for your run-time libraries (see **Creating and Updating Tag Files** on page 50). In addition, you should create a project and add your source files to it so your source is tagged as well. If you still have a problem, see **"Context Tagging(TM) Limitations"** in the online help.

Q: Why doesn't Auto Parameter Info work for me?

A: Make sure you have created tag files for your run-time libraries (see **Creating and Updating Tag Files** on page 50). In addition, you should create a project and add your source files to it so your source is tagged as well. If you still have a problem, see **"Context Tagging(TM) Limitations"** in the online help.

Q: What is a project in Visual SlickEdit?

A: A project in Visual SlickEdit is a great way of organizing your work into a neat package. This package includes a working directory, compile/build command lines (where you tell Visual SlickEdit what to execute for your compile/build), and which source files you want as part of the project. Once you supply this information, you can do compiles/builds from within Visual SlickEdit and have your error messages automatically processed. See **"Creating Projects"** on page 32 for more information.

Q: What is a workspace in Visual SlickEdit?

A: The purpose of a workspace is to allow you to share projects in multiple workspaces and to define dependencies between projects. For example, building project A might depend on building project B first, which is a utility library linked with project A. You might find that you only need one project for each of your workspaces. In this scenario, workspaces will not

help you. Another feature you get with workspaces is compatibility with other vendors' workspaces. Visual SlickEdit supports opening Visual C++ and Tornado workspaces. This allows more seamless integration with these products. The data for each workspace is stored in an ASCII (EBCDIC for OS/390) file which has the extension ".vpw". See "[Creating Projects](#)" on page 32 for more information.

Q: Why did my key bindings change when I installed version 7.0?

A: If you installed version 7.0 on top of your previous installation, your key bindings should remain the same. If you did not, some emulations in version 7.0 have some minor key binding changes. Version 7.0 has a debugger for Java, so some key binding changes were made because frequently used debugging commands needed to be on function keys.

The F11 key was changed from the "javadoc_editor" command to "debug_step_into" for all emulations except ISPF emulation. The new binding for the "javadoc_editor" command is CTRL+SHIFT+D.

The F9 key was changed from the "undo" command to "debug_toggle_breakpoint" for CUA, GNU Emacs, and Epsilon emulation. In CUA emulation, the "undo" command is bound to CTRL+Z and ALT+BACKSPACE. For GNU Emacs and Epsilon emulation the "undo" command is bound to CTRL+/ and CTRL+X 'u'.

The F5 key was changed from the "config" command to "project_debug" for CUA, GNU Emacs, and Epsilon emulation. The "config" command is no longer bound to a key in CUA emulation. For GNU Emacs and Epsilon emulation the "config" command is bound to ALT+P.

If you prefer some of the old key bindings, use the Bind Command to Key dialog ("Tools", "Configuration", "Keybindings...") to bind the old command to the appropriate key.

Q: What is tagging?

A: "Tagging" (or "Procedure Tagging") is a facility that allows you to quickly and easily build a database of function definitions, function prototypes, global variables, and various other constructs depending on the language you are working with. This tag data is also used to implement the **Context Tagging(TM)** features, which include Auto List Members and Auto Parameter Info. Visual SlickEdit automatically creates and maintains a tag database for files in your project. Once a tags database is built, you can simply jump to the location of the tag (maybe a function definition), by putting your cursor on the name and pressing a hot-key (CTRL+PERIOD for CUA emulation). If you want to "pop" back to where you were, simply press another hot-key (CTRL+COMMA for CUA emulation). See "[Dynamic Tagging](#)" on page 57 for more information. The tag databases are also used by the Class Browser. See "[Class Browser](#)" on page 60 for more information.

Q: I use Visual C++. How do I compile from inside of Visual SlickEdit?

A: See "[Using Visual SlickEdit with Visual C++ or Visual Studio .NET](#)" on page 178.

Q: How do I compile from inside of Visual SlickEdit?

A: There are two ways to do it:

1. The most common way is to set up a Project within the editor by clicking the "New..." menu item under the Project menu.
 - Select the appropriate package (or choose "None"), enter the project name, choose a directory, enter the executable name.
 - Add your source files to the project by using the "Project Files" tab ("Project", "Project Properties...", select "Files" tab).
 - Fill in the Working Directory using the project Directories Tab ("Project", "Project Properties...", select "Directories" tab).
 - Use the project Tools Tab ("Project", "Project Properties...", select "Tools" tab) to set your build and compile command strings.
 - To load the source files that you specified in your project, use the "Load Files..." menu item under the Project menu.

The compile and build commands are on the Project menu and will show you key bindings specific to your emulation. See "[Creating Projects](#)" on page 32 for more information.

2. The second way to add compile/build support is to bring up the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the "Advanced" tab).
 - Select the extension you want to affect
 - Click "Extension Specific Project..."
 - Type in your compile/build command strings

This is an extension-wide setting, but remember, if you have a project open with a compile/build command defined, the extension-specific compile/build command is ignored. See "[Building from Within the Editor](#)" on page 146 for more information

Q: Why does it take so long to start Visual SlickEdit?

