## Multi-Edit 9.10

1

What's New? .................................................................................................................1

## Getting Started

5

Multi-Edit ..................................................................................................................... 5
What's Installed with Multi-Edit 9.10...........................................................................5
Upgrading From Previous Versions .............................................................................7
Upgrading Existing Multi-Edit Installations .................................................................7
Finding Multi-Edit 9 Menus in Multi-Edit 9.10 ..............................................................7
New Menu Commands in Multi-Edit 9.10 .................................................................9
Installation Requirements (Hardware/Software) ........................................................10
Add-On Integration.....................................................................................................10
Key / Command Assignments ....................................................................................11
Multi User Configurations ..........................................................................................11
Installation Overview.....................................................................................11
Configuration.................................................................................................11
Things to Do Before Installing Multi-Edit...............................................................12
Stand-Alone and Server Installation .......................................................................13
User Client Installation (Network Install Only)......................................................16
Installing on a NetWare Server ..............................................................................17
Ordering Information..................................................................................................18
License Agreement ...................................................................................................18

## Getting and Using Help

21

Technical Support.................................................................................................21
How to get Technical Support ..................................................................................21
Forum Support ........................................................................................................21
FAX Support ............................................................................................................22
Web Support............................................................................................................22
Other ways to get help ..............................................................................................22
How to get Help on CMAC .....................................................................................22
Language Specific Help............................................................................................23
Tip of the day.............................................................................................................23

## Intro to Multi-Edit

25

Introduction..............................................................................................................25
Opening and Viewing Files .....................................................................................25
Opening Files..........................................................................................................25
Saving Files .............................................................................................................26
Viewing Files .......................................................................................................................... 26
Navigating Windows .................................................................................................................. 26
Viewing Modes .......................................................................................................................... 27
Editing and Formatting Text ....................................................................................................... 27
Editing and Formatting Text ....................................................................................................... 27
Blocks and Block Marking ........................................................................................................... 27
Search and Replace ...................................................................................................................... 28
Searching Within Multi-Edit ....................................................................................................... 28
Global Expression Highlighting .................................................................................................... 28
Incremental Searching ................................................................................................................. 28
Book marking Cursor Position .................................................................................................. 28
Goto Line / Column ....................................................................................................................... 29
Language Support ........................................................................................................................ 29
Templates .................................................................................................................................. 29
Compiling .................................................................................................................................. 29
Comparing Files .......................................................................................................................... 29
Multi-Tags .................................................................................................................................. 30
Project Management .................................................................................................................... 30
Session Manager / Session Management ..................................................................................... 31
Version Control Support .............................................................................................................. 31
What are Macros .......................................................................................................................... 32
WebLair ...................................................................................................................................... 33

The Multi-Edit Interface 35

Command Sets .......................................................................................................................... 35
Choosing a Command Set .......................................................................................................... 35
Command Map ............................................................................................................................ 36
What is a Command Map? .......................................................................................................... 36
Modifying a Command Map ....................................................................................................... 36
Special WCMD Identifiers ........................................................................................................... 40
Customizing Menus ....................................................................................................................... 41
Toolbars .................................................................................................................................... 43
Customizing Toolbars .................................................................................................................. 43
Using Custom Toolbar Buttons ................................................................................................... 45
Interface .................................................................................................................................... 45
Tools Pane .................................................................................................................................. 45
Navigation Pane .......................................................................................................................... 46
Status Bar ................................................................................................................................... 47
Tab Bar ....................................................................................................................................... 47
List Boxes ................................................................................................................................... 48
Mouse Interface ............................................................................................................................ 49
Colors ....................................................................................................................................... 51
Fonts ....................................................................................................................................... 53

Working in Multi-Edit 55

Files and Windows ...................................................................................................................... 55
What is the difference between a File, Buffer and Window? ...................................................... 55
Organizing Windows .................................................................................................................... 56
Navigating Windows ..................................................................................................................... 58
Files ....................................................................................................................................... 59
Tool and Navigation Panes ................................................................. 63
Search and Replace ............................................................................. 66
   Find and Replace ........................................................................... 66
   File Find and Replace ................................................................. 67
   File Replace ................................................................................. 68
   Using Regular Expressions ......................................................... 68
   Find Word Under Cursor ............................................................ 76
   Using a Find View ....................................................................... 76
   Incremental Search .................................................................... 76
   Expression Highlight ................................................................. 77
   Examples ...................................................................................... 78
Blocks .................................................................................................. 91
   Cut / Copy / Paste ....................................................................... 91
   Using Paste Buffers View .......................................................... 91
   Internal Buffer vs. Windows Clipboard ...................................... 92
   Block Types ............................................................................... 92
   Indenting Blocks ....................................................................... 92
   Block Operations ....................................................................... 93
Editing ................................................................................................. 94
   Aligning Operators ................................................................. 94
   Commenting .............................................................................. 96
   Line Numbers .......................................................................... 96
   Center Line ............................................................................... 96
   Time/Date Stamp ...................................................................... 97
   Ruler ......................................................................................... 97
   Page Break ............................................................................... 97
   RefORMAT Paragraph ............................................................. 97
   Justify Paragraph .................................................................... 98
   Unjustify Paragraph .................................................................. 98
   UNDO and REDO ...................................................................... 98
   Formatting Lines ...................................................................... 98
   Hex Mode ................................................................................. 99
Navigating Code ................................................................................ 99
   Match Constructs ..................................................................... 99
   Goto Line / Column .................................................................. 99
   Scrolling .................................................................................. 100
   Collapse ................................................................................ 100
   Using the Preview Pane .......................................................... 102
   Multi-Tags ............................................................................... 102
   Bookmarks ............................................................................... 105
Comparing Files ................................................................................ 106
   How does it work? ................................................................. 106
   File Compare Toolbar ............................................................. 108
   Next Difference / Prev Difference ........................................... 108
Compiling Files ................................................................................ 109
   Adding a new Compiler ......................................................... 109
   Adding Error Processing ....................................................... 112
   Error Parsing Expressions .................................................. 114
   Running your compiler ......................................................... 118
   Filtering Output ...................................................................... 119
   Tracking Line Numbers ......................................................... 120
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Projects</td>
<td>152</td>
</tr>
<tr>
<td>What are Projects</td>
<td>152</td>
</tr>
<tr>
<td>Using Projects</td>
<td>152</td>
</tr>
<tr>
<td>Creating Projects</td>
<td>152</td>
</tr>
<tr>
<td>Using the Project View</td>
<td>153</td>
</tr>
<tr>
<td>FTP</td>
<td>154</td>
</tr>
<tr>
<td>Sessions</td>
<td>155</td>
</tr>
<tr>
<td>What are Sessions?</td>
<td>155</td>
</tr>
<tr>
<td>Protecting Sessions</td>
<td>156</td>
</tr>
<tr>
<td>Session Manager</td>
<td>156</td>
</tr>
<tr>
<td>WebLair – HTML Editing</td>
<td>156</td>
</tr>
<tr>
<td>Internet Programming Support</td>
<td>156</td>
</tr>
<tr>
<td>Embedded Language Support</td>
<td>156</td>
</tr>
<tr>
<td>What are Markup Language Tag Databases?</td>
<td>157</td>
</tr>
<tr>
<td>Using the HTML Formatter</td>
<td>157</td>
</tr>
<tr>
<td>Common Code Manager</td>
<td>158</td>
</tr>
<tr>
<td>Editing HTML Tags</td>
<td>158</td>
</tr>
<tr>
<td>Configuring a Browser</td>
<td>158</td>
</tr>
<tr>
<td>Automating Tasks Using Macros</td>
<td>159</td>
</tr>
<tr>
<td>Recorded Macros</td>
<td>159</td>
</tr>
<tr>
<td>Record</td>
<td>160</td>
</tr>
<tr>
<td>Repeat</td>
<td>161</td>
</tr>
<tr>
<td>A Brief intro to CMac</td>
<td>161</td>
</tr>
<tr>
<td>Other Useful Tools</td>
<td>162</td>
</tr>
<tr>
<td>Spell Check</td>
<td>162</td>
</tr>
<tr>
<td>ASCII Table</td>
<td>163</td>
</tr>
<tr>
<td>Create HTML from Code</td>
<td>163</td>
</tr>
<tr>
<td>Compile on a remote computer</td>
<td>121</td>
</tr>
<tr>
<td>METerm</td>
<td>122</td>
</tr>
<tr>
<td>Finding Errors</td>
<td>125</td>
</tr>
<tr>
<td>Language</td>
<td>125</td>
</tr>
<tr>
<td>Property Strings</td>
<td>125</td>
</tr>
<tr>
<td>Matching</td>
<td>126</td>
</tr>
<tr>
<td>Indenting</td>
<td>127</td>
</tr>
<tr>
<td>Filename Extensions</td>
<td>128</td>
</tr>
<tr>
<td>TipWin</td>
<td>132</td>
</tr>
<tr>
<td>Templates</td>
<td>134</td>
</tr>
<tr>
<td>Adding a New Language</td>
<td>136</td>
</tr>
<tr>
<td>Version Control</td>
<td>143</td>
</tr>
<tr>
<td>What is VCS?</td>
<td>143</td>
</tr>
<tr>
<td>What is SCC?</td>
<td>144</td>
</tr>
<tr>
<td>Why have VCS support in Multi-Edit?</td>
<td>144</td>
</tr>
<tr>
<td>How Does the VCS Support Work?</td>
<td>144</td>
</tr>
<tr>
<td>Configuring the VCS System</td>
<td>145</td>
</tr>
<tr>
<td>Using Associate Directories</td>
<td>146</td>
</tr>
<tr>
<td>Understanding Setup Verification</td>
<td>147</td>
</tr>
<tr>
<td>Using the VCS Interfaces</td>
<td>148</td>
</tr>
<tr>
<td>Using The File Open And Close Interface</td>
<td>148</td>
</tr>
<tr>
<td>Difference List</td>
<td>150</td>
</tr>
<tr>
<td>Errors</td>
<td>150</td>
</tr>
<tr>
<td>Supported VCS Aliases (or Metacommands)</td>
<td>150</td>
</tr>
<tr>
<td>Supported Version Control Systems</td>
<td>151</td>
</tr>
<tr>
<td>Using Projects</td>
<td>152</td>
</tr>
<tr>
<td>What are Projects</td>
<td>152</td>
</tr>
<tr>
<td>Creating Projects</td>
<td>152</td>
</tr>
<tr>
<td>Using the Project View</td>
<td>153</td>
</tr>
<tr>
<td>FTP</td>
<td>154</td>
</tr>
<tr>
<td>Sessions</td>
<td>155</td>
</tr>
<tr>
<td>What are Sessions?</td>
<td>155</td>
</tr>
<tr>
<td>Protecting Sessions</td>
<td>156</td>
</tr>
<tr>
<td>Session Manager</td>
<td>156</td>
</tr>
<tr>
<td>WebLair – HTML Editing</td>
<td>156</td>
</tr>
<tr>
<td>Internet Programming Support</td>
<td>156</td>
</tr>
<tr>
<td>Embedded Language Support</td>
<td>156</td>
</tr>
<tr>
<td>What are Markup Language Tag Databases?</td>
<td>157</td>
</tr>
<tr>
<td>Using the HTML Formatter</td>
<td>157</td>
</tr>
<tr>
<td>Common Code Manager</td>
<td>158</td>
</tr>
<tr>
<td>Editing HTML Tags</td>
<td>158</td>
</tr>
<tr>
<td>Configuring a Browser</td>
<td>158</td>
</tr>
<tr>
<td>Automating Tasks Using Macros</td>
<td>159</td>
</tr>
<tr>
<td>Recorded Macros</td>
<td>159</td>
</tr>
<tr>
<td>Record</td>
<td>160</td>
</tr>
<tr>
<td>Repeat</td>
<td>161</td>
</tr>
<tr>
<td>A Brief intro to CMac</td>
<td>161</td>
</tr>
<tr>
<td>Other Useful Tools</td>
<td>162</td>
</tr>
<tr>
<td>Spell Check</td>
<td>162</td>
</tr>
<tr>
<td>ASCII Table</td>
<td>163</td>
</tr>
<tr>
<td>Create HTML from Code</td>
<td>163</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Calculator</td>
<td>164</td>
</tr>
<tr>
<td>Notebook</td>
<td>166</td>
</tr>
<tr>
<td>Linedraw</td>
<td>167</td>
</tr>
<tr>
<td>System View</td>
<td>168</td>
</tr>
<tr>
<td>Metacommands</td>
<td>169</td>
</tr>
<tr>
<td>What are Metacommands?</td>
<td>169</td>
</tr>
<tr>
<td>Long Filename Metacommands</td>
<td>169</td>
</tr>
<tr>
<td>General Command Line Metacommands</td>
<td>170</td>
</tr>
<tr>
<td>Execute Program Metacommands</td>
<td>173</td>
</tr>
<tr>
<td>Printing Metacommands</td>
<td>174</td>
</tr>
<tr>
<td>Compiler Metacommands</td>
<td>174</td>
</tr>
<tr>
<td>VCS Metacommands</td>
<td>175</td>
</tr>
<tr>
<td>Help File Metacommands</td>
<td>176</td>
</tr>
<tr>
<td>Template Metacommands</td>
<td>177</td>
</tr>
<tr>
<td>Modifying Multi-Edit Startup</td>
<td>181</td>
</tr>
<tr>
<td>Command Line Switches</td>
<td>181</td>
</tr>
<tr>
<td>What are Command Line Switches?</td>
<td>181</td>
</tr>
<tr>
<td>No Restore - /NR</td>
<td>181</td>
</tr>
<tr>
<td>Read Only - /RO</td>
<td>181</td>
</tr>
<tr>
<td>Read/Write - /RW</td>
<td>182</td>
</tr>
<tr>
<td>No Splash - /NoSplash</td>
<td>182</td>
</tr>
<tr>
<td>No Restore, No Save - /NS</td>
<td>182</td>
</tr>
<tr>
<td>Run A Macro - /R[macro_name]</td>
<td>182</td>
</tr>
<tr>
<td>Display Session Manager - /SM</td>
<td>183</td>
</tr>
<tr>
<td>Start A Session - /SN[session_name]</td>
<td>183</td>
</tr>
<tr>
<td>Start The Last Session - /SR</td>
<td>183</td>
</tr>
<tr>
<td>Goto Line n - /L[n]</td>
<td>184</td>
</tr>
<tr>
<td>Goto Column n - /C[n]</td>
<td>184</td>
</tr>
<tr>
<td>Bypass VCS - /NV</td>
<td>184</td>
</tr>
<tr>
<td>Load A File – [filename]</td>
<td>184</td>
</tr>
<tr>
<td>Load A DOS File - /*[filename]</td>
<td>185</td>
</tr>
<tr>
<td>Load A Unix File - /!/filename</td>
<td>185</td>
</tr>
<tr>
<td>Load A Binary File - /&amp;[filename]</td>
<td>185</td>
</tr>
<tr>
<td>Multiple Instances - /NI</td>
<td>186</td>
</tr>
<tr>
<td>Startup.cfg and Startup2.cfg Files</td>
<td>186</td>
</tr>
<tr>
<td>The Update Macro Processor</td>
<td>187</td>
</tr>
<tr>
<td>Update Script Reference</td>
<td>187</td>
</tr>
<tr>
<td>Line Comments</td>
<td>188</td>
</tr>
<tr>
<td>Block Comments</td>
<td>188</td>
</tr>
<tr>
<td>Conditionals</td>
<td>188</td>
</tr>
<tr>
<td>General Commands</td>
<td>190</td>
</tr>
<tr>
<td>Data Commands</td>
<td>192</td>
</tr>
<tr>
<td>Set Commands</td>
<td>197</td>
</tr>
<tr>
<td>Add-On Packages Installation Files</td>
<td>204</td>
</tr>
<tr>
<td>Integrating External Applications</td>
<td>206</td>
</tr>
<tr>
<td>DDE</td>
<td>206</td>
</tr>
<tr>
<td>COM</td>
<td>210</td>
</tr>
</tbody>
</table>
External Application Integration

Borland Integration ................................................................................................... 213
  Borland ........................................................................................................ 213
  Delphi 1.0 ................................................................................................. 213
  Delphi and C++ Builder ............................................................................... 215
Microsoft Integration ................................................................................................. 215
  Microsoft ...................................................................................................... 215
  DevStudio 6.0 .......................................................................................... 216
  Visual Basic 6.0 ........................................................................................ 216
BradSoft..................................................................................................................... 217
  Top Style..................................................................................................... 217
Scooter Software’s.................................................................................................... 218
  Beyond Compare .......................................................................................... 218
AI Internet Solutions.................................................................................................... 219
  CSE HTML Validator ................................................................................ 219

Add-Ons

Polystyle ..................................................................................................................... 221
  About Polystyle .......................................................................................... 221
  Installing and Updating Polystyle ............................................................... 221
  Customizing Polystyle to Suit Your Coding Style ...................................... 221
  Formatting Files ............................................................................................ 222
  Getting Support for Polystyle ...................................................................... 222
Evolve ........................................................................................................................ 223
  Introduction .................................................................................................. 223
  Language Dialect Support .......................................................................... 223
  Configuration ............................................................................................... 224
  Multi-Edit's Language Style ....................................................................... 224
  Setup ............................................................................................................ 225
  Template Expansion API .............................................................................. 232
  Commenting Support .................................................................................... 236
  Same-Line-Comment .................................................................................... 236
  Block Style Comments ................................................................................ 236
  Function Header Comment ........................................................................... 237
  Procedure Header Comment ....................................................................... 237
  Class Header Comment ................................................................................ 237
  Program Header Comment .......................................................................... 237
  #INCLUDE Header Comment ..................................................................... 238
  Using Programmable Comment Headers .................................................... 238
  User Definable Meta Commands .................................................................... 238
  No Information For A Meta Command ........................................................ 239
  Multiple Function Arguments ..................................................................... 240
  Cursor Positioning With {Entry} ................................................................. 240
  Cursor Positioning With {Field} ................................................................. 240
  Prototype Comment Table ........................................................................... 241
  Active Productivity and Template Editing .................................................. 242
  Template Files ............................................................................................. 242
  Control Types .............................................................................................. 243
  Template Expansion .................................................................................... 243
Multi-Edit 9.10

What's New?

Perl Regular Expression Support
Multi-Edit’s already powerful search and replace functionality has been enhanced with the introduction of Perl 5.8 regular expressions support. Use the latest and most powerful programming equations to date when searching cross folders, files and projects.

Template Preview
Templates can now be easily previewed in the Preview pane by selecting the "Preview" entry from the context menu of the TmplPane dialog.

Export Functionality
Basic support for exporting configuration settings has been added to the Tools | Customize dialog.

VCS (Subversion)
Added basic Subversion version control support.

End of line processing
Changed processing of line ending characters to better support editing of DOS (CrLf), UNIX (Lf) and Mac (Cr) formatted files. Files with multiple line ending characters will be correctly handled.

Many new navigation panes
- Templates - Shows two sets of template lists usually the template set for the current file and the global template set for normal language files.
- File List - A user defined list of commonly used files to allow quick loading into Multi-Edit.
- Window List - A list of all loaded windows. This pane was released as an Add-On on our site but it has been updated to allow all the functions of the mail Window List dialog.

Resizeable Dialogs
Dialogs can now be created as well as resized.
**Beyond Compare integration**

Allows Beyond Compare to be launched when comparing files from within Multi-Edit.

**Tagging Support**

Full function tagging support for source files of the following languages: Ada, C, C++, C#, JavaScript, Perl, PL_SQL, Progress, Python, SciLab, TCL/TK, MatLab, Modula-2, VBasic, VBScript, Verilog, Vhdl, CMac, xBase, Fortran, Java, Lisp, Pasal, PHP, Rebol and more.

**Code formatting via Polystyle**

A fully registered copy of the Polystyle source code formatter is included and will format the following languages.

C#, CSS, Java, JavaScript, JSP, HTML, PHP, XML, Python, Cmac, Perl, Delphi and C++.

**Support for Ant, NAnt, and Want, build environments**

Adds compiler entries for Want/Ant/NAnt programs and provides minimal template support.

**C/C++ language support**

CFindFunction macro completely rewritten and should work with any reasonably written code. Reports correct context information for functions, classes, structs, unions, enums and namespaces.

"Align continuation lines to opened braces" option added to Formatting/Coding Properties Setup dialog. When enabled, it aligns code as follows:

```
ALongFunctionCall(Param1, Param2, Param3, Param4, Param5);
```

Added function header parsers which allow writing macros and templates parametrized by real functions. As an example there is a macro which creates Doxygen comment block and a macro which creates TDBG traces displaying all function parameters on enter and return value on leave.

**Updated BSC support**

Support added for using VS.NET compiler.

**Improved TipWin support**

- New tooltip windows used for displaying function definitions.
- Multi-line, highlight current parameter, ability to use colors, and configurable.
- Generic language support allowing you to add new languages. Currently C/C++, CMAC and Delphi/Pascal are supported. MeTags, BSC and CMAC kernel function list are used as info source and it is possible to add new ones via hooks.

**Automatic function templates**

This feature allows automatically expanding function parameters as a template when open parameter parenthesis is written. Currently configured via TipWin configuration dialog (TipWin^TipWinGlobalConfig). Supports the same set of languages as TipWin.
Macro Debugging

DebugLog macro uses standard Win32 debug output for log number 0 and GExpert Debug Window application for log number 1.

Added TDBG trace library (see tdbg.sh for details) which allows you to write code without conditional compilation directives and configure traces via dialog whenever necessary. Supports traces from external DLLs.

Updated Telnet support

Added a McTerm program to handle all telnet compiling. Telnet server information has been moved into its own database so a compiler entry no longer needs to be tied to a single telnet server. Telnet configurations now support scripts as well.

Microsoft Integration

Added an option to allow all loaded files or only the current file in the Microsoft IDE to be synchronized in Multi-Edit.

Borland Integration

Added an option to allow all loaded files or only the current file in the Borland IDE to be synchronized in Multi-Edit.

Evolve

The Evolve Add-On is now included with Multi-Edit and provides enhanced support for Arago for DOS, dBASE III Plus and IV, dBASE V For DOS, dBASE For Windows, FoxPro For DOS, FoxPro For Windows, CA-Visual Objects and dBFast For Windows.

New Updated Help

There is a new updated version of Multi-Edit's Help in HTML Help (.Chm) form. This will require IE 4.0 or greater and the HTML Help engine to be installed. The Help Engine will be automatically updated for you from the install. This is all pre-installed if you have Windows 98 or later.
Getting Started

Multi-Edit

Multi-Edit is a programmer's text editor with powerful features designed to deliver the ease of operation and timesaving functions you need to meet demanding deadlines.

With Multi-Edit, you can manipulate text with unsurpassed ease and compile source files while you are still within Multi-Edit. Your ability to handle files is greatly enhanced. Up to 256 files can be edited simultaneously, and Multi-Edit 9.10 effortlessly handles large files with line lengths up to 16K!

You will not spend a lot of time getting up to speed with Multi-Edit 9.10 either. You will be doing productive work very quickly with the aid of our intuitive user interface. Drop down menus and special Key Assignments are available that will help you execute commands in a heartbeat. Plus, features like our smart indent, template editing, and construct matching make our language support second to none. And Multi-Edit's Help System and Technical Support guarantees you will never encounter a problem without a solution near at hand.

What's Installed with Multi-Edit 9.10

This topic provides reference information about where you can find specific components in the Multi-Edit set up, and what is installed.

In order for Help to work, Multi-Edit will install the latest HTML Help controls if they are not currently installed. In addition, it may be necessary to install Internet Explorer version 4 or greater.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Default Install</th>
<th>Location in Setup</th>
<th>What's installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Files</td>
<td>Yes</td>
<td>Program Files</td>
<td>Executable files, libraries, and system macro binaries (*.mac, mew.mcl) necessary to run the Multi-Edit.</td>
</tr>
<tr>
<td>Program Help</td>
<td>Yes</td>
<td>Program Help</td>
<td>Main program help (Me.chm)</td>
</tr>
<tr>
<td>Feature</td>
<td>Default Install</td>
<td>Location in Setup</td>
<td>What's installed</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------</td>
<td>----------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>Macro Compiler</td>
<td>No</td>
<td>Macro Compiler, Headers, and Help</td>
<td>Cmacwin.exe</td>
</tr>
<tr>
<td>Macro Headers</td>
<td>No</td>
<td>Macro Compiler, Headers, and Help</td>
<td>System macro headers (*.sh)</td>
</tr>
<tr>
<td>Macro Help</td>
<td>No</td>
<td>Macro Compiler, Headers, and Help</td>
<td>Macro Help (Cmac.chm)</td>
</tr>
<tr>
<td>Macro Source</td>
<td>No</td>
<td>Macro Source</td>
<td>System macro source files (*.s)</td>
</tr>
<tr>
<td>Html Reference</td>
<td>No</td>
<td>HTML Reference Help</td>
<td>HtmlRef.chm</td>
</tr>
<tr>
<td>Cold Fusion/Homesite Integration</td>
<td>No</td>
<td>Macromedia Integration</td>
<td>Library and source files necessary to integrate. (Macromedia folder)</td>
</tr>
<tr>
<td>Delphi Integration</td>
<td>No</td>
<td>Borland Integration</td>
<td>Options</td>
</tr>
<tr>
<td>C++ Builder Integration</td>
<td>No</td>
<td>Borland Integration</td>
<td>Options</td>
</tr>
<tr>
<td>Visual Basic Integration</td>
<td>No</td>
<td>Microsoft Integration</td>
<td>Options</td>
</tr>
<tr>
<td>Visual C++ Integration</td>
<td>No</td>
<td>Microsoft Integration</td>
<td>Options</td>
</tr>
<tr>
<td>Visual C++ Browse Support</td>
<td>No</td>
<td>Microsoft Integration</td>
<td>Options</td>
</tr>
<tr>
<td>Watcom C++ Integration</td>
<td>No</td>
<td>Watcom Integration</td>
<td>Library files and source files necessary to integrate. (Watcom folder)</td>
</tr>
<tr>
<td>Feature</td>
<td>Default Install</td>
<td>Location in Setup</td>
<td>What's installed</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------</td>
<td>-------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>Oracle Integration</td>
<td>No</td>
<td>Oracle Integration</td>
<td>Library files and source files necessary to integrate. (PL_Sql folder)</td>
</tr>
<tr>
<td>TopStyle Integration</td>
<td>Yes</td>
<td>Program Files</td>
<td>Library files and source files necessary to integrate. (TopStyle folder)</td>
</tr>
<tr>
<td>CSE HTML Validator Integration</td>
<td>Yes, if CSE validator installed</td>
<td>Program Files</td>
<td>Library files and source files necessary to integrate. (CSEValidator folder)</td>
</tr>
</tbody>
</table>

**Upgrading From Previous Versions**

**Upgrading Existing Multi-Edit Installations**

All upgrade installations require Multi-Edit 9.10 to be installed in a different directory than the existing install. However, Multi-Edit 9.10 can keep your customizations from all Multi-Edit products by copying your configuration files to the new installation directory and updating them with the latest changes.

Components not updated
- Any updated system macro source will not be copied and updated. This will have to be done manually.
- Language templates are not copied and updated.
- Session files are not copied and updated.

> If your session directory in your previous version is an absolute path and you update your configuration, Multi-Edit 9.10 will continue to use that path. It is better to base your session directory on a Metacommand.

- New CMAC Macros are not copied.

> If you are updating from a Multi-Edit for DOS product you will need to re-enter any changes you have made to your Multi-Edit for DOS keymaps.

**Finding Multi-Edit 9 Menus in Multi-Edit 9.10**

**File menu**

<table>
<thead>
<tr>
<th>Multi-Edit 9 command</th>
<th>Changes to this in Multi-Edit 9.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>Renamed Properties</td>
</tr>
</tbody>
</table>
## Block menu

<table>
<thead>
<tr>
<th>Multi-Edit 9 command</th>
<th>Changes to this in Multi-Edit 9.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy Block</td>
<td>Removed from menu. You can use Cut/Paste as default.</td>
</tr>
<tr>
<td>Move Block</td>
<td>Removed from menu. You can use Cut/Paste as default.</td>
</tr>
<tr>
<td>Delete Block</td>
<td>Moved to the **Edit</td>
</tr>
<tr>
<td>Block Operations</td>
<td>Moved to the Edit menu.</td>
</tr>
<tr>
<td>Save Block to Disk</td>
<td>Moved to the File menu</td>
</tr>
<tr>
<td>Indent</td>
<td>Moved to the **Edit</td>
</tr>
<tr>
<td>Undent</td>
<td>Moved to the **Edit</td>
</tr>
<tr>
<td>Math Operation</td>
<td>Moved to the Edit menu.</td>
</tr>
<tr>
<td>Window Copy</td>
<td>Removed. Use the Cut/Paste combination by default. Can still be used if enabled in command map.</td>
</tr>
<tr>
<td>Window Move</td>
<td>Removed. Use the Cut/Paste combination by default. Can still be used if enabled in the command map.</td>
</tr>
<tr>
<td>Mark Lines of Text</td>
<td>Moved to Edit menu.</td>
</tr>
<tr>
<td>Mark Columns of Text</td>
<td>Moved to Edit menu.</td>
</tr>
<tr>
<td>Mark Streams of Text</td>
<td>Moved to Edit menu.</td>
</tr>
<tr>
<td>End Marking</td>
<td>Moved to Edit menu.</td>
</tr>
<tr>
<td>Turn Marking Off</td>
<td>Moved to Edit menu.</td>
</tr>
<tr>
<td>Persistent Blocks</td>
<td>Moved to Edit menu.</td>
</tr>
</tbody>
</table>

## Project menu

<table>
<thead>
<tr>
<th>Multi-Edit 9 command</th>
<th>Changes to this in Multi-Edit 9.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Removed. Replaced by using either Open Project or Select Project</td>
</tr>
<tr>
<td>View</td>
<td>Moved to View menu, item Project</td>
</tr>
<tr>
<td>Options</td>
<td>Renamed to Properties</td>
</tr>
</tbody>
</table>
Macro menu

<table>
<thead>
<tr>
<th>Multi-Edit 9 command</th>
<th>Changes to this in Multi-Edit 9.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>List all Globals</td>
<td>Consolidated in the View</td>
</tr>
<tr>
<td>List all Macros</td>
<td>Consolidated in the View</td>
</tr>
</tbody>
</table>

Tags menu

<table>
<thead>
<tr>
<th>Multi-Edit 9 command</th>
<th>Changes to this in Multi-Edit 9.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>List tags</td>
<td>Consolidated in the View</td>
</tr>
<tr>
<td>Browse current file</td>
<td>Consolidated in the View</td>
</tr>
</tbody>
</table>

New Menu Commands in Multi-Edit 9.10

The following summarizes the commands that are new in Multi-Edit 9.10 from Multi-Edit 9.

Edit menu

<table>
<thead>
<tr>
<th>This Multi-Edit 9.10 command</th>
<th>Allows you to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Operations</td>
<td>Align Operators</td>
</tr>
</tbody>
</table>

View menu

<table>
<thead>
<tr>
<th>This Multi-Edit 9.10 command</th>
<th>Allows you to</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Menu</td>
<td>Brings up the appropriate view</td>
</tr>
</tbody>
</table>

Project menu

<table>
<thead>
<tr>
<th>This Multi-Edit 9.10 command</th>
<th>Allows you to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Project</td>
<td>Open a project file (*.mep)</td>
</tr>
<tr>
<td>Create Project</td>
<td>Create a new project file</td>
</tr>
<tr>
<td>Select Project</td>
<td>Select a project to open from a quick list</td>
</tr>
<tr>
<td>Close Project</td>
<td>Closes the current project</td>
</tr>
<tr>
<td>Notebook</td>
<td>Open the project specific notebook</td>
</tr>
</tbody>
</table>
Tools menu

<table>
<thead>
<tr>
<th>This Multi-Edit 9.10 command</th>
<th>Allows you to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create HTML from Code</td>
<td>Takes the current file and generates a syntax highlighted version in HTML.</td>
</tr>
</tbody>
</table>

Help menu

<table>
<thead>
<tr>
<th>This Multi-Edit 9.10 command</th>
<th>Allows you to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Map Report</td>
<td>Detailed report of all Keymap, Toolbar and Menu entries.</td>
</tr>
<tr>
<td>Tip of the Day</td>
<td>A random tip about Multi-Edit.</td>
</tr>
</tbody>
</table>

Installation Requirements (Hardware/Software)

Multi-Edit 9.10 requires Microsoft Windows 9x/2000/NT 4.0/Me/XP or greater to run. To use FTP features in Multi-Edit, you must have Microsoft Internet Explorer 4.0 or greater (or Wininet extensions) installed.

Multi-Edit 9.10 can run on an IBM-compatible 486 or Pentium computer with a minimum of 16 MB of available memory. Disk space requirements are approximately 18 MB (assuming you install everything). Intellimouse support requires that you have the Microsoft Intellimouse.

Add-On Integration

Several options designed to work seamlessly with Multi-Edit have been developed to substantially enhance productivity. Here is a list of the currently available Add-On options:

- Borland Delphi/C++ Builder Integration - Drop-in replacement for Borland's Delphi editor. Adds transparent synchronizing for Delphi. This option comes automatically with Multi-Edit but will need to be selected for installation when installing Multi-Edit.
- Sybase Watcom C/C++ Integration - Provides all hooks into Watcom IDEs. Supports 10-11 C and Fortran.
- Macromedia ColdFusion Studio 5 and Homesite 5 IDE integration - When installed, files are synchronized when editing in either of these IDEs and Multi-Edit at the same time.
- Microsoft DevStudio and Visual Basic IDE integration - When installed, files are synchronized when editing in either of these IDEs and Multi-Edit at the same time.
- Bradsoft's Top Style - TopStyle helps you create cross-browser style sheets by alerting you of problems as you work. What's more, TopStyle's powerful style checker validates against multiple CSS implementations, alerting you not only to invalid entries in your style sheets, but also to bugs in popular browsers that may affect their display.
- AI Internet Solutions' CSE HTML Validator Integration - When this Add-On is installed, HTML files may be validated using a user-installed copy of CSE HTML Validator. Please review the HTML Validator topic for more information.
Key / Command Assignments

The "default" command map in Multi-Edit uses CUA-Style mapping. There are also additional command maps available for use within Multi-Edit and these include Brief, WordStar, Borland IDE, and Visual Studio IDE.

Command Maps can be modified or created from scratch. For more information on creating command maps, please refer to "What is a Command Map."

To view the current key assignments click on HELP | COMMAND MAP REPORT.

Multi User Configurations

Installation Overview

This section tells you everything you need to know to install Multi-Edit 9.10, whether you have acquired a single-user Stand-Alone license or a multi-seat network license. The installation procedure is remarkably similar for the two types. In fact, should you have any reason for doing so, you could install a single-user copy to run networked, exactly the way a multi-seat version does, save that you can only run one seat. The Configuration section below explains this in detail.