A: The most common delay comes from auto-restoring remote files. If a remote file is no longer accessible, it can take 30 seconds or more for Visual SlickEdit to start. To correct this problem, simply close the remote file or all files if you do not know which one is causing the problem.

If you are auto-restoring a project with thousands of files, and you have the Project toolbar docked, this can increase Visual SlickEdit's start-up time. There is not currently an effective workaround for this performance slow down.

Q: Why can't I run my compiler in Visual SlickEdit?

A: Windows 95 or later, and Windows NT or later:

Some older 16-bit compilers will not run in the concurrent process buffer. From the Project Properties dialog ("Project", "Project Properties..."), turn off the "Output to process buffer" check box for the "Compile" and/or "Build" tools. Some compilers which prompt for input cannot be run in the concurrent process buffer. If this is your problem, turn off the "Capture output" check box on the Project Properties dialog ("Project", "Project Properties...") for the "Compile" and/or "Build" tools.

A: UNIX:

Some compilers which prompt for input cannot be run in the concurrent process buffer. If this is your problem, turn off the "Capture output" check box on the Project Properties dialog ("Project", "Project Properties...") for the "Compile" and/or "Build" tools.

Q: Why am I getting a VSLICKERRORPATH line in my ".process" buffer?

A: This fixes an old error process limitation where the compiler's error output was dependent upon the current directory. This allows you to change the current directory after a compile and still process your error output. This change was necessary to implement the new project dependencies features.

It is possible to suppress this output for a Project command. However, you should not suppress this output if the command creates error output that you want processed. Go to ("Macro", "Set Macro Variable..."), enter the variable **def_no_error_info_commands**, and add the name of the program your project command executes to the list of program names.

Q: Why doesn't my alias expand when I press SPACEBAR or CTRL+SPACEBAR?

A: SPACEBAR: Aliases will only expand if the alias is the first and only word on the line.

A: CTRL+SPACEBAR: The new binding for the "expand_alias" command is CTRL+SHIFT+O. If you want to change this binding, use the Key Bindings dialog ("Tools", "Configuration", "Key Bindings..."). (The command bound to CTRL+SPACE was changed because context tag completion was more important and this kept our key binding for this the same as Visual C++.)

Q: How do I get my foreign language characters to display properly?**A: Windows:**

The default font uses the OEM character set. This font will not display foreign characters correctly. You need to choose a font which uses the "Western" Script character set. To change the font, bring up the Font Configuration dialog ("Tools", "Configuration", "Font..."), select the "Window Text" element, and choose the courier font (or any other font which has, or defaults to, the "Western" script character set).

Q: How do I type my foreign language characters?**A: UNIX:**

You need to invoke Visual SlickEdit with the "-sua" option. This will allow the right ALT key to support foreign language character entry. To avoid typing this option every time you run Visual SlickEdit, set the VSLICK environment variable to "-sua".

Q: How do I see the name of the function I am editing?

A: There are three techniques for seeing the name of the current function:

For Visual SlickEdit 2.0 and above:

Use the prev-proc command (CTRL+UP) to jump to the top of the current function and look at its signature, then use Undo (ALT+BACKSPACE) to return to where you were.

For Visual SlickEdit 3.0 and above:

With the project toolbar visible, select the "Procs" tab. All functions in the current buffer will be visible and the current function will be highlighted.

For Visual SlickEdit 4.0 and above:

From the Visual SlickEdit menu bar, select "View-->Toolbars", switch to the "Categories" tab and select the "Bookmarks and Tags" category. Drag the "Current Context" combo box control to a button bar. This control will dynamically display the tag currently being edited; that is, the symbol or local variable in the scope of the cursor. In addition, you can pull down the combo box to see all locals and tags in the current buffer and select one to jump to.

Q: What is the "Symbol" tab on the Output toolbar?

A: The "Symbol" tab on the "Output" toolbar displays either a list of symbols matching the symbol under the cursor in the current buffer, or, if exactly one tag matches, it shows the source code for that tag. When the editor focus is on the "Project" toolbar, "Tag Properties" toolbar, List Members dialog, or Inheritance dialog, the Symbol Window "previews" the currently selected tag in the list or tree control. For more information see "Symbols Tab" on page 57.

Q: Why can't I set environment variables in the ".process" buffer?

A: **Windows 95 or later:**

The problem is the amount of environment space being used for a process. Edit your "config.sys" file and add or edit a SHELL line such as the one below:

```
SHELL=c:\command.com c:\ /p /e:2048
```

Where /e:2048 specifies an environment size of 2048 bytes. Make this larger if you still are out of environment space.

Q: How do I get my Tab key to insert a real tab character?

A: Bring up the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Indent tab) and select the extension you want to affect. The "fundamental" extension is used for files with no extension or which are not in the list. Turn on the "Indent with tabs" check box and click OK.

Q: Where are the small or large nice-looking fonts?

A: **Windows:** Choose the "Terminal" font for the largest selection of attractive, fixed fonts. These are the fonts used in the DOS box.

Q: Why is the left edge of my window a different color?

A: Bring up the Color Settings dialog box ("Tools", "Configuration", "Colors...") and set your "Line Number" color to match your "Window Text" color.

Q: How do I set up a new file extension in Visual SlickEdit?

A: Bring up the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...") and click on the "New..." button.

To set up color coding for a new language, bring up the **Color Coding Setup dialog box** ("Tools", "Configuration", "Color Coding...") and click on the "New..." button on the far right of the dialog box. If you create a new color coding lexer, you will need to fill in the "Lexer Name" text box on the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup...", select the Advanced tab) to see your color coding.

Q: How do I add or modify keywords in my color coding?

A: Bring up the **Color Coding Setup dialog box** ("Tools", "Configuration", "Color Coding...", select the "Lexer Name" you want to affect) and use the "New..." and "Delete" buttons on the left/middle of the dialog box.

Q: How do I add color coding for my language?

A: To set up color coding for a new language, bring up the **Color Coding Setup dialog box** ("Tools", "Configuration", "Color Coding...") and click "New..." (on the far right of the dialog

box.) Fill in the "Lexer Name" text box on the Extension Options dialog box ("Tools", "Configuration", "File Extension Setup..."), select the "Advanced" tab) to see your color coding.

Q: My toolbar has disappeared. How do I get it back?

A: Go to ("View", "Toolbars...") and click on the "Toolbars" tab. Double-click on the name of the toolbar to bring it back.

Q: My toolbar is undocked. How do I redock it?

A: To redock the toolbar, click on the gray area of the toolbar (for example, the gray area next to the tabs on the "Output" and "Project" toolbars) and drag it to the location where you would like to dock it. When the outline of the toolbar changes shape and/or clarity, you are in docking position. Release the mouse to dock.

Q: How do I toggle the Project or Output toolbar on/off using the keyboard?

A: To toggle the toolbars on and off, there are two macros: "toggle_output()" and "toggle_project()" that will show and hide their respective toolbar in the previous location. You can call the two macros from the Visual SlickEdit command line, or you can bind them to keys under ("Tools", "Configuration", "Key Bindings..."). To bind them to a key, type the name of the macro on the line marked Command and click "Add Key or Mouse Click". Then, press the key combination you want to bind the macro to and click "Bind". Repeat this with the other macro, then click "Done."

Q: How do I reset all my toolbars to their original docked position?

A: To restore the toolbars to their default positions, exit the editor, and delete the "vrestore.slk" file (located in your "\$HOME/.vslick" directory for UNIX, and in your executable directory for Windows). The toolbars will be displayed in their default positions when you restart the editor.

Q: How do I add a new button to a toolbar?

A: Go to ("View", "Toolbars...") and click on the "Categories" tab. Select a button category. Click on one of the toolbar buttons shown and drag it to the toolbar you would like to add to and release. The button will now be added to the toolbar.

If you would like a user-defined button, select the "User Definable Tools" category and drag one of the buttons to the toolbar. Then, right click on the toolbar button and select "Properties" from the menu. In the Properties dialog, provide the button with a command, a message to be displayed when you place your mouse pointer on the button, and even a bitmap file if you would like.

Q: How do I remove a button from a toolbar?

A: Bring up the Toolbar Customization dialog box ("View", "Toolbars..."). Now drag the button you want to remove off the toolbar and release the mouse.

Q: How do I keep the currently-selected text selected after I copy it to the clipboard?

A: Go to ("Macro", "Set Macro Variable...") and set the variable "def_deselect_copy" to a value of **0**. Remember to save your configuration.

Q: How do I get the cursor to snap to the end of a line when I click?

A: Set the macro variable "def_click_past_end" to **0** ("Macro", "Set Macro Variable...").

Q: How do I un-install Visual SlickEdit?

A: **Windows:**

Run the vsuninst.exe program in the "\vslick\win" directory.

A: **UNIX:**

Delete the /usr/lib/vslick directory and everything under it.

Q: I am using Reflection X 6.0 for Windows. Why does my left ALT key cause keyboard problems?

A: The problem lies with the Reflection X server. WRQ has a patch to correct the problem on their FTP site called **xpatch01.zip** and unzips into a replacement **rx.exe**.

Q: How do I break out of a long search?

A: To break out of a long search, hold down CTRL+ALT+SHIFT for at least 3 seconds.

Q: How do I break out of an infinite Slick-C loop?

A: To break out of a an infinite loop in the Slick-C macro language, hold down CTRL+ALT+SHIFT for at least 3 seconds. If the infinite loop occurs in a timer hold down CTRL+ALT+SHIFT+F1.