Multi-Edit 9.10 has only a few requirements, but you should verify that you can meet them before beginning installation. The section titled “Things to Do Before Installing Multi-Edit” gives a detailed walk-through of all pre-installation issues.

The section titled “Installing Multi-Edit” gives a similarly detailed walk-through of the actual installation, broken into two sections: installing a Stand-Alone copy or the server-side copy of Multi-Edit, and installing the client software for users. The second of these sections is only necessary for an installation that runs across a network.

The section titled “Installing on a NetWare Server” gives ad-hoc directions for getting the networked version of Multi-Edit 9.10 working in a Novell environment.

Configuration

There are two basic configurations: Stand-Alone and networked. The sole difference between them is not, as you might surmise, the number of seats licensed, but whether you set Multi-Edit 9.10 up to run as a local client or as a client/server application. This differs somewhat from the way in which previous versions of Multi-Edit worked. The salient features of the two Multi-Edit 9.10 set ups are as follows:

- A local client version installs to a local directory on the same machine it will be run from, and its various configuration files are by default located in subdirectories of the installation directory, requiring the user to have Read, Execute, and Modify access rights to several of these locations;
- A client/server version installs on a file server in a similar manner, but requires a directory, called the Access File Path, to contain the Network Access File. In addition, each user client installed must have a configuration directory (which can be anywhere the user has full access rights) and an environment variable, ME91_CFG_DIR, set to the configuration directory path. This mechanism allows each user...
Getting Started Multi-Edit 9.10

complete control over every aspect of Multi-Edit’s configuration and customization, while requiring only one installed server executable. Each user’s personal configuration, including macros, templates, keymaps—any and all possible customization—is stored in his/her personal configuration directory, and takes precedence over its counterpart in the server installation.

The determining factor in which kind of installation occurs, is simply the existence of the ME91_CFG_DIR environment variable. If it isn’t defined, then you’re performing either a Stand-Alone or server-side installation; else, a user client installation. You should not define it for your installation account before installing the server portion of a networked installation, or all you’ll end up with is a single-user Stand-Alone copy. In addition, there are two internal path metacommands you can use after installation that assume different values, depending on the type of installation.

See the examples below:

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Value in Stand-Alone</th>
<th>Value in Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;USER_PATH&gt;</td>
<td>&lt;ME_PATH&gt;</td>
<td>ME91_CFG_DIR</td>
</tr>
<tr>
<td>&lt;MAC_PATH&gt;</td>
<td>&lt;ME_PATH&gt;Mac</td>
<td>&lt;USER_PATH&gt;Mac</td>
</tr>
</tbody>
</table>

**Things to Do Before Installing Multi-Edit**

1. Make sure you have enough disk space in the location where you are going to install Multi-Edit. For a Stand-Alone or server-side installation, you will need approximately 28 MBytes, when all components are selected for installation. Bear in mind that for a networked installation, you must install all components on the server that your users might need, so each configuration will have available the components necessary for customization.

2. Verify that you have available the Serial Number and Release Code issued with your purchase. Should you lose this, you will be unable to install Multi-Edit 9.10 until you contact our Sales Department (sales@multieditsoftware.com) and arrange for it to be regenerated.

3. Important note: if you have an existing installation of a previous version of Multi-Edit, install Multi-Edit 9.10 to a different directory! Installing over the old copy can cause problems. If you want the new version in the old directory, simply rename the old directory before installation. Should you wish, you will have the opportunity to import the settings from any existing installation. It is very important that you use the installer to perform a full install. Simply copying old configuration files over the new ones can easily lead to corrupt files, and depending on the age of the version, you are copying from, important run-time files may not be updated correctly. If you want to make sure you are installing to a clean environment, make a backup copy of all your old Multi-Edit–related files wherever they are located, including the old Multi-Edit installation directory, then uninstall the old copy and reboot your machine. Except in very unusual circumstances, this will eliminate all trace of the old program, including all registry entries other than file class entries. Since you made a backup of the original directory, you can still import its settings into Multi-Edit 9.10; an installed copy of the old version is not necessary for this process.

4. Log on to Windows using an account with full rights. On a Windows NT/2K/XP system, this account should be a member of the Administrators group.
5. If you are doing a network installation, there is a bit more to prepare. If you want, you can defer steps (b) and (c) until you are ready to install your user clients.
   a) First, create a directory on the server. It does not matter where it is located or what the name is, so long as all users have Read and Write access rights to it.

   This will allow Multi-Edit’s network licensing to function. Later on, when the installer asks for your Access File Path, the path to this directory is what you will enter.

   b) Second, create a configuration directory for each user. Again, it does not matter where it is located or the name, so long as the user has full access rights to the directory. Make sure that each user has adequate disk space in which he or she will keep his personal configuration files. Potentially this can include not only all of the configuration files, but personalized copies of the Multi-Edit macro source, header, and compiled files, as well as portions of the Add-Ons that he or she intend to install. Since this is also likely to be where temporary files will be stored, and possible work files, you will have to estimate how much space this might take. Again, 28 MBytes is an upper bound, but most users are not likely to come close to that, unless they save work files there.

   c) Third, create the environment variable ME91_CFG_DIR for each user, setting it to the path of that user’s configuration directory.

   **Important:** Do not create this variable for your own account until after you have completed the server installation. In essence, you have created a Stand-Alone installation accessible by your installation account; thus you might prefer to have two accounts: the installation account, from which you can run Multi-Edit 9.10 from the server’s console; and a separate user account, which will be exactly like any other user client account.

---

**Stand-Alone and Server Installation**

1. After completing the preparations described in the previous section, run the installer. Remember, you must have Administrative rights for this installation, the Access File Path must exist and be accessible to your account, and you should have no ME91_CFG_DIR defined for the account logged in.

2. The first thing you will see is the welcome screen; if you haven’t read it before, do so, then click on the Next button.

   It is a good idea, as the screen cautions, to be running as few other programs and processes as possible, preferably start the process just after rebooting. This will minimize the chance of another process locking a file that the installer will have to modify, and it will make available as much contiguous free memory as possible (Incidentally, this is good advice when installing any program.)

3. The next dialog displays Multi-Edit’s license agreement. When you are satisfied you understand and agree to the terms set forth, check the box titled “I Agree to the terms of the License Agreement” and click on the Next button.

4. The Registration Validation dialog that follows is where you will place your Serial Number and Release Code.

   If you have not purchased Multi-Edit 9.10, but are interested in evaluating a time-limited but fully functional copy, click on the blue hotlink “Request Evaluation Code” immediately above the first textbox to request your code. This will unlock Multi-Edit in evaluation mode. (Step 4 continued)
Fill in the first four fields appropriately.

For a Stand-Alone installation, leave the “Net Access File Path” field blank.

When you have filled the dialog in appropriately, click on the Next button.

The hotlink above the first textbox, lets you request a time-limited evaluation code. If this is the server portion of a network client/server installation, supply the path to the Access File Path directory you created.

You must give this path in UNC format.

5. The next dialog proposes a default installation path. If you want to install Multi-Edit 9.10 to a different location, click on the Browse button to display a dialog that allows you to specify any currently reachable location. Once you are satisfied with the install location, click on the Next button.

6. The installer now gives you the opportunity to import configuration settings from an already-installed copy of Multi-Edit. Simply click on the appropriate radio-button, then click on the Next button. Note: Although the dialog currently says that template files will not be copied, in fact, they will be if they are located in the <ME_PATH>Config directory of the existing installation.

If you chose not to copy a prior configuration, skip to Step 8 below.

7. If you have chosen to copy a previously existing configuration in Step 6, you will now see a dialog with instructions and a Browse button. That button displays a dialog that lets you navigate to any reachable and existing installation of Multi-Edit.

As noted, this can be any version 7, 8, or 9 for Windows. Once you have located the base directory of that installation, click on the Next button.

If you change your mind or cannot locate any such installation, click on Cancel to proceed without copying a configuration.

8. The following dialog (whether you arrived via Step 6 or Step 7) is where you specify which components you want installed.

First, you should always select both Program Files and Program Help; any other choices are optional, but some are always advisable.

If you do Web work, you will certainly like the HTML Reference Help, and possibly one or more of the Web tool integrations.

If you work in any of the languages the Trita formatter supports, you will want to give this handy formatter a spin.

The Evolve support is a big win for XBase programmers, and a full version of it, as with Trita, comes with Multi-Edit 9.10.

The three items that include the word Add-On are actually new portions of Multi-Edit 9.10, supplying new functionality. Do not pass them by thinking they will not relate to your work.
Important: If you are performing the server portion of a network install, you must include all components that any of your users may want, or they will be unable to install them into their local clients later on. This is because the client installers draw their files from the server installation.

We strongly recommend that you install the macro source and compiler. Not only is it very likely that one of your users will want to write macros at some point, but we occasionally issue bug-fixes, some of which you can easily apply to the source files yourself, rather than wait for the next checked release.

Next, the checkbox near the bottom of the dialog gives you the choice of only updating program shortcuts and registry entries, but not actually installing any files; this is useful for restoring corrupted items to their default states.

Once all is to your liking on this dialog, click on the Next button to continue.

9. The next dialog lets you specify where in Windows’ Start Menu you want your Multi-Edit shortcuts. If you leave the “Add to NT/2K/XP Common Shortcuts” checkbox unchecked, each user will have a personal set; check it, and all users will share the same shortcuts. Again, click Next to pass on.

10. At this point, you will see a dialog listing the install options you have chosen, giving you the option of clicking Back to return to the previous dialog (see Step 9) to make changes, or Next to begin copying files.

11. The installer displays a progress bar while copying files. Upon a successful completion, a dialog appears from which you can check for any more-recent updates, or click on the Finish button to complete the installation.

To check for updates, you need an active Internet connection. If you don’t want to check at this point, you can always do so later, either from the link in Multi-Edit’s Help menu, or by browsing our web site and looking in the Download area. ( http://www.multieditsoftware.com/downloads )

12. At this point, Multi-Edit will run its update scripts, which display the next few dialogs you’ll see.

The first dialog asks you to choose a command map from several predefined ones. These files contain maps that bind specific Multi-Edit functionality to a particular key combination, as well as a few other related items.

Unless you have a predilection for, say, Brief or WordStar key commands, or have a prebuilt command map of your own, your best choice is the Default map.

Note that if your command map is a part of an installation you copied the configuration from, this task is done; simply select the predefined command map you based your own version on originally.

To use a command map of your own devise, click on the Insert button.

You can also edit any displayed command map by clicking on Edit.

When you have clicked on your preferred map to highlight it, click the Select button to continue.
13. If, in Step 8, you selected any components that require configuration, such as Microsoft Support, you will see a sequence of dialogs. Each dialog is different for each component and this will allow you to configure them.

If you are installing a Stand-Alone, fill out all the required configuration information.

If you are doing a networked install, do not bypass any of these menus by pressing Cancel; instead, enter each one and click on whatever the equivalent to an OK button is in all of them. You may have to click through some dialogs that complain that you have not configured something, but ignore these; all that is important for a server installation is that the files are copied. Of course, if you happen to have the necessary programs installed on the server, you can certainly fill in the required information.

14. After any component configuration that may be required, you will reach a dialog titled “Configuration Update.” Unless you’ve reached this point while trying to restore a default configuration where you have corrupted files, always answer by clicking the Yes(full) button.

This is very important, as failing to do so may mean that some key files necessary for Multi-Edit’s correct operation may not be updated correctly.

15. At this point Multi-Edit displays its user interface, with the “Tip of the Day” dialog in the foreground, and some readme files loaded (and yes, you should certainly read them).

You can, if you so choose, skip this registration.

We do recommend, however that you take the time to fill it out and submit it. It not only gives you an easy way to get on our private mailing list which will notify you of future updates, site changes, and special offers, but it will also activate your support account.

If you were installing a Stand-Alone, you are done at this point, and ready to write that killer app you have been planning.

If you were installing the server portion of a networked installation, you now need to install your users’ client software so they can access the server installation remotely using their own configurations.

You no longer require full access rights, merely Read and Execute, to the server copy, unless you want to modify its default configuration.

**User Client Installation (Network Install Only)**

1. For each user, if you have not already done so, create, or have him create, a configuration directory. This directory can be called anything legal within the filesystem, and can be located anywhere, so long as it is always accessible while the user is running Multi-Edit.

The user must have full access rights to this directory, as well as Read and Modify rights to the Access File Path directory.

2. Next, create an environment variable called ME91_CFG_DIR for each user, set to the path of his or her configuration directory. The easiest way to do this is to add it to the user’s login script. Under Windows NT/2K/XP, you can also add it to either the User or System variables section of the Environment variables section of the Advanced tab of the System Properties dialog.
(User for different individual user paths, System for one common path for all users). You will require Administrator access to set variables in this manner. Under Windows 95/98/ME, you can also set the variables in the AutoExec.bat file instead of a login script, but this is not the preferred method.

3. Make sure no copy of Multi-Edit is running on the server or client machine, and have each user run the Client Install Program NetSetup.exe from the server directory into which you installed Multi-Edit.

The environment variables created in the previous step must exist during client installation, and there must be no copy of Multi-Edit running, or the client installation will fail. The installer will prompt you for the server directory containing the installation you created in the previous section, followed by options for that particular client.

Should the user wish to keep his configuration from an earlier version of Multi-Edit, he should copy the files from that installation’s <ME_PATH>Config directory into the new directory pointed to by his ME91_CFG_DIR environment variable before running the Client Install Program.

Once you have gotten this far, your users can log in and run the network installation of Multi-Edit using their own configurations, up to the user-count limit specified by your license.

It does not matter how many clients are installed; what matters is how many are active at any one time. Should a user try to run Multi-Edit at a time when all licensed seats are already active, Multi-Edit will display an error message. Once one of the other users shuts his client down, the seat becomes available, and the previously locked out user can then log on.

حذر: A user’s ME91_CFG_DIR must exist and point to his or her configuration directory each time he runs Multi-Edit.

### Installing on a NetWare Server

Multi-Edit 9.10 is primarily a Windows program at present, and the installer makes the assumption that the server you are installing to can execute a Windows program, and that you will be installing from the server console (a plus on a Windows server, as it gives the sysadmin an extra console login for use on the server itself.

This is a problem if you are in a NetWare environment. We are currently working on making the installation compatible, but in the meantime, you can still get running using a work-around technique.

First, log on to your NetWare server as Supervisor from a Windows client machine, and run the installation just as you would ordinarily. After you have entered all the information in the dialog for Step 8, things will go as expected until the installer attempts to write to the registry on the server. Since there is no such thing on a NetWare server, the installer will cause a kernel fault on the remote console, forcing a power-down and reboot. Before this occurs, however, all needed files will have been copied to the server installation directory. This is disturbing, but we have had no reports of any problems caused on the remote machine because of this process.

At this point, you can perform your user client installations exactly as described for the Windows environment with no further issues. The only functionality that will be unavailable will be the console copy of Multi-Edit, and the default set up values you would have entered later in the installation process. These default values are inherited by any clients you install, which simply means that all clients will install
Multi-Edit’s default values, instead of any customizations you might have created for your users. It’s still possible to create these files separately and install them on the server by hand, but there’s no substitute for the missing registry entries, unless you care to hand-craft registry files for your users to install.

Ordering Information

For all North American orders:

Multi Edit Software, Inc.
3370 North Hayden Road
#123, PMB 712
Scottsdale, AZ 85251
USA
Voice: (480) 610-2700
Fax: (480) 610-2714
Email: sales@multieditsoftware.com
World Wide Web: http://www.multieditsoftware.com/

For all inquiries pertaining to shipped copies of Multi-Edit outside North America, please see our website for a complete list of authorized international resellers.

License Agreement

Multi Edit Software and the Multi Edit logo are trademarks of Multi Edit Software, Inc. © 2003 Multi Edit Software, Inc. All rights reserved. Multi Edit software is furnished under license and may be used or copied only in accordance with the terms of such license.

End User License Agreement For Multi Edit Software

Section 1: Multi Edit Software, Inc. (hereinafter referred to as Mesi) grants to you a nonexclusive, nontransferable license to use Multi Edit Software (“Software”) only upon the condition that you accept all of the terms contained in this Software License Agreement. Your use of the Software is limited to the licensing options set forth on your invoice or receipt or as otherwise described in your original purchase and/or upgrade of Multi Edit Software.

End User License: Software provided to you under this option may only be installed and used on a single PC or CPU. You may neither network the Software, install it on a Server nor otherwise use it on more than one computer system at the same time. The Software may be transferred to and used on another computer used by you, so long as the Software is de-installed from the original computer, and under no circumstances is used on more than one computer at a time. As used herein, “Server” computer shall mean a data station that provides facilities to other stations in a computer network.

Network License: Software provided under this option may be installed on and processed by the Server computer located at a site indicated on the invoice or sales receipt, and may be installed on and simultaneously processed by the number of workstations and/or personal computers which have electronic access to the Server computer located at any site indicated on the invoice or sales receipt, limited to the
number of users as indicated on the invoice and or sales receipt. You may make one copy of the Software for
backup or archival purposes in support of your permitted use of the Software.

**Section 2:** You may not transfer the Software to another entity or person without specific written agreement
from MESI. Any such transfer terminates your license under this Agreement. You shall not modify,
translate, reverse assemble or reverse compile in whole or in part the Software that is provided in object code
only (the “Object Code”). You shall not rent, sell, time-share, lease, sublicense, transfer, publish, disclose,
display or otherwise make available the Object Code or copies thereof to others. You may not copy the
Object Code except as provided herein. You shall not tamper with, bypass or alter any security features of the
Software or attempt to do so.

**Section 3:** Multi-Edit 9.10 now makes use of the following open source libraries: TurboPower OnGuard and
TurboPower Internet Professional. The downloadable source code for these libraries can be found on the
Multi-Edit website at [http://www.multieditsoftware.com/downloads_win.htm](http://www.multieditsoftware.com/downloads_win.htm) under file names,
tpguard_1_13.zip and tpipro_1_15.zip.

**Section 4:** Polystyle Software is incorporated in its entirety within version 9.10 of Multi-Edit Software. Upon
your acceptance of the terms of this Agreement, you hereby agree that all protections extended to MESI as
pertains to your use of its Software shall also extend to Polystyle Software as pertains to the use of its
Software. A License Agreement specific to Polystyle Software can be found in the Polystyle root directory
within the Multi-Edit installation.

**Section 5:** This license is not a sale. Title, ownership rights, and intellectual property rights in and to the
Software and Documentation shall remain wholly vested in MESI. You agree to abide by the copyright laws
and all other applicable laws of the United States. You acknowledge that the Software in source code remains
a confidential trade secret of MESI. Failure to comply with the above restrictions will result in automatic
termination of this license and will make available to MESI other legal remedies. You may not disclose any
of MESI’s proprietary or trade secret information relating to the Software to any third party. You agree to
secure and protect the Software, Documentation and copies thereof in a manner consistent with the
maintenance of MESI’s rights therein and to take appropriate action by instruction or agreement with anyone
who is permitted by you to have access to the Software, to satisfy your obligation hereunder.

**Section 6:** MESI warrants the software media, if provided in a MESI-packaged CD format, for a period of 30
days. If during this period, the software media is found to be defective, it may be returned to us for a free
replacement.

MESI makes no other warranties, express or implied, with respect to the Software, and/or Documentation,
and specifically disclaims the implied warranties of merchantability and fitness for particular purposes. MESI
does not warrant that the operation of the Software will be uninterrupted or error free.

To the maximum extent permitted by law, in no event shall MESI be liable for any special, incidental,
consequential or punitive damages whatsoever, including any claims for lost profits, business interruption,
lost information or other damages arising out of the use or inability to use the Software. In all events, MESI’s
sole obligation or liability under this license is the replacement or repair of the Software or, at our option, a
refund of the purchase price.

**Section 7:** The Software and Documentation are “commercial computer software” and “commercial
computer software documentation,” respectively, and are provided with Restricted Rights under Federal
Acquisition Regulations and agency supplements to them. Use, duplication or disclosure by the U.S.
Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data
and Computer Software clause at DFARS 252.227-7013 et seq. and DFARS 252.227-7015, and
subparagraphs (a) through (d) of the Commercial Computer Software Restricted Rights at FAR 52:227-19, as
applicable, or similar clauses in the NASA FAR Supplement. Manufacturer is Multi Edit Software, Inc. 3370
N. Hayden Road, # 123, PMB 712, Scottsdale, AZ  85251-6632.
Section 8: MESI may terminate this Agreement immediately and without notice if you fail to comply with any item or condition of this Agreement. In the event of the termination of this Agreement, all the sections of this Agreement will survive except for Section 1.

Section 9: The laws of the State of Arizona, U.S.A., shall govern the interpretation of this Agreement, irrespective of the fact that one of the parties now is nor may become a resident of a different state or country. The parties shall submit all disputes, which arise under this Agreement to state or federal courts located in the County of Maricopa, State of Arizona, for resolution. In such event, the parties to this Agreement each consent to the in personam jurisdiction and venue of such courts. The parties agree that service of process upon them in any such action may be made if delivered in person, by courier service, by telegram, by telefacsimile or by first class mail, and shall be deemed effectively given upon receipt.

Section 10: This Agreement sets forth the entire agreement between you and MESI pertaining to the licensing of the Software, and supercedes in its entirety any and all written or oral agreements previously existing between the parties with respect to such subject matter. If any provision of this Agreement is held invalid or unenforceable, such provision shall be revised to the extent necessary to cure the invalidity or unenforceability, and the remainder of this Agreement shall continue in full force and effect. In the event of any conflict between any provision of this Agreement and any applicable law, the provision or provisions of this Agreement affected shall be modified to remove such conflict and permit compliance with such law and as so modified this Agreement shall continue in full force and effect. This Agreement may not be assigned without the written consent of MESI.
Getting and Using Help

Technical Support

How to get Technical Support

We want you to be able to easily find what you need to most effectively use Multi-Edit as your text editor. The online Help System, as well as the Multi Edit Software web site (www.multieditsoftware.com) and FTP site offer detailed solutions to most, if not all of the questions you may have. If you haven't found a solution after reviewing these resources, you can reach our technical support representatives via email or facsimile.

One of the easiest ways of contacting MESI's support representatives is by using Multi-Edit's bug report feature. By clicking on Help | Create Bug Report, Multi-Edit will guide you through creating a detailed bug/problem report that you can then send via email directly to our support staff.

Using the Help | Create Bug Report option will require an active internet connection.

We sincerely hope you never have to contact Multi-Edit for technical support, but if you do, you should expect no less than to receive prompt and courteous service. When you are ready to contact technical support, have your Multi-Edit serial number on hand (found on the CD sleeve or Help | About) and know the exact revision number that you're using (found under Help | About).

Forum Support

Another way to reach Multi Edit for support is through our online Multi Edit Forums. When you visit the forums for the first time, you should register.

You may not have to register – it is up to the administrator of the board as to whether you need to register in order to post messages. However, registration will give you access to additional features not available to guest users such as definable avatar images, private messaging, emailing to fellow users, usergroup subscription, etc. It only takes a few minutes to register so it is recommended that you do so.

There are many forums available such as the General Forums with Announcements, General Chat and a board for test message postings. Additionally, there are Multi Edit forums for support, FAQs, suggestions and user created macros.
FAX Support

MESI FAX Number: (480) 610-2714

To receive technical support via fax, simply send a fax to the number listed above. When faxing, please remember to include the exact revision number and letter of Multi-Edit you’re using (for example: 9.10.03). Revision number can be found under Help | About. You can expect to receive an answer to your fax within two business days, except on weekends and holidays.

When contacting Technical Support, please use Help | Create Bug Report to assist us with solving your problem.

Web Support

Stay current with Multi-Edit developments by visiting the Multi Edit web site at http://www.multieditsoftware.com/. There you will find product information, FAQs, technical articles, updates (to the program and the Help system), patches, and other links.

MESI also has an anonymous FTP site where Multi-Edit users can download the latest Multi-Edit patches, user supplied macros, and learn about new Multi-Edit developments. To contribute something to the site, read the UPLOAD.TXT file.

One of the easiest ways of contacting MESI’s support representatives is by using Multi-Edit’s bug report feature. By clicking on Help | Create Bug Report, Multi-Edit will guide you through creating a detailed bug/problem report that you can then send via email directly to our support staff.

E-mail: support@multieditsoftware.com
World Wide Web: http://www.multieditsoftware.com/

Other ways to get help

How to get Help on CMAC

The Macro Language Reference Guide is installed if the macro source install option was enabled when Multi-Edit was installed. Typically, all help files are contained in the HELP subdirectory of the main Multi-Edit directory.
Language Specific Help

Help on specific languages is not available by default in Multi-Edit. However, each language type can be set up to allow access to any language specific help provided by the language vendor by configuring the Filename Extension Setup. Within this setup dialog, you can configure help files using a semi-colon delimited list of help file names to be accessed with context sensitive help.

Tip of the day

Multi-Edit now offers a "Tip of the day" dialog that is displayed upon startup. The "Tip of the Day" dialog can be enabled/disabled under Tools | Customize | User Interface and tips can be added or modified by editing the tips.db file located in the config subdirectory.
Intro to Multi-Edit

Introduction

Multi-Edit has many powerful features, some of which you may not use or even be aware of. Many times, users call technical support asking: "How do I do this?" or "How do I set this up?" To help remedy this and save you some frustration, we've put together this section to show you some of the more important (and basic) features of Multi-Edit, thus getting you up and running faster and with a minimum of fuss. As you become more familiar with Multi-Edit, we encourage you to explore and "play" with some of the other features.

Opening and Viewing Files

Opening Files

Files can be opened by many methods; however, the standard method would be to use the file open dialog. Select File | Open, or File | Load. Selecting one of these options will display a file dialog in which a file can be selected.

Alternatively, files may be quickly selected for editing by choosing from the list of most recently opened files at the bottom of the File Menu, or opened from the file prompt in the Window List dialog which is available by selecting Window | List.

Multi-Edit also offers an "Open File Under Cursor" feature that allows you to open the filename (where the cursor resides) within a new window.

When opening a file, Multi-Edit allows you to select the "line terminator," used to determine both where to break lines when loading a file and which characters to insert when the Enter key is pressed during normal editing. DOS files use a carriage return and a line feed to mark the end of the line. UNIX uses only line feeds to mark the end of the line. Binary files have no line terminators—they are continuous streams of data; however, for easier reading, binary files use a Binary Record Length that can be specified on the fly or for a particular extension.

Each extension can have its own default "Line Terminator" type. By changing the Type field when opening a file, it is possible to override the default file type. It is also possible to tell Multi-Edit to automatically detect a file’s line terminator when loading it. The default file type is set in Tools | Customize | Filename Extensions.
Saving Files

Multi-Edit offers several methods to save files. File | Save will cause the current file to be saved to disk under the existing filename. If the file has not been previously saved, you may select a filename and other options. Use File | Save As to save a file for the first time, or to change the filename and other options. File | Save All allows you to save all modified files in open windows. Use File | Close to close the current window and save modified files. Select File | Close All to close all Editing Windows and save modified files. You will also be prompted to save modified files when exiting Multi-Edit.

In addition, files may be saved using the Autosave operation. The options for Autosave and Backups can be configured in the Tools | Customize | Files dialog.

Viewing Files

Normally, Multi-Edit uses the standard MDI style windowing arrangement (each window contains one file with that file staying in the same window). In addition, Single Window mode (configure in Tools | Customize | Windowing) may optionally be used where a window is a view where each file is cycled through (this will be familiar to Brief users).

A variety of commands for navigating between, sizing, and manipulating windows are available. Many of these options are available from either the Window Menu or the Window List.

The following are some of the organizing features and their definitions:

- Hide/Unhide conceals/reveals the current window. When hidden, Next and Previous will skip this window.
- Zoom allows you to instantly maximize a window within the Multi-Edit environment or return it to its previous size. Mouse users can achieve the same results by clicking the left button on the Minimize/Maximize buttons, or double-clicking on the window's title bar.
- Cascade provides ways to group your windows in a cascade arrangement (like tabbed index cards).
- Tile vertical arranges non-minimized windows into a vertically tiled arrangement.
- Tile horizontal arranges non-minimized windows into a horizontally tiled arrangement.
- Split will divide the current window in half, having two adjacent windows occupying the screen area previously occupied by the original window. You can use Split to view two or more files at the same time, or view two or more parts of the same file simultaneously. The latter is referred to as linking.

Navigating Windows

Multi-Edit provides numerous ways of navigating windows, the easiest of which is to use either the window tabs or buttons which are located above the window(s) and just below the tool bar on a default installation. Another way of navigating windows can be accomplished by using the Window List dialog. When a window is initially created, it is assigned a letter that is displayed just above the status bar. Clicking on the window letter will display the Window List dialog that contains all of the open windows as well as window properties such as read-only, modified, and linked. Finally, the window menu offers the following navigation commands that are also accessible via keystrokes.

- Next (F6) moves you to the next window in alphabetical order.
- Previous (SHIFT+F6) moves you to the previous window in reverse alphabetical order.
- Window List (CTRL+F6) Brings up the window list.

Hidden, minimized and system windows are skipped over during navigation.
Viewing Modes

Using one or more of the following modes can modify the way a file is displayed in Multi-Edit:

**Hex**
Displays the current file in Hex format. This is primarily used for editing binary files or for viewing unprintable characters in ASCII files. This mode can be enabled under Tools | Customize | Editing.

> Selecting hex mode results in a side-by-side split: the left side is in hex; the right side is in ASCII. When editing in the left side, characters may only be entered in hex, with the overwrite mode always on. Hex mode is simply a different view of the current file. It does not assume that the file is binary, nor does it change the "file type". Thus, template expansion, smart indent and other features work (if they are configured for that file) while you are editing a file in hex mode. If you wish to view line terminators for a file then you will need to load it as a binary file.

**Collapse**
Collapse mode allows you to view and edit a file in an outline-style format. You could for example, choose only to view lines that start at columns less-than-or-equal-to 8, or perhaps you only want to see lines that contain the word "ERROR."

- You can also have separate collapse mode settings for each open file within Multi-Edit. Thus, you can have one file set to collapse on column 1, while another file collapses based on the keywords PROCEDURE or FUNCTION.
- The difference between using Collapse Mode and performing a Find All operation is that Collapse is more interactive, allowing you to quickly edit the viewed lines, selectively unhide the hidden lines, and perform block operations that either include or exclude hidden lines.

**Line Numbering**
Line Numbering displays line numbers along the left margin of the editing window and may be enabled for the current file or all files of the same language.

Editing and Formatting Text

**Editing and Formatting**

Multi-Edit has many editing features, such as the ability to **Undo** and **Redo** changes to your files. This feature can be found by selecting Edit | Undo, Edit | Redo respectively. Of course there are many other features as well, including the ability to indent/undent blocks of text, change case, fill white space with tabs or spaces, automatically format comments, center lines and justify paragraphs. For more information on any of these features, please reference the additional resources below. In addition, there is an enhanced Cut and Paste functionality with multiple buffers.

**Blocks and Block Marking**

Multi-Edit supports three types of block marking: **line**, **stream** and **column**. Each method of marking can be accessed from hot-keys, by using the mouse or from within the Edit menu. Additionally, you can mark lines or a stream of text by holding down the SHIFT key while pressing the arrow keys.
Search and Replace

Searching Within Multi-Edit

Search and Replace is one of Multi-Edit's most powerful features, enabling the ability to search for text, search and replace text, even across multiple files. All of these features have been combined into one tabbed dialog for quick and easy access. The word "search" has now been changed to "find" to be consistent with most Windows applications. All the search and replace functions can be found under the Search menu in the default command map.

Besides just finding literal strings, Multi-Edit allows patterns, called regular expressions, to be entered and found. In addition, those regular expressions may be combined and saved as regular expression aliases, for future reference.

Global Expression Highlighting

With Global Expression Highlight, you can specify a search string or expression and see all occurrences of the found text highlighted in one or all of your specified files.

This is persistent; you may edit the files as usual without disturbing the highlighting. This function allows paging through a file to quickly see how a string or variable is used.

There is a limitation to Global Expression Highlight. Search strings cannot go across line boundaries.

Incremental Searching

Multi-Edit's powerful incremental search feature allows you to perform simple searches with ease. Place the cursor at the position from which you wish to begin searching, select Search | Incremental Search, and begin typing the string of characters for which you would like to search. Multi-Edit will search for the string as you type it in, highlighting the next occurrence of the search string in the file. Thus, you may not need to input the whole string to find the text for which you are searching. Once a string is found, the next or previous occurrence of that found string is located by using the Alt+N or Alt+P keys.

Book marking Cursor Position

Multi-Edit supports two types of marking positions in text, a marker stack and a random access mark. Each file can contain up to ten of each type of mark and are saved and restored across sessions.

The marker stack can actually be larger but only 10 per file are saved in the status file when exiting.

The marker stack is useful for saving a position in a file so that you can quickly return to it after moving elsewhere in the file. The system uses the marker stack for its own use such as when doing a file compare or matching a language construct (this should not interfere with the user dropped marks).

Random access marks allows you to drop a mark and reposition to that mark in any order. This is useful for switching between multiple locations in a file.
Goto Line / Column

Jumping to a specific line/column is easy within Multi-Edit and can be accomplished by clicking on Search | Goto Line / Column or by clicking on the Status Bar where the current line/column position is displayed.

Language Support

The language support in Multi-Edit is composed of a set of keywords, symbols, and comment characters that define that particular programming language, thus enabling syntax highlighting. However, to really take advantage of the language specific features of Multi-Edit, each language will have macros to allow the following:

- **Construct Matching** – The ability to position the cursor at a matching construct, such as moving between a Begin and End statement.
- **Smart Indent** – The ability to indent the cursor in the correct position after a carriage return.
- **Tagging** – The ability to find code and mark for later positioning. Such as finding the functions and being able to position to that function with one key.
- **Properties** - For example, how to expand curly braces for 'if' constructs in C-style languages, and highlighting of matching parenthesis.
- **Tools | Customize | Languages** contains fields that allow you to enter keywords and other syntax highlighting options, along with a Set Properties button that controls language specific items.

Templates

Template expansion is one of Multi-Edit's most powerful and timesaving features. Template expansion allows you to easily and consistently write repetitive tasks in your code that would otherwise bog you down with extra keystrokes. For example, in C, one can type "i" and press the space bar and receive an if construct complete with curly braces and parenthesis.

These templates are completely editable and are not limited specifically to code items. The flexibility of the template system in Multi-Edit allows you to set up a template for virtually anything you can think of, everything from a comment header to an entire form letter is feasibly possible.

Compiling

Since programmers spend quite a bit of time in their editor, Multi-Edit includes an interface that allows you to easily run the compiler or other programs from within Multi-Edit. As with most things in Multi-Edit, it is completely user-configurable and may be configured by clicking on the Compiler/Program setup button within the Filename Extension dialog.

Comparing Files

Comparing files within Multi-Edit is a breeze! To compare two files, first load them both into Multi-Edit and then select the Compare Files option located under the text menu. The Compare Files Split Window dialog will be displayed, allowing you to make changes to the comparison criteria.
When the comparison is complete, the two files can be moved through and edited. The cursor movement between the two files will be linked and synchronized. By default, you may use the Ctrl+PgUp and Ctrl+PgDn keys to move quickly from difference to difference.