Glossary

absolute

A file name that has a complete path specification.

binding

The attachment of a command to a key.

buffer

A file that has been loaded into Visual SlickEdit. Once a file has been loaded, you can safely perform modifications to the buffer without modifying the file on disk until you save the buffer.

clipboard

A temporary storage area used to transfer text or dialog box controls from one place to another. Since the Microsoft Windows operating system does not have adequate multiple clipboard support, Visual SlickEdit stores multiple text clipboards internally.

container variable

A Slick-C variable of type **typeless**.

concurrent

Running at the same time as something else.

delimiter

A separator of two objects.

edit window

A rectangular viewing area used to display buffers. More specifically, an MDI (Multiple Document Interface) child window which displays a buffer.

file name

Name of a file on disk or the name used when storing a file on disk.

global

A variable or identifier that can be accessed by more than one module.

keyin

A sequence of characters entered into edit window or command line as if typed from the keyboard.

local

A variable that can be accessed by only one procedure.

make

A process started when a source file has a later date than a target file.

mark

Highlighted text. **select** and **selection** is also used instead of mark.

module

Source code whose function is to provide a set of tasks or a task. A single unit that can be compiled.

octal

Base 8 arithmetic.

pcode

The binary result of a translation of source code. The translation is done to speed up the interpretation of source code.

relative

A file name with the shortest name possible that distinctly identifies the location of a file.

selection

A highlighted region of text which is typically operated by a command which affects only the region. In the dialog editor, your selection is indicated by 8 square handles which surround the control.

state file (file extension “.sta” for Windows, “.stu” for UNIX)

A file which stores enough configuration information to enable the editor to load this file and restore the configuration in subsequent edit sessions.

static

A macro variable whose scope is more than a single procedure but limited to the module it is defined in.

view

A view contains the information necessary for the editor to remember your location in a **buffer**.

window

A rectangular viewing area. In the more advanced sections of this manual this term also refers to the operating system resource known as a window. Do not confuse this term with the Microsoft Windows operating system.

Index

- #pragma, 286
- / command, 98
- [] operator, 268
- @SLICK, 136
- [] operator, 267
- [_command](#), 279
- [_control](#), 288, 311
- [_delete_window](#), 301, 303
- [_inherit](#), 291
- [_resume](#) built-in, 346
- [_str](#), 263
- 3 Way Merge Editing, 69
- Accessing And Editing FTP Files, 159
- Active form, 311
- Active object, 311
- Ada, 7
 - syntax expansion/indenting options, 218
- add command, 81, 89
- Add Key or Mouse Click, 225
- Add Selected Expr menu item, 81
- Add User 1, 86
- Add User 2, 86
- Adding completion to command, 280
- Adding language support, 10
- Adjust Block menu item, 80
- [adjust_block](#) command, 80
- Affect all modes, 225
- ALIAS facility, 154
- Alias Filename, 219
- Aligning Controls, 296
- Allow in non-MDI editor control, 92
- Allow in read only mode, 92
- Alphabetic, 110, 113, 116
- Alt menu, 206
- Alt menu hotkeys, 206
- AND operator, 261
- Anonymous Unions, 267
- API Apprentice, 5
- append command, 80
- Append Cut menu item, 79
- Append EOF character, 235
- Append to Clipboard menu item, 79
- [append_cut](#) command, 79
- [append_to_clipboard](#) command, 79
- [arg](#) built-in, 278
- [Argument Declarations](#), 277
- Array length method, 267
- Array variables, 267
- [Arrays](#), 267
- Assembler, 243
 - Intel, 7
 - OS/390, 7
- Associative array variables, 268
- Attribute, 163
- Auto exit process, 212
- Auto List Members, 207
- Auto list params, 208
- Auto Parameter Info, 208
- Auto reload, 232
- [Auto Restore](#), 209
- Automatic Directory Mapping, 69
- AutoSave Tab, 236
- AWK
 - syntax expansion/indenting options, 218
- Backspace key configuration, 228
- backup
 - hard disk, 160
- [Backup options](#), 235
- Batch macro, 255
- Begin/end Pairs, 223
- [bigfloat](#), 263
- [bigint](#), 263
- [bigstring](#), 263
- Binary Files
 - Editing, 70
- Binary math result, 89
- Binary operators, 89
- Bind, 225
- Bind Command to Key dialog box, 224
- Bitmaps
 - Adding to image control, 325
 - Adding to list box, 319
- Bitwise
 - and, 261
 - complement, 260
 - or, 261
 - xor, 261
- Blank Lines After Header, 132
- Blank Lines Before Footer, 132
- Bookmarks, 125
- [Bookmarks dialog box](#), 126
- [boolean](#), 263, 273
- brace style, 15
- [break](#), 274
- Break Process menu item, 146
- Breakpoints
 - setting, 150