To stop the comparison, close either of the windows. If you select Compare again while either of the two comparison windows is active, the two files will be automatically re-compared. This is useful for updating the comparison after significant changes have been made. You may also generate difference reports. You may change the file compare keys and the colors used to highlight the differences in the Color Setup dialog.

When performing a file compare, Previous Difference and Next Difference will take the cursor to the next difference or the previous difference in the compared files.

These are active only while performing a file compare operation.

Composite Diff allows you to compare two files and build a third file that merges the differences of the two compared files into one. You can then view, save or edit this file as necessary. This is extremely useful for merging changes to a single file from multiple sources.

Multi-Tags

Multi-Tags is an easy-to-use, hypertext-like source code browser.

Run your source files through the Multi-Tags scanner to produce a database of functions/procedures, structures, types, etc., depending on the language being scanned.

Once the database is created, position your cursor on any function name (or other supported language object) and use Find tag under cursor. Multi-Tags will then locate the source file where that tag was defined and take your cursor to the definition. If a tag was defined in more than one place or file, then Find Again will locate the next occurrence.

Tags can also be located using the Tag View, which resides on the Navigation Pane and allows for quick access to tags in the currently loaded tag database. The Tag View also allows you to load and browse tag databases which aren't associated with the current file.

Most of the languages supported by Multi-Edit currently support tags.

Any text file may contain tags via explicit tags.

Project Management

Project Management allows for containment of the many specific settings related to a particular project. A project contains a list of files for easy access, including non-text files, directories, tools, and FTP configuration. All Project options take precedence over other options (i.e., Filename extension setups).

While the Project Manager keeps information on each individual project, it does not contain information on the files that were last opened or a search history. The Session Manager handles this function. When switching Projects, you should also switch to a matching session. This can be done automatically by setting
the **Tools | Customize | Project | Synchronize Projects to Session** option. When enabling this option, every project change will also change to the corresponding named session.

---

**Encoded sessions must be enabled in Tools | Customize | Sessions.**

To start using projects, you will first need to create a project. Using the **Project | Create Project** function can do this. Project files begin with the `.mep` extension and will by default be created in the directory specified in the create function.

A Project consists of a ROOT section, which is all files that are contained in or are children of the project root folder. The Project Root folder will also be the default working directory when the project is opened. There is also an EXTERNAL section that contains all files that are outside the Root file.

In order to view a project, select the **View | Project** option.

---

**Session Manager / Session Management**

A session is a history of what was last done in Multi-Edit. This includes things like which files are open, how the windows are arranged, history of strings used in search and/or replace operations, the location of the cursor in each file, and what project was being edited.

Sessions can be stored in several different ways, configurable in **Tools | Customize | Sessions**.

Descriptive names for each environment, such as "Database Project", store all editor environments related to that editing session. Multiple projects can each have different layouts and settings. This allows you to configure work environments for each project or session. The session list can be sorted and sessions can be protected so that environment settings will not change. For example, Session X has two files open. If it is protected and another file is opened, the next time the session is started, only the two original files would be opened. If the session were not protected, all three files would be opened. Encoded status files are stored with a `.mew` extension.

Command line options are available to allow you to start a named session, start the last session (which is the default), or to bring up the Session Manager on startup to allow you to pick from the list of sessions.

---

**Version Control Support**

One of the main purposes of Version Control Support (VCS) programs is to help maintain and track versions or revisions of a set of files. Multi-Edit's VCS support consists of a set of Multi-Edit macros that acts as a front end to the more popular third-party version control systems.

The VCS support actually contains four different interfaces to the selected third-party VCS programs. This was done to provide convenience and to help increase your productivity. With these interfaces most, if not all, of your VCS work can be done without exiting Multi-Edit.

The first interface is essentially the same interface used to open and close files. When the VCS support is enabled, the file open and close functions have the ability to check files out from VCS archives when loading non-existing files, and to check files back into VCS archives when closing them or upon exiting Multi-Edit. For more detail on how these function, refer to Using the File Open and Close Interface.

The other three interfaces are all accessed through the VCS Menu or toolbar buttons; the most used being the Current File Menu/Toolbar Interface. This allows you to do most of your VCS operations using the file that is loaded into the current Editing Window.
The Directory of Archives Dialog and the Multi-Edit Files From Archives Dialog use dialogs that allow you to select the files on which you want the VCS operations to be run. These are powerful ways to check in and out files and perform other functions in an interface similar to Windows Explorer.

The last interface involves selecting **VCS | Run VCS Interactively**. This starts the User Interface that comes with your VCS program without exiting Multi-Edit. This would be used for features that you require that are not currently fully supported in the VCS support macros.

---

**What are Macros**

A "macro" is a way of automating tasks, particularly repetitive tasks. Multi-Edit uses two different kinds of macros. The first, known as **Keystroke Macros**, is simply a sequence of keystrokes that have been recorded and may be played back later. The second is much more advanced and is created using Multi-Edit's high-level CMAC Macro Language. The Multi-Edit CMAC Macro Language is a complete programming language, designed specifically for the automation of text manipulation tasks.

A Multi-Edit macro is a sequence of instructions that the user may save, retrieve and execute repeatedly. Unlike a simple keystroke macro, which merely repeats a series of operations already performed by the user, the CMAC macro language allows for conditional action, user interface and manipulation of a variety of data types, including strings of text, integers, real numbers, and structures.

CMAC is a compiled programming language similar in syntax to C. While not as sophisticated as a full-fledged, structured programming language like C, CMAC does provide:

- If/else/else if, do/while, while, for and switch constructs
- String, integer, character and floating point data types
- Local and Global variables
- String, integer, character, real and void function types
- Parameter passing, with variable arguments and return values
- Preprocessing directives #ifdef, #ifndef, #else, #endif, #define, #undefine
- Complex, nested expression evaluation
- Full access to all Multi-Edit functions
- DLL Access
- Easy access to the screen, keyboard, mouse and hardware

The creation of a CMAC macro is a relatively simple process: you create the source code to the macro in a Multi-Edit window and then select **Tools | Execute Compiler**. Once compiled, the macro becomes immediately available for your use. You can make changes to the macro, compile again, and the changes will instantly go into effect. If you make any syntactical errors while creating or modifying the macro, Multi-Edit will display the appropriate error message and take your cursor directly to the position where the error occurred.

For additional help with the CMAC Macro Language, select **Help | CMAC Language** or click the CMAC Help button on the toolbar of any Multi-Edit Help window.

The Macro Menu contains tools for executing and debugging Multi-Edit macros. In addition, keystroke macros may be edited from this menu.
WebLair

WebLair encompasses all of Multi-Edit's web development technologies. This includes web browser interaction via WebLair's **browser manager**, Markup language tag highlighting and editing, Tag databases (with a new implementation in version 9.10), and the configuration of **helper applications**.

Multi-Edit makes use of a tag database for markup language tags. This consists of a file containing tag information, such as browser compatibility and attribute names and types. The tag database file is identified by the extension `.tdb` and resides in the `config` subdirectory of the Multi-Edit install directory. The `.tdb` file(s) are created, deleted, or modified through the WebLair configuration dialog(s).
The Multi-Edit Interface

Command Sets

Multi-Edit uses a single component called a Command Set to configure Key/Command Mapping, Toolbars/Toolboxes, and Menus.

When you load a Command Set, you load all of the above-mentioned items for the Command Set. If you create a new Command Set, you will have to create Keys/Commands, Toolbars and Menus. Therefore, we suggest that you copy an existing Command Set and modify it to fit your needs rather than create your own completely from scratch.

A Command Set is more than just keys; it contains menus and toolbars.

Choosing a Command Set

You can select a Command Set by clicking on Tools | Customize | General and clicking the "..." button.

Press the "..." button to the right of the text box below Command set. This will display a standard Multi-Edit list box with a list of existing Command Sets.

This dialog box allows you to create, manage and select whole key/command mappings while keeping those command maps separate from each other. It is here that you can select some of the optional key mappings, such as the Brief keymap. Buttons along the side of the list box allow you to Edit, Insert, Delete or Copy the selections. You can also rearrange the list by moving them up and down in the menu list. The Search and Again buttons allow you to quickly find the list member you are looking for. When you press the Select button, Multi-Edit will enable the currently highlighted command mapping.
Select an item in the list and press the Edit or Insert button to display the Edit Command Set dialog box with the following fields:

- **Description:** A descriptive name for the Command Set (i.e., Brief Emulation mode).
- **Filename:** The actual filename of the command set. The .db extension is assumed. The file is searched for first in the Multi-Edit \Defaults subdirectory and then in the Multi-Edit \Config subdirectory. Once located, it will be copied to the Config directory if any changes are made.
- **InitMacro:** The macro to run when loading the command set, can be used to set global variables, etc.

---

**Command Map**

**What is a Command Map?**

No matter if you're adding a new key mapping entry, adding a toolbar entry, or modifying the menus, you will probably end up looking at the **Keys/Commands** dialog box, which is located under **Tools | Customize | General | Keys/Commands**. This dialog contains all of the commands and their associated keys. Modifying or adding to the list of commands is the first step to creating or updating a toolbar or menu item.

To create a key command report for the current command set being used, click on **Help | Command Map Report**.

To see what command is associated with a key, press **Alt+k**, enter the key you want to find the command for, and the command will be displayed in the status bar.

**Modifying a Command Map**

**Keys / Commands** allows you to change almost every key assignment in Multi-Edit. We have included several example keymaps to help you set up, modify and customize the key/command map to your liking. Because of the complexity of creating a new keymap, we suggest that you copy an existing keymap and modify it, rather than create your own completely from scratch.

To view the **Command Mapping** dialog, select Customize from the Tools menu and click on General in the tree. Press the **Keys/Commands** button to display a standard Multi-Edit list box with a list of existing command mappings. Buttons along the side of the list box allow you to Edit, Insert, Delete or Copy the selections. You can also rearrange the list by moving them up and down in the menu list. The Search and Again buttons allow you to quickly find the list member you are looking for. When you press the Select button, Multi-Edit will enable the currently highlighted command mapping.
To **add** an item to this dialog, move the highlight bar to where you want to insert the new entry and press the Insert button. A new dialog will appear with many fields. See below for a description of these fields. Follow these steps to **add** a Command Map entry:

1. Type a title in the Name field.
2. Type the appropriate command line for this entry in the Command line field. Command line formats will vary depending on the nature of the entry:
   - For Multi-Edit macros, the proper format is `[macro_file]^macro_name`[parameters](/P1=xxx/P2=yyy/P3=zzz). See information on running macros for more information.
   - For external programs, simply type the command line as it would be run from a DOS prompt or from a Windows run prompt:
     `c:\wscan\wscan.exe /SX /F1b2` (Run a Virus Scanner, with parameters)
   - `<FILE>.EXE` (Run the compiled version of the current filename to test. Notice the command line uses Metacommands.)
   - For easy access to external help files, use the following examples for differing options:
     `c:\bc45\bin\bcw` (This will bring up the help file and show the table of contents)
     `c:\bc45\bin\bcw^` (This will bring up the help file and show the search dialog, allowing you to easily search for a topic)
     `c:\bc45\bin\bcw^command-line options` (This will bring up the help file and search for the topic "command-line options" in the search list)
   - To quickly load a commonly used text or source file into Multi-Edit, simply type the full path and filename of the file to be loaded.
3. Select the appropriate Type for the command map. Each type corresponds to each different command line type shown above. The Command type is used only for internal Multi-Edit commands, which are already all mapped for you in the default keymap.
4. Select a Primary and Secondary Key for the entry, if desired. If you only want this item accessible from the menus or from the toolbar but not from the keymap, enter nothing.
5. Click on the large and small buttons in the Buttons field to specify toolbar buttons for this command to use. You can select a button from the list given, or use your own. For more information on how to use your own custom buttons, see Using Custom Buttons on your Toolbars.
6. For special options used when running external programs, including setting the Working Directory and Show Options, click the Program setup button.

To **delete** a key assignment or **change** a current key assignment to a different key:

1. Find the key mapping entry that you want to change using the search feature in the dialog or by browsing through the list. Once you have found the entry, highlight it and press the Edit button.
2. The Command Mapping dialog is shown, with many different fields. You will be primarily interested in the Primary Key and Secondary Key fields. Use the mouse to select the "…” button to the right of the Primary Key field, or use Tab or Alt+Tab to select the button and press Enter.
3. The Press Key to Assign dialog appears. If you want to remove this key assignment (thus freeing the key up for something else you had in mind), press the Delete button. Otherwise, press the new key(s) you want assigned to this command.
4. Repeat steps 2 and 3 for the secondary key field, if desired.
5. Press OK, Close, OK in the successive dialog boxes.


Command Map Details:

The Keymap Lines

Each keymap line displays a command description or macro description, toolbar icons associated with the command (if one exists), button text and the Key Assignments for invoking that command or macro. Many of the lines have two Key Assignments. Multi-Edit allows you to assign a primary key and an alternate key to a command or macro. Only the first will show if a menu item is associated with the command.

The Edit button allows you to edit existing commands or macro descriptions, and change key assignments. Select an item in the list and press the Edit or Insert button to display the Command Mapping dialog box with the following fields for editing:

- **Name:** An informative phrase to help you remember the function of this command or macro.
- **Command Line:** The full command line, including any parameters of your program (or macro, command, etc.) to be executed, should be entered here. Before creating parameters for a macro, you must have a solid understanding of Multi-Edit's system macros or of the macros being used.
- **Type:** This option button instructs Multi-Edit how to interpret the command line given above. There are five different options:
  - **Macro:** This option defines the command as a Multi-Edit macro. If you pass parameters to the macro, you must use string parameters.
  - **Program:** Is used to map the command line to an external program. The full path must be designated in the command line.
  - **Help file:** This option will link the command line to a Windows help file.
  - **Text file:** This button links the command line to a frequently loaded text file name.
  - **Command:** This option must be selected for Multi-Edit to recognize that the command line refers to an internal Multi-Edit macro command.
- **Mode shift:** The ability to use multi-keystroke assignments (for example, Ctrl+K J), has been added via Key Modes. To use key modes, you assign a key, like Ctrl+K to do a "Mode Shift" to, for example, mode 1. Then commands that you want to execute only after Ctrl+K has been pressed would be assigned to their respective keys, but the new "Mode" field would be set to one. The default-editing mode is zero. When a key is pressed that causes a mode shift, the next key pressed will reset the mode back to zero, even if no command is found to execute. We will be enhancing this later to allow for the creation of "persistent" modes that do not automatically reset.

⚠️ This can be confusing. The mode number for a mode shift needs to be put in the Command Line field, not the Mode field. By putting it in the Mode field you are telling it that the mode shift should only take place if Multi-Edit is already in that mode.
• **Primary Key:** Enter the chief key assignment you wish to assign to this command or macro.
• **Secondary Key:** Enter the secondary key assignment you wish to assign to this command or macro. This field is optional.
• **Buttons:** This field shows the icons associated with the command (if any). This field is also optional. Two buttons are displayed: a large icon and a small one.
• **Button Text:** This field lists the text to be displayed on a button if there is no icon associated with it.
• **Help Index:** This field lists the help search string to use if help is requested for this command or icon. This does not refer to creating a command entry to access a help file. Help search strings are entered in the following format: [path][help file]'[search string]. If no path is specified, Multi-Edit will look in the Multi-Edit \HELP subdirectory. If no help file is specified, Multi-Edit will default to the default help file Me.chm. If no search string is specified, Multi-Edit will leave you in that help file's search dialog.

Examples:

c:\bc45\bin\bcw^  
This will open the help file BCW.HLP and wait for you to enter a help search string for which to search.

CMAC^set_Multi-Edit_attr  
This will open the help file CMac.chm and look for a topic matching the search string 'SET_MULTI-EDIT_ATTR'.

• **Program Setup:** This button will bring up the Program Setup dialog which is used to set options. These options control how the program specified in the Command Line is launched. The options in this dialog are similar in function to the ones in the Compiler Setup dialog.

---

*The options in this dialog are only meaningful when the Type is set to Program*

---

• **Working Directory**
  • **Current:** Uses the current Multi-Edit working directory as the working directory of the compiler.
  • **Source file:** Uses the path to the current source file as the working directory.
  • **Program:** Uses the path to the compiler as the working directory.
  • **Specified:** Allows you to specify the working directory.
• **Executable Type:** This drop-down list box is used to select the type of compiler executable that has been specified in the Command entry. This is used by Multi-Edit to determine how to start the compiler running. It is usually set to Auto Detect; however, if problems arise in compiling then try another setting.
  • **Show:** Note that PIF settings (under Windows) take precedence over these settings.
    • **Normal:** The compiler execution window will appear in the default position and size.
    • **Minimized:** The compiler execution window will appear minimized.
    • **Maximized:** The compiler execution window will start full screen.

**Options**
• **Wait for program to finish:** When running, the command line program will wait for it to finish before returning to Multi-Edit.
• **Run in background:** Will start program and immediately return, processing errors after the program is finished.
• **No auto file save when program is run:** When checked, will cause Multi-Edit not to do an auto save before running the program.
Menu/Toolbox Control

- **Disabled identifier:** This field designates a global variable or macro used to instruct Multi-Edit when to disable this command if it exists in a menu. If it refers to a global variable, the variable name is entered here. If it refers to a macro, the macro name must be preceded by an equals sign (=), and the macro must be a string function with no parameters (C-style). The variable or macro must return a 1 (TRUE) if the menu selection is to be disabled. /LS=lang may also be used, where "lang" is the name of the language that the menu item is to be shown. For example, entering /LS=DELPHI would tell Multi-Edit that the menu item is only to be shown when a Delphi file is being edited.

- **Checked identifier:** This field works like the disabled identifier, except this variable or macro designates when to put a check mark in front of the menu selection. The Persistent Blocks menu selection under the Block menu uses this.

- **Wcmd identifier:** This field designates a global integer used to hold the WCMD Identifier when the command is called. This is useful only to those users who want to write their own macros and need to enable a function to identify what WCMD executed it.

- **Mode:** The key mode; means this key belongs in mode "n". The initial mode command needs to be entered in order for this command to be executed.

Options

- **Disable for minimized window:** This command is dimmed if the current editing window is minimized.
- **Reset mode:** Check if this key should reset the key mode after executed.

Special WCMD Identifiers

In order for Multi-Edit's MDI support to work correctly, the following global integers must be used to identify key Multi-Edit functions to the kernel. If a user wishes to change the macro Multi-Edit executes for these commands, he/she must ensure that the WCMD Identifier for that command is set to the appropriate global. The key functions and their corresponding globals are listed below:

```
WLIST_WCMD_ID = Window List
CLOSE_WCMD_ID = Close Window
MIN_WCMD_ID = Minimize Window
MAX_WCMD_ID = Maximize Window
RESTORE_WCMD_ID = Restore Window
MOVE_WCMD_ID = Move Window
SIZE_WCMD_ID = Size Window
```

If the command does not exist in the key/command map then it is not necessary to make one; the windows default will be used in such cases.
Customizing Menus

The menus in Multi-Edit are fully configurable. You can easily create, delete, or modify existing menus by using the following dialog boxes.

Open the menus dialog by selecting **Tools | Customize | Menus**. A standard Multi-Edit list box with a list of existing menus is displayed. Buttons along the side of the list box allow you to Edit, Insert, Delete or Copy the selections. You can also rearrange the list by moving them up and down in the menu list. The Search and Again buttons allow you to quickly find the list member you are looking for.

Adding and modifying items in the menu is a snap when you follow these steps:

1. Bring up the Menus dialog by selecting **Tools | Customize | General | Menus**.
2. Select the menu you want to modify. A dialog will appear displaying the current layout of the selected menu. If you want to insert a menu item, position the highlight bar below where you want the new item to appear and press the Insert button. If you want to modify an existing menu item, highlight that item and press the Edit button. If you want to rearrange the menu items, you can use the Move Up/Move Down buttons to move existing menu items.
3. Selecting the insert or edit button will display the "edit menu item". See the menu dialog fields section below for this dialog's field details.
4. To set or change the command map entry linked to this menu item, press the button to the right of the Command field. The Command Mapping dialog will appear. Highlight the command you want to link to the menu item and press the Select button. If no command-mapping item exists for this menu item, you may create one. See Adding/Modifying Command Map Entries for more information.
5. Press OK.

**Menu Dialog Fields:**

**Main**
This refers to the Menu items below the title bar of the Multi-Edit window. Click on the down arrow to view items in the list; click on the menu to be used by default.

**Context**
This pop-up menu is shown when you right-click in an editing window. Click on the down arrow to view items in the list; click on the menu to be used by default.
MDI
This is a window specific menu shown when you click on the upper left icon of any editing window. Click on the down arrow to view items in the list; click on the menu to be used by default.

If you select a menu to edit (or press Insert), you will be presented with another list box (Edit Menu Set) that shows the menu items you have configured for this menu. At the top of this dialog is a Menu Type drop-down list box from which you can select main, context or MDI. Press Edit or Insert to display the Edit Menu Item dialog box with the following fields for editing:

- **Menu Text**: This is the text that will appear in the menu itself. The ampersand (&) character precedes the hot key for that menu item. An underline will be placed under that character, and it will then be accessible from the menu by pressing that character on the keyboard.
- **Level**: This option button field designates the level in the menus where this item should appear. If it is a main menu selection, Level One should be selected. If it is to be placed in a sub-menu under a main menu selection, Level Two should be selected, and so on.
- **Command**: Select this button to be taken to the Command Mapping list box to select a command mapping for this menu item. The menu selection will also inherit the keys assigned to a command, if any, and the primary key will be displayed next to the menu item when the menus are displayed. A separator may also be selected as a command in order to group items in the menus.
- **Context Sensitive Menu Additions**: It is possible to add items to the context menu (right mouse button menu) on a language sensitive basis. There are two ways to do this. First, if you create a menu with the exact name of the language being used (for example JAVA), then that menu will automatically be added to the context menu. Also, for more complex tasks, we have added a "Menu" field to the language properties dialog that allows you to specify a macro that is to be run whenever the context menu is about to be popped up.
Here is an example of a macro that modifies the menu:

```c
#include Multi-Edit.sh
#include menus.sh
macro TestMenu

/***************************************************************************/
Function: Example of adding menu entries to the context menu by using the language specific "Menu" macro. This macro assumes that you have created a menu called "TEST".
Entry   : /MENUHANDLE=x The handle of the menu to pass to CreateMenuFromDBEx
          /LANGUAGE=str The name of the language
Exit    : Nothing
***************************************************************************/
{
    int menu = parse_int("/MENUHANDLE=", mparm_str);
    str dbname = wcmdmapname;
    str menuname = "TEST";
    CreateMenuFromDBEx( dbname, menuname, MENU_PopUp, menu  );
} // TestMenu
```

---

**Toolbars**

### Customizing Toolbars

Many common editing tasks already have an icon associated with the command and a button set up in the toolbar for easy mouse access. By default, the Main Toolbar is positioned below the Main Menu. You can customize the toolbar to suit your editing needs by modifying, removing, or creating buttons and commands. The toolbars can also be displayed along any window border or, you can even have them "float" in the Multi-Edit workspace.

While most of the toolbar configuration can be done from within the Edit Toolboxes dialog, a right mouse click on a toolbar will display options for performing on the fly modifications.

When you press the **Toolbars/Boxes** button on the customize dialog (Tools | Customize | General), a list box will appear containing all of the currently defined toolbars. Buttons along the side of the list box allow you to Edit, Insert, Delete or Copy the selections. You can also rearrange the list by moving them up and down in the menu list. The Search and Again buttons allow you to quickly find the list member you are looking for. When you press the Select button, Multi-Edit will enable the currently highlighted command mapping.

---

Some toolbars may be hidden. To verify if a toolbar is hidden, select a toolbar from the list and click on edit, if the hidden option is enabled, this toolbar will not be displayed.
To add or modify items in the toolbars, follow these steps:

1. Bring up the Toolbars/boxes dialog by selecting Tools | Customize | General | Toolbars/boxes.
2. Highlight the Toolbar you want to modify and press the Edit button or press the Insert button to create your own custom toolbar.
3. Give your toolbar a name (if you're creating one), ensure its hidden/unhidden state is set correctly, and set the icon size and position as you desire.
4. To add items to the toolbar, press the Insert button. The Command Mapping dialog will appear. Highlight the command you want to put into the toolbar and press the Select button. You can also create your own command mapping items on the fly. See Adding/Modifying Command Map Entries for more information.
5. To replace one toolbar item with another, press the Edit button. Again, the Command Mapping dialog will appear. Highlight and Select the item you want to put into your toolbar.
6. Use the Move Up/Move Down buttons to organize / move items in the toolbar.

**Edit Toolbox fields:**

- **Title:** This text box contains the name of the toolbox you are editing.
- **Hidden:** If checked, then the toolbox will not be active. In order to use the Toolbox, this must be unchecked.
- **Position:** This drop-down list box allows you to configure where the Toolbox will appear in Multi-Edit. There are six options available: along one of the four Multi-Edit borders, floating or roving. Floating Toolboxes can be positioned within Multi-Edit window; Roving Toolboxes can be positioned anywhere on the Windows desktop. They may be sized with multiple rows and columns. If multiple toolbars are positioned in the same location, the order in which the toolbars will be displayed is based on the order in the Toolboxes dialog.
- **Style:** This list box lets you choose between large or small icons in your toolbox.
- **Show On:** This field designates a global variable or macro used to instruct Multi-Edit when to hide or show the toolbar. If it refers to a global variable, the variable name is entered here. If it refers to a macro, the macro name must be preceded by an equals sign (=), and the macro must be a string function with no parameters (C-style). The variable or macro must return 1 (TRUE) if the toolbar is to be hidden. /LS=lang may also be used, where "lang" is the name of the language that the toolbar is to be shown. For example, entering /LS=DELPHI would tell it that the toolbar is only to be shown when a Delphi file is being edited. The /LS= syntax has also been added to the "Disabled identifier" in Command Map Editing.
Using Custom Toolbar Buttons

In order to add custom buttons to your Multi-Edit toolbars, you must have access to a resource editor—you cannot use Multi-Edit to edit your toolbar buttons. Such resource editors are included with many of the popular IDEs (especially for C) or are available separately. Whatever application you use to edit your bitmaps, it must be able to save to a .DLL file, which is where Multi-Edit accesses the toolbar bitmaps.

The toolbar images in Multi-Edit are not icons in the strict sense—they are bitmaps.

Included with Multi-Edit is a USRBMP32.DLL file, where you can edit and add your user bitmaps. Use your resource editor to open USRBMP32.DLL and add your bitmaps. So your bitmaps match the current Multi-Edit standard, large buttons are 25x25 pixels, and small buttons are 15x15 pixels. Use the resource editor to create or copy bitmap images into the USRBMP32.DLL file. Take note of the names of the resources you wish to incorporate in Multi-Edit (for example: USER_100).

Once you have your custom images within the USRBMP32.DLL, start up Multi-Edit and click on Tools | Customize | Keys/Commands to bring up the Command Mapping dialog. Highlight and edit the item for which you wish to use your custom button, and then press the large or small button in the Buttons field (whichever one you would like to set). You may then highlight and Select that bitmap for use by your command mapping.

Interface

Tools Pane

The Tools Pane is a tabbed pane at the bottom of the Multi-Edit screen that holds tabs for Bookmarks, FTP Results, Compiler Results, Collapse Mode, Find Results and Background Task List.

The Tools Pane is a displayed item. On the Status Bar, the small window icon in the lower right corner can be clicked to toggle the display of the Tools Pane.

When displayed, the Tools Pane can also be minimized by clicking the minimize icon in the upper left corner of the Tools Pane. Use the Tools Pane maximize button to cause it to fill the specified percentage of the screen (defined under Tools | Customize | Windowing | Tool Pane). When the Tools Pane has keyboard focus, use the Esc key to hide it and return to editing.

Multiple tabs can be displayed at one time and brought to the front by clicking on a tab. Close a tab by right clicking on the tab and selecting Close tab. Resize the window by pausing the mouse over its top edge until a double arrow cursor appears. Click and drag the edge of the Tools Pane to the desired size.

In addition to this option, each tab may have options specific to its function.
Navigation Pane

The Navigation Pane is a resizable tabbed pane that can be docked on the right or left side of the editor. This pane allows for quick access to files using the Project Manager, tags using the Tag View, globals and macros using the System View, and Multi-Edit's HTML common code feature.

The Navigation Pane is displayed when any of the previously mentioned dialogs are executed or by selecting the Navigation Pane item in the View menu.

To delete a dialog contained on the Navigation Pane, right click on the dialogs tab and select Close Tab.

**TmplPane Add-On**

The TmplPane Add-On provides convenient access to a set of templates appropriate for the currently selected file. After installation, which you've just completed, a new Template Pane option will be available on the View menu. Selecting it will turn the pane on. By default, the Template Pane will 'follow' the current window; as the focus changes, the template sets shown will match the file extension for the current window.

For most languages, the top section will show language specific templates, while the bottom section will show the global templates. For web related extensions (where the LANG buffer property equals HTML) you can choose ASP, Script, WebLair, or HTML templates for both the top and bottom section. ASP and Script templates are found by looking at the ASP_DEF and SCRIPT_DEF buffer properties.

The template set choices can also be overridden by setting a buffer property TMPLPANE_TOP or TMPLPANE_BOTTOM in the extension or language setup. To use the Template Pane, select a template from the list and double-click it to expand it at the current cursor position.

The TmplPane Add-On can also be used to preview a template in the current file. The Preview option is available from the context menu. Selecting it will open the Preview tab of the Tools Pane, copy the current file, and expand the template at the current cursor position. You will be prompted for user input where appropriate, just as you would during a real template expansion.

For large files, a confirmation dialog will be displayed.

**FilePane Add-On**

The FilePane Add-On provides convenient access to a set of commonly used files or directories, similar to the Favorites menu in a web browser. Files and directories can be added via the toolbar buttons on the Favorites Pane, or the current file/directory can be added via the right-click context menu. Directories default to a depth of 1 level, which can be adjusted via the Favorites List dialog. The depth controls how deep the tree will be under a directory favorite. Setting depth too high can have performance impact when the tree is created and refreshed, you may need to play with these settings to find what works best for your environment.
Window List Add-On

The WinList Add-On is a fully functional replacement for the Ctrl+F6 Window List dialog found on the Window menu. Via this navigation pane, you can hide/unhide, minimize/restore, save, print, and close edit windows. In addition, you can bring up the file prompt in order to open a new file. The window list can be set up to automatically sort open files by filename, full path, extension or date. Or, you can select a manual sort, and reorder the windows using the up and down buttons. Note that manually reordering the windows will turn off the automatic sort.

Status Bar

The Status Bar is displayed at the bottom of the Multi-Edit screen.

![Status Bar](image)

From left to right, the Status Bar displays the following:

- **Status Line**: Displays messages as necessary.
- **Cursor Position**: Displays the current line and column position of the cursor in the current Editing Window. Click in this area to display the Goto Line dialog box.

*The number displayed within the "[ ]" as shown in the Status Bar above indicates the "original line number" before any changes.*

- **Ins/Ovr**: Displays the Insert/Overwrite mode. Pressing this button (or the Ins key) toggles between Insert mode (characters typed are inserted at the current cursor position) and Overwrite mode (typing replaces characters at the current cursor position).
- **Caps**: Displays the Caps Lock status. Pressing this button (or the Caps Lock key) toggles the keyboard Caps Lock status.
- **Num**: Displays the Num Lock status. Pressing this button (or the Num Lock key) toggles the keyboard Num Lock status.
- **Rec**: Will display in red text when a macro is being recorded. Press this button to record keystrokes; when completed, select Rec again to stop recording and save the macro.
- **Tools Pane Status**: Pressing this button will toggle the Tools Pane display (minimize or restore).

Tab Bar

The Tab Bar, a tabular window selector, appears by default across the top of the Editing Window under the toolbars. The Tab Bar can be placed above or below the Editing Windows, or turned off. Click on a tab in the Tab Bar to bring the selected Editing Window to the top.

<table>
<thead>
<tr>
<th>Block (my edits).s</th>
<th>Blocks</th>
<th>Blocks</th>
<th>Block.sh</th>
<th>ConMenu.s</th>
<th>ConMenuD.s</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEW Macro Sample.bat</td>
<td>Projects</td>
<td>RShared.s</td>
<td>WinList.s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bitmaps on the tabs show whether the file has been modified (yellow pencil through the file folder), is read only (red circle/slash on the file folder), is minimized (compressed folder icon), or is hidden (grayed title if Show Hidden has been enabled).
The Tab Bar is customizable via the **Customize | Windowing | Tab Bar** dialog.

The Tab Bar can be placed at the top or bottom of the screen or turned off.

When enough windows are open such that the Tab Bar is filled on one line, remaining tabs can be displayed on additional lines or a single line (arrow buttons will be displayed on the right that can be clicked to move through the lines of tabs) depending on the configuration.

The Tab Bar will initially be displayed with the filenames in tabs. However, when the Tab Bar expands to multiple lines, clicking on a tab brings it to the bottom and the order in which the tabs are displayed changes. To avoid this, you can customize the Tab Bar to display files in "Button" style. Changes to the "Button" style option will not affect tabbed dialogs.

File names can be displayed in Proper Case, which saves space but does not reflect true file name case.

Normally, windows that are hidden or minimized are not shown on the Tab Bar. Customizing the windowing can change this.

Fonts, font colors, and tab colors can be customized as well.

---

*Changes to font and tab colors will affect all tabbed dialogs in Multi-Edit.*

---

**Right Click Options**

Right click on a tab in the Tab Bar to display a pop-up menu which includes several window functions (Restore, Move, Size, Minimize, Maximize, Close, Hide, Previous, Next), Extension Setup, Edit Templates, Language Setup, Ruler, and Customize windowing.

---

**List Boxes**

Several dialog boxes in Multi-Edit appear as standard Multi-Edit list boxes, such as the one below. Select an item from the list and choose from the buttons on the right.

---

**Toolboxes**

- **Edit**: Press the Edit button to modify the selection in the list box.
- **Insert**: Press the Insert button to create a new item in the list.
- **Delete**: Press the Delete button to delete the highlighted item in the list.
- **Copy**: Press the Copy button to copy the highlighted item in the list and create a new item with identical properties.
- **Move Up**: Press the Move Up button to move the highlighted item in the list up one line.
- **Move Dn**: Press the Move Dn button to move the highlighted item in the list down one line.
- **Search**: Press the Search button to search the list for a particular item.
- **Again**: Press the Again button to repeat the last search on the list.
- **Print**: Press the Print button to print the list.
- **Select**: Press the Select button to select the highlighted item in the list.
- **Close**: Press the Close button to close the List Box.
Mouse Interface

Block Operations

The status bar displays information about the text being marked. This information is dependent on the style of block used.

- **Columnar blocking:** The number of lines and columns are displayed.
- **Stream Blocking:** The number of characters blocked is displayed.
- **Line Blocking:** The number of lines blocked is displayed.