- BRIEF, 13
- Buffer cache size, 215
- Buffer Input/Output Functions, 336
- Build
 - menu item, 146
- Building within the editor, 146
- C#, 7
- C/C++, 7
- C++
 - syntax expansion/indenting options, 218
- Calculate Expression menu item, 89
- Calculator dialog box, 88
- Calls and uses of a symbol, 61, 62
- Cancel button
 - Adding, 300
- Capture output
 - Check Box, 39
- Cascade menu item, 76
- cascade_windows command, 76
- case, 275
- case statement, 275
- casting types, 271
- cd command, 6, 74
- CFScript, 7
- Change, 86
- Change All, 86
- Change dir, 206
- change_window command, 76
- Changing
 - active drive (See also change-drive command), 6
 - directory (See also cd command), 6
- Changing current directory, 74
- Changing Key Bindings, 224
- Check Box control, 316
- CICS, 7
- Class Browser, 60
- class path, 37
- Clear, 98, 131
- Clear process buffer
 - Check Box, 39
- Clipboard Inheritance(R), 306
- Clipboards
 - multiple, 83
- Closing a dialog box, 300
- Cmd Help
 - Button, 226
- Cmdline prompting, 206
- COBOL, 7
 - syntax expansion/indenting options, 218
- CodeWarrior, 13
- Color Coding, 137
- Color Settings dialog box, 238
- Colors, 238
- Columns per Line, 132
- Combo Box control, 316
- Command Button control, 315
- Command history, 73
- Command line editing, 71, 72
- Command line interfaces, 343
- Command line options, 23
- Command retrieval, 73
- Commands
 - defining, 279
- Comments
 - defining, 256
- Common Dialog Boxes, 331
- Common List, 87
- Compile
 - menu item, 146
- Compiling macros, 256
- Completion, 72
 - Adding to command, 280
- compression with tabs, 243
- Concatenation, 258, 261
- Conditional breakpoints, 150
- Configuration
 - changing, 204
- Configurations
 - Project, 36
- const, 259
- Constants
 - defining, 259
 - numeric, 258
- Context Tagging(TM), 45
- continue, 274
- Control characters
 - Inserting, 127
- Converting SlickEdit 2.3 Macros, 355
- Copy
 - selected files, 164
- Copy menu item, 78
- copy_to_clipboard command, 78
- copy_to_cursor command, 79
- Count number of lines, 231
- create_tile command, 75
- Creating
 - projects, 33
 - workspaces, 33
- Creating dialog boxes, 294
- CUA, 211
- CUA Text Box, 213
- Current directory
 - Changing, 74

- Cursor wrap, 228
- cursor_error command, 146
- cut command, 78
- Cut menu item, 78
- dBASE
 - syntax expansion/indenting options, 218
- DD, 196
- DDNAME, 196
- debugging macros, 293
- Declaring (See defining), 256
- def, 289
- Default Arguments, 278
- defeventtab, 290
- defeventtab primitive, 289
- Defining
 - comments, 256
 - constants, 259
 - event tables, 291
 - functions, 277
 - global variables, 262
 - keys, 289
 - procedures, 277
- definit, 292
- defload, 292
- defmain, 284
- Del key configuration, 228
- Delete
 - select files, 164
- delete_tile command, 76
- Delimiter used by search command, 100, 105
- Delphi, 7
 - syntax expansion/indenting options, 218
- Deselecting files, 163
- Dialog Box Retrieval, 326
- Dialog Boxes
 - Common, 331
- Dialog editor, 294
- Dialog Editor Properties dialog box, 298
- Difference editing, 63
- Differencing Files with DIFFzilla™, 63
- Diffing
 - All Symbols of Two Files, 68
 - Source Trees, 65
 - Symbols or Parts of Files, 67
 - two Files, 63
- dir command, 161
- Directory List Box control, 323
- DLL Interface, 342
- do, 274
- Document math, 90
- DOS
 - access to, 73
 - dos command, 73
- double, 263
- Drive List control, 321
- DTD Caching, 168
- Duplicate Window menu item, 76
- duplicate_window command, 76
- Dynamic Difference Editing™, 63
- Dynamic Tagging, 57
- Edit "A B C" start on file A, 212
- Editing
 - binary file(s), 24
 - file name with space characters, 25
 - file name with wild cards, 24
 - MACINTOSH format files, 25
 - record files, 154
 - selected files, 165
 - UNIX format files, 25
- Editor Control, 315
- EMACS, 13
- Embedded Language Support, 9
- Encoding
 - extension specific, 220
 - global setting, 233
- Enter key configurations, 227
- Environment
 - setting, 246
- Environment variables
 - VSLICK, 26, 245
 - VSLICKALIAS, 245
 - VSLICKBACKUP, 245
 - VSLICKBIN, 245
 - VSLICKBITMAPS, 245
 - VSLICKCONFIG, 166, 244
 - VSLICKINCLUDE, 287
 - VSLICKLOAD, 245
 - VSLICKMACROS, 245
 - VSLICKMISC, 245
 - VSLICKPATH, 166, 245, 287
 - VSLICKRESTORE, 244
 - VSLICKSAVE, 245
 - VSLICKTAGS, 245
 - VSLICKXTERM, 74, 245
 - VST, 246, 287
 - VSUSER, 246
- Epsilon, 13
- Error message recognition, 147
- Event driven dialog box, 253
- Event driven dialog boxes, 291
- Event tables, 289
- exception breakpoints, 151
- execute_selection command, 80
- Executing programs, 73