Marking A Block

Double clicking with the left button on a word in a window will cause that word to be marked as a columnar block. Triple clicking will cause the whole line to be marked.

To mark a stream block, position the cursor at the desired starting point. While holding down the left mouse button, move the mouse cursor to the desired ending point and release the left mouse button.

If you move the text cursor to the desired starting position, then move the mouse cursor to the desired ending point and press the **Shift** left button, a block will be marked from the starting position to the ending point. Now, leaving that block marked, you can extend it by pointing farther in the text, and pressing **Shift** left button.

Additionally, while marking a block with the mouse, you can hit the right mouse button (while still holding down the left button), and the block mode will change. Repeatedly clicking the right mouse button will cycle through the line, column, and stream marking modes.

Whenever you use the mouse to mark a block, a cursor will be displayed next to the mouse cursor indicating the type of block being marked.

---

*By default, the mouse marks streams of text.*

Moving and Copying Blocks

If you click and hold the left mouse button on any position inside a marked block, you can drag the mouse cursor to the position you want to move the block. You will notice a small icon displayed next to the mouse cursor, indicating that a block move operation is occurring. When you release the mouse button, the block will be moved to that position.

To copy a marked block to another location without removing it from the first location, hold the **Alt** key down while keeping the left mouse button pressed. Move the mouse cursor to the desired position and release both buttons. The block will then be copied to that position.

---

**Alternative Mouse functionality**

The right mouse button has a special feature in Multi-Edit. During normal editing, you can click the right mouse button to access a special menu. Exactly which menu gets accessed depends on where the mouse cursor is when clicked.

In **Any Editing Window**

Click the right mouse button (or press the **Esc** key) within an Editing Window to pop up the Context Menu.
The Context Menu is fully configurable under **Customize | Menus**, including the addition of language-specific entries. By default, the Context Menu contains the following choices taken from various other menus.

- **Copy:** This selection copies the current marked block to the Multi-Edit buffer.
- **Cut:** This choice cuts the currently marked block and places it in the Multi-Edit buffer.
- **Paste:** This choice pastes the contents of the Multi-Edit buffer to the current cursor position.
- **Find tag under cursor:** This Multi-Tags operation searches for a tag with the same name as the text under the cursor.
- **Open file under cursor:** If the text cursor is sitting on a filename, you can open that file with this command. This is especially useful for opening up files listed in #INCLUDE statements.
- **Extension-Specific help:** Multi-Edit has the ability to access help files on an extension-specific basis. For example, all files with a .C extension can have HTML help files associated with it. The Extension Specific Help selection will search for a help topic with the same name as the text under the cursor. If a matching topic is not found, the HTML Help Search Dialog will remain open, showing topics that most closely match the search string. If an extension does not have a Default Help File specified, this feature will search for a matching topic in the Multi-Edit help file (Me.chm). For additional information about the Default Help File, see the Filename Extensions section.
- **Customize this files settings:** Choose from Extension setup, Edit templates, Languages setup.
- **Edit this menu:** This selection allows you to add, remove, or modify entries in the right mouse button menu.

**On A Toolbar/Box**

A right mouse click on the toolbar or toolbox displays a pop-up menu with the following options:

- **Left vertical, Right vertical, Top horizontal, Bottom horizontal, Floating, Roving:** Only one of these items may be selected at a time. They allow you to change the location of your toolbar/box "on the fly." Toolbar/box positions may be along any border, "floating" in the Multi-Edit workspace, or "roving" on the Windows desktop.
- **Big icon, Small icon:** Controls whether the buttons displayed in the toolbar/box are large or small icons.
- **Hide toolbox:** If you do not want your toolbox to appear in the Multi-Edit workspace, select this item. The toolbox selected will be hidden from view. To make it reappear, uncheck the Hidden check box in the Edit Toolbox dialog.
- **Edit toolbox:** Brings up the Edit Toolbox dialog for that toolbar/box. From it, you may add, remove, or edit buttons in the toolbar.
- **Toolbox Manager:** Brings up the Toolbox Manager.

**On the Tab Bar**

Right click on a tab in the Tab Bar to display a pop-up menu which includes several window functions (Restore, Move, Size, Minimize, Maximize, Close, Hide, Previous, Next), Extension setup, Edit templates, Language setup, Ruler, and Customize windowing.

**On the Tools/Navigation Pane**

Close a tab on the Tools Pane by right clicking on the tab and selecting Close tab. In addition to this option, each tab may have options specific to its function.
Drag and Drop

The "drag and drop" feature is available in Multi-Edit. From the File Manager, you can open a file by selecting the file you want to open with the mouse and dragging it to the Multi-Edit screen. If you have minimized Multi-Edit, you may also drag the file to the minimized icon. When using the drag and drop feature, Multi-Edit opens a new window for each file loaded.

Drag and Drop Files

From the Windows Explorer, you can open a file by selecting the file you want to open with the mouse and dragging it to the Multi-Edit screen. If you have minimized Multi-Edit, you may also drag the file to the minimized icon. When using the drag and drop feature, Multi-Edit opens a new window for each file loaded. Files are opened in the default style for its extension.

Intellimouse Support

Scrolling in Editing Windows works with the Microsoft Intellimouse. The wheel will scroll editing windows and list boxes up or down one line at a time. Hold down the Ctrl key to scroll one page at a time.

Multi-Edit 9.10 supports the standard wheel messages that have been built into Windows. Intellimouse support in Multi-Edit may work with other wheeled mice, but only the Intellimouse has been tested.

Colors

The title bar, menu bar and workspace background colors are user configurable and can be modified in the Windows Appearance Dialog located under the Windows - Control Panel - Display options dialog.

The editing window colors within Multi-Edit are also user configurable and can be saved as a color template. Color templates are extension specific and are used to define colors such as reserved words, changed text, and search highlight text. Each color setting includes a foreground and background color definition.

Follow the instructions below to create or modify a color template.

1. Select Tools | Customize | General and click on the Colors button to display the Colors Setup dialog box.
2. Select the screen display item whose color you want to change. FG (foreground) and BG (background) will appear in the Color Chart corresponding to that item's current colors.
3. The Color Chart contains numerous background and foreground color combinations. Click the left mouse button to change the foreground color to the color currently under the mouse cursor. Click the right mouse button to change the background color.

The color chart contains pre-mixed colors that may not be displayable under your current Windows configuration. Multi-Edit will pick the color from your Windows color palette that most closely matches the one selected in the color chart. Some colors will not be displayable at all (show up as black).
Colors Dialog

This area allows you to define several color templates that you can select and use at any time, depending on your editing needs. You have the option to Save your current color template, or Remove a color template.

Be sure to click on the SAVE button after modifying a color template.

Color Fields

The list box below the color templates field contains examples of the different types of text that exist within Multi-Edit editing windows and how they currently look on your monitor. The top section contains Editing selections that will always be colored when in Multi-Edit. The second section represents Syntax Highlighting selections that will only be colored when the Syntax Highlighting | On check box is marked. If this check box is not marked, they will appear as normal text. The next section shows the colors that will be used during Multi-Edit's interactive file compare. The final section shows the Tag Highlighting colors.

- **On**: When checked, Multi-Edit's context sensitive syntax highlighting is enabled. If unchecked, keywords, strings, comments, etc. will appear as normal text.
- **Override Current Line**: When checked, syntax highlighting will override the current line color, if one exists. Otherwise, the current line color will take precedence over syntax highlighting.
- **Basic Colors**: This section shows the entire default foreground and background colors to choose from in Multi-Edit.
- **Custom Colors**: Any custom colors that you define appear here. You can choose these colors exactly the same way as a basic color. You must define colors by pressing the Define button for any colors to appear here.
Multi-Edit allows you to set some special options when defining your window colors:

- **Control:** Allows you to instruct a Syntax Highlighted item to use its currently defined color or use the normal text color instead.
- **Use Color:** Allows you to use the Font Style option button field to set a bold or italic font for this Syntax Highlighted item.
- **Use Font:** Allows you to set a standard Editing item to off (disabled). Note this button is not accessible for syntax highlighting options.
- **Font Style:** If the Use Font check box has been checked, there are additional syntax highlighting options:
  - **Bold:** The Syntax Highlighted text will appear in bold.
  - **Italic:** The Syntax Highlighted text will appear in italics.
  - **Define ...:** Press this button to display the Custom Color dialog box in which you can create a color and add it to the Custom Colors palette. You create a color by adjusting its hue, saturation, and luminosity or by specifying its red, green and blue values.

## Fonts

Multi-Edit does not save the font with a file. You cannot have different fonts within the same file. Changing the font only affects how the files are displayed and, in some cases, how characters are entered.

### Browse Buttons

To make changes to any of the font settings in the Tools | Customize | Editing dialog box, press the "..." button. The resulting DOS/OEM Font Select dialog contains the following fields:

- **Font**
  - Select a font from the list of fonts available. As you change the font, the example box will change to reflect how the new font will appear. You will probably notice that there are not as many fonts in this list as you have installed in Windows. This is because Multi-Edit can only use non-proportionally spaced fonts. Proportionally spaced fonts vary in character width and cannot be used.

---

> Multi-Edit does not force either an OEM or ANSI "character set" while you are editing or printing files. Instead, Multi-Edit uses the currently selected font to display or print the characters in the file. What actually appears on your screen or printout will depend on which font you have selected in Multi-Edit. If you have an ANSI font selected and wish to insert or print a special OEM font character, you will most likely need to switch to that OEM font for that character to display properly. You should experiment with the fonts you have available to find one that's appropriate for your needs. For more information about entering high-ASCII value characters in Multi-Edit, see OEM Translation below.

### Font Style

There are four font styles: regular, italic, bold, and bold italic.

> If you select a bold and/or italic font, this may reduce the effectiveness of Multi-Edit's syntax highlighting by making highlighted text appear the same as normal text. See the section Colors for more information.

### Size

Select a font size from the list given. Larger numbers correspond to larger text, as shown in the example box.

### Script

The style of the font (i.e., Western or OEM/DOS).
OEM Translation

As you may know, there are two "categories" of fonts under Windows: ANSI and OEM. OEM fonts are fonts that use the older DOS ASCII character set, while ANSI fonts use the newer ANSI (Windows standard) character set. While using the ANSI character set is fine when you're only dealing with Windows, if the files you're editing have to be compatible within DOS as well as Windows, it is necessary to use the ASCII character set. This is especially true when you are using the DOS compilers and using high ASCII value characters (characters above ASCII 127 - like special International characters or line drawing characters).

You also may or may not know that when running Windows, Windows "performs" an OEM to ANSI translation when you enter characters. This translation "converts" your OEM character into a comparable character in the ANSI character set. For example, let’s say you want to enter an ASCII 164 character (the ñ character) into your file using an older (pre-OEM Translate) version of Multi-Edit. So you hold down the ALT key and use the numeric keypad to type out 164. The Windows keyboard driver takes that ASCII 164 and "turns" it into an ANSI 241 (also an ñ character) (in actuality, the keyboard driver simply returns 241). If you then save your file and use a DOS program to edit it, you'll see that your ñ character is now a ± character. Obviously, any problem where the character you type isn't the character you get is serious.

In Multi-Edit, this can be remedied by selecting the OEM radio button under Tools | Customize | Editing. This does two things: First, it enables the font you have selected as your default OEM font. And second, it activates the OEM Translate feature that takes the ANSI character that Windows has just converted, and uses an ANSI to OEM function to convert the character back to an ASCII character. Thus, your ñ character will be present both within Windows (under an OEM font) and DOS. You can also enable this feature by selecting OEM font for a specific extension and enabling OEM translation for that extension.

Obviously, when using this feature, OEM fonts will be used to view your files. When not using it, ANSI fonts will be used to view your files. It would not be useful to use a font based on a different set than the one the file was created in.

If you are going to be using this feature in Multi-Edit, you'll need to be familiar with which of your fonts are truly OEM and which are truly ANSI. Even though fonts have a flag, which identifies them as OEM or ANSI, some font designers will purposefully set the flag to ANSI to keep the font from showing up in certain font lists in Windows. It is for this reason that all the available fonts show up in the OEM font list while only the ones with the ANSI flag set show up in the ANSI lists. This allows you to select an OEM font whose flag has been set to ANSI for one reason or another. But you'll certainly run into problems if you select a true ANSI font as your OEM font.

You can set the OEM / ANSI option to be the default for an entire session (all files loaded in a session), or you can specify a particular extension to use OEM fonts and another extension to use ANSI fonts, or you can specify a specific file to use the OEM font at load-time by checking the OEM Mode box in the File | Open dialog.

This explanation of OEM and ANSI character sets is not meant to be exhaustive - there is more to the issue of OEM and ANSI character sets than this. However, this is a fairly accurate description of what happens when you develop for DOS while using Windows. This feature was included in Multi-Edit to provide flexibility, not to confuse anyone. If you don't understand what's presented here (as it is difficult to explain in written terms without several more pages), more information about the differences between OEM and ANSI is available. The November, 1995 issue of “Windows Developer Journal” contains an article titled Understanding NT, which does an excellent job of describing the differences between OEM and ANSI characters sets, explaining Code Pages, and showing the next evolution in character sets - Unicode (a true two-byte character set). Also, you can always drop a line to our tech support department for an explanation if you don't know how to use this feature.
Files and Windows

What is the difference between a File, Buffer and Window?

Files, buffers and windows all have different meanings in different contexts. When working within Multi-Edit, the term **files** refers to the data stored on media, such as a hard drive, and is accessed via a path and filename. **Buffer** refers to the file data that is stored by Multi-Edit in memory. The term **window** refers to the interface used to view and manipulate the data within a **buffer**.

There are typically two types of text editors: **stream editors** and **line editors**. A stream editor deals with all of the data as one long stream of text, while a line editor breaks the data into lines based upon delimiter characters that denote the end of a line. Multi-Edit is a line editor and thus must break the data into lines of text. How it does this depends upon the type of file that is being loaded.

When working with text files there are a number of different ways that lines are represented. Under DOS/Windows, a line is denoted by a two-character string, carriage return (0x0D - CR) and linefeed (0x0A - LF), as the end of line terminators. Under UNIX systems, lines are terminated only by a linefeed character (0x0A - LF) and Apple machines only use the carriage return character (0x0D - CR). Some files use a combination of these line terminators, or even use a different character altogether.

When loading a file in Multi-Edit, the file type must be determined first. Multi-Edit accomplishes this automatically or uses a user-defined option. If the File Prompt dialog is used to open a file, the file type option may be used. If this option is set as "Default", then Multi-Edit checks the file extension settings for this file. If there is no set up data for the extension or the file option is set to Auto-Detect, Multi-Edit will load the first 4096 characters and count all CR and LF characters. Based upon the results of this test, the line terminators are set to the type found most often and the text is then scanned and broken into lines, which are stored in a linked list excluding the line terminating character(s). When the user defines the file type, then the line terminators are set to this type and the above process is used to split the file into lines. When a file is saved, the lines in the linked list are reassembled into a stream, the line terminators are inserted at the end of each line, and then the file saved to disk.

A new feature in Multi-Edit is the **FilePane Add-On**. This provides convenient access to a set of commonly used files or directories, similar to the favorites menu in a web browser. Files and directories can be added via the toolbar buttons on the Favorites pane, or the current file / directory can be added via right-clicking on the Context menu. Directories default to a depth of one level, which can be adjusted via the Favorites List dialog. The depth controls how deep the tree will be under a directory favorite. Setting the depth too low can have a performance impact when the tree is created and refreshed.
A Multi-Edit file buffer is actually a data record that contains important information about a file such as its filename, size, date and time loaded, and pointers to the first and last line records in a double linked list. Since there is no direct way a user can access this information, a Multi-Edit window is created containing a field that points to a buffer record. Each buffer record has at least one, and sometime more than one, window pointing to it.

A Multi-Edit window is used to manipulate the data of a file via the Multi-Edit buffer. As mentioned before, each file is stored internally in a Multi-Edit buffer and has at least one window pointing to it. A window can be visible or hidden and is what is seen and used by the user. Each window generally shows a portion of a file, except for small files that can be viewed completely within the window. A cursor is used to represent the position that text will be inserted at. Since Multi-Edit is a line editor, only the line the cursor is on can be directly manipulated, therefore the cursor must first be positioned to a line and column before editing can be accomplished. When a second window points to the same buffer, the windows are said to be linked. This is indicated by the linked chain bitmap beside the window letter in the lower left hand corner of each window. Deleting a window will remove the associated buffer.

**Splitting** a window doesn't literally split a window, but rather creates a new window and sizes both the original window and the new window to each be half the size of the original window so they appear in the same area that the original window occupied. The big difference between two separate windows and split windows is that split windows are considered to share the same screen number. When two windows share the same screen number, they will both react to the same commands. For instance, when one is brought to the foreground, both are shown in the foreground.

A related feature is called Single Window mode. This feature was implemented to provide an emulation of Brief style windows. When enabled, each window is resized to fill the entire client area and only one window is viewed at a time unless the split window feature is used. If a window is split while in Single Window mode, a new window is created and is automatically linked to the current window, and then the windows are split as previously described. The difference is seen when switching to another window. When a new window is selected, it is first determined if the window is already being shown on the screen, and if it is, a new window is created, resized to the same size and position as the original window, linked to the selected buffer, and then shown. If the original window was a linked window, then it is deleted. If the selected window is not one of the windows shown already, then it will be resized to the same size and position as the original window and shown. The only way to get back to showing a single window is to do an unsplit. When closing a window in Single Window mode, the current window is deleted and replaced with the next visible window.

### Organizing Windows

**Split** allows you to divide your current window, having two adjacent windows occupying the screen area previously occupied by the original window. You can use Split to view two or more files at the same time, or view two or more parts of the same file simultaneously. The latter is called linking.

Windows created with Split differ from full screen windows in one important aspect. All "children" of the original window share what we refer to as a virtual screen. All windows associated with the same virtual screen are updated simultaneously - if one is redrawn, they all will be. For this reason they are tiled and cannot overlap.

Resizing of split windows resizes other windows sharing the screen. In addition, windows are resized when the client area changes.

When Split is invoked, a dialog box with four arrows appears. Each arrow points in a different direction. Your arrow choice determines where the next window appears. After choosing where you want a split window to appear, the Link Window dialog box appears (a standard Multi-Edit Window List).
Linking is optional. Select a window from the list to which you would like to link the first window, or press Esc to cancel linking. If you decide not to link, the original window's file will be loaded into the split one.

Next will move you to the next window in alphabetical order. Hidden, minimized and compiler error windows are skipped over.

Previous moves you to the previous window in reverse alphabetical order. Hidden, minimized and compiler error windows are skipped over.

The Adjacent Window function switches you to an adjacent window in a split-window or tiled situation. This is useful for toggling between two windows that you have set up by splitting the screen. When implemented, a Split Window dialog box appears. Select the direction of the split by pressing an arrow button on the dialog, or select Cancel to cancel the operation.

Hide conceals the current window. After that, Next and Previous will skip over it. To unhide windows, select List, highlight the desired window and press the Unhide button, or select it from The "Quick Pick" Window List.

Zoom allows you to instantly maximize a window within the Multi-Edit environment or return it to its previous size. Mouse users can achieve the same results by clicking the left button on the Minimize/Maximize buttons, or double-clicking on the window's title bar.

Minimize All changes all editing windows into icons. The icon displays the letter name of the window and the file name.

To return a minimized window to its previous size, first make the minimized window the current window using the Window List, then select Zoom from the Window menu, or select it from the "Quick Pick" Window List. Mouse users can double-click the left button on a minimized window's icon to return it to its previous size.

Link allows you to have two or more windows containing the same file. You may edit in any one of the linked windows, and the changes will be reflected in the other(s). This is very handy if you need to edit a file in two or more places.

Link will display a standard Multi-Edit Window List box, from which you can choose the file to which you wish to link.

---

*By default, files loaded multiple times in Multi-Edit are not linked.*

Unlink cancels the link between the current window and any other windows linked to the current window.

Arrange Icons arranges your minimized file icons into neat, orderly rows and columns.

Cascade provides ways to group your windows in a cascade arrangement (like tabbed index cards).

Tile vertical allows you to arrange your non-minimized windows into a vertically tiled arrangement.

Tile horizontal allows you to arrange your non-minimized windows into a horizontally tiled arrangement.
"Quick Pick" Window List

Also shown at the bottom of the Window menu is a list of loaded files. You can select a file from this list, and Multi-Edit makes that window the active window. All files will be listed, even minimized ones. If you select a file that you currently have minimized, Multi-Edit will return it to normal size.

Only nine files can be listed in the "quick-pick" window list due to space limitations. If you have more than nine files loaded and want to choose one of the files not listed in the list, select the More Windows menu selection. You will be presented with a list box of all your currently loaded files. Select one, and it will be made the active window.

Navigating Windows

Using the Tab Bar

The optional tab bar provides another way of quickly switching between windows. When it is enabled, a tab will be available for every visible window and possibly minimized and hidden windows when the appropriate option is enabled, that Multi-Edit has open. The appearance of the tabs is very configurable and can be found in Tools | Customize | Windowing | TabBar.

Clicking on a tab is all that is required to switch to the selected window. This is much easier than bringing up the Window list and selecting a window from it. Also being able to have the tabs sorted can make it easier to locate a desired window.

Using the Window List

The Window | List command opens a standard Multi-Edit Window list box that displays information about open windows. You will see the letter names of the windows, the names of the files loaded into the windows, and the path of each loaded file.

While you are in this dialog box, you can delete a window (press the Delete key on a selection), save a window, hide a window (hiding a window allows you to skip over a window when you move sequentially from window to window) and minimize a window. Each operation affects only the highlighted file in the window list, except those buttons that specifically indicate all windows, such as Hide All. The Select button will select the highlighted file and allow you to edit it.

This list box, like all list boxes, has an incremental search feature. By typing the first few characters of a file name or list item, you can highlight that file name or list item in the list box. In addition, you can swap the position of the file name for the window letter, making the incremental search key off the window letter. See the appendix titled THE STARTUP MACRO for more details.

Window List Add-On

The WinList Add-On is a fully functional replacement for the Ctrl+F6 Window List dialog found on the Window menu. Via this navigation pane, you can hide/unhide, minimize/restore, save, print, and close edit windows. In addition, you can bring up the file prompt in order to open a new file. The window list can be set up to automatically sort open files by filename, full path, extension or date. Or, you can select a manual sort, and reorder the windows using the up and down buttons. Note that manually reordering the windows will turn off the automatic sort.
**Key Commands**

Use the Help | Command Map Report command to generate a list of all defined commands for the selected command map. This report will generate a text file in a Multi-Edit window and will contain a list of each of the menu and toolbar layouts as well.

**Files**

**Opening and Saving Files**

A new File Prompt has been implemented for Multi-Edit 9.10, which replaces the old Classic style prompt.

- **Up one level**: Selects the directory one level higher than the current selected directory.
- **Refresh**: Refreshes the tree and file list view.
- **Desktop**: Selects and views the desktop.
- **Properties**: Displays the currently selected files properties.
- **Create New Folder**: Creates a new folder in the currently selected directory.
- **Select All**: Selects all of the files listed in the file list view.
- **Deselect All**: Deselects the files listed in the file list view.
- **Small Icons**: Displays the files using small icons.
- **Large Icons**: Displays the files using large icons.
- **List View**: Displays the files using a list.
- **Detail View**: Displays the file details in the file list view.

**File List View**

The File list view is designed to help you search for the file you want to open. It also displays the contents of the current directory or folder, confined to the appropriate directory mask, if any. Use the arrow keys or mouse to scroll through files in search of the one you want. Use the arrow keys or mouse to highlight the file and select OK to open it. Or, place the mouse cursor on the file you want and double-click the left mouse button to open it.
Directory Tree View
The Directory tree view displays the selected directory and its subdirectories. Use the arrow keys or mouse to scroll through directories in search of the one you want. Use the arrow keys or mouse to highlight the directory and select OK to open it. Or, place the mouse cursor on the directory you want and double-click the left mouse button to open it.

Name Field / History List
If you aren't sure of a file's name or location, type as much as you know (the Name field also accepts wildcards, such as *.*). As you type, Multi-Edit will highlight the first file matching the currently entered filename (or part thereof) in the File List Box. If you find the file you're looking for, press enter or select OK. Multi-Edit will then load the currently highlighted file. If you entered a wildcard, press Enter or select OK to accept that wildcard entry and continue your search. Multi-Edit can also load multiple files at a time. Simply separate filenames by a space to load more than one file. The history list contains up to fifteen file or directory names that were opened during the current editing session. To access the History list box, click on the down arrow to the right of the field. If you want to access the items in the history list without viewing them, you can press the down arrow to cycle through the list's entries when the field has focus.

Mask List
Below the Name Field is a drop-down list box of common file masks. To access the Mask list box, click on the down arrow to the right of the field. Move through the list with the scroll bar. If you want to access the items in the list without viewing them, you can press the down arrow to cycle through the list's entries when the field has focus.

Read Only checkbox
This check box allows you to toggle the read/write status of a file when loading it.

If the file is not marked as read-only, you may check this box upon loading that file to open a read-only "copy" of the file. This does not change the read/write status of that file. The original file on disk will still be read/write, but you will have only a read-only copy of that file.

If the file is marked as read-only, you may uncheck this box upon loading that file to open that file as read/write. This enables the read/write status of that file.

Changing the Locked/Read Only setting for a file on a network server might cause unexpected problems, since it's likely some process, even one spawned through Multi-Edit, has set the attribute for a specific purpose. Changing the setting, especially to release a lock, might confuse that process as to the file's actual state. For instance, if someone has the file checked out in a VCS program, the setting might be how the program prevents others from modifying the file.

OEM Mode checkbox
Checking this box will open the current file in OEM Translate mode. It does not alter the OEM Translate setting for the rest of your Multi-Edit session. To change that setting, look under Tools | Customize | Fonts.

Type field
When opening a file, the Type field allows you to change the line terminator for the file. Multi-Edit uses this "file type" to determine both where to break lines when loading a file and which characters to insert when the Enter key is pressed during normal editing.
Default: Each extension can have its own default file type. By changing the Type field when opening a file, it is possible to override the default file type. It is also possible to tell Multi-Edit to automatically detect an extension's file type when loading it.

**DOS:** This option will load the file as a DOS file. DOS files use a carriage return and a line feed to mark the end of the line.

**Unix:** This selection will load the file as a UNIX file. UNIX uses only line feeds to mark the end of the line.

**Mac:** Selecting Mac file type will load the file as a Macintosh file. Mac files only use a carriage return to mark the end of the line.

**Binary:** Binary files have no line terminators as they are continuous streams of data; however, for easier reading, binary files use a Binary Record Length that can be specified on the fly or for a particular extension in Filename Extension Setup.

**Record Length field**
This field specifies the Binary Record Length of the file that is about to be opened. This field is only used if Binary is selected as the File Type. The record length is the number of characters that will be displayed before Multi-Edit wraps the data to a new line. This can be useful for locating addresses in binary files. If no value is specified here and the file is loaded as a binary file, the default record length as given in Filename Extension Setup will be used.

The displays and buttons at the bottom of the Edit New File dialog box pertain to specialized file manager options available from within Multi-Edit. These operations allow you to quickly switch directories to your session's "working directory" (as defined in the Session Manager), copy files from directory to directory, move files, etc. For more information, see File Manager Operations.

**Work (working directory) Field**
The Work field contains the current working directory. To the right of this field are two buttons, which allow the working directory to be set, or displayed in the file list view and directory tree view.

Alternatively, the Explorer-style dialog (shown below) offers an interface similar to the common Windows dialogs with which you are familiar. To use this dialog, enable the **Explorer Style** option under **Tools | Customize | Files**.

**Properties**

**File | Properties** displays the current file’s name, the amount of disk space the file occupies, and the date and time the file was last modified. You can also change the file Locked/Read only status of the file and "type". See the Open section or the Filename Extensions section for more information on file type. If the current file type is binary then the option to change the line length will be enabled.
Changing the Locked/Read Only setting for a file located on a network server can yield undesirable results. The undesirable results would occur if the remote file were locked/unlocked by the server for a reason unrelated to what you were doing while opening the file. For instance, it might be locked by a VCS program, even one invoked through ME; unlocking it manually is bad juju, because it would throw the VCS' state out of sync with reality.

Sharing

Multi-Edit's Network File Locking feature prevents more than one person at a time from having write-access to any file. Other network users may load the file, but it will be marked Read Only, and no modifications will be permitted. Multi-Edit is always shipped with Network File Locking disabled. To enable this feature, from the Tools menu go into the Customize dialog box and select the Backups | Temp files | Autosave option. At the bottom of that dialog box you will find the Network file locking option. Toggle it on [X].

If Multi-Edit was not exited properly, i.e. there was a power failure, reboot or system crash, any files which were loaded at the time will be marked Read Only. In order to unlock these files, you can:

Run the DOS program ATTRIB.

OR…

Load the restricted files into Multi-Edit and change the locked status from the File Information Dialog Box (Main Menu | File | Information).

File Locking also includes configuration and data files used by Multi-Edit. If the configuration files are locked, you will see this message and menu:

If File Locking is turned ON, your configuration files may become locked under the following conditions:

IF MULTI-EDIT WAS NOT EXITED PROPERLY. If you turn off your computer without exiting Multi-Edit first or if you re-boot your computer while in Multi-Edit, your configuration files will become locked. If this is the case, select the 'Unlock configuration files' option.

IF TWO USERS ARE LOGGED IN UNDER THE SAME USER ID. If two users are logged in under the same user ID, only one user can have write-access to the files. The second user may load Multi-Edit and select the 'Ignore and leave files locked' option. The 'Unlock Configuration Files' SHOULD NOT be used in this situation.

Backup and Autosave

Autosave allows you to configure Multi-Edit to automatically save all modified files based on two different methods. The first method allows you to save files based on x seconds of keyboard inactivity. The second method allows you to save files at absolute intervals (every x seconds). Each method also includes the option of saving only the files loaded or the complete editor status at the same time (this includes all files loaded and the window sizes, etc). Setting either field to zero will turn the option off.

<table>
<thead>
<tr>
<th>Seconds of keyboard inactivity</th>
<th>0 = OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save status</td>
<td></td>
</tr>
<tr>
<td>Absolute interval in seconds</td>
<td>0 = OFF</td>
</tr>
<tr>
<td>Save status</td>
<td></td>
</tr>
</tbody>
</table>
Tool and Navigation Panes

Working with the Tool and Navigation Panes

The View Menu allows you to select and display tabbed dialogs on the Tool and Navigation panes. A checkmark preceding a menu item indicates that that item is active and will be displayed in the appropriate Pane. Selecting the Tool or Navigation Pane menu item will display that pane with the corresponding (active) tabbed dialogs. Note that the menu items below the Tool and Navigation Pane are related specifically to the Tool Pane while the items following the second separator are related to the Navigation Pane. The View menu is used to control the Tools and Navigation panes and is separated into three sections: main pane entries, the Tool panes entries and the Navigation pane entries.

The Tools pane is a horizontal window at the bottom of Multi-Edit screen that can contain a number of dialogs for showing results from different macros and the Navigation pane is a window that can be shown on either the left or right of the Multi-Edit screen and is used to show dialogs used for navigation of source files.

The main pane entries are used to show the Tool or Navigation pane. These entries will be grayed if there are no active dialogs of the appropriate type enabled. When a dialog is active, selecting one of these entries will toggle the showing of the specified pane.

The Tools pane entries are used to create the selected Tools pane dialogs, or, by default, toggle the focus between the dialog and the client area. A checkmark will be shown in the menu when the dialog has been created. The global variable, !ViewDlgToggle can be set in the Startup.cfg file that will change the behavior of the Tools pane menu entries. When this global variable is set, instead of toggling the focus between the dialog and the client area, it will will toggle the existence of the dialog. Below are the Tools pane entries.

<table>
<thead>
<tr>
<th>Dialog</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bookmarks</td>
<td>Show the list of bookmarks.</td>
</tr>
<tr>
<td>Collapse</td>
<td>Show the collapse dialog.</td>
</tr>
<tr>
<td>Find List 0</td>
<td>Show the first Find list.</td>
</tr>
<tr>
<td>Find List 1</td>
<td>Show the second Find list.</td>
</tr>
<tr>
<td>Ftp Results</td>
<td>Show the results of the last Ftp session.</td>
</tr>
<tr>
<td>Output</td>
<td>Show the results of the last compile.</td>
</tr>
<tr>
<td>Paste Buffers</td>
<td>Show the paste buffers.</td>
</tr>
<tr>
<td>Preview Pane</td>
<td>Show the preview pane.</td>
</tr>
<tr>
<td>Tasks</td>
<td>Show the list of background tasks.</td>
</tr>
</tbody>
</table>
The Navigation pane entries are used to create the selected Navigation pane dialogs, or, by default, toggle the focus between the selected dialog and the client area. A checkmark will appear in the menu when the dialog has been created. The global variable !ViewDlgToggle can be set in the Startup.cfg file that will change the behavior of the Navigation pane menu entries.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Manager</td>
<td>Show the WebLair code manager dialog.</td>
</tr>
<tr>
<td>Project</td>
<td>Show the Project manager dialog.</td>
</tr>
<tr>
<td>System</td>
<td>Show the System dialog, which shows the list of global variables and loaded macros.</td>
</tr>
<tr>
<td>Tags</td>
<td>Show the Tags dialog.</td>
</tr>
</tbody>
</table>

**TmplPane Add-On**

The TmplPane Add-On provides convenient access to a set of templates appropriate for the currently selected file. After installation, which you've just completed, a new Template Pane option will be available on the View menu. Selecting it will turn the pane on. By default, the Template Pane will 'follow' the current window; as the focus changes, the template sets shown will match the file extension for the current window.

For most languages, the top section will show language specific templates, while the bottom section will show the global templates. For web related extensions (where the LANG buffer property equals HTML) you can choose ASP, Script, WebLair, or HTML templates for both the top and bottom section. ASP and Script templates are found by looking at the ASP_DEF and SCRIPT_DEF buffer properties.

The template set choices can also be overridden by setting a buffer property TMPLPANE_TOP or TMPLPANE_BOTTOM in the extension or language setup. To use the Template Pane, select a template from the list and double-click it to expand it at the current cursor position.

**Add-On package setup**

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install.lst</td>
<td>The package installation file</td>
</tr>
<tr>
<td>TmplPane.dat</td>
<td>Empty file</td>
</tr>
<tr>
<td>TmplPane.del</td>
<td>Update script to remove command and menu entries.</td>
</tr>
<tr>
<td>TmplPane.s</td>
<td>The TmplPane macro file</td>
</tr>
<tr>
<td>TmplPane.txt</td>
<td>This readme file</td>
</tr>
<tr>
<td>TmplPane.upd</td>
<td>Update script to add commands and menu entries.</td>
</tr>
</tbody>
</table>
**FilePane Add-On**

The FilePane Add-On provides convenient access to a set of commonly used files or directories, similar to the Favorites menu in a web browser. Files and directories can be added via the toolbar buttons on the Favorites Pane, or the current file/directory can be added via the right-click context menu. Directories default to a depth of 1 level, which can be adjusted via the Favorites List dialog. The depth controls how deep the tree will be under a directory favorite. Setting depth too high can have performance impact when the tree is created and refreshed, you may need to play with these settings to find what works best for your environment.

### Add-On package setup

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install.lst</td>
<td>The package installation file</td>
</tr>
<tr>
<td>FilePane.dat</td>
<td>Empty file</td>
</tr>
<tr>
<td>FilePane.del</td>
<td>Update script to remove command and menu entries.</td>
</tr>
<tr>
<td>FilePane.mac</td>
<td>The compiled FilePane macro</td>
</tr>
<tr>
<td>FilePane.s</td>
<td>The FilePane macro source file</td>
</tr>
<tr>
<td>FilePane.txt</td>
<td>This readme</td>
</tr>
<tr>
<td>FilePane.upd</td>
<td>Update script to add commands and menu entries</td>
</tr>
</tbody>
</table>

**Window List Add-On**

The WinList Add-On is a fully functional replacement for the Ctrl+F6 Window List dialog found on the Window menu. Via this navigation pane, you can hide/unhide, minimize/restore, save, print, and close edit windows. In addition, you can bring up the file prompt in order to open a new file. The window list can be set up to automatically sort open files by filename, full path, extension or date. Or, you can select a manual sort, and reorder the windows using the up and down buttons. Note that manually reordering the windows will turn off the automatic sort.

### Add-On package setup

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install.lst</td>
<td>The package installation file</td>
</tr>
<tr>
<td>WinList.dat</td>
<td>Update script to update MeConfig.db file</td>
</tr>
<tr>
<td>WinList.del</td>
<td>Update script to remove MeConfig and command entries</td>
</tr>
<tr>
<td>WinList.s</td>
<td>WinList macros</td>
</tr>
<tr>
<td>WinList.txt</td>
<td>This file</td>
</tr>
<tr>
<td>WinList.upd</td>
<td>Update script to add commands to the command file</td>
</tr>
</tbody>
</table>
Search and Replace

Find and Replace

When it comes time to locate text and possibly replace it, nothing is easier than using the Find and Replace dialogs. From these dialogs, it is easy to set up different kinds of searches from literal text to search using a regular expression. Below are some tips on how to use these dialogs to quickly locate the text that needs to be found.

- The Find and Replace dialogs are actually part of a set of search related dialogs that appear in a tabbed dialog with tabs for each of the other dialogs. The Find and Replace dialogs are used to quickly search and replace text in files that are currently loaded into Multi-Edit. The main difference between the two dialogs is the Replace dialog has an additional text field used to specify the replacement text. Most of the other options operate exactly the same in either dialog.
- The first thing to set when doing a find is to set the type of search that is desired. Selecting one of the Type radio buttons does this.
- The Literal type is used when simple text is being searched for. It will cause the search engine to locate text exactly as entered into the Search for field, ignoring the case of the characters unless the Case sensitive option is checked. Since the search engine can only find simple text, it can do so at a much faster rate than the other types of searches.
- Regular expression searches are useful for locating variable length text, multiple words, and more complex text patterns. When this option is checked, a number of characters, called metacharacters, take on special meaning (see the section "Regular Expressions" for more details about what each of the supported characters does). If the string entered into the "Search for" field does not contain any of the special characters the search engine will revert to doing a literal search, which is usually faster. The time it takes to search using regular expressions depends upon the particular expression. The Alias button makes it easier to use regular expressions by saving commonly-used expressions under a name you can quickly enter into the "Search for" field. In addition, clicking the arrow button to the right of the "Search for" field drops down a menu listing the metacharacters available for the regular expression style currently in effect; clicking an item inserts it into the field at the current caret position.
- The Word/Phrase search is used when looking for a literal phrase that might cross line boundaries. Note that this option is only available when searching loaded files and thus cannot be used when doing multiple file searches.
Searches normally scan the complete line but can be restricted to search between specific columns by enabling the Columns option and setting the Start and Stop column numbers. Usually a search only searches the current window and when it finds a match, it stops and highlights the matching text. If the Search all windows options is checked, when the search engine can't find a match in the current window, it will continue searching in other windows until a match is found or it wraps back to the current window. Also if the "Search auto wraps at TOF/EOF option in the Search Options dialog is checked, then the search engine will continue searching after wrapping to the top or bottom of the window. See the Search Options help for more information about each of these options.

If a list of all matches needs to be found, then selecting the All button instead of the Search button will cause the search engine to continue finding matches and inserting an entry for every match in a Find list. There can be multiple Find lists but the user interface only supports two and is selected by picking one from the File list option. The purpose for multiple Find lists is so a File Find can be performed, then allow for searching the results of that File Find for some other text without the need to navigate through the tree to load the next result. See the Find List topic for more information.

Most of the above holds true for the Replace dialog as well, except that now the Replace button becomes activated when a match is found so that the matching text can be replaced with the replacement string. Regular expressions are also supported in the Replace string and the additional button for inserting them is also available.

File Find and Replace

The File Find and Replace dialogs are also part of the Search tabbed dialog and are used when multiple files need to be searched which are not loaded in Multi-Edit. Searching multiple files can take a long time, especially if they each need to be loaded into Multi-Edit and then searched. Therefore, File find uses a slightly different search engine to allow speeding up the searching of files. Since only part of each file is ever loaded, the Phrase type searches are not supported but regular expressions are still available.
When searching for text in multiple files, a filespec and a starting path can be provided. Both of these fields can have multiple entries by separating them with a semicolon. When the Filespec field is left blank, the search engine will use *.* as the filespec and when the Starting path field is blank, the current path is searched. If the Search for field is left blank, then a list of all of the files matching filespec will be added to the list.

When a project is defined, the "Limit search to project list" option is made available. This option, when checked, will cause the search engine to search the files in the project list of the files in the paths specified in the Starting path field.

A File Find or File Replace always generates an entry in one of the Find lists. See Find and Replace for the reason there are multiple find lists. When a search is started, a new top-level tree node is inserted showing the parameters of the search. If there are matches found in a file, then a second level node for each file is inserted into the tree. When the "List all occurrences" option is checked, then a third level node for each found item in the file is inserted into the tree. See the Find List topic for using the lists.

**File Replace**

*Search | File Replace* is used when replacements are to be done in multiple files, including those that are not currently loaded into Multi-Edit.

This is similar to the File Find dialog, but after all of the matches are found and listed in the Tools Pane, the files are then opened and the replacements can be performed.

---

*All find dialog options and search/replace strings are saved in the Session information.*

---

### Using Regular Expressions

**Regular Expressions**

Regular expressions behave like sophisticated wildcards (actually it's more accurate to say that wildcards are a very small subset of regular expressions), using symbols with special meanings, called *metacharacters*.

Regular expressions, or regexes for short, let you do a number of things not possible in a literal-text search, such as requiring the search text to occur at the beginning or end of a line, searching for text that fits a certain pattern but isn't bound to specific characters, or searching across multiple lines. Multi-Edit provides support for three regex styles: Multi-Edit Classic, Unix-compatible, loosely based on the X/Open standard for extended expressions, and Perl 5.8-compatible. The Perl and UNIX styles support maximal and minimal matching, and UNIX expressions can span lines (the Perl style will soon have this capability as well).
So given the following text in a document:

    Test-drive the car of your choice.
    Wow!!! This handles great!
    What is the price?

The UNIX regex pattern

    Test?*$Wow?*$What

Will match everything from "Test" on the first line to "What" on the third line.

Regular Expressions are a very powerful search tool, but one that carries a cost. Unless you use them often enough to become thoroughly familiar with them, on the odd occasion you do use them you'll find yourself puzzling through a pattern to make it find exactly what you're after, or worse, continually looking up the meanings of the various metacharacters. This can be particularly annoying when you've spent a lot of time figuring out a really great regex for some esoteric search, only to find that you can't remember it later. As well, regular expressions can appear quite cryptic and difficult to read, and leaving out or misplacing a single character can cause your search to fail, or to match against a totally unexpected string. Multi-Edit supports three different regex styles, which means you must be careful not to forget which style you're using.

Multi-Edit provides two tools to make regexes easier to use: drop-down reminder menus, and expression aliases. The drop-down menus list and briefly define all the available metacharacters for the currently-selected regular expression style. To access them, simply click on the arrow button to the right of the "Search for" text box on any of the search menus; selecting a menu item inserts its associated metacharacter in the text box at the current cursor position. Note that the Perl-style drop-downs contain submenus grouped by type, with some of the most commonly-used metacharacters at the top of the main menu. If you're performing a replacement as well, there's a second drop-down menu button to the right of the "Replace with" text box, listing the expressions valid in a replace expression. Note that since replacement expressions aren't part of most regular expression standards, they vary from program to program; we've chosen replacement expressions in common use in major packages, such as Perl, with occasional variances where an expression wouldn't make sense in Multi-Edit's context, or would collide with something else.

Expression aliases allow you to save expressions under names of your choice, in any or all regex styles, with a short description; to use such a saved expression later, simply enter its name in the text box, and Multi-Edit will automatically expand it into the full expression internally. There are a number of aliases already defined for your use, and you can easily add your own to the list. The only syntactic requirement on alias names is that you must enclose them in angle-brackets, like this: <alias_name>. You'll probably also want to keep those names short, for ease of use. Aliases representing filenames and paths fully support long filenames, with embedded spaces.

Many of Multi-Edit's system macros use regex aliases; the search macro itself is one of these. You can use aliases in macros of your own composition, as well, using the techniques illustrated in the system macro header file RegExp.sh. The most important of these is to call the ReTranslate() macro to expand any aliases in your search string into their actual text before you run your search. You can also call ReSelectAlias() to display the Regular Expression Aliases dialog, which lists the currently-defined aliases; the macro's return value is the selected alias.

**IMPORTANT:** never rename or modify any of the predefined aliases unless you're very familiar with the workings of Multi-Edit's system macros, and quite sure of what you're doing; it's all too easy to break major program subsystems, since, as mentioned above, many system macros depend on them.
The Classic- and UNIX-style regular expressions deviate from the X/Open standard in that they don't treat strings of characters as groups, unless they're explicitly grouped using the grouping expressions specific to each style. The Perl 5.8-style regexes follow the spec. For example, Given the text

"The cat ate the car"

and the regular expression

"cat|r"

which is valid in all three styles, the Classic and UNIX styles match on the words "cat" and "car", while the Perl 5.8 style matches on "cat" and the final "r" in "car". This is because the Perl 5.8 engine interprets the regex (including group metacharacters for clarity) as

"(cat)|(r)"

while the other engines interpret it as

"ca(t|r)"

which does not, as observed, follow the X/Open specification. And now for something completely different.

---

**Classic Regular Expressions**

**Search String Expressions**

**Expressions that anchor a match to a location in the text:**

<table>
<thead>
<tr>
<th>exp</th>
<th>Text matched by &quot;exp&quot; must occur at the beginning of a line</th>
</tr>
</thead>
<tbody>
<tr>
<td>%exp</td>
<td>exp$</td>
</tr>
<tr>
<td>exp$</td>
<td>Text matched by &quot;exp&quot; must occur at the end of a line</td>
</tr>
</tbody>
</table>

**Quantifiers that determine how many times an expression can match:**

<table>
<thead>
<tr>
<th>C*</th>
<th>Match zero or more occurrences of the previous character (&quot;C&quot;) or grouped expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>C+</td>
<td>Match one or more occurrences of the previous character (&quot;C&quot;) or grouped expression</td>
</tr>
</tbody>
</table>

**Expressions that allow an "either/or" match:**

| C|C  | Match either the left or right character ("C") or grouped expression |

**Expressions that group and capture text:**

<table>
<thead>
<tr>
<th>{exp}</th>
<th>Group the text matched by &quot;exp&quot; and capture it in one of #0, #1, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(determine group numbers by counting &quot;{&quot; characters from the left of the search regex)</td>
</tr>
</tbody>
</table>

**Expressions that match single characters:**

<table>
<thead>
<tr>
<th>?</th>
<th>Match any character</th>
</tr>
</thead>
<tbody>
<tr>
<td>[list]</td>
<td>Match any character in the bracketed list of characters</td>
</tr>
<tr>
<td>[~list]</td>
<td>Match any character except one in the bracketed list</td>
</tr>
<tr>
<td>@C</td>
<td>Match the character &quot;C&quot; as a literal character</td>
</tr>
<tr>
<td>@a</td>
<td>0x07 BEL</td>
</tr>
<tr>
<td>@b</td>
<td>0x08 BS Backspace</td>
</tr>
<tr>
<td>@f</td>
<td>0x0C FF Page Break (Formfeed)</td>
</tr>
<tr>
<td>@n</td>
<td>0x0A LF Newline (Line Feed)</td>
</tr>
<tr>
<td>@r</td>
<td>0x0D CR Carriage Return</td>
</tr>
<tr>
<td>@t</td>
<td>0x09 HT Tab (Horizontal Tab)</td>
</tr>
<tr>
<td>@v</td>
<td>0x0B VT Vertical Tab</td>
</tr>
<tr>
<td>@xHH</td>
<td>Byte specified by the two hexadecimal digits &quot;HH&quot;</td>
</tr>
</tbody>
</table>
Replace String Expressions

\$
\quad\text{Insert a line break}

\#n
\quad\text{Insert the text captured by group number "n" (the group beginning with the (n - 1)th "}", since Classic groups are zero-based)}

&
\quad\text{Insert the complete matched text}

%
\quad\text{Delete the previous character}

^\quad\text{Leave cursor at this position in the replaced text}

Unix Regular Expressions

Search String Expressions

Expressions that anchor a match to a location in the text:

\^exp
\quad\text{Text matched by "exp" must occur at the beginning of a line}

exp\$
\quad\text{Text matched by "exp" must occur at the end of a line}

Quantifiers that determine how many times an expression can match:

C*
\quad\text{Match zero or more occurrences of the previous character ("C") or grouped expression (as many as possible)}

C+
\quad\text{Match one or more occurrences of the previous character ("C") or grouped expression (as many as possible)}

C@
\quad\text{Match zero or more occurrences of the previous character ("C") or grouped expression (as few as necessary)}

C#
\quad\text{Match one or more occurrences of the previous character ("C") or grouped expression (as few as necessary)}

Expressions that allow an "either/or" match:

C|C
\quad\text{Match either the left or right character ("C") or grouped expression}

Expressions that group and capture text:

(exp)
\quad\text{Group the text matched by "exp" and capture it in one of \0, \1, etc. (determine group numbers by counting "(" characters from the left of the search regex)}

Expressions that match single characters:

.  
\quad\text{Match any character}

[\text{list}]
\quad\text{Match any character in the bracketed list of characters}

[^\text{list}]
\quad\text{Match any character except one in the bracketed list}

\|C  
\quad\text{Match the character "C" as a literal character}

\|a  
\quad0x07 \quad\text{BEL}

\|b  
\quad0x08 \quad\text{BS Backspace}

\|f  
\quad0x0C \quad\text{FF Page Break (Formfeed)}

\|n  
\quad0x0A \quad\text{LF Newline (Line Feed)}

\|r  
\quad0x0D \quad\text{CR Carriage Return}

\|t  
\quad0x09 \quad\text{HT Tab (Horizontal Tab)}

\|v  
\quad0x0B \quad\text{VT Vertical Tab}

\|xHH  
\quad\text{Byte specified by the two hexadecimal digits "HH"}
Replace String Expressions

$  Insert a line break
\n  Insert the text captured by group number "n" (the group beginning
with the (n - 1)th ")", since UNIX groups are zero-based
&  Insert the complete matched text
%  Delete the previous character
^  Leave cursor at this position in the replaced text
\a 0x07  BEL
\b 0x08  BS  Backspace
\f 0x0C  FF  Page Break (Formfeed)
\n 0x0A  LF  Newline (Line Feed)
\r 0x0D  CR  Carriage Return
\t 0x09  HT  Tab (Horizontal Tab)
\v 0x0B  VT  Vertical Tab
\xHH  Byte specified by the two hexadecimal digits "HH"

Perl 5.8 Regular Expressions

Search Strings Expressions

Expressions that match single characters:
\a 0x07  BEL
\b 0x08  BS  Backspace; only supported in character classes
\e 0x1B  ESC  Escape
\f 0x0C  FF  Page Break (Formfeed)
\n 0x0A  LF  Newline (Line Feed)
\r 0x0D  CR  Carriage Return
\t 0x09  HT  Tab (Horizontal Tab)
\OOO  Character specified by three octal [0-7] digits "OOO"
\xH, \xHH  Character specified by one "H" or two "HH" hexadecimal digits
\x{H..H}  Character specified by any string of hexadecimal digits "H..H"
Mainly intended for multi-byte Unicode, but can be corrupted by some regex
operations, such as backtracking
\cC  Named control character: \cZ = Ctrl-Z; some platforms require that C be capital

.  Match any character
[list]  Match any character in the bracketed list of characters
[^list]  Match any character except one in the bracketed list
\C  Match the character "C" as a literal character
\w  Word character, [a-zA-Z0-9_]  
\W  Non-word character, [^a-zA-Z0-9_]  
\d  Decimal digit, [0-9]  
\D  Non-digit, [^0-9]  
\s  Whitespace character, [\n\r\f\t ]  
\S  Non- whitespace character, [^\n\r\f\t ]
Expressions that anchor a match to a location in the text, or test a condition without consuming characters:

^exp      Text matched by "exp" must occur at the start of the searched text or after a line break
exp$      Text matched by "exp" must occur at the end of the searched text or before a line break
\A       Start of search string
\Z       End of string or before EOS newline
\z       End of string
\G       Beginning of current search
\b       Word boundary; between \w and \W or BOS/EOS
\B       Not-word boundary

(?!exp)   Negative lookahead; matches if the text after the current position doesn't match, but doesn't consume that text
(?!<exp)  Negative lookbehind; matches if the text preceding the current position doesn't match; consumes no characters

Expressions that allow an "either/or" match:

exp|exp    Match the left expression; if that fails, attempt to match the right one

Expressions that group and capture text:

(exp)     Group the text matched by "exp" and capture it in one of \1, \2, etc. (determine group numbers by counting "(" characters from the left of the search regex)
(?P<name>exp) Group and capture "exp" into named group "name"
\n   Results of nth earlier group submatch
(?>exp)   Group "exp", disallowing backtracking
exp|exp    Attempt to match the left expression; if that fails, try the right expression

Quantifiers that determine how many times an expression can match:

exp*      Match "exp" zero or more, but as many as possible
exp+      Match "exp" one or more times, but as many as possible
exp?      Match "exp" zero or one times
exp{n}    Match "exp" exactly n times
exp{n,}   Match "exp" at least n times, but as many as possible
exp[min,max] Match "exp" between "min" and "max" times, but as many as possible
exp*?     Match "exp" zero or more times, but as few as necessary
exp+?     Match "exp" one or more times, but as few as necessary
exp?? Match "exp" zero or one times, but as few as necessary

texp{n,}? Match "exp" at least n times, but as few as necessary

exp{min,max}?
Match "exp" between "min" and "max" times, but as few as necessary

exp*+ Match "exp" zero or more, but as many as possible; never backtrack
exp++ Match "exp" one or more times, but as many as possible; never backtrack
exp?+ Match "exp" zero or one times; never backtrack
exp[n] Match "exp" exactly n times; never backtrack
exp[n,]+ Match "exp" at least n times, but as many as possible; never backtrack
exp{min,max}+
Match "exp" between "min" and "max" times, but as many as possible; never backtrack

(?(condition)true_exp|false_exp)
If/else match; if "condition" is true, attempt to match "true_exp"; else attempt to match "false_exp"; the condition can be a capture group or a lookahead number

(?(condition)exp)
If match; if "condition" is true, attempt to match "exp"; the condition can be a capture group or a lookahead number

Modes:

i Case-insensitive match
m Multi-line: ^ and $ match next to embedded \n
s Single-line: . dot metacharacter matches newlines
x Ignore whitespace and allow EOL comments in expression

Mode modifiers:

(?!msx) Turn listed modes on
(?!msx) Turn listed modes off
(?!modes:msx) Toggle listed modes within parentheses

Quote delimiters:

\Q Quote all following metacharacters
\E End a span started with \Q

Replace String Expressions

$ Insert a line break
\n Insert the text captured by group number "n" (the group beginning with the nth "," since Perl 5.8 groups are one-based)
\0 Insert the complete matched text
% Delete the previous character
^ Leave cursor at this position in the replaced text
!(macro)
Run Multi-Edit macro within the parentheses; if the left parenthesis is absent, the text is interpreted normally, while a missing right parenthesis will display an "Error 1018" dialog.
To force the string "!(" to be interpreted as normal text, preface the "!" with a backslash: \!\(".
\0 Insert the complete matched text
\a 0x07 BEL
\b 0x08 BS Backspace
\f 0x0C FF Page Break (Formfeed)
Regular Expression Quantifier Types

Quantifiers are the metacharacter expressions that determine how many times a preceding expression can repeatedly match. There are three basic kinds: lazy, greedy, and possessive. Not all kinds are supported by all three regular expression types in Multi-Edit.

Minimal, or lazy, quantifiers match only as many characters as are necessary, even though a longer match might be possible. The lazy quantifiers for each style are:

- **Classic**: N/A
- **UNIX**: C@, C#
- **Perl 5.8**: exp*?, exp+?, exp??, exp{n}?, exp{n,}?, exp{min,max}?

Maximal, or greedy, quantifiers match as many characters as possible, although (Perl-compatible only) they will backtrack to shorter matches if necessary to find a match for the entire search expression. Note that unless otherwise qualified, Perl 5.8 quantifiers are greedy by default, although they are not multi-line (use the multi-line mode modifier "(?m)" before a Perl regex to make it match across line boundaries). The older UNIX expressions do not match across line boundaries. The greedy quantifiers for the three styles are:

- **Classic**: C*, C+
- **UNIX**: C*, C+
- **Perl 5.8**: exp*, exp+, exp?, exp{n}, exp{n,}, exp{min,max}

Possessive quantifiers (Perl-style only) behave like greedy quantifiers, except that they never backtrack; they always match the longest possible string, which can cause the search string to fail to match where a greedy quantifier might backtrack to an overall match, but they also avoid certain infinite-loop conditions that can happen with poorly-constructed regexes. The possessive quantifiers are:

- **Classic**: N/A
- **UNIX**: N/A
- **Perl 5.8**: exp*+, exp++, exp?+, exp{n}+, exp{n,}+, exp{min,max}+

For example, given the following code fragment

```c
if ( ( Wrap_Stat && Doc_Mode ) || ( Format_Stat && Indent_Style ) ) {
  Use_Format_Line = True;
}
```

the greedy regex "\(\.*\)\)", which matches the text between opening and closing parentheses, inclusively, matches

\( ( \text{Wrap\_Stat} \&\& \text{Doc\_Mode} ) \) \( \| \) \( ( \text{Format\_Stat} \&\& \text{Indent\_Style} ) \) \)

whereas the lazy regex "\(\(.@\)\)" for UNIX, "\(\(\.*\)\)" for Perl, when run against the same text, matches

\( ( \text{Wrap\_Stat} \&\& \text{Doc\_Mode} ) \) \( \| \) \( ( \text{Format\_Stat} \&\& \text{Indent\_Style} ) \) \)
The Perl 5.8 possessive regex "\(.*+\) \) \{", run against the same text, finds no match, because it's already consumed the last closing parenthesis with the subexpression ".*+" and can't backtrack to "give up" the space and parenthesis necessary to match the entire expression. Changing the regex to the greedy "\(.*\) \) \{" matches against

\(( ( \text{Wrap\_Stat} && \text{Doc\_Mode} ) || ( \text{Format\_Stat} && \text{Indent\_Style} ) ) \) \{
because the non-possessive form can backtrack to an earlier partial match that allows the rest of the expression to match as well.

### Find Word Under Cursor

A quick way to find the next or previous occurrence of the word under the cursor can be done by using the \text{Ctrl}+\text{Alt}+\text{Up} or \text{Ctrl}+\text{Alt}+\text{Dn} keys available in most keymaps or by using \text{Shift} or \text{Ctrl} with the right mouse button. If the word under the cursor is not already in a block mark, then hitting one of these keys will block mark the word and will move to the next or previous matching word on the next keystroke. A phrase can also be searched for in this manner, but it will need to be block marked before hitting the search keys.

### Using a Find View

The find lists are the results of doing either a Find File or a Find All. Multiple search results can be shown in a single find list since it uses a tree view. A new top-level node will be added to the top of the tree whenever a new search is done. There can be up to three levels for each search depending upon the options selected. The top-level entry, Find nodes, contains the information about the search such as the string being searched for, the file mask and the total found count. The second level entry, File nodes, contains the name and path of the file where a match was found. It can also contain the number of times the string was found in the file. The third level entry, Found nodes, are generated whenever a Find All or the List all occurrences option in the File find dialog is enabled. This level shows the line number the string was found on and the text of the line.

Moving through the tree list can be done by using either the mouse or the cursor keys. To expand and collapse the tree, use either the left/right arrow keys or click on the +/- icon in the first column. The Open and View buttons have the same function except for where the focus is left, the focus will be in the tree list after the View button is used and in the opened file if the Open button is used. The Open/View buttons change behavior depending upon which level node is active when they are selected. If a Find node is selected, the appropriate search dialog will be opened and set up to allow a new search with the same parameters to be done when one of these buttons is hit. When a File node is selected, the file specified will be loaded and the cursor moved to the first match. When a Found node is selected, the specified file is loaded and the cursor positioned to the line of the selected found line. To remove a node, the Delete key can be used.

In addition, a context menu is provided via a right click on the mouse, that allows doing all of the above with the mouse.

A special menu button (a button with a Right arrow on it) provides the ability to manipulate the tree list as a whole. From this menu, the data can be saved to disk, loaded from disk, the data exported to a text file and the list cleared. To quickly access this menu using the keyboard when in the tree control, hit the Tab key and then the space key to bring up the menu.

### Incremental Search

Multi-Edit's powerful incremental search feature allows you to perform simple searches with ease. Place the cursor at the position from which you wish to begin searching, select \text{Incremental Search}, and begin typing the string of characters for which you would like to search.
Multi-Edit will search for the string as you type it in, highlighting the next occurrence of the search string in the file. Thus, you may not need to input the whole string to find the text you’re looking for.

Incremental Search will not enter the last entered character into the search string if the search fails. In addition, the search string is not inserted into the normal search string variable so that a repeat search can be done.

While in incremental search the Alt+N and Alt+P keys will cause the search for next or previous match. This has always been available but is hard-coded into the macros and not set through the keymaps.

The incremental search feature has no dialog box (keep an eye on the Status Bar for progress), but the following keys control its actions:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esc:</td>
<td>Cancels the search and returns the cursor to its original position before the search began.</td>
</tr>
<tr>
<td>Alt+P:</td>
<td>Searches backward for the next occurrence of the current search string.</td>
</tr>
<tr>
<td>Alt+N:</td>
<td>Searches forward for the next occurrence of the current search string.</td>
</tr>
<tr>
<td>Backspace:</td>
<td>Removes the last character from the search string and returns the cursor to the previously found string.</td>
</tr>
<tr>
<td>Alt+W:</td>
<td>Performs a word search. When this option is invoked, the incremental search will ignore instances of the search string that are not complete words (i.e. will not find the string &quot;ear&quot; in the middle of the word &quot;search&quot;).</td>
</tr>
<tr>
<td>Alt+C:</td>
<td>Activates case-sensitive searching. When this option is selected, the incremental search will treat the search string as case sensitive. By default, the string is considered case-insensitive.</td>
</tr>
</tbody>
</table>

To end the incremental search, press any cursor movement key (home, end, arrow keys, etc.).

**Expression Highlight**

This allows you to specify a search string or expression and see all occurrences of the found text highlighted in all of your files. This is dynamic; you may edit the files just like normal without disturbing the highlighting.

There is a limitation to Global Expression Highlight. Search strings cannot go across line boundaries.

Select Prompted Search to find one occurrence at a time.
The Search | Global Expression Highlight dialog contains the same fields as in the Tools | Customize | Search | Hilite dialog (where you may modify the default search settings), with the addition of the following:

- **Highlight**: Specify a string or regular expression to highlight throughout your currently loaded files.
- **Alias**: Search Aliases are pre-defined regular expressions that you can use to easily set up, modify, and use in your search strings.
- **Reg Ex Help**: Press the Reg Exp button to invoke Help on Regular Expressions.
- **Regular Expression Insert button**: This button allows selecting a regular expression character from a popup menu to be inserted into the string field at the caret position.

If you do not want to go through the dialog, you can call the macro directly and pass parameters to it. For more information, please refer to Global Expression Highlight in Search.sh.

### Examples

#### Search for the '$' character

**EXAMPLE 1:**

Search for the '$' character.

**SEARCH EXPRESSION:**

(Classic Style)  @$

(Unix and Perl Style)  \\$

**DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:**

(Classic Style)

- `@` Find literal.
- `$` Search for '$' char

(Unix and Perl Style)

- `\` Find literal
- `$` Search for '$' char

**SUMMARY:**

Since the '$' symbol is a wildcard character and has special meaning in search and replace, the '@' symbol must be placed before it. The '@' tells Multi-Edit to treat the next character literally.

The '@' symbol must be used when searching for any of the wildcard characters literally. The wildcard characters include:

(Classic Style)  * + [ ] % $ @ { } | & # ~ ^

(Unix Style)  ^ $ * + @ . [ ] | ( ) \n
(Perl Style)  ^ $ * + @ . [ ] | ( ) \ { }
Note that within a character class the metacharacters are different. In general, in UNIX and Perl classes the only metacharacters are -, [, ], and \. Note also that \b has a different meaning in a character class than outside one in Perl. The wildcard characters Multi-Edit recognizes in replace expressions are:

(Classic Style)  $ # & % ^
(Unix Style)      $ \ & % ^
(Perl Style)     $ \ % ^

**Find a parenthesis set**

**EXAMPLE 2:**
Find a parenthesis set.

**SEARCH EXPRESSION:**
(Classic Style)  (?*)
(Unix Style)     \(.*\)
(Perl Style)     \(.*\) or (?m)\(.*\)

**DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:**
(Classic Style)

\(  \)                Search for an open parenthesis
  \(?*\)              Followed by any number of occurrences (>=0) of any character
 \)                    Followed by a close parenthesis

(Unix Style)

\(  \)                  Search for an open parenthesis
 \(\.*\)               Followed by maximal match of any number of occurrences (>=0) of any character
 \)                      Followed by a close parenthesis

(Perl Style)

(?m)                      Multi-line search, if present
  \(  \)                Search for an open parenthesis
  \(\.*\)              Followed by maximal match of any number of occurrences (>=0) of any character
 \)                      Followed by a close parenthesis

**SUMMARY:**
Finds parenthesis sets by searching for an open parenthesis followed by any number of characters, of any kind, followed by a close parenthesis. The Perl expression performs a multi-line search if prefixed by ‘(?m)’.

---

For UNIX and Perl-style users, take care to select the proper minimal/maximal matching character. See Understanding Minimal vs. Maximal Closure for more information.
Find any occurrence of single or double quote sets

EXAMPLE 3:
Finds any occurrence of single or double quote sets.

SEARCH EXPRESSION:
(CClassic Style) {"\*"}|{"\*"}
(Unix Style) (".*")|('.*')
(Perl Style) (["'"].*?\| or (\?)(["'"]).*?\|)

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(CClassic Style)
```pseudocode
{ Begin Group 0
    " Match the opening double quote
    Group 0
      ?* Match a (possibly empty) string of any characters
    " Match the closing double quote
  } End Group 0
  | - OR -
  { Begin Group 1
    ' Match the opening single quote
    Group 1
      ?* Match a (possibly empty) string of any characters
    ' Match the closing single quote
  } End Group 1
```

(Unix Style)
```pseudocode
{ Begin Group 0
    " Match the opening double quote,
    Group 0
      .@ Minimally match a (possibly empty) string of any characters
    " Match the closing double quote
  } End Group 0
  | - OR -
  { Begin Group 1
    ' Match the opening single quote
    Group 1
      .@ Minimally match a (possibly empty) string of any characters
    ' Match the closing single quote
  } End Group 1
```

(Perl Style)
```pseudocode
( Begin Group 1
  Group 1
    ['"] Match the opening single or double quote
  ) End Group 1
  .*? Minimally match a (possibly empty) string of any characters
  \1 Match the character matched by Group 1
```

SUMMARY:
Finds double or single quotes sets by searching for a quote, followed by any number of characters, of any kind, followed by a quote.

For UNIX- and Perl-style users, take care to select the proper minimal/maximal matching character. See Understanding Minimal vs. Maximal Closure for more information. Since Perl expressions can use backreferences, there’s no need to repeat verbatim strings; instead, the alternation can be encapsulated as a character class and a backreference. Note that this can’t be extended to allow matching nested structures, as they usually don’t use the same character sequence to begin a nesting level as they do to end one, and there’s no way to refer to match counts within a regex.
Find the next word

EXAMPLE 4:
Finds the next word.

SEARCH EXPRESSION:

(Classic Style)  %|[-a-z0-9_][a-z0-9_]
(Unix Style)    ^|[^[a-z0-9_][a-z0-9_]
(Perl Style)    (?<!\w)(?!\w)

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)
%  Search for beginning of line
   -OR-
|  Search for any character that is not a letter between a-z, a number
  between 0-9, or an underscore
[-a-z0-9_]  Search for any character that is a letter between a-z, a number
            between 0-9, or an underscore

(Unix Style)
^  Search for beginning of line
   -OR-
|  Search for any character that is not a letter between a-z, a number
  between 0-9, or an underscore
[^a-z0-9_]  Search for any character that is a letter between a-z, a number
            between 0-9, or an underscore
[a-z0-9_]   Search for any character that is a letter between a-z, a number
            between 0-9, or an underscore

(Perl Style)
(?<!\w)  Search for a position not preceded by a word character (negative look-behind)
(?!\w)   and followed by a word character (look-ahead)

The Unix expression would have also worked as a Perl expression, but is not as efficient as the Perl
expression shown. If it doesn’t matter whether the expression matches the beginning or the ending of
the word, the simple:
\b
to find a word boundary is sufficient. Note that the Perl expression, unlike the other two expressions, doesn’t
select any text. That’s because it consists of two anchors. That also makes it quite nice as a sort of prefix to
an extended expression; the prefix locates the start of a word, and the rest of the expression matches from
there.