- Executing programs from macro, 342
- Expand tabs to spaces, 232, 235
- Exponent, 90
- Expressions, 89, 260
- Extension Options dialog box, 216
- false**, 263
- File differencing, 63
- File Extension Setup, 216
- File Filters Tab, 238
- File List Box control, 322
- File locking**, 232
- File manager**, 160
- File merging, 63, 69
- File Options dialog box, 230
- Fileman mode, 161
- fileman_mode command, 161
- Files/Buffers, 97, 130
- Fill menu item, 79
- fill_selection command, 79
- filter_selection procedure, 337
- Filtering marked text, 337
- find command, 98
- Find dialog box, 96
- Find Slick-C Error menu item, 294
- find_error command, 294
- find_matching_paren command, 87
- find_proc command, 294
- find_tag command, 57
- Floating point numbers, 90, 258
- Fonts
 - changing defaults, 240
- Footer, 129, 132
- For**, 274
- Format line**, 136
- Format Selection menu item, 80
- Formatting disks, 165
- Fortran, 7
 - syntax expansion/indenting options, 218
- Frame control, 315
- FTP Support, 159
- Function Pointers, 270
- Functions
 - defining, 277
- Gauge control, 323
- General Configuration dialog box, 205
- Generating File Lists, 68
- get_string procedure, 344
- Global directory, 236
- Global nested directories, 236
- Global scope variables, 262
- Global search and replace, 106
- GNU C/C++ Debugger, 148
- GNU Emacs, 13
- Go to Error/Include menu item, 146
- goto_bookmark command, 127
- Hack tabs backspace**, 229
- Hash Tables**, 268
- Header, 129, 132
- Help, 29
- Hex character code, 257
- Hex math result, 89
- hexadecimal numbers, 258
- Hidden files, 161
- History
 - command, 73
- Horizontal scroll bar, 207
- Hscroll Bar control, 321
- Hsplit menu item, 75
- hsplit_window command, 75
- HTML, 7
- i_search command, 95
- Identifier characters, 262
- IDL, 7
- Ignore, 86
- Ignore All, 86
- Image control, 324
- Include Directories, 35
- Include hidden files, 161
- Include System Files, 161
- Including macro header files**, 287
- Incremental searching, 95
- Indent Amount, 14
- Indent menu item, 79
- Indent Style, 217
- indent_selection command, 79
- indenting style, 14
- Inheritance, 291, 306, 309
- initializing
 - macro modules, 292
- Inserting control characters, 127
- InstallScript, 7
- Instances, 311
- int**, 263
- Intel Assembly, 7
- Invert, 98, 130
- Invocation options, 23
- ISPF, 13
- Java, 7
 - syntax expansion/indenting options, 218
- Java Debugger, 148
- Java options, 37
- JavaScript, 7
- JES, 192
- JSP, 7

- Jump over tab characters, 229
- Justification dialog box, 136
- Justify Style, 219
- Key bindings, 224
- Key message delay, 214
- Label control, 313
- Languages
 - adding support for other languages, 10
- Line insert style, 214
- Line modify, 209
- Line wrap on text, 229
- Lines per Page, 132
- Link Window menu item, 75
- link_window command, 75
- List All Occurrences
 - Check Box, 97
- List Box control, 319
- List Clipboards dialog box, 85
- list command, 161
- list_clipboards command, 78
- load command, 256, 289
- Load entire file, 231
- Load options, 230
- Loading macros, 256
- long, 263
- Loops, 274
- Lowcase menu item, 79
- lowcase_selection command, 79
- MACINTOSH, 25
- Macro
 - prompting, 343
 - Selection filters, 337
- Macro converting
 - From SlickEdit 2.3, 355
- Macro Programming, 248
- Macro recording, 90
- Mantissa, 90
- Margins, 129, 134, 218
- margins command, 135
- Margins dialog box, 135
- Margins menu item, 135
- Mark_id, 337
- Match case
 - Check Box, 96
- Match groups (see Tagged search expressions), 123
- Match Paren menu item, 87
- Match whole word
 - Check Box, 96
- math command, 89
- mathb command, 89
- Mathematical expressions, 89
- matho command, 89
- mathx command, 89
- Max clipboards, 215
- Max undo steps, 233
- Maximal matching, 125
- Maximize first window, 206
- Menu
 - Changing or Adding, 327
- Menu Macro Programming, 327
- Merging changes, 63, 69
- Methods, 312
- Minimal matching, 125
- Modal and modeless dialog boxes, 304
- Mode Name, 219
- Modified lines
 - resetting, 137
- Modula-2, 7
 - syntax expansion/indenting options, 218
- mou_click, 78
- mou_click_copy, 78
- mou_copy_to_cursor command, 78
- Mouse Operations, 30
- Move
 - selected files, 164
- Move menu item, 79
- move_edge command, 76
- move_to_cursor command, 79
- Moving Controls, 296
- Multi-File Diff Output dialog box, 66
- Multiple clipboards, 83
- Multiple file search and replace, 106
- Multiple Instances, 311
- name_info Attributes, 280
- Next Error menu item, 146
- Next menu item, 76
- Next Occurrence menu item, 102
- Next word style, 214
- next_error command, 146
- next_window command, 76
- no_code_swapping, 345
- NOT operator, 260
- null, 267
- Number Lines Every, 132
- Numeric constants, 258
- Numeric sort, 93
- Object Instances, 311
- Octal numbers, 89
- OK button
 - Adding, 300
- One file per window, 206
- One Window menu item, 76
- one_window command, 76