SUMMARY:  Search for the beginning of the line OR search for the first occurrence of a blank followed by a
letter, digit, or underscore. In the Perl expression, search for a look-behind and look-ahead match that
together define the start of a word; it isn’t necessary then to cover the case where the word occurs at the start
of a line as a separate subexpression, as the look-behind condition covers it.
Search for the following operators

EXAMPLE 5:
Searches for any of the following operators: ' = > < ! | &'

SEARCH EXPRESSION:
(Classic Style) [>=!<@|@&]
(Unix and Perl Styles) [=<!|&]

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)
[ Begin character class
 = Search for '=' character
 > Search for '>' character
 < Search for '<' character
 ! Search for '!' character
 @| Search for '|' character literally. Since the | is a wildcard character an @ must be placed before it in order to search for the character literally (as opposed to interpreting it as one of the search expression operators)
 @& Search for '&' character
 ] End character class

(Unix and Perl Styles)
[ Begin character class
 = Search for '=' character
 < Search for '<' character
 > Search for '<' character
 ! Search for '!' character
 | Search for '|' character literally. Since the | is a wildcard character, it would ordinarily have to “escaped” by preceding it with a backslash to prevent the regex engine from interpreting it as such. Occurring as it does, however, inside a character class, this is not necessary, as the alternation operator would be meaningless in that context.
 & Search for '&' character
 ] End character class

Search for the occurrence of a string and replace it with the found string in parenthesis

EXAMPLE 6:
Searches for the occurrence of a string and replaces it with the found string in parenthesis, demonstrating the use of '&' and '\0' as replace expression wildcard characters.

CURRENT STRING: Multi-Edit
DESIRED RESULT: (Multi-Edit)
SEARCH EXPRESSION:
(All Styles) Multi-Edit

REPLACE EXPRESSION:
(Classic & UNIX Styles) (&)
(Perl Style) (\0)

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENTS:

(Classic and Unix Styles)
( Print a literal open parenthesis,
  & followed by the entire string matched by the search expression,
) and then a closing literal parenthesis

Search for a blank line and delete it

EXAMPLE 7:

Searches for a blank line and deletes it.

SEARCH EXPRESSION:
(Classic Style) %$
(Unix and Perl Styles) ^$

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)
% Search for the beginning of a line
$ Followed by the end of a line

(example continued…)

(Unix and Perl Styles)
^ Search for the beginning of a line
$ Followed by the end of a line

REPLACE EXPRESSION:
(All Styles) %

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENTS:

(All Styles) % Delete a character

SUMMARY:
Finds a blank line by searching for the beginning of a line immediately followed by the end of line. Using '%' (delete character) as the replace expression deletes the blank line.

In order for this expression to work properly the 'Leave Cursor At' switch must be set to leave the cursor at the beginning of the replace string.
Search for the first occurrence of a ';' and delete the rest of the line

The use of curly braces {} in Classic Style and parentheses in Unix and Perl Styles to define capture groups within search expressions adds power and flexibility to Multi-Edit's Search and Replace operations. This grouping capability enables you to do such things as delete portions of the found string or change the sequence of matched groups within the found string. Both of these operations are demonstrated in the following examples. Perl has several other types of more specialized grouping operators; refer to the documentation for the regular expression library or a text on regular expression usage for more detail, as such material is quite outside the bounds of this manual.

EXAMPLE 8:
Searches for the first occurrence of a semicolon and deletes the rest of the line after it.

CURRENT STRING:  goto_line(1);del_line;

DESIRED RESULT:  goto_line(1);

SEARCH EXPRESSION:

(Classic Style)  {?*;}{?*}$
(Unix Style)    (.@;)(.*)$
(Perl Style)   (.*?;)(.*)$

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)

{  Begin capture Group 0
  ?*  Match a (possibly empty) string of any characters
  ;  Match a literal semicolon
  }  End capture Group 0
  - OR -
  {  Begin capture Group 1
  ?*  Match a (possibly empty) string of any characters
  }  End capture Group 1
  $  Anchored at the end of the line

(Unix Style)

(  Begin capture Group 0
  .@  Minimally match a (possibly empty) string of characters
  ;  Match a literal semicolon
  )  End capture Group 0
  - OR -
  (  Begin capture Group 1
  .  Match a (possibly empty) string of characters
  )  End capture Group 1
  $  Anchor match to the end of the line

(Perl Style)

(  Begin capture Group 1
  .*?  Minimally match a (possibly empty) string of characters
  ;  Match a literal semicolon
  )  End capture Group 1
  - OR -
  (  Begin capture Group 2
  .  Match a (possibly empty) string of characters
  )  End capture Group 2
  $  Anchor match to the end of the line
SUMMARY:
Find any character that occurs any number of times, (including 0 times) that is followed by a semicolon. This constitutes Group 0 (Group 1 in Perl). Following the previous match, find any character that occurs any number of times. This will constitute Group 1 (2 in Perl).

While greedy (maximal) operators are generally more efficient than lazy (minimal) ones, it's fairly easy to construct pathological regexes where the opposite is very much the case, due to excessive “super-linear” backtracking in search of a match. In particular, the Kleene star, ‘*’, consumes everything when used in an expression like ‘.*’, potentially the entire file if multi-line matching is enabled (although it can also deliver some surprising results in other ways, since, while it attempts to match the longest possible string consistent with an overall match, if necessary, it will match against nothing at all). Judicious use of “possessive” operators and capture groups, which prevent certain kinds of backtracking, can make searches much faster, and can often prevent an exponential explosion in search time that can cause Multi-Edit to appear hung. A good source book on these topics at the time of this printing is Jeffrey E.F. Friedl’s Mastering Regular Expressions, under the O’Reilly imprint.

REPLACE EXPRESSION:
(Classic Style) #0
(Unix Style) \0
(Perl Style) \1

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENTS:
(Classic Style)
#0 replace found string with group 0 from search string.

(Unix Style)
\0 replace found string with group 0 from search string.

(Perl Style)
\1 replace found string with group 1 from search string. Perl counts its sub-pattern matches starting at one, rather than zero, like the Classic and Unix styles. \0 is reserved for the text matched by the entire pattern.

SUMMARY:
In this example we are searching for two groups of strings, everything up to the ';' and everything else to the end of the line. When we replace the string we only replace the first group, which has the effect of deleting to the end of the line.

Swap the parameters of the procedure gotoxy

EXAMPLE 9:
Swaps the parameters of the procedure 'gotoxy'.
CURRENT STRING: gotoxy(x,y);
DESIRED RESULT: gotoxy(y,x);
SEARCH EXPRESSION:

(Classic Style)  gotoxy({?+},{?+});
(Unix Style)  gotoxy((.#),(.#));
(Perl Style)  gotoxy((.+?),(+.?));

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)

gotoxy( Match the literal string "gotoxy("
{ Begin capture Group 0
  ?+ Match a non-empty string of any characters
} End capture Group 0
, Match a literal comma
{ Begin capture Group 1
  ?+ Match a non-empty string of any characters
} End capture Group 1
); Match the literal string ");"

(Unix Style)

gotoxy( Match the literal string "gotoxy("
( Begin capture Group 0
  .# Minimally match a non-empty string of any characters
) End capture Group 0
, Match a literal comma
( Begin capture Group 1
  .# Minimally match a non-empty string of any characters
) End capture Group 1
); Match the literal string ");"

(Perl Style)

gotoxy( Match the literal string "gotoxy("
( Begin capture Group 1
  .+? Minimally match a non-empty string of any characters
) End capture Group 1
, Match a literal comma
( Begin capture Group 2
  .+? Minimally match a non-empty string of any characters
) End capture Group 2
); Match the literal string ");"

SUMMARY:

Finds the literal string “gotoxy(”, followed by any string at least one character long (the first argument, captured as Group 1 by a Perl expression, Group 0 by the other sorts), followed by the comma that delimits the two arguments; then another capture expression identical to the first for the second parameter (Group 0 or 1), followed by the literal string “);”.

REPLACE EXPRESSION:

(Classic Style)  gotoxy(#1,#0);
(Unix Style)  gotoxy(1,0);
(Perl Style)  gotoxy(2,1);
DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENTS:

(Classic Style)

```
gotoxy( Replace with the opening string “gotoxy(”
  #1 followed by Group 1 from the search expression
  , followed by the argument-delimiter comma
  #0 followed by Group 0 from the search expression
  ); followed by the closing string “);”
```

(Unix Style)

```
gotoxy( Replace with the opening string “gotoxy(”
  \1 followed by Group 1 from the search expression
  , followed by the argument-delimiter comma
  \0 followed by Group 0 from the search expression
  ); followed by the closing string “);”
```

(Perl Style)

```
gotoxy( Replace with the opening string “gotoxy(”
  \2 followed by Group 2 from the search expression
  , followed by the argument-delimiter comma
  \1 followed by Group 1 from the search expression
  ); followed by the closing string “);”
```

Notice that the literal parentheses that form the argument list had to be backslash-escaped in the search expression, but not in the replace expression, because they’re metacharacters in the former, but couldn’t be in the latter. It also becomes obvious in this example that sub-pattern capture groups are zero-based for Classic and Unix expressions, but one-based for Perl.

SUMMARY:
This example demonstrates the use of groups to change the order of your search string. Since we defined each parameter as a group, it was easy to swap them by changing their order in the replace expression.

**Search for two semi-colon delimited statements on a single line**

**EXAMPLE 10:**
Searches for two semi-colon delimited statements on a single line. Once found, a carriage return is inserted between them leaving each statement on its own line.

CURRENT STRING: `goto_line(1);del_line;`

DESIRED RESULT: `goto_line(1);`
`del_line;`

SEARCH EXPRESSION:

(Classic Style) `{;}@x20*{(~ )?*;}` (@x20 is the space character a literal space would also work)

(Unix Style) `(;)x20@[([ ]).@;]` (x20 is space in this style)

(Perl Style) `(;)s*?(S.*;)`
DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)

Group 0

{ Begin capture Group 0
 ; Match a literal semicolon
 End capture Group 0
}

@x20* Match a (possibly empty) string of spaces

Group 1

[- ] Match any one non-space character

?* Match a (possibly empty) string of any characters

; Match a literal semicolon

End capture Group 1

(Unix Style)

Group 0

{ Begin capture Group 0
 ; Match a literal semicolon
 End capture Group 0
}

@\s*\ Minimally match a (possibly empty) string of spaces

Group 1

[^ ] Match any one non-space character

. *? Minimally match a (possibly empty) string of any characters

; Match a literal semicolon

End capture Group 1

(Perl Style)

Group 1

{ Begin capture Group 1
 ; Match a literal semicolon
 End capture Group 1
}

\s*? Minimally match a (possibly empty) string of spaces

Group 2

[\ ] Match any one non-space character

. *? Minimally match a (possibly empty) string of any characters

; Match a literal semicolon

End capture Group 2

SUMMARY:

Find the first occurrence of a semicolon on a line. This will constitute Group 0 (1 for Perl). Following this, find any number of occurrences of a space. Then find Group 1 (2 for Perl), which consists of the shortest matching string of at least one character, followed by a semicolon.

REPLACE EXPRESSION:

(Classic Style) #0$#1
(Unix Style) \0$\1
(Perl Style) \1$\2

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENTS:

(Classic Style)

#0 Replace the matched string with whatever matched capture Group 0,
S followed by a line break
#1 followed by whatever matched Group 1

(Unix Style)

\0 Replace the matched string with whatever matched capture Group 0,
S followed by a line break,
\1 followed by whatever matched Group 1

(Perl Style)

\1 Replace the matched string with whatever matched capture Group 1,
S followed by a line break,
\2 followed by whatever matched Group 2
SUMMARY:
In this example, two groups are defined. The first consists of the first occurrence of a semicolon on a line. The second consists of any number of characters (excluding a space) followed by a semi-colon. When the found string is replaced a carriage return is inserted between the two groups, leaving each one on its own line.

Search and Replacing Phrases

EXAMPLE 11:
Searches for the phrases “this is a test” OR “this is not a test” and replaces “a test” with “a pizza”

CURRENT STRING:  this is a test
- OR -
this is not a test

DESIRED RESULT:  this is a pizza
- OR -
this is not a pizza

SEARCH EXPRESSION:
(Classic Style)  this i{s not}|{s} a test
(Unix Style)    this i((s not)|(s)) a test
(Perl Style)    this is( not)? a test

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:
(Classic Style)

( Group 0

  ( Group 1
    this i
    Begin capture Group 0
  )

  Group 1
  s not
  Begin capture Group 1

  Group 2
  End capture Group 1

  Group 2
  s Match the literal character “s”

  Group 0
  a test
  End capture Group 0

  a test
  Match the literal string “a test” (first character is a space)

(example continued…)

(Unix Style)

( Group 0

  ( Group 1
    this i
    Begin capture Group 0
  )

  Group 1
  s not
  Begin capture Group 1

  Group 2
  End capture Group 1

  Group 2
  s Match the literal character “s”

  Group 0
  a test
  End capture Group 0

  a test
  Match the literal string “a test” (first character is a space)

(Perl Style)

( Group 1

  this is
  Begin capture Group 1

  Group 1
  not
  Begin capture Group 1

  Group 1
  a test
  End capture Group 1

  Group 1
  a test
  Match the literal string “a test”
SUMMARY:
Match “this i”, followed by either “s not” OR “s” (which constitutes group #0), followed by “a test”. For the Perl version, find “this is”, possibly followed by just one “ not”, followed by “ a test”. This is only possible in the Perl version, since only it has a “none or one” control, which is necessary to lock out false matches (repeated “ not”).

REPLACE EXPRESSION:
(Classic Style)  this i#0 a pizza
(Unix Style)  this i0 a pizza
(Perl Style)  this is\1 a pizza

SUMMARY:
Replace the complete matched string with the literal-string prefix “this i” (with an extra “s” in the Perl version), concatenated with Group 0 (1 for Perl) from the search expression, then with the literal-string suffix “a pizza”. There is a very important detail to be noted in the Classic and Unix search expressions in this example: “s not” came before “s” in the alternation.

This was done intentionally. Since “OR” expressions are evaluated from left to right, you must pay close attention to the order in which its components appear. Once the regex engine matches any component of the expression, it won’t evaluate any of the remaining components for a match. Since the Perl version doesn’t use alternation, it isn’t an issue here; however, in Perl expressions that do use alternation, it is still an issue. If “s” had been placed before “s not” the following would result when encountering the line “this is not a test” (the Unix version only differs in that the curly braces become parentheses):

```
this i { {s} | { s not} } a test
```

FOUND TEXT: xx this is not a test xxx xx
RESULT:  Equivalent of no match, no replacement is made.

The “s” portion of the line would match, terminating evaluation of the other branch of the alternation. The remaining component “s not” would never be evaluated. At that point, the engine would attempt to match the next unmatched part of the expression, “a test”, and would fail, because the searched text at that point is “not a test”.

The engine will always attempt to backtrack until it finds a match (whenever possible, expressions attempt to “give up” already-matched text to backtrack to a prior state from which a match down a different path may be possible), unless it exhausts the text without finding a match, is prevented from backtracking (note the operators explicitly intended for that), or arrives at a state where the available search space expands exponentially (you’ll likely think Multi-Edit is hung, but it’s the result of a poorly-formulated regex). Always keep this in mind when constructing your own search expressions.
Blocks

Cut / Copy / Paste

Multi-Edit offers the standard cut, copy, and paste operations as well as a Paste Buffer interface that allows you to maintain several copied or cut text snippets. Performing these operations can be accomplished by using the following methods.

- **Using the keyboard:** Use the key command Ctrl+C to copy or Ctrl+X to cut the currently selected text. To paste text at the current cursor position, the key command Ctrl+Y may be used.
- **Using the edit menu:** The edit menu contains the menu items cut, copy and paste as well as a show buffer item for viewing and editing Multi-Edit's internal buffer.
- **Paste Buffers:** To view the Paste Buffer interface, click on View | Paste Buffers. This will display the Paste Buffers dialog in the Tools Window or as a floating dialog depending on how it was last invoked. Alternatively, the key command Alt+Ctrl+V may also be used. Use the key command Alt+Ctrl+X or Alt+Ctrl+C to cut or copy (respectively) to the multiple buffers.

Using Paste Buffers View

The content of the paste buffers may be viewed and accessed through the use of a floating dialog or a view contained in the Tools Pane. This dialog (or view) can be executed by clicking on the View menu and selecting the Paste Buffers menu item.

**Paste Buffer View**

The paste buffer view has two windows, one containing the list of buffers and their status that states the style of blocking used (stream, column, or line) or empty. The second window serves as a preview of the buffer selected in the list. Double clicking on a selected buffer in the list will paste its contents at the current cursor position.

**Paste Buffer Floating Dialog**

When the floating dialog is used, a single window containing the list of buffers and their status is displayed. Similar to the Paste Buffer View, double clicking on one of the buffers in the list will paste its contents at the current cursor position.

**Options**

The dialog and view both contain the following additional options for working with buffers:

- **Paste:** The paste button may be used as an alternative to double clicking on a buffer contained in the list.
- **Paste All:** The paste all button will paste all buffers containing text at the current cursor position. The contents of the buffers are pasted in the order that they are listed within the buffer list.
- **Show:** Clicking on the show button will display the contents of the selected buffer in an editable window. Note that the paste feature depends on the buffer's contents being block marked.
- **Clear Selected / Clear All:** The 'clear selected' and 'clear all' buttons will delete any text contained within the selected buffer or all buffers.
- **Float / Dock:** Depending on which version of the paste buffers interface is being displayed (view or floating dialog), a float or dock button will be displayed. Selecting this button when in view mode will close the view and enable the floating dialog; while selecting this in floating mode will do the opposite.
Internal Buffer vs. Windows Clipboard

The Multi-Edit buffer can be regarded as a temporary storage area to place a block or blocks of text, which can then be retrieved later. The Multi-Edit buffer is similar to the Windows clipboard, except that the Multi-Edit buffer holds all cut and copy operations for the entire session. When a block of text is cut or copied to the Multi-Edit buffer, it is also copied into the Windows clipboard. Select Edit | Show buffer to view the contents of the Multi-Edit buffer.

Block Types

Multi-Edit supports three block types: stream, line, and column. Marking text using one of these block types can be accomplished using any of the following procedures.

Using the Mouse

- Click and hold down the left mouse button and drag to create the desired block. Release the left mouse button when the entire block has been selected.
- Double clicking with the left button on a word in an Editing Window will cause that word to be marked as a columnar block. Triple clicking will cause the whole line to be marked.
- To mark a stream block, position the cursor at the desired starting point. While holding down the left mouse button, move the mouse cursor to the desired ending point and release the left mouse button.
- Another way to create a stream block is to place the text cursor to the desired starting position, then click and release on the desired ending point while pressing the SHIFT key. A block will be marked from the starting position to the ending point. Extend the block marking by pressing the SHIFT key while clicking at the desired ending point for the block.
- While marking a block with the mouse, you can click the right mouse button (while still holding down the left button), and the block mode will change. Repeatedly clicking the right mouse button will cycle through the line, stream, and column marking modes. Whenever you use the mouse to mark a block, a cursor will be displayed next to the mouse cursor indicating the type of block being marked.

By default, the mouse marks streams of text.

Using the Keyboard

- To create a stream block using the keyboard, hold the SHIFT key while using the RIGHT or LEFT arrow keys.
- Creating a line block is dependent on the option "Shift+UP/DOWN begins line block" located under Tools | Customize | Editing | Blocks. When enabled, pressing SHIFT and the UP or DOWN arrows keys will begin line blocking.
- To create a column block press SHIFT+F7 to start the block then use the ARROW keys to encompass the text followed by SHIFT+F7 to end the block.

Indenting Blocks

To indent/undent a block one tab stop to the right, mark the block you would like to indent, select Edit | Block Operations | Indent or undent. Alternatively, TAB and SHIFT+TAB can be used to indent/undent the marked block.

Using the TAB key and SHIFT+TAB key to indent/undent can be disabled/enabled by modifying the "Tab key indents/undent block" option under Tools | Customize | Editing | Blocks.
Block Operations

Block features are a group of powerful text editing tools designed to minimize repetitive tasks associated with the editing of text and data files. The macro file BLOCK.S contains the macros listed below, which are described here in detail. These macros are accessible from the default Block Operations menu option.

Clear: Performs simple clearing of a marked column of text, converting all text contained within the marked block to blank lines or blank characters.

Fill: Fills a columnar block with a character or a series of characters by prompting the characters to use, then duplicating the characters downward to fit the block. If you select "Fill entire width of marked block" then the block will be filled, repeating the characters until the entire width and height of the block is full.

FillDown: Used to make multiple copies of the top line contained within a block. Start by marking the area of text with which you want to work (at least two lines) using a Line or Column block. By invoking FillDown, the top line of text contained within the marked block is copied downward to fit the block.

FillUp: The opposite of FillDown with the exception that the bottom line of text contained within the marked block is copied upwards to fit the block.

FillParagraph: Will duplicate the currently marked block or current unmarked line to the next available line, paragraph, or blank line. If you pass /S=1 on the command line it will duplicate the marked block to the next paragraph. If you pass /S=2 it will duplicate the marked block to the next available blank line.

SeriesFill: SeriesFill will enumerate a block of text. It works by obtaining values from the first occurrence of a whole number from the first and second lines within the block. If the first number is 20, then numbering continues upward from that number, applying an increment based on the difference between the first and second numbers. For example, when the first line begins with 20 and the next is 21 then subsequent lines are numbered 22, 23, 24, and so on. If the second line begins with 22, however, subsequent lines are numbered 24, 26, 28, and so on.

If "/M=1" is omitted from the command line then a dialog will show allowing custom increment and start values, hex conversion, etc.

Fill Series Dialog
- Start Value: Number value on first line of marked block.
- Increment Value: Value to increment by.

Options:
- Use Hexadecimal equivalents: Converts values within block to hex.
- Force Values onto block: Fills in values on empty spaces within the block.
- Skip empty values: Leaves spaces within the block blank.
- Allow only in block width: Does not extend values outside of the blocked area.

SmartFill - Evaluates the selected area of text and determines what sensible fill concept to apply. It works by applying rules to the marked block of text.

(continued...)
Case A:
- If no block is marked or only one line style block marked
- If the above conditions are true, then FillParagraph is performed on the current line.

Case B:
- If at least two lines are marked.
- There are numbers contained within the marked block.
- The numbers are not yet enumerated.
- If the above conditions are true then a FillSeries will automatically be performed with one exception. If the increment value for SeriesFill is 0, then SmartFill will apply 1 as the increment value.
- Exception: If at least two lines are marked, and there are numbers contained within the marked block. And the numbers are not yet enumerated, and all the lines selected below the first two lines are blank. Then SeriesFill will FillDown the remaining lines while still applying a SeriesFill using the predetermined increment value.

Case C:
- If at least two lines are marked.
- There are numbers contained within the marked block that are already enumerated, or there are no numbers contained within the marked block.
- The cursor is sitting on the last line of the marked block.
- If the above conditions are true then a FillDown will be performed.

Case D:
- If at least two lines are marked.
- There are numbers contained within the marked block that are already enumerated, or there are no numbers contained within the marked block.
- The cursor is sitting on the first line of the marked block.
- If the above conditions are true then a FillUp will be performed.

Case E:
- A columnar block is marked.
- Contains no numbers.
- Is blank or was already filled using FillDown or FillUp.
- If the above conditions are true then FillBlock will be invoked.

ConvertBlock
The ConvertBlock macro will prompt to convert a stream block to a line or column style block. All the Fill features listed above use this macro.

Editing

Aligning Operators

The Align Operators menu item aligns the first operator (based on the language being used which is defined in the filename extension setup) on the first line of a marked block with the same operators found within that marked block. The indent level is determined by the operator with the greatest indent level.
The following examples highlight how this operation works.
Take, for instance, the following code snippet:

```c
    case 3:
        TStr1 = " if ( )" ;
        TStr2 = " { ++I;" ;
        TStr3 = " )" ;
        TStr4 = " " ;
        break;
```

Using Align Operators would produce the following result:

```c
    case 3:
        TStr1 = " if ( )" ;
        TStr2 = " { ++I;" ;
        TStr3 = " )" ;
        TStr4 = " " ;
        break;
```

If multiple operators exist on one line, columnar marking can be used to determine which operator to align to. For example, highlighting the `=` with a columnar block as below:

```c
    case 3:
        TStr1 => " if ( )" ;
        TStr2 = " { ++I;" ;
        TStr3 = " )" ;
        TStr4 = " " ;
        break;
```

(example continued…)
will produce the following when Align Operators is used:

```c
    case 3:
        TStr1 = " if ( )" ;
        TStr2 = " { ++I;" ;
        TStr3 = " )" ;
        TStr4 = " " ;
        break;
```
Commenting

The **Comment/Uncomment** feature (available from the Tools Menu) works for any language for which you have set up comment characters. It works best if you have set up both open/close comments and end of line comments, but it will work just as well with only one comment style defined. To set up comment characters, use **Tools | Customize | Languages**.

The comment feature has several functions built into one command. Which function gets performed depends on how you use the features. There are four methods of using the comment/uncomment command. Each method has two different behaviors depending on the conditions when using the feature.

**Method 1 – No block marked:** If you invoke the comment/uncomment command when no block is marked, Multi-Edit will comment out the line that the cursor is sitting on. If that line has already been commented out, it will uncomment that line.

**Method 2 – Line block marked:** When a line block is marked and you invoke the comment/uncomment feature, Multi-Edit will comment out the marked lines much the way it does with Method 1. Again, if the marked lines are already commented, it will uncomment them.

> Multi-Edit cannot "reverse" comment. Thus you cannot comment one line and uncomment another at the same time.

**Method 3 – Stream block marked:** When you mark a stream block and then invoke the comment/uncomment command, Multi-Edit will comment your code from the beginning of the stream block to the end, and vice-versa if the marked code is already commented.

> If you have only defined end-of-line comments for the language you are using (or begin/end comments do not exist), then you may not be able to end your comment at the point specified in your stream block (i.e., your last line will be completely commented out, since only end of line comments exist).

**Method 4 – Single column of text marked:** If you mark a single column of text and then invoke the comment/uncomment feature, your code will be commented starting at the column marked and extending to the end of each line.

Line Numbers

Select **Line numbers (Text | Line Numbers)** to display line numbers along the left margin of the Editing Window.

Center Line

**Center Line** lets you center the line at the current cursor position between the first column and the right margin.

The Center Line command does not dynamically center text as you type it. To use the Center Line command correctly, first type the text you want to center, and then center the text. If you add to that text later, you may want to re-center it to correct for unbalanced space on either side of the centered text.

> To avoid surprises, it is a good idea to know in advance where you have defined your right margin.
Time/Date Stamp

**Time/Date Stamp** places the time and date at your cursor position. The format of the time and date depends on how it is configured in Windows. To change the format of your Time/Date Stamp, double click on the International icon in your Windows Control Panel.

Ruler

The **Ruler** is a useful tool for lining up your source code and quickly moving your cursor to specific columns in your files. When you activate the ruler, it will appear along the top of the current file.

Across the face of the ruler are sets of marks and numbers that indicate the column numbers across the screen. Also on the ruler are two gray triangular icons. To move these, click on them and drag them to a new position on the ruler.

The first, which sits on the top of the ruler and points down, indicates the "center" of the ruler. If you drag it to the middle of the ruler, then the "center" of the ruler will be placed there, with column numbers extending to both the left and right of the "center".

The second, which sits on the bottom of the ruler pointing up, indicates the current caret (cursor) position. You will notice that as you use the arrow keys to move the caret left and right, the bottom gray triangle mirrors it. In addition, if you want to quickly move the caret to a specific column on the ruler, you can drag the bottom triangle to the desired column and the caret will follow accordingly.

Page Break

Select **Page Break** to insert a page break string (on a line by itself) on the line before the current cursor position. The page break string is normally an ASCII 12 (form feed). See Edit Settings for details on redefining the page break string.

Reformat Paragraph

**Reformat Paragraph** is typically used after you have changed the right margin and want to rewrap a paragraph using the new right margin setting. This command will start reformatting from the cursor position and continue to the end of the paragraph.
Justify Paragraph

**Justify Paragraph** inserts spaces in each line of a paragraph until their ends are flush to the right margin. This command will start justifying from the cursor position to the end of the paragraph.

Unjustify Paragraph

**Unjustify Paragraph** changes a justified paragraph into one that is ragged right. This command will start unjustifying from the cursor position and continue to the end of the paragraph. Typically, the Unjustify Paragraph command is used with the Reformat Paragraph command to get rid of unnecessary spaces. You can then leave the paragraph unjustified or justify it without risking big holes in your text.

The steps for using the Unjustify Paragraph command in combination with the Reformat and Justify Paragraph commands are:

1. Unjustify the paragraph.
2. Reformat the paragraph.
3. Justify the paragraph.

Undo and Redo

**Edit | Undo** allows you to cancel any text change you made. For example, you can restore a character you deleted, reverse the effects of a Search And Replace operation, or return a marked block to its previous position after its been moved. Undo may be selected repeatedly. It has the ability to cancel up to 65,000 changes (should you choose to configure that many changes in Tools | Customize | Editing).

**Edit | Redo** reverses the effects of the Undo command. You can redo as many changes as were "undone" with the following rule:

After a change is made to a file following a redo, previous undo operations cannot be reversed. For example, suppose you used Undo 10 times and then used Redo 3 times. If you then make a change to the file, you will not be able to redo the remaining 7 Undo steps.

Formatting Lines

Tabs vs. Spaces

There are two methods for placing white space within a file when using tabs. The first uses the tab character (0x09) followed by a number of virtual space characters *(at the time of this writing, Multi-Edit uses 0xFF as a virtual space character)*. The second method uses just space characters (0x20). The number of virtual or space characters used is based on either the **Format Line** or **Tab Spacing** settings under File Extension Setup. There are two important notes to be aware of when deciding which method to use. Using spaces when expanding tabs ensures the exact spacing but increases file size as each space character is saved with the file. Using tabs decreases file size but does not ensure exact spacing since it relies on the **Tab Spacing** and **Format Line** settings and these settings may be differ from user to user.
Hex Mode

You may switch into Hex mode by selecting **Text | Hex Mode**. Under **Tools | Customize | Editing**, you can configure Multi-Edit to edit binary files in hex mode by default. Selecting hex mode results in a side-by-side split window: the left side is in hex, the right side is in ASCII.

When editing in the left side, characters may only be entered in hex, with the overwrite mode always on.

Hex mode is simply a different view of the current file. It does not assume that the file is binary, nor does it change the “file type”. Thus, template expansion, smart indent and other features work (if they are configured for that file) while you are editing a file in hex mode. If you wish to view line terminators for a file then you will need to load it as a binary file.

Navigating Code

Match Constructs

Multi-Edit provides construct matching for most of the languages it supports. Construct matching is the ability to find the open/closing characters or keywords that encapsulate classes, conditions, scope, etc. By positioning the cursor under a construct and using the menu **Tools | Match Language Structure** or the keys Ctrl+F9, the cursor position will be toggled between the beginning and ending construct.

![Construct matching is language dependent and accomplished through a macro that is defined in the Language Setup dialog.](image)

Goto Line / Column

The **Goto line-column** dialog allows the user to specify a line number (based at 1) and column number (based at 1) for the cursor to jump to. The current line and column number will be set by default when the dialog is first displayed. This dialog can be invoked using one of the three following methods.

**Search Menu**

Selecting **SEARCH – GOTO LINE NUMBER** on the main menu bar.
Status Bar
The current line / column number is displayed on the right side of the status bar. Clicking on this area will enable the Goto line /column dialog.

Macro Call
The Goto line / column dialog can be called manually by clicking on MACRO – RUN MACRO on the main menu bar, then specifying mesys^gotoline as the macro to run. It can also be called within a macro using the RM macro call.

Scrolling
When working with large files, quick navigation is essential. Multi-Edit offers many methods of scrolling through a file to help you obtain this.

- **Page Up/Page Down:** The page up/down buttons will scroll a full-page length in either direction maintaining the cursor position within the window.
- **Ctrl+ Page up/down:** Scrolls either up or down until a page break, top of file, or end of file is reached. The cursor is positioned at the page break, top of file, or end of file depending on which is found.
- **Ctrl+B/Ctrl+T:** Moves the cursor to the top or bottom of the window maintaining the files position within the window.
- **Ctrl+Home/End:** Moves the cursor to the top/bottom of the file.
- **Alt+Shift+Up/Down:** Moves the cursor to the next changed line, either in the up or down direction.

Options
Two options under Tools | Customize | Editing, affect the cursors position. These are Restrict Cursor and Lock Cursor On V-Scroll. Restrict cursor keeps the cursor within the bounds of the file. This means that the cursor is not allowed to go past the end of the file marker or the end of a line. Lock cursor on v-scroll maintains the cursor position as the document is scrolled.

Collapse
Collapse Mode is an evolution of the Condensed Mode feature in Multi-Edit for DOS, allowing you to specify a column number or a search string on which to collapse your code. Lines that do not meet the collapse mode parameters are hidden and a plus character is shown to the right of the line where this hidden (or collapsed) text resides.

To view the collapsed text, click on the plus button. The text will be expanded, and a minus button will replace the plus button.

When you are ready to re-collapse your text, click on the minus button.

Example:

You can have separate collapse mode settings for each open file within Multi-Edit. Thus, you can have one file set to collapse on column 1, while another file collapses based on the keywords PROCEDURE or FUNCTION.
Collapse Modes

The collapse mode Tools Pane has the following fields:

Column
Selecting this option enables collapse mode by column number. The Column # field will be displayed when this option is selected.

- **Column #**: This is the column number on which the file should be collapsed. All lines that do not start at a column equal to or less than the specified column are collapsed.

Search
Selecting this option enables collapse mode by search string. Additional fields pertaining to search strings appear in the dialog.

- **For**: This field specifies the search string you want to collapse on. All lines that do not contain the search string will be collapsed. Regular expressions are also supported if the "regular expressions" box is checked. For example, to search for the beginning of your 'C' functions, you might search for `{void}|{str}|{int}|{real}` (assuming Multi-Edit Classic regular expressions).
- **Regular exp**: This checkbox enables regular expressions within the For field. When this is not checked, the search strings are literal. When checked, regular expressions are enabled. The default regular expression is set under Tools | Customize | Search Defaults.
- **Case Sensitive**: Checking this box makes the collapse search case sensitive. For example, if VOID was the search string and this box was checked, void (lowercase) would not be matched.
- **Invert**: By default, all the lines that do not match the specified search string are collapsed. If you want all the lines that do match the search string to be collapsed, check this box.
- **Alias**: Press this button to display the Regular Expression Aliases dialog. Select the alias representing the metacommand you wish to use.

Tags
Selecting this option enables collapsing by using the tag function for the language of the current window to specify the lines to collapse. All functions that would be listed in the tag file will be the first line of the collapsed text.

User
Selecting this option will enable collapsing by using a macro that will locate the starting line of the text to collapse. This functions just like the tag mode, but uses different user selected macros.

Currently there are three defined macros:

- **_CollapseParagraph**: This macro will collapse all but the first line of a paragraph.
- **_CollapseChangedLine**: This macro will collapse and show only the changed lines in the current file.
- **_CollapseComment**: This macro will Collapse all comments showing only the first line of the comment.

Other fields

Active
This checkbox allows you easily toggle the collapse mode feature on and off globally without having to uncollapse and re-collapse all the collapsed text.

Scan
The scan button takes the current collapse mode settings and collapses (or re-collapses) the current file according to them.

Exclude collapsed text on

- **Block copy**: Check to exclude hidden (collapsed) text on a block copy.
- **Search**: Check to exclude hidden (collapsed) text during a normal search operation.
Using the Preview Pane

Multi-Edit's Preview Pane is a convenient feature used to preview files or Multi-Tags without having to manually load them. When a file or tag is selected within the Project View or Tag View and the Preview Pane is displayed, a preview of the file or tag is automatically generated within the Preview Pane's main window.

To display the Preview Pane, click on View | Preview Pane. A history list is displayed above the main window allowing quick access to previously viewed files or tags. Clicking on the Edit button will load the currently displayed file or tag into Multi-Edit's editing window.

An option to auto-show the Preview Pane is offered under Tools | Customize | Windowing | Tool Pane. When enabled, the Preview Pane will be displayed when a dialog that supports it is used.

Multi-Tags

What is Multi-Tags

Multi-Tags is an easy-to use, hypertext-like source code browser for C/C++, CMAC, Turbo-Pascal, Java, FORTRAN, ASM, Modula-2, dBase/XBase/CLIPPER and Paradox. Any text file may contain tags via explicit tags.

Run your source files through the Multi-Tags scanner to produce a database of functions/procedures, structures, types, etc., depending on the language being scanned.

Once the database is created, position your cursor on any function name (or other supported language object) and right click the mouse to select Find tag under cursor (or hit a hot key if one is defined). Multi-Tags will then locate the source file where that tag was defined and take your cursor to the definition. If a tag was defined in more than one place or file, then Find again will locate the next occurrence.

View | Tags opens a tree list of the current tag file that allows you to select any tag and move immediately to its definition.

Explicit Tags

A line of source may be explicitly tagged by adding @Metags by a tag identifier on the line, probably in a comment. For example:

```c
if( jx == 10 ) // @METAGS wow
    fprintf("Wow!!");
```
Creating Tags

Multi-Edit maintains tags using a flat file database system. Tag databases use the naming convention <language>.tag where <language> is the language the tags were created for. Tag databases either are created by scanning the current file, scanning the files within a project or by using the wildcard tag scan feature.

Not all languages support the Multi-Tags feature. To see if the language you are using supports Multi-Tags, verify that a Find Tag macro has been created and is specified in the language setup dialog.

Tag Scan
To scan the current file for tags, click on Tag | Scan Tags For Current File. Alternatively, if the Tag View (View | Tags) is displayed, the Scan File button may be used.

Project Scan
Scanning a project for tags will scan ALL files contained in the project. If a project has been created, click on Tags | Scan Project. Alternatively, if the Tag View is displayed, the Scan Project button may be used.

Wildcard Tag Scan
The Wildcard Tag Scan feature scans based on a user defined directory and file mask. Primarily, this is used for scanning multiple files with the same extension. For example, specifying the following path and mask would scan all CMac source files creating the Cmac.tag database.

C:\multiedit\src\*.s

Click on Tags | Wildcard Tag Scan to view the Wildcard Tag Scan dialog. Alternatively, if the Tag View is displayed, the Wildcard Tag Scan button may be used.

Using the Tag View

Multi-Edit's Tag View is a graphical user interface for accessing and maintaining Multi-Tags. Tag databases can be created or loaded (automatically or manually) and source files with corresponding tags can be selected and viewed quickly.

Here's how it works...

When a file is opened or switched to, the Tag View automatically loads the corresponding Multi-Tags database and attempts to select the file from within the database and display the tags contained in the file.

If the file is not found, the first file in the database is displayed. Alternatively, a Multi-Tags database can be manually loaded by clicking on the load tag file icon or switched to using the drop down list. Manually loading a database is most often used for locating tags (function implementations) in a file that is not already loaded or referenced from a loaded file.
Source and Tag Lists

The Tag View has two lists that work together. The first is a list of all source files within the currently loaded Multi Tag database. The second is a list of all tags contained in the selected file from the first list. Both source files and tags may be deleted from these lists (and the Multi Tag database) by selecting the file or tag and selecting the "remove tag / file" option. Double clicking on a tag or selecting a tag and pressing the enter key will load the file the tag is located in and position the cursor at the tag.

Creating / Updating a Multi Tag Database (Scanning Files)

Multi Tags are created by scanning a file for tags.

---

Tags generally include functions, subroutines, classes, and in some cases defined variables. What tags are found within a source file is dependent on the find tag macro that is defined under the language setup.

---

Files can be scanned using the tag view by selecting one of the four methods below.

- **Scan File button:** Scans the current file
- **Scan All Files button:** Re-scans all files within the Multi Tags database
- **Scan Project:** Scans all files contained in the current project
- **Wildcard Scan:** Allows the use of wildcards to scan multiple files in one or more directories.

Using any one of the above methods will create and / or update the Multi Tags database.

---

Displaying all Tags vs. Source: Tag Display

The Tag View offers the option of displaying all of the tags contained in the Multi Tags database. This is different from the default display, which requires a source file to be selected and displays only tags for the selected source file. When the Display All Tags button is clicked, the source file list is disabled but when a tag is selected the corresponding source file will be automatically selected for reference.

---

Tags and Projects

When a project is specified, Multi-Tags are configured within the project settings that take precedence over the Multi-Tags language based configurations. By default, tag databases are stored in the projects root directory. However, this can be changed by clicking on Project | Properties and specifying the desired path on the Directories tab.

Once the desired Tags directory is defined, the entire project can be scanned by clicking on Tags | Scan Project or by using the Scan Project button on the Tag View.
Bookmarks

Push / Pop

The current cursor position can be push/popped using the marker stack thus allowing you to quickly return to the saved position after moving elsewhere within the file. The Multi-Edit system uses the marker stack for its own use such as when doing a file compare or matching a language construct (this should not interfere with the user dropped marks).

Select Search | Push Position onto Marker Stack to drop a mark at a position. When a marker is dropped, a little red check mark ☑️ appears in the left-hand column of the line to notify the user of the dropped mark. When more than one mark is placed in the same location, a multiple checkmark icon ☐️ is displayed.

Select Search | Get Position from Marker Stack to return to a marked position. This causes the cursor to be located at the saved position and the little red check mark will be removed.

Random

Use random access marks to drop a mark and reposition to that mark at any time regardless of mark order. This is useful for switching between multiple places in a file.

To drop a random access mark, select Search | Set Random Access Mark. A dialog with ten numbered buttons will be shown; select one of the numbered buttons to cause that number mark to be dropped.

When this happens, a small red number corresponding to the dropped mark will appear in the left-hand column to indicate the location of dropped mark. A little red check mark will appear above all of the numbered buttons that have active marks dropped.

To return to a saved position, select Search | Retrieve Random Access Mark. This will bring up a similar dialog as the one for dropping a mark. Select a numbered button to cause the cursor to be restored to the position saved for the selected mark. The little red check mark above a numbered button is used to indicate which random access marks have been dropped.

To clear any or all-random access marks in a file, select Search | Clear Random Access Mark. This will display a dialog similar to the ones used for setting and retrieving random access marks. Now, when selecting a button, the number mark is cleared and removed from the current file. The Clear All button removes all random access marks from the current file. When a random access mark is cleared, the little red number in the left-hand column is removed, as well as the little red check mark above the number button.

Using the Bookmark View

The bookmark view is a dialog that is shown in the Tools pane at the bottom of the Multi-Edit screen. It is shown by selecting the View | Bookmark menu or using the key assigned to it, Ctrl+M in the default command map. This dialog is another interface for using the random marks in a window but with a few additional features.

Every file loaded into Multi-Edit can have up to 10 random access marks dropped in them so that a position can be returned to quickly. The Bookmarks view allows associating a descriptive text to each random mark and thus becomes a bookmark. From this dialog, it is possible to set and clear bookmarks as well as goto and/or view the text at a bookmark. There is also a single check box option called Global, which changes the behavior of random access marks. Normally each file can contain up to 10 marks but when the Global option is checked, then only 10 mark for all the files is available but the marks can be in any file and going to a mark will first switch to the correct file.
To set a bookmark the following steps should be followed.

- Select the window the bookmark is to be dropped in.
- Position the cursor to the location to be bookmarked.
- Open the Bookmark dialog.
- Select the number of bookmark to be set by clicking on the bookmark number.
- Set the text to associate with the bookmark. If left blank then the text at the cursor will be used.
- Select the Set button.

Bookmarks are saved in the Session information.

Comparing Files

How does it work?

To compare two files, load them both into Multi-Edit and make one the current file. Select Compare Files.

File compare has been updated to use the new original line number-tracking feature of Multi-Edit. When doing a file compare the two windows are synchronized now by searching for the original line number.

You can now compare the Editing Buffer with the Compiler Results by simply clicking on the "Edit" button in the Compiler Results pane.

The FC Split Window dialog box allows you to make changes to the comparison criteria (default settings can be changed in Tools | Customize | File Compare or by checking the Save as default settings checkbox before initiating the compare).

If you have selected anything other than Prompt in the Window split field, the default settings will be used and you will not see the FC Split Window dialog box, but instead be taken directly to the Link Window dialog box, a standard Multi-Edit list box that lists the open files from which to choose for the comparison. Highlight the desired file and press the Select button.

When the comparison is complete, the two files may be moved through and edited. The cursor movement between the two files will be linked and synchronized. By default, you may use the Alt+PgUp and Alt+PgDn keys to move quickly from difference to difference.

To stop the comparison, close either of the windows. If you select Compare again while either of the two comparison windows is active, the two files will be automatically re-compared. This is useful for updating the comparison after significant changes have been made. You may also generate difference reports. You may change the file compare keys and the colors used to highlight the differences from the Tools | Customize | Colors dialog.
File Compare macro will leave a marker behind so that you can return to the file position before the compare started.

The File compare macros have also been updated to add support for easily merging changes. After a compare is done, a File Compare toolbar is shown, by default at the bottom of the screen, see the following topic, File Compare Toolbar, for the description of each button, and the context menu has some file compare specific entries added.

The following entries are inserted at the top of the context menu when bringing up the context menu in a window that is showing the differences.

- **Fc – Replace**: Copies the block of differences the cursor is on to the other window replacing the corresponding differences.
- **Fc - Insert after**: Copies the block of differences the cursor is on to the other window inserting it after the corresponding differences.
- **Fc – Delete**: Deletes the block of differences the cursor is on for the current window.
- **Fc - Previous difference**: Move the cursor to the previous difference.
- **Fc - Next difference**: Move the cursor to the next difference.

Syntax highlighting during file compare: If checked, syntax highlighting will be enabled, otherwise the default screen colors will be used.

- **Ignore Case**: If checked, the file compare will ignore the case of the characters in the file during the compare.
- **Ignore indent**: If checked, the file compare will ignore indent changes between two files being compared.
- **Ignore blank lines**: When this box is checked, the file compare will ignore blank lines inside the files being compared.
- **Ignore Tabs**: When this option is checked, tabs are treated as spaces when doing a compare.

No window list: Normally, when a file compare is executed, a list of the currently open windows is displayed, allowing you to select a file already loaded to compare. If you do not want this list to appear and instead have Multi-Edit prompt you for the file to compare to, check this box.

Save as default settings: Check to save your selections to be used in future file comparisons.

Restrict compare to columns: When checked, the comparison will be limited to the specified columns.

Window split: Depending on your default File Compare Settings, you may need to manually select a direction in which to split the window:

- **Right/Left/Up/Down**: Splits the window in the selected direction.
Full Screen: This option is much different than the other split options, which display the file compare interactively side-by-side. The full screen option builds a third window that contains the original window text, and supplemented by the different text from the two compared files. In effect, this option creates a "merge file" while highlighting the differences between the first two files.

File Compare Toolbar

File Compare: Recompare files to include any changes made.

Stop Compare: Stops the compare. Removes highlighting but does not delete the windows.

Previous difference: Move the cursor to the previous difference.

Next difference: Move the cursor to the next difference.

Replace change: Copy the difference block at the cursor to the other window replacing the corresponding differences.

Insert change: Copy the difference block at the cursor to the other window inserting it after the corresponding differences.

Delete change: Deletes the difference block at the cursor for the current window.

Undo: Undo the last change made to the current window.

Redo: Redo the last undo made to the current window.

Change Window Left: Moves the cursor to the window on the left of the current window.

Change Window Right: Moves the cursor to the window on the right of the current window.

Change Window Up: Moves the cursor to the window above the current window.

Change Window Down: Moves the cursor to the window below the current window.

Next Difference / Prev Difference

The Previous Difference and Next Difference selections are active only while performing a file compare operation. When performing a file compare, they will take the cursor to the next difference or the previous difference in the compared files.
Difference Report allows you to generate either a detailed difference report or a summarized difference report of a file compare. The report will be written to a new window.

Summarized difference reports contain the date and time the file compare was done, the names of the two files compared, and the line numbers where the two files differed.

Detailed difference reports contain the same information as summarized reports, but also show the actual lines of text that differed between the two files.

The Composite Difference option allows you to compare two files and build a third file that merges the differences of the two compared files into one. You can then view, save, or edit this file as necessary. This is extremely useful for merging changes to a single file from multiple sources.

Compiling Files

Adding a new Compiler

Each filename extension may have any number of compiler/program interfaces associated with it. Each compiler/program interface contains its own command line, and configuration. Thus you might have multiple compilers, linkers and debuggers set up for .C extensions. You might have a grammar analyzer set up for .DOC files. When invoked, a list box appears containing a list of compilers available for the active filename extension.

To view the Compiler/Program Setup dialog, select Customize from the Tools menu and click on the Customize tab. Press the Filename extensions button to display a standard Multi-Edit list box with a list of existing Filename extensions. Select the filename extension from the list and press the Edit button. Press the Compiler/Program setup button in the resulting Edit Filename Extension Setup dialog box to display a standard Multi-Edit list box with a list of existing Compiler/program configurations.

Buttons along the side of the list box allow you to view the Filename Extension Setup dialog, select Customize from the Tools menu and click on the Customize tab. Press the Filename extensions button to display a standard Multi-Edit list box with a list of existing Filename extensions. Buttons along the side of the list box allow you to Edit, Insert, Delete or Copy the selections. You can also rearrange the list by moving them up and down in the menu list. The Search and Again buttons allow you to quickly find the list member you are looking for. When you press the Select button, Multi-Edit will enable the currently highlighted command mapping.
Select an item in the list and press the Edit or Insert button to display the Compiler/Program Setup dialog box with the following fields for editing:

- **Description**: Enter a name or descriptive phrase for the compiler. Your entry will appear in the Compiler/Program Setup list box.
- **Command**: Enter the command-line just as you would if you were running this compiler from the DOS prompt or the Windows File, Run prompt. The only exceptions to this are the Multi-Edit Metacommands that are used to tell the compiler what file you want compiled.
- **Working Directory**: This is an important field! The working directory, as defined in compiler setups, is used by Multi-Edit while executing your compiler. The working directory is the current directory while your compiler is executing. For example, if your working directory is D:\MYPROJ, imagine yourself at the DOS prompt manually typing your compiler command line in with D:\MYPROJ as the current directory.

There are four different working directory settings:

- **Current**: Uses the current working directory (as defined in either the Session Manager or in the File, Open dialog or in the File Item Properties under Windows) as the working directory for the compiler.
- **Source File**: Uses the path to the current source file as the working directory.
- **Program**: Uses the path to the compiler as the working directory.
- **Specified**: Allows you to specify the working directory. This is the only option that accepts input from the text box below the Working Directory radio buttons.

**Program Type**

This brings up a list box containing currently supported compilers. Multi-Edit uses the selection you make here to correctly parse the compiler errors out of the error file your compiler generates. If you don't see your compiler or are using a newer version of a compiler that isn't supported, see the Advanced Compiler Setup section.

*Very often, even though a specific compiler is not listed, it is compatible with one that is listed. The best thing to do is to try several of the more popular compilers listed. For example, many compilers are compatible with the generic MICROSOFT listing.*
EXE Type
This drop-down list box is used to select the type of compiler executable that has been specified in the Command entry. This is used by Multi-Edit to determine how to start the compiler running.

- **Auto Detect:** This option will cause Multi-Edit to run the IdExe macro to determine the executable type of the compiler. This is the default and will work for most cases but sometimes the executable will not be identified correctly and thus one of the other options can be used to force the correct compiler invocation.
- **DOS:** Selecting this option will cause the compiler to be run as a DOS program where the DOSEXEC program is used to start the compiler and capture its output to an error file.
- **Windows:** Selecting this option will cause the compiler to run as a Windows program.
- **OS/2:** Selecting this option will cause the compiler to be run as an OS/2 character mode compiler where the OS2EXEC program is used to start the compiler and to capture its output to an error file.
- **Win32 Console:** Selecting this option will cause the compiler to run as a Win32 Console application.
- **Macro:** The command line specifies a Multi-Edit macro to be run.

Show
Note that PIF settings (under Windows) take precedence over these settings.

- **Normal:** The compiler execution window will appear in the default position and size.
- **Minimized:** The compiler execution window will appear minimized.
- **Maximized:** The compiler execution window will start full screen.

Options
The following check box options are used to configure the operation of the compile macro.

- **Save All Files:** Enable this option to cause every file that has changed and not been saved to be saved to disk before the compiler/program is started.
- **Reload File:** If you expect the compiler/program to modify the current file, then enable this option to have Multi-Edit reload the file after the compiler/program has finished.
- **No stdout capture:** Enable this option if you do not want to have the standard output stream redirected to the Multi-Edit error file.
- **No stderr capture:** Enable this option if you do not want to have the standard error output stream redirected to the Multi-Edit error file. If you disable both Stdout and Stderr, then the program will be run directly instead of using one of the redirection programs, i.e. DOSEXEC.EXE
- **No error processing:** Enable this option to have Multi-Edit not process errors after the compiler/program has finished. This would be enabled for compiler/programs that are not currently supported.
- **Command line prompt:** Enable this option when you want to have the command line come up in a prompt before you run a compiler. This allows you to make any last minute adjustment to the command line, such as adding or deleting parameters.
- **Use cmd processor:** Enable this option when you want the command processor to be run, which will start the compiler running. This would be enabled when the command line specifies the compiler/program as a Batch file.
- **Run in background:** Enable this option to cause Multi-Edit to start the compiler/program running and allow you to continue editing or processing errors from a previous compile while the current compile is being run in the background. If this option is enabled when a compiler is started, a small Task Dialog will appear in the upper right hand corner of the Multi-Edit window, showing all of the running background tasks. A check mark will appear next to the task number when that task has completed. By double clicking on any task that has finished, you can start the error processing for that compile.
- **Load error file only**: Enable this option to cause Multi-Edit to load the error file specified in the Command field without actually starting a compiler/program. This allows a user to compile a program outside of Multi-Edit and then load and process the errors from the captured error file. The Program type still needs to be configured for this option to function correctly.

> Only one DOS and one OS/2 program can be run in the background at once, although multiple Windows programs can be run.

### Adding Error Processing

For users who want to set up a compiler type that is not currently supported by Multi-Edit, we have added extra functionality to the Compiler Parsing feature.

To set up your own compiler error-parsing support, you need to set up your own Compiler/Program Type. To view the Compiler/Program Type Setup dialog, select Customize from the Tools menu and click on the Customize tab. Press the Filename extensions button to display a standard Multi-Edit list box with a list of existing Filename extensions. Select the filename extension from the list and press the Edit button. Press the Compiler/Program setup button in the resulting Edit Filename Extension Setup dialog box.

Select the "..." button in the Program Type field to display a standard Multi-Edit list box with a list of existing Compiler/Program Types. Select an item in the list and press the Edit or Insert button to display the Compiler/Program Type Setup dialog box with the following fields for editing:

- **Type**: This stringfield contains a description of the compiler or program that is being setup This is what is show in the Compiler Program Type List Dialog.

- **Exe File**: This string field is used to enter the default file name of the compiler executable to run. This field can contain the full path of the executable file as well.

> This will be used in the Project support to allow setting the compiler executable when the default button is selected.

- **Release Parms**: This string field is used to enter the default command line parameters that you would use when running the compiler to produce a release version of the compiled code.

> This will be used in the Project support to allow setting the new Release Parms entry when the Default button is selected.

- **Develop Parms**: This string field is used to store the default command line parameters that you would use when running the compiler when developing your programs.
This will be used in the Project support to allow setting the new Develop Parms entry when the Default button is selected.

Compile Macros
These options are used to expand the default behavior of the compile macro.

- **Pre:** This string field is used to enter the name of an optional macro that you want to run before the actual compiler executable is started.

- **Post:** This string field is used to enter the name of an optional macro that you want to run after the compiler executable has finished, but before the error processing is done. This macro should return with the following:
  - Return_Int - False Continue and run error processing macro.
  - Return_Int - True Continue but not run the error processing macro.
  - Return_Str - The string to display on the message line when Return_Int = True

Error (Processing)
Error-processing within Multi-Edit can be accomplished by the use of a macro or regular expressions, which is the suggested method.

Version 9.10 contains newly added code to allow the compiler error processing code to not show an empty file when an error is located in a file that doesn't exist. There were two global variables added that can be set in Startup.cfg that control this. They are:

- **!CmpErrorNoFile:** When set will cause the empty window to be removed.
- **!CmpErrorNoDlg:** When set will disable the File not found dialog.

  (Only used when !CmpErrorNoFile is also set)

- **File:** This string field is used to enter an optional error file name. This would be set for compilers that generate their own error files. When a file name is specified in this entry, the Compile macro will not capture the output to its temporary error file and will instead use specified file name when the process error macro is run.

- **Macro:** This string field is used to enter an optional error macro that will be called to search for errors in the error file. When an error macro is specified here the main error processing macro will try to run this string as a macro to find the errors.

This should only be used when a regular expression (documented below) cannot be designed to locate error message lines in the captured error file. There are no plans to provide any additional specific compiler error processing macros, but feel free to write your own.

Regular Expressions
These fields are used to specify regular expressions that the built in error processing macro will use to try to locate error messages in the captured error file. This is the preferred way of processing errors and can easily be updated by the user to provide new compiler/program types. See Regular Expression Error Processing for more details.

- **Search:** This string field is used to enter a regular expression that will be used to search the captured error file for a line that contains an error message. This string is a UNIX style regular expression and should be such that it will match the complete line of the error message. Thus it should always start with the ^ (beginning of line) character and end with the $ (end of line) character. This string can contain regular expression aliases, which make creating these expressions much easier to understand.
• **Replace**: This string field is used to enter a replacement string for extracting the specific error information from the error message line located in the Search field.

• **Extra**: This string field is used to enter additional regular expression strings to be used to locate error information that is not located in the error message. For example, this could be used to locate the column the error occurred on for compilers that show the error message on one line, the line with the error on the next line and following that a ^ character to show the column in which the error was found.

### Error Parsing Expressions

Setting up error processing regular expression strings in the Compiler/Program Type Set up dialog is reviewed in this section. We feel this is the best and easiest way to process errors in the captured error file because the user can update these fields and add support for their own compilers without needing to write a macro to do so.

To use this feature the Search and Replace strings, at a minimum, must be defined. This is usually sufficient to handle most compilers. The Extra field is provided so built-in error processing macros can do further searching to locate additional error information for those compilers that do not provide all of the needed information on a single line.

To process errors, the built-in error macro uses the text in the Search string to scan the captured error file in an attempt to find a match for a line that contains an error message. There are four items that the error processing macro is looking for to be able to intelligently show you where the error occurred:

- **File Name (Optional)**: The file name of the file in which the error occurred. This is needed to be able to locate the file that has the error, so that Multi-Edit can load it and position the cursor on the line that contains the error. Not all compilers show the file name of the file that the error was in, so the default is the name of the file that was in the current window when the compile was started.

- **Line Number (Optional)**: The line number on which the error occurred. If the line number is located, the cursor will be positioned at the new line; otherwise, the cursor will remain where it was when the compiler started.

- **Column Number (Optional)**: The column number in which the error occurred. When provided, Multi-Edit will position the cursor on this column, which is assumed where the error occurred. Otherwise the cursor will be positioned to column 1.

- **Error Message**: The located error message. This will be what is shown on the status line when an error is located.

The Search string is a UNIX style regular expression that should match complete lines that contain error messages. Thus it should always start with the ^ character and end with a $ character. You can use regular expression aliases in this string to make them more readable, which will be translated before the search is done. When designing this string, you should use the grouping characters "(" and ")" to group the expression into groups that match the file name, line number, column number, and the error message so that they can be extracted from the line and used to show the source line where the error occurred.

The extraction of the above information is done by using the Replace string to do a regular expression replace of the located error line, read the replaced line in to a variable and then change the error line back to what it was originally. Then the contents of the replaced line are parsed for one or more of the following strings to extract the provided information. The replacement string should contain one or more of the following where /# is the number of the associated replacement group.

\/^F=#\#
When parsed this will return the file name of the file with the error.

\/^L=#\#
This will parse to the line number on which the error occurred.
/C=\#
This will parse to the column number at which the error occurred.

/M=\#
When parsed this will be the located error message. There can be multiple \# with other text so that you can format the message the way you desire.

The Extra string is used to locate additional information that does not appear on the same line as the error message. This string will actually consist of multiple parts. The first part is used to specify the type of information for which to search. Specifying more than one search type allows for searching more than once piece of information. All of the possible values for the search type string are shown below.

/X=x
The search type x is one or more of the following letters. These specify the type of information that will be searched for.

<table>
<thead>
<tr>
<th>String Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>File Name</td>
</tr>
<tr>
<td>L</td>
<td>Line number</td>
</tr>
<tr>
<td>C</td>
<td>Column number or position</td>
</tr>
<tr>
<td>M</td>
<td>Message</td>
</tr>
<tr>
<td>M+</td>
<td>Append to Message</td>
</tr>
</tbody>
</table>

Following the search type string are sets of three more strings, one set for each specified type. These strings specify the direction in which the search will be done, a search string and a replace string. The search and replace strings are defined just like the above search and replace strings but are used to find another line that contains the information specified. In these strings x is replaced with one of the type letters shown above.

/XP=\#
The value of str would be one of the following listed below and is the direction the search should take to locate a new line. This direction is from the current position of the cursor, which is usually the last found position. The # is replaced by a number specifying the number of lines to search to try to locate a matching line. The number 0 will be used to cause the search to have no limit, i.e. search to start of file or end of file. Leaving # blank will be the same as entering a 0.

<table>
<thead>
<tr>
<th>String Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D#</td>
<td>Search forward # lines starting with the next line down</td>
</tr>
<tr>
<td>U#</td>
<td>Search backward # lines starting with the previous line up</td>
</tr>
<tr>
<td>F#</td>
<td>Search forward # lines</td>
</tr>
<tr>
<td>B#</td>
<td>Search backward # lines</td>
</tr>
</tbody>
</table>
The search string 'str' is a UNIX style regular expression that is designed to match a complete line that contains the information specified by x.

The replace string 'str' is a Unix style regular expression replacement string used to replace the found string to allow for easy parsing to get the specified information. For more information about this string, see the Replace string documentation above.

If the found line contains other information than the one specified then the replace string can also get this information without needing to do a further search.

---

**Examples**

Below are some examples of how to set up the Search and Replace strings.

**CMAC compiler**

When the CMAC compiler finds an error it will output an error message in the following format:

Filename(line num, col num): ERROR errornum: message

Example:

LANGUAGE.S(92,22): ERROR 1: Syntax Error: 'THIN'

The Search and Replace strings to locate this message in the error file would then be entered as follows:

**Search:**

```
^(<p>)(<i>,<b0>(<i>)): (Error <i>: . # ) $
```

**Replace:**

```
/F=/0 /L=/3 /C=/4 /M=/5
```

Following is the description of each element of the Search string.

- **^** - Match the beginning of a line.

- **(<p>)** - Match group 0 start, followed by an Alias for a filename with full path, followed by group 0 end. This will match filename in the error message line.

- **\** - Match a (character. The \ is needed to make the ( match literally because the ( character by itself is used to start a group.

- **(<i>)** - Match group 3 start, followed by an Alias for an integer followed by group 3 end. This will match line num in the error message line.

- **(<b0>)** - Alias to match 0 or more blanks (spaces or tabs).

- **(<i>)** - Match group 4 start, Alias to match an integer, Match group 4 end. This will match column num in the error message line.

- **\):** - Match a ) followed by a : and then a space. The \ is needed in front of the ) because the ) character is normally the end group character.

- **(Error <i>):** - Match group 5 start followed by the word Error followed by a space followed by an integer alias followed by a :.

---

*This is match group 3 because the <p> alias uses two match groups in its definition.*
.\#) - Match any character any number of times followed by group 5 end. This and the line before will match Error errornum: message in the error message line.
$ - Match end of line.

The description of the Replace string is as follows:

<table>
<thead>
<tr>
<th>String Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/F=\0</td>
<td>Replace group 0 with /F=filename.</td>
</tr>
<tr>
<td>/L=\3</td>
<td>Replace group 3 with /L=lin num.</td>
</tr>
<tr>
<td>/C=\4</td>
<td>Replace group 4 with /C=col num.</td>
</tr>
<tr>
<td>/M=\5</td>
<td>Replace group 6 with /M=Error #: message.</td>
</tr>
</tbody>
</table>

So for the error line with LANGUAGE.S on as shown above, the replace string would be replaced as

/F=LANGUAGE.S/L=92/C=22/M=ERROR 1: Syntax Error: 'THIN'

**Borland Pascal 7.0**

When the Borland Pascal compiler finds an error it will output an error message in the following format.

```
filename(line num): Error errornum: message line with error;
```

-or-

```
filename(line num): Warning warningnum: message line with warning;
```

Example:
```
D:\ERROR.PAS(7): Error 3: Unknown identifier.
j := 0;
```

The Search and Replace strings to locate this message in the error file would then be entered as follows:

**Search:**
```
^(<p>)\((<i>)\): ((Error)|(Warning) <i>:.+)$
```

**Replace:**
```
/F=\0/L=\3/C=\4
```

**Extra:**
```
/X=C/CP=D2/CS=^(<b0>\^)$/CR=//C=\0
```

Following is the description of each element of the Search string.

- ^ - Match the beginning of a line.

- (<p>) - Match group 0 start followed by an Alias for a filename with full path followed by group 0 end. This will match filename in the error message line.

- \ - Match a \ character. The \ is needed to make the \ match literally because the \ character by itself it is used to start a group.
(<I>) - Match group 3 start followed by an Alias for an integer followed by group 3 end. This will match line num in the error message line.

This is match group 3 because the <p> alias uses two match groups in its definition.

\): - Match a ) followed by a : and then a space. The \ is needed in front of the ) because the ) character is normally the end group character.

((Error)|(Warning) <i>: - Match group 4 start followed by either Error or Warning followed by a space followed by an integer alias followed by :.

.#: - Match any character any number of times followed by group 5 end. This and the line before will match Error errornum: message in the error message line.

$ - Match end of line.

The description of the Replace string is as follows:

<table>
<thead>
<tr>
<th>String Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/F=\0</td>
<td>Replace group 0 with /F=filename.</td>
</tr>
<tr>
<td>/L=\3</td>
<td>Replace group 3 with /L=lin num.</td>
</tr>
<tr>
<td>/M=\4</td>
<td>Replace group 6 with /M=Error #: message.</td>
</tr>
</tbody>
</table>

Notice that the column number is not in the found string. Since the column is marked with the ^ character two lines below the found message string an Extra string will be used to locate this line and determine the column the error was located on.

The meaning of the Extra string is shown below.

/X=C - Do an extended search for a line specifying the column number.

/CP=D2 - Search for the column message down 2 lines.

/CS=^(<b0>\^)$ - Match start of line followed by (<b0>\^) group 0 start followed by 0 or more blanks or tabs followed by the ^ character followed by group 0 end followed by $ match end of line.

/CR=//C=\0 - The replacement string will return the column as a string containing the ^ character.

Running your compiler

Tools | Execute Compiler allows you to use your favorite compiler to compile source code in a file without leaving Multi-Edit. The compiler you use and the manner in which Multi-Edit works with it depends on the Filename Extension Setup of the file being edited.
Multi-Edit is designed to support any compiler, and you can easily add your own compiler support. If you experience any trouble adding support for your compiler, please feel free to contact our Technical Support by email: support@multieditsoftware.com.

When Execute Compiler is selected, a standard Multi-Edit List Box appears containing compiler interface menu items from which you may select, add and modify compiler interfaces. Highlight and select the compiler interface you want to run. It will immediately start compiling source code in the window you are editing. The output to the screen, while the compiler runs, is user-configurable. Most users, however, choose to have the screen output displayed in a pop-up window.

When finished compiling, Multi-Edit will place you on the first compiler error in the source file, assuming you have at least one compiler error. The Compiler Results are displayed in the Tools Pane (unless you have the Auto Show Compiler Error Window option disabled). To move to subsequent compiler errors, use Tools | Find Next Compiler Error or press the Next button in the Compiler Tools Pane. Instead of automatically tracking the errors as you move the cursor, the Compiler Tools Pane requires you to hit Enter or double-click on the mouse. In addition, the errors in the Tools Pane are displayed in a different color when the window comes up.

Execute Compiler can also be used to run other programs, such as a Lint utility or debugger.

If you only have one compiler interface defined and you have configured Multi-Edit to bypass compiler menus with only one entry, the Select A Program list box will not appear and the compiler will run immediately. See Tools | Customize | User Interface.

Select Project | Properties | Tools (Tab) to specify the command setups for Project release, make, build and debug tasks.

Filtering Output

The compiler support in Multi-Edit is very configurable and can allow users to do things that weren't originally thought of. In the Compiler Setup dialog there is a field "Program Type" which brings up the "Compiler/Program Type Setup" dialog where the data needed to do error processing for different compilers is set. In this dialog are two "Compile Macros" fields that allow the user to specify Pre and Post compile macros, i.e. macro that can be run before and after the compiler/program is run. By using these fields with a couple of system macros, the user can create a filter program, i.e. a program that takes a file as input, does something to the text and then write its output to stdout and/or stderr.

The two system macros that can be used to run "filter" programs are provided in the Compiler.s file and are called PreCmpFilter and PostCmpFilter. See the Compile.s source file for more details about each macro.

The PreCmpFilter macro checks the current window for a marked block and copies the contents of it to a file named "FilterIn.tmp" located in the temporary directory. It also sets a global string variable, !FilterFile, with the full filename of this file. When this macro is specified in the "Pre Compile Macro" field of the "Compiler/Program Type Setup" dialog, it will be run before the compiler program is started so a compiler command line can use the metacommand <~!FilterFile> to pass this filename to the compiler.

The PostCmpFilter macro loads the captured output file that the compiler macro is currently using and copies the contents of it to the current window at the current cursor position overwriting a marked block if one is marked. Thus when this macro is specified in the "Post Compile Macro" field of the "Compiler/Program Type Setup" dialog, it will be run right after the compiler program has completed, allowing the output of a program to be pasted into the current file.
When both of these macros are specified in the Pre/Post Compiler Macros fields a filter program is created. But each can be used by itself to create other kinds of functions. If the PreCmpFilter macro is used by itself, then a marked block of text can be passed to a program but not be replaced when the program is complete. On the other hand, if only the PostCmpFilter macro is used, then the output of a program can be inserted into the current window without sending any text to the program.