- Operators, 260
- Options
 - Extension, 217
 - Spelling, 87
- OR operator, 261
- OS Shell menu item, 73
- OS/390 Assembler, 7
- Output to process buffer
 - Check Box, 38
- Overlay Block menu item, 80
- overlay_block command, 80
- p_active_form, 311
- p_window_id, 311
- Paren match style, 213
- Pascal, 7
 - syntax expansion/indenting options, 218
- paste command, 78
- Paste menu item, 78
- Perl, 7
- PHP, 7
- Picture Box control, 323
- Pictures
 - Adding to image control, 325
 - Adding to list box, 319
- PL/I, 7
- PL/SQL, 7
- Place cursor at end
 - Check Box, 97
- Pointer Variables, 269
- Pointers to Functions, 270
- Preprocessing, 285
- Preserve column on top/bottom, 213
- prev_window command, 76
- Previous menu item, 76
- Print Command, 132
- Print Device, 132
- Print dialog box, 128
- Print Filter, 132
- Printing, 127
- Procedures
 - defining, 277
- Product Support, 2
- Project
 - Extension, 224
- Project Build Commands tab, 40
- Project Configuration Settings Dialog Box, 34
- Project Directories tab, 35
- Project Management, 32
- Project Open Command tab, 41
- Project Properties dialog box, 34
- Project Tools tab, 35
- project_build command, 146
- project_compile command, 146
- Projects, 32
- prompt procedure, 344
- Prompting
 - from macros, 343
- Properties dialog box, 298
- prototypes, 282
- Pull chars backspace, 229
- put command, 80
- PVCS, 41
- PV-WAVE, 7
- Python, 7
- quote_key command, 127
- Radio Button Control, 315
- rc variable, 345
- RCS, 41
- Read-only, 165
- Record Files
 - Editing, 154
- Recording macros, 90
- Redefine Common Keys, 227
- Redefine Common Keys dialog box, 227
- Refer To, 217
- References, 35
 - finding references of a symbol, 62
- Reflow next, 212
- reflow_selection command, 80
- Regular expressions, 107
- Reinsert After Current, 233
- Remove EOF character, 235
- Repeat Command on Selected dialog box, 165
- Replace dialog box, 102
- Requires editor control, 92
- Resetting modified lines, 137
- Restore clipboards, 209
- Restore process, 209
- Restore selective display, 209
- Restore working directory, 209
- Restore workspace files, 209
- Restrict to selection
 - Check Box, 97
- resume command, 105
- return statement, 277, 279
- reverse_i_search command, 95
- REXX, 7
 - syntax expansion/indenting options, 218
- Run
 - Button, 226
- Run-time error finding, 294
- Same directory as *.BAK, 236
- Save files on loss of focus, 235
- Save Macro dialog box, 91

- Save options**, 234
- Save/Restore file pos, 233
- SCCS, 41
- scope of variables, 262
- Screen Element, 239
- Screen layout, 27
- Scroll style, 213
- Search Functions Overview, 333
- Searching
 - delimiter, 100, 105
 - for variables, 99, 103
 - incremental, 95
 - multiple file replace, 106
 - next occurrence, 102
 - options, 96
 - regular expressions, 107
 - replace, 103
- Select Styles Tab, 211
- Selecting Controls**, 295
- Selection functions Overview, 337
- Selection Only, 132
- set command, 246
- set_bookmark command, 126
- Setting environment variables, 246
- Setting Properties**, 295
- Shelling from macro, 342
- Shelling programs, 73
- Shift Left menu item, 79
- Shift Right menu item, 80
- shift_selection_left command, 79
- shift_selection_right command, 80
- Short key names, 212
- Sizing Controls**, 296
- Slick-C, 7, 248
 - syntax expansion/indenting options, 218
- Slick-C Language, 248
- SlickEdit, 13
- SlickEdit 2.3
 - Converting macros, 355
- Smart indenting (see syntax indenting), 81
- SmartPaste(TM), 83
- Sort buffer, 93
- Sort dialog box, 93
- Sort on selection, 93
- Sort on Selection menu item, 81
- Sort within selection, 93
- Sort Within Selection menu item, 80
- sort_buffer command, 94
- sort_on_selection command, 81, 94
- sort_within_selection, 94
- sort_within_selection command, 80, 94
- Sorting, 93
- SourceSafe, 41
- Spell checking, 85
- Spill file path, 215
- Spin control, 313
- st command, 256
- Start in Insert Mode, 212
- State cache size, 215
- State file
 - alternate, 24
 - swapping, 24
- State file caching, 345
- static**, 262
- stop_process command, 146
- string concatenation, 261
- String Functions Overview, 331
- String literals, 257
- Strip trailing spaces, 235
- struct**, 264
- Subdirectories, 161
 - Check Box, 130
- Suppress reload prompt, 232
- suspend built-in, 346
- switch**, 275
- Symbol references, 62
- Symbol Tagging, 57, 60
- Tab
 - compression when saving, 243
 - expansion when loading, 25
 - indenting with, 218
- Tab key configuration, 14
- Tab stop
 - setting, 134
- tab_ext command, 243
- Tabs, 218
 - indenting with, 135
- tabs command, 134
- Tabs dialog box**, 134
- Tag Files, 50
- Tag Files dialog box**, 50
- Tagged search expressions, 123
- Tagging, 57, 60
- Tcl, 7
- Template editing, 81
- Text Box control, 314
- Text box editing, 71
- Text Mode Print dialog box, 131
- Throw Away File Lists, 212
- Tile menu item, 76
- tile_windows command, 76
- TLIB, 41
- Tool tips, 214
- Top of file line, 206

Transact SQL, 7
true, 263
Truncate file at EOF, 231
Type Casting, 271
typeless, 263, 272
typeless variable declaration, 262
TypeName, 263
Unary operators, 89, 260
Unbind, 225
Undo Last, 87
Unicode and SBCS/DBCS Macro
 Programming, 339
Unicode Support, 5, 167, 170, 220, 221, 240
Unindent menu item, 79
unindent_selection command, 79
union, 266
UNIX, 25
Unmodified block spilling, 233
Up/Down on Text, 229
Uppcase menu item, 79
upcase_selection command, 79
Use child directory, 236
User List 1, 87
User List 2, 87
Uses of a symbol, 61, 62
Using Unicode, 170
Validating and Well-formedness Checking of
 XML Documents, 167
var, 279
Variable
 characters supported, 262
 declaring, 262
VBScript, 7
Version Control, 41
Vertical line column, 210
Vertical scroll bar, 207
VHDL, 7
VI, 13
Visual C++ emulation, 13
Visual C++ Integration, 178
vsbuild, 37, 40
Vscroll Bar control, 321
VSLICK, 26, 245
vslick.sta, 24, 345
vslick.stu, 24
VSLICKALIAS, 245
VSLICKBACKUP, 245
VSLICKBIN, 245
VSLICKBITMAPS, 245
VSLICKCONFIG, 166, 244
VSLICKINCLUDE, 287
VSLICKLOAD, 245
VSLICKMACROS, 245
VSLICKMISC, 245
VSLICKPATH, 166, 245, 287
VSLICKRESTORE, 244
VSLICKSAVE, 245
VSLICKTAGS, 245
VSLICKXTERM, 74, 245
Vsplit menu item, 75
vsplit_window command, 75
VST, 246, 287
vstw program, 256
vstw.exe program, 256
VSUSER, 246
what_is command, 227
while, 274
wid_expression, 311
Window
 operations, 74
Window left margin, 214
window_above command, 76
window_below command, 76
window_left command, 76
window_right command, 76
Word Characters, 223
Word Help, 21, 22
Word help filename, 209
Word processing features, 134
Word wrap
 justification, 136, 219
 margins, 135
 on/off, 135, 219
Working directory
 Changing, 74
 project, 35
Wrap at beginning/end
 Check Box, 97
Write Selection menu item, 80
xcom command, 285
XML, 5, 167
Zoom Toggle menu item, 76
zoom_window command, 76

Reader's Comments

Thank you for using Visual SlickEdit! SlickEdit Inc. invites you to give us your comments and suggestions for improving Visual SlickEdit and this User Guide. Please take a moment to complete this Comments Sheet. SlickEdit Inc. values your input and will consider your suggestions for the next product update.

Please rate the Visual SlickEdit User Guide on the following items:

	Excellent	Very Good	Good	Fair	Poor
Ease of finding information.					
Completeness of information.					
Accuracy of information.					

Please list your suggestions for improvements to the Visual SlickEdit User Guide:

Please list your suggestions for improvements and/or enhancements to Visual SlickEdit, including language support, version control support, the code beautifier, etc.:

Please fax this sheet or email the information to:

SlickEdit Inc.
Attn: Director of Product Management
Fax +1 919.473.0080
marketing@slickedit.com

Thank you for your comments.