Multi-Edit 9.10 comes with a "Filter" Program Type record already defined which will meet most needs, but additional Program Type entries can be created to handle different things. Below are examples of how to set up these types of programs.

Examples:

```
(Compiler/Program Type Setup)
Type: Filter
Compile Macros
Pre: Compile^PreCmpFilter
Post: Compile^PostCmpFilter

Type: Filter (Col Block)
Compile Macros
Pre:
Post: Compile^PostCmpFilter /BT=2

Type: Filter (Line Block)
Compile Macros
Pre:
Post: Compile^PostCmpFilter /BT=1

Type: HtmlTidy Filter
Compile Macros
Pre: Compile^PreCmpFilter
Post: Compile^PostCmpFilter /F=<~!FilterFile>

Regular Expressions
Search: ^line (<i>) column (<i>) - ((Warning)|(Error): .@)$
Replace: /L=0/C=\1/M=\2
Extra: /X=F/FP=U/FS=^Tidy \(.@\) Parsing (<q>)$/FR=//F=\0

(Compiler/Program Setup)
Description: Perl Reverse Filter
Command: Perl.exe c:\bat\PLRev.pl < <~!FilterFile>
Program Type: Filter

Descriptions: HtmlTidy Filter
Command: d:\Mew32\HtmlTidy -wrap 79 -ium <~!FilterFile>
Program Type: HtmlTidy Filter
```

**Tracking Line Numbers**

Most compilers add a line number to their error report so that locating the errors can easily be done. Multi-Edit uses this information to locate errors when processing a compile that was started from within it. This usually works very well as long as the source file isn't changed by adding/deleting lines or the errors are fixed from the bottom up. If the source file is changed, starting with the first error onward, eventually the error processing routine will not be able to correctly locate the next error since the line numbers have changed. File compares also uses line numbers to determine where differences happen and adding/removing a line will affect the ability to locate the next difference. To overcome this limitation, Multi-Edit 9.10 has been updated to add a new feature called Original Line Numbers.
The way this works is an additional line number variable is added to each line record. When a file is loaded, every line's original line number variable is set to be the same as the physical line number. Where they differ is when new lines are added or lines are deleted, the physical line number changes due to the inserted/deleted line while the original line number does not change. The compiler error processing and the file compare routines were also updated to position the cursor to the physical line number specified in the error or difference report as before but now a check is done of the original line number for the current line. If the original line number matches the physical line number no further action is taken. On the other hand, if the original line number doesn't match, the routine will start scanning the file forward or backward, depending upon the difference, for an original line number that matches the error line number.

The above works fine for the first compile or compare. But once a new compile or compare is done, the original line numbers need to be reset to match the physical line number, since the compiler uses the physical line numbers to generate the new error report. If the original line numbers didn't match the physical line numbers the correct line would no longer match. To resolve this issue, the ability to reset the original line number of a file was added. The conditions for resetting the original line numbers can be different for different situations, so the compile interface was changed to add a new "Reset Original Line Number” option. This option allows the user to determine which files are reset before and after a compile is done. The following values are currently supported:

- **Special (default):** When this option is specified the following string field can contain a macro command line that will be run before and after the compile is run. This macro is responsible for calling the macro to reset the original line numbers. When the macro is run after the compile, it is passed the "/POST=1" parameter. If there is no macro command line specified, then the original line numbers are not reset.
- **Current:** This option causes the original line numbers for only the current file to be reset before a compile.
- **Project:** This option causes the original line numbers for all files specified in the current project that are loaded in Multi-Edit to be reset before a compile.
- **All Files:** This option causes all loaded files to have their original line numbers reset before a compile.

For file compares, the original line numbers of both files are always reset before the compare is done.

An option was also added that will enable the original line number to be shown on the status bar with the physical line number. The original line number will appear before the physical line number and be enclosed in [ ]s. A [*] shown for the original line number represents a new line.

In support of the Original Line Number, the new OrigLineNumTime buffer variable was added. When a file is first loaded, this variable is set to the file time of the file. Whenever the original line numbers are reset, this variable will be updated to contain a file time equivalent to current time of when the reset was done. This variable could be used to determine the status of the original line numbers when using an external tag database to determine the best method for locating lines.

### Compiling on a remote computer

The ability to run a program on a remote system via a Telnet login will support any system that allows a telnet login. Any command that can be issued from a telnet command line can be run, the output captured and brought into Multi-Edit for error processing. This feature is accessed through the Compiler setup for individual extensions. In order to use this feature, you must select from the Compiler setup dialog the exe type to "Remote Telnet" then pick a Telnet server for the host.

#### Setup Dialog

- **Name:** This is just a descriptive name to identify the server. Must be unique to other servers.
- **Add:** This is the host add. Can be an IP address or a DNS address.
- **Port:** The port that the telnet server uses, usually 23.
• **UserID:** The name of the user to login as on the server.
• **Password:** The password of the user being logged in.

There is no local encryption on this password yet.

• **User ID Prompt:** This is the text that precedes a user typing in their login name. Usually, "login:" the last 4 chars of "Login:"  
• **Password Prompt:** This is the text that precedes a user typing in their password. Usually, "sword:" the last 5 chars of "Password:".

Both of these could use the full text all that is needed is a unique pattern match.

• **Shell Prompt:** This is the text that precedes a user typing a command line. This one is more difficult since every user could have a different prompt based on shell used and the environment setup. Possibly, ":>" or ":$".  
• **Eol Seq:** This is the set of characters used to terminate a new command line entry. Most popular would be "\r" or "\r\n". Also, all standard C style escape characters may be used.  
• **Login Timeout:** Time in milliseconds to wait for a login prompt.  
• **Compile Timeout:** Time in milliseconds to wait for the program to finish compiling.  
• **Chdir Command:** This is the command to be used by the remote system to change to a directory. A common way to type this would be "cd". "cd" will be used if no entry is made.

Issues to address: Conversion of path delimiters and filenames. If a file is mounted remotely, we need to be able to pass the filename and change the path entries for the remote system. For now, set the working directory to the directory of the files and use filename only no path.

**METerm**

**METerm Introduction**

Telnet support in earlier versions of Multi-Edit supported remote compilation in a very user-unfriendly fashion. For example, each compiler entry had to have a telnet server assigned to it, so if you used the same compiler on multiple remote hosts, you had to set up duplicate compiler entries for each server.

METerm is a new approach to telnet support in Multi-Edit 9.10. The following notable enhancements and new features are provided:

• Compiler entries are now independent of telnet servers, as they should be. If a compiler entry is set up as a Remote Telnet compiler, then the user will be prompted for the desired telnet server after selecting the compiler from the compiler list box. This eliminates the requirement for multiple compiler entries being locked into a single telnet server.  
• Telnet configurations now support scripts that can be executed on the remote host before and after a compile operation. This is useful for preparing for a compile on the remote host, and for cleaning up after a compile. Multi-Edit metacommands are fully supported in these scripts, including the new metacommand PROJECT_FTPROOT, which will allow you to reference the remote FTP directory assigned for the Project. This allows the telnet operation to work more seamlessly with the FTP configuration used to transfer the source files to the remote host.
• METerm can run Stand-Alone, providing a simple interactive telnet terminal for quick access to your
telnet hosts. While METerm isn’t designed to be a full-features terminal emulator, it is functional
enough for most purposes. It is useful for quick chores and can conveniently be launched from within
MULTI-EDIT. Currently, VT100 and TTY emulations are supported, which cover the vast majority of
telnet servers.
• Passwords within the Telnet configuration are now hashed to prevent clear-text passwords from
appearing in MULTI-EDIT configuration files. Note that this isn’t intended to be a secure encryption,
but simply a scrambling algorithm to thwart casual glances and minor hack attempts.

Due to some outstanding issues with the EncodeStr macro, password hashing is currently disabled,
meaning passwords are still clear-text until this is fixed.

• Full telnet trace logging is now supported in each telnet configuration. This feature is enabled when a
filename is entered in the Log File field on the configuration dialog. This is useful for troubleshooting
communications problems.

Setup Instructions

First, create a compiler entry for the remote compiler. This will require knowledge of the command-line
syntax and options for the specific compiler, as well as configuring of error-parsing support. MULTI-EDIT
already ships with some compiler setups that support remote compiles, such as GCC. It is beyond the scope
of this document to go into detail on compiler setup, beyond just the note that “EXE Type” should be set to
“Remote Telnet.” If you use the same compiler for both local and remote compiles, you will also need to
create a separate remote compiler entry from a local compiler entry. For example, if you use javac or GCC
for local and remote projects, you must configure a local compiler setup and a remote compiler setup.
However you only need a single remote compiler setup no matter how many remote servers you will target
(assuming of course the compiler syntax and options are compatible with all remote servers). This differs
from the previous telnet support where each remote server required its own compiler setup.

Next, create the telnet configuration to be used for the desired remote server(s). This can be done either by
directly running the macro `Telnet^TelnetSetup` or during compiler execution when you are prompted to
select a Telnet Setup. Telnet Setup uses the conventional Multi-Edit DB dialog format and stores its database
in Defaults\Telnet.db. You create a new setup by copying an existing setup, including the Blank Template.

The `Insert` button on the DB dialog is disabled for future use.

The server `tbird1200` is currently selected. If this dialog were presented following a compile command with a
compiler configured as “Remote Telnet,” then clicking Select would initiate the compile on the selected
server. If this dialog is cancelled with the Close button following a compiler selection, an error dialog will
appear informing the user that the compile is aborting due to a telnet server not being chosen.

To create a new entry, copy the “A Blank Server Template” entry, rename it as desired, and edit the options in
the configuration dialog on the following page. Existing entries are edited by clicking the `Edit` button.
Many of these parameters are carried over from the original telnet support and thus should be familiar to those who have already used (or tried to use) telnet compiling in prior versions. Below is an item-by-item description:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>The name of this configuration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>The host name or IP address of the remote server.</td>
</tr>
<tr>
<td>Port</td>
<td>The remote port number (default is 23 for standard telnet servers).</td>
</tr>
<tr>
<td>User ID</td>
<td>The user ID to log on to the remote server.</td>
</tr>
<tr>
<td>Password</td>
<td>The password to log on to the remote server.</td>
</tr>
</tbody>
</table>

**Terminal Settings**

<table>
<thead>
<tr>
<th>Answer</th>
<th>The telnet answer string sent to the remote server during telnet option negotiation. This is regardless of emulation type specified in the next option, but should coincide in most cases. If blank, the telnet client responds with “NETWORK-VIRTUAL-TERMINAL” according to the telnet RFC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The terminal emulation to be used. Currently VT100 and TTY (basic text with no control codes) is supported.</td>
</tr>
<tr>
<td>EOL</td>
<td>The end-of-line sequence used by the remote server. Expressed using C/CMAC style escape characters, usually \n for LF (common on Unix systems) or \r\n for CR/LF (common on Microsoft systems).</td>
</tr>
</tbody>
</table>

**Prompts**

<table>
<thead>
<tr>
<th>User ID</th>
<th>The string sent by the server to prompt for user ID during login. Used to facilitate automatic login to the server. This may be a partial string, and usually the last few characters of the prompt. The match will be case sensitive, so avoid including characters that may be presented in different cases at different times.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>The string sent by the server to prompt for user password during login. Used to facilitate automatic login to the server. Case-sensitive.</td>
</tr>
<tr>
<td>Shell</td>
<td>The shell (command-line) prompt sent by the server following a successful login, where the server waits for user input before proceeding. Usually set to the last few characters, but be sure not to include dynamic data that might change, such as current path information. For example, on Linux hosts with standard prompts, using ]$ for non-root users or ]# for root users would be sufficient.</td>
</tr>
</tbody>
</table>
Timeouts

<table>
<thead>
<tr>
<th>Login</th>
<th>The maximum amount of time allowed for the login process to succeed. Specified in milliseconds (1/1000sec).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile</td>
<td>The maximum amount of time allowed for a compile to complete before METerm gives up. Specified in milliseconds (1/1000sec).</td>
</tr>
</tbody>
</table>

Other Options

| Log File     | An optional trace file, useful for troubleshooting connection problems. If set to a filename, all telnet communications will be logged to this file. Path is relative to the MULTI-EDIT root directory, unless an absolute path is specified. |

Entering Pre and Post Compile Scripts

The other tabs in the Telnet Setup dialog are for entering optional scripts that will be executed on the remote host before and after the compiler command is executed.

Finding Errors

Find Next Compiler Error

Most compilers give you a listing with more than one error when they've finished compiling. This command allows you to move to each successive syntax error. The compiler results are displayed in the Tools Pane (unless you have the Auto-arrange Compiler Error Window option disabled). To move to subsequent compiler errors, use **Find Next Compiler Error** or press the Next button in the Compiler Tools Pane.

Find Previous Compiler Error

Most compilers give you a listing with more than one error when they've finished compiling. This command allows you to move to each successive syntax error. The compiler results are displayed in the Tools Pane (unless you have the Auto-arrange Compiler Error Window option disabled). To move to subsequent compiler errors, use **Find Previous Compiler Error** or press the Previous button in the Compiler Tools Pane.

Language

Property Strings

Multi-Edit now makes use of Property Strings allowing language specific functionality to be added without having to add more and more fields. An example of this can be seen in the HTML language/extension setup where Property Strings are used to define the embedded scripting language.

Property Strings may be assigned numeric or string values. When a file is loaded, first the language Property Strings are set, followed by the extension Property Strings. Therefore, in the case of duplicate Property String settings, the extension Property Strings will override the language Property Strings.
Property Strings are configured under Filename Extension Setup or Language Setup. The following settings are currently accepted:

- **SCRIPT=1**: Process embedded scripts with the <SCRIPT> tag
- **SCRIPT_DEF=language**: The default language to use for the SCRIPT tag. If this is not defined, it defaults to JavaScript.
- **STYLE=1**: Process embedded style sheets with the <STYLE> tag
- **STYLE_DEF=language**: The default language to use for the STYLE tag. If this is not defined, it defaults to CSS
- **ASP=1**: Process ASP code. This enables the use of <%= %> style tags.
- **ASP_DEF=language**: The default language to use for ASP tags. If this is not defined, it defaults to VBSCRIPT.
- **COLDFUSION=1**: Process cold fusion scripts (CFSCRIPT and CFQUERY).
- **PHP=1**: Process embedded PHP. This enables the use of <? ?> style tags.

### Matching

The Match Language Structure command enables locating a language construct, such as ( ), { } and begin/end, matching the one under the cursor. It is also used for quickly navigating through a source file and to check that the code is correctly formed.

Using this feature is as simple as placing the cursor on a construct and hitting the key assigned to it or selecting the **Tools** | **Match Language Structure** menu entry. If the language setup for the current extension does not support this feature, then the "NOT Supported for this extension." message will be shown on the status line. Otherwise, the cursor is on the matching construct when one is found, and the text between it and the starting construct could be highlighted.

Most languages supported by Multi-Edit already have this feature configured and the few that don't can have it added by following the instructions in the Adding Language Support topic. This feature usually works without any user set up but the few options the user can change are set via the Language Setup dialog for each specific language. The main user selectable option would be the ability to enable the highlighting of text between the opening and closing structure on a successful match. This option is enabled by checking the "Match language structure highlight" option in the Properties Setup dialog, shown by clicking on the Formatting/Config button from the Language Setup dialog.

For the matching to function correctly the following must be set in the specific Language Setup dialog:

1. The Init Macros field should contain a macro command line for the language specific Init macro.
2. The Match Macros field should contain a macro command line for the language specific Match macro.
3. The Config Macros field should contain a macro command line for showing the language specific Properties Setup dialog.

Below is a general overview of how the Match Language Structure feature is implemented.

Whenever Multi-Edit loads a file, it runs the ExtSetup macro, which locates the language record for the current extension and executes the specified Init macro. This Init macro is responsible for setting three global variables that contain the data that specifies the constructs that can be matched. The first global specifies three sets of delimiter characters, ones that can appear before the construct, ones that can appear after a construct and a delimiter character used to separate construct names that are defined together in one record. The other two global variables contain the patterns and search parameters, one to specify the opening constructs and the other to specify the closing constructs.

Now when the Match Language Structure command, i.e. the Language^Match macro, is selected, the Match macro will run the macro specified in the Match Macros field of the specific Language Setup. This macro is
usually a wrapper for the general purpose Language^LangDoMatch macro. This macro first calls a language specific XxxGetMatchPat macro, which is responsible for locating and returning a pattern to match.

This macro can be as simple as to only check the cursor position and returning the pattern or as complex as to scan the line searching for one of a number of supported patterns. When this macro returns, the LangDoMatch macro uses the found pattern to scan first the opening construct pattern data global and then the closing construct pattern global for a pattern match. If a matching pattern is found, the search parameters from the found record of the appropriate global pattern data are set. Then a search is stated, a forward search for a match in the opening pattern data global or a backward search for a match in the closing global pattern data. The search continues until either a match or no more matches are found. Once a match is found, the LangDoMatch macro will check the search parameters to see if it needs to continue searching or to finish by highlighting the text if the highlight option is enabled and both the opening and closing construct can be shown in the window. When the pattern data is correctly set up, nested constructs are skipped so that the correct ending construct is found.

**Indenting**

For most languages the use of indenting is ignored by the compiler and is used mainly to help the programmer understand the flow and grouping of code. Although there are a few languages such as Python where indenting is a requirement of the language. Since Multi-Edit has support for multiple languages, it also supports multiple indent styles. These are set up, as most of the language support, via the Filename Extension Setup dialog.

Since Multi-Edit tries to format the code as it is being entered, the Indent Style setting come into play when the Enter key is hit. Where the cursor is positioned on the following line depends upon the setting of the Indent Style as well as the language being used. Setting the Indent Style for a file is done by first selecting the Edit Filename Extension Setup dialog and choosing one of the three Indent style radio buttons.

The Off Indent Style option, when enabled, causes the cursor to always be positioned in column 1 of the following line. This option is useful when no indenting is desired such as for documentation or text files.

The Auto Indent Style option, when enabled, causes the cursor to be indented so that it will line up under the first word of the current line. This is useful for quickly creating columns of text or for entering code that is to be indented but no extended language support currently exists for it.

The Smart Indent Style option would be selected for languages that have extended language support macros and will cause the cursor position to be positioned based upon the context of the code. Most of the major languages have support macros that include smart indenting.

The Off and Auto Indent Styles require no more set up after selecting them, but the Smart Indent Style requires additional set up. When the Smart Indent Style is enabled, the correct Language must be selected and the items in the Language Setup dialog must be correctly set. Since each language can have different settings, a language specific Properties Setup dialog is used to set these features. Languages that use this dialog are set up to include a macro command line in the Config Macros field of the Language Setup dialog, which specifies the macro to run to display the dialog. Also for the Smart Indenting to function, the Indent Macros field must contain the macro command line of the language specific indent macro.

When the Config Macros field has a valid macro command line, the Formatting/Config button will bring up this custom Properties Setup dialog for the selected language. For languages that support different indent styles such as C, there is usually an entry to specify the style of indent with an example of how it appears in code. Also there could be options that control outdenting and/or auto adjustment of ending constructs for alignment with the opening construct, i.e. the closing brace of an if statement. It might take some experimenting with the different options to get the desired results for each language.
Once the options are correctly set, the Enter key will cause the specific language Indent macro to be called, which analyzes the code context and then positions the cursor to the correct column based upon the context and the other property settings. The auto adjust features are activated when the appropriate key is entered or the Enter key is used.

Filename Extensions

Almost everything in Multi-Edit is filename extension driven. File by file configuration, based on the file name extension of the file, is one of the keys to the power of Multi-Edit. For example, you can have completely different configurations for .C files as opposed to .ASM files. Multi-Edit utilizes Filename Extension Setup in order to place the tab settings and to identify the programming language being used.

One of the list selections is Default. Multi-Edit will not allow you to delete the Default item since it is used as a 'none of the above' file extension.

To view the Filename Extension Setup dialog, select Customize from the Tools Menu and click on the Customize tab. Press the Filename extensions button to display a standard Multi-Edit list box with a list of existing Filename extensions.

Create “Edit with Multi-Edit” shell extension

The Edit with Multi-Edit shell extension is grouped with the Include and Exclude.

- **Include:** If selected, the option to Edit with Multi-Edit will only show up in the right-click context menus of those file types you enter their extensions for in the box to the right of Include.
- **Exclude:** If selected, the option to Edit with Multi-Edit will show up in the right-click context menus of ALL file types excepting the ones whose extensions you’ve listed in the box to the right of Exclude.

The “Default file open extension(s)” is to let you list any extensions Multi-Edit should try when handed a file with no extension. It will try them in sequence until it finds one that works. If that fails, it will open the file under the extension DEFAULT. You'd mainly use it for extensions that are either executables or scripts with associations in the file system. Its a mechanism for Multi-Edit to try and execute a file when the user didn't specify an extension. If nothing in the list works, Multi-Edit just opens it as a file, using the settings for the DEFAULT extension, which is linked to the language NONE.

Buttons along the side of the list box allow you to **Edit, Insert, Delete** or **Copy** the selections. You can also rearrange the list by moving them up and down in the menu list. The Search and Again buttons allow you to quickly find the list member you are looking for. When you press the Select button, Multi-Edit will enable the currently highlighted command mapping.

Select an item in the list and press the Edit or Insert button to display the Edit Filename Extension Setup dialog box with the following fields for editing:

**Extension(s)**

This field will allow you to enter as many extensions as you wish with the following rules:

- A space, comma, or semicolon must separate each extension.
- Do not include the period before the extension.
Example:

C CPP H HPP
This would make the extensions .C, .CPP, .H and .HPP all have the same extension specific options.

Edit Mode

• **Text:** In this mode, the line number shows its position relative to the entire document, and not relative to any individual page. Also, the page number display is turned off.
• **Document:** Use this mode if you have page breaks in a document and want to display whichever page you are viewing. Line numbers reflect the relative position on each page.

Indent Style

• **Off:** With this style, pressing ENTER positions the cursor on the new line at the currently defined indent level. The indent level is set through the use of the Indent and Undent commands that are set up in the Command Map by default.

---

*If your document is in Overwrite mode, pressing ENTER will not create a new line, but merely move the cursor to the next line. This is true regardless of the Indent Style chosen.*

• **Auto:** With the Auto style, the cursor position on the new line is where the first letter of the first word was on the old line.
• **Smart:** This will indent according to the language type you configured for this file extension.

Options

• **Word Wrap:** This will cause text to wrap to the next line upon reaching the right margin. Even when text is inserted in the middle of a line, text at the line's end will be dynamically wrapped.
• **Auto-Template Expansion:** This will cause language-specific templates to be detected and expanded upon hitting the space bar. If this checkbox is not checked, then template expansion must be forced by invoking the Build Template command. See Template Editing for information on how to customize templates.
• **Line numbers:** Check to turn on display of line numbers in the editing window.

Tab settings mode

• **Use tab and margin settings:** Mark this if you want to use the tab settings entered in the Tab Spacing and Right Margin fields.
• **Tab spacing:** Allows you to change the default tab spacing.
• **Right margin:** Allows you to specify the column number at which you want words to start wrapping to the next line (see Word Wrap).
• **Use format line:** Check this option to use the format line for this extension.
• **Edit:** This option gives you the ability to define a custom Format Line for this extension. The Format Line can be changed if you want unevenly spaced tab stops.
The following keystrokes are available:

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>Inserts (or overwrites) a tab stop</td>
</tr>
<tr>
<td>R or r</td>
<td>Sets the right margin at the column where it's entered</td>
</tr>
<tr>
<td>Del</td>
<td>Deletes the character under the cursor</td>
</tr>
<tr>
<td>Backspace</td>
<td>Deletes the character to the left of the cursor</td>
</tr>
<tr>
<td>End</td>
<td>Moves the cursor to the end of the Format Line</td>
</tr>
<tr>
<td>Home</td>
<td>Moves the cursor to the beginning of the Format Line</td>
</tr>
<tr>
<td>Left Arrow</td>
<td>Moves you to the left on the Format Line</td>
</tr>
<tr>
<td>Right Arrow</td>
<td>Moves you to the right on the Format Line</td>
</tr>
<tr>
<td>Enter</td>
<td>Exits Format Line editing and saves the changes</td>
</tr>
<tr>
<td>Esc</td>
<td>Exits Format Line editing without saving changes</td>
</tr>
</tbody>
</table>

- **Expand to spaces:** When pressing a tab, spaces are entered to fill the space. This overrides the default in Tools | Customize | Editing.
- **Language:** Allows you to select the language type you wish to use. See Language Support for more information.
- **Template:** Allows you to specify a template set to use with this extension. See Templates for more information.
- **Colors:** Allows you to set up extension specific window colors. Selecting this will bring up the Window Colors dialog.
- **Font:** You may set up a specific font for each extension set up, if you wish. If one is set up for a particular extension, it will appear in this field. To the right of the field is a button, press it to select a specific font for this extension. The OEM three-state box in this dialog serves an important function dealing with OEM Translation. When the box is checked, OEM translation is turned on. When unchecked, OEM translation is not active. When in the gray state, Multi-Edit uses the default OEM translation setting under Tools | Customize | Fonts. This three state is provided to let you turn on OEM translation for a font that is not listed as an OEM font.

**File Type**

This control allows you to set the default line terminator for the file extension. DOS files typically use a carriage return and a line feed to mark the end of the line. UNIX, however, uses only line feeds to mark the end of the line. Binary files have no line terminators - they are continuous streams of data. Mac files only use a carriage return to mark the end of the line. Multi-Edit uses this file type to determine both where to break lines when loading a file and which characters to insert when the ENTER key is pressed during normal editing.
• **Auto Detect**: This option will enable the Multi-Edit automatic file type detection feature. When loading a file of the specified extension, Multi-Edit will attempt to determine the file's type. You should use caution when using this feature with files that mix "file types." In other words, the file detection feature may not work correctly for files that terminate some lines with a CR and LF and terminate others with only a CR.

• **MSDOS text**: This option will load the file as a DOS file. A carriage return/line feed character combination is the line terminator for this file type.

• **UNIX text**: This selection will load the file as a UNIX file. A line feed is the only line terminator for this file type.

• **Mac**: Selecting Mac file type will load the file as a Macintosh file. Mac files only use a carriage return to mark the end of the line.

• **Binary**: Binary files have no line terminators and are displayed as a continuous stream of data. This option is for files that you want loaded as fixed length records, such as Binary data files. Even .EXE files can be "patched". All characters, tabs, line terminators, etc. are editable and are given no special interpretation.

**Record Length**
If you selected the Binary file type, you need to enter a value here. We recommended a value of 78 or less so you can see the entire file on the screen without needing to scroll horizontally. For example, if you want to view your binary file as 16 byte records, set the Binary Record Length of that extension to 16. Each line in the binary file will have a length of 16 characters. No line terminators will be placed at the end of each 'line'; the characters will be simply wrapped at that point.

**Post-load macro**
Enter the name of a macro that you want to run immediately after loading a file with this extension. This field is optional.

**Pre-save macro**
Enter the name of a macro that is to run immediately prior to saving a file with this extension. This field is optional.

**Default Help file**
Allows you to name an extension-specific help file, providing context sensitive help for that extension. Such a help file for Multi-Edit's Macro Language (.S) is supplied to all users. For example, if you want to use the file BCW.HLP for context sensitive help on all your .C and .CPP files, you might put C:\BC45\BIN\BCW.HLP in this field. You can separate multiple help files by a semicolon. In order to ease entry, you may use metacommands to obtain help files from the Help Manager.

**Add-On templates**
Allows entry of additional templates such as a Windows API template. Enter the name of the templates separated by semi-colons.

**Default directory**
Allows you to name a directory or group of directories where the program first looks when a file is loaded. This option is handy if you don't want to always type in the directory when loading a file with this extension. A semicolon must separate multiple directories. You may also use metacommands, which are explained in Multi-Edit Metacommands.

**Property Strings**
Allows the user to specify optional settings for this extension. See Property Strings.

**Compiler/Program Set up**
Allows you to specify and customize compiler set ups for the extension you have selected.
TipWin

TipWin Overview

The TipWin feature displays a function's definition in a pop-up as you type the function name, greatly reducing time wasted hunting for help when all you need is a quick reminder of the function prototype. It can also automatically create function templates that fill in the functions' definition for you.

At present, TipWin supports several languages, retrieving the necessary information from several sources: C and C++ from MeTags and BSC databases; CMac from MeTags and a database of kernel and import functions; Delphi/Pascal from MeTags; and Java, using experimental MeTags support. Since TipWin's functionality is dependent on the quality of data available to it, you should create tag files and, where applicable, BSC browse databases, and keep them as up-to-date as is reasonable for your project by incrementally updating files that change. For the truly ambitious, it's always possible to create function databases from scratch, using the CMac database, CMac.fct as a model.

You can independently configure TipWin's two functions, hint popups and template expansion, to run automatically, manually, or not at all from the TipWin section of the Customize dialog. To display a TipWin popup manually, position the text-insertion cursor anywhere over a function call and run the macro TipWin^DisplayFunctionTip. If you do this frequently, you'll probably want to assign its command mapping to a hotkey, button, or menu item. Likewise, to manually expand a template, type the function name, followed by the opening parenthesis of its argument list, then run the TipWin^ExpandFunctionTemplate macro. Note that with the single exception already mentioned, All TipWin functionality requires the presence of a recognized function name followed by a left parenthesis. If either option is set on automatic, it triggers the instant you type the opening parenthesis. The first argument in an expanded template will be selected by default, and the matching argument in the TipWin popup. Since the arguments are enclosed in backticks, you can use the Ctrl+I and Ctrl+U keys to move quickly between them, or move the cursor manually. If you wish, TipWin will syntax-highlight in the popup the argument the cursor is on. Once you move the cursor out of the function or its argument list, the popup disappears. If you use the TipWin popups, but not the template expansion, the highlighting follows the cursor as you type in the arguments.

Automatic function templates

TipWin generates and expands function templates from the same set of information sources it uses to display hint popups.

You can undo a template expansion in one undo, should you wish. Note that at present TipWin supports function templates for the same set of languages as for hint popups, with the exception of Delphi/Pascal. Don't confuse TipWin's automatic templates with the template facility discussed in the next section; they're quite similar in concept, but use a different mechanism.
TipWin configuration

To display TipWin's configuration dialog, select Tools | Customize... from the main menu bar, then click on the TipWin item in the topic tree in the left pane of the Customize dialog. You can then set the following items to suit your preference:

Options

- **Automatic parameter info**: If checked, typing a recognized function name followed by an opening parenthesis causes TipWin to display popup hints automatically.
- **Use colors**: Popup hints appear in the default system colors (usually black text on a yellow background) unless this box is checked, in which case TipWin uses seven custom colors instead. The current user interface doesn't provide a means of changing these colors, but it can be done using the TipWin^TipWinSetTwColors and TipWin^TipWinSaveTwColors macros. Examine their source code in TipWin.s to see how to use them.
- **Follow cursor**: By default, when TipWin displays a hint for a multi-line function, it positions the hint popup just below the current line, where it stays, so long as the cursor is within the function. Checking this box causes the hint popup to move, tracking the cursor, so that it's always immediately beneath the current line.
- **Prefer position above**: TipWin displays hint popups below the current line by default; checking this box causes it to display them above the current line instead. Whatever the state of this option, if there isn't room for the popup at the preferred location, TipWin displays it in the other location.
- **Display info source**: When this option, currently supported for CMac and Delphi/Pascal, is selected TipWin displays the source of the current information within square brackets in the hint popup. The possible values are MeTags [t], DLL imports [i], and Multi-Edit system functions [m].

Display default parameters

This option controls how TipWin displays any default values a function's parameters may have. The possible behaviors are:

- **Always**: Always display default values.
- **When selected**: Display an argument's default value only when it's selected.
- **As a hint**: Display default values as a hint popup.
- **Never**: Never display default values.

Function Templates

- **Automatic expansion**: When this option is checked, typing a known function name followed by an opening parenthesis automatically expands the function's definition as text in the editing window.
- **Expand empty parameters list**: When checked, this option permits TipWin to expand functions that have empty argument lists, or ones containing only "void" or "VOID" arguments.

Expand style

This option, available only when automatic expansion is enabled, determines whether to expand a template based on the current cursor position. The possible behaviors are:

- **Always**: TipWin always expands templates automatically.
- **End of line**: TipWin only expands templates if the line is empty after the cursor.
- **Smart**: TipWin uses heuristics to determine whether to expand templates. It will expand a template if the rest of the line is empty, or if a function expansion makes sense in the current context.
For example, in a C-like language, TipWin would expand a template if the character following the cursor were a semi-colon, but not if it were a left parenthesis. The key here is that after expansion, the first case would result in valid code, but the second would not.

**Expand parameters as**

This option determines how template arguments will be formatted. In all three cases TipWin expands the argument within enclosing backticks, making selection easy using the Ctrl+I and Ctrl+U keys.

- **Both type and name:** TipWin expands each argument as it would appear in a function definition, with both the data type and the variable name.
- **Type only:** TipWin expands each argument to just its data type.
- **Name only:** TipWin expands each argument to just its variable name.

---

**Templates**

**Understanding Templates**

Templates provide a way to reduce the amount of typing needed when entering large amounts of repetitive code. Some good candidates for templates are flow control statements, documentation headers for routines and API functions. For example, with a routine header, a template could be created that enters the basic outline of the header with template field markers placed where the routine-specific information would be entered and, with a few keystrokes, would be fully expanded with the cursor positioned on the first field marker.

Still don't know what templates are? Well then, lets use a C `if` condition for an example. In the language C, a typical `if` condition has the following structure.

```c
if ( condition ) {
    statement
}
```

With the C templates provided with Multi-Edit, after typing `if` followed by a space, the structure above would be automatically generated for you and the cursor would be resting at the first letter of the word `condition`.

The template data for each template set is stored in a special file with an extension of `.TPT` and are associated with a file extension in much the same way as a language type. You can set the template set for each extension by selecting **Tools | Customize | Filename Extensions**. Templates are easy to create and you can create customized sets of templates on the fly for programming languages, documentation, etc. using the **Tools | Edit Templates** dialog box. This dialog is modeless, which means you can leave the box up and edit simultaneously.

Global templates, if enabled, are available to any Editing Window regardless of file extension and can be used in the same way as regular templates (i.e., expanded via a keyword and/or via a template expansion metacommand from any other template set).

**Enabling Templates**

Templates are organized in what are called 'template sets'. Each template set is stored in a file with an extension of `.TPT`. There are three classes of templates sets: language templates, global templates and other templates.
Every language that Multi-Edit supports comes with a template set that contains templates specific for that language and is named after the language (i.e., C.TPT, PASCAL.TPT and JAVA.TPT). A language template set is specified in the Template field of the Extension Setup dialog for each extension (Tools| Customize | Filename Extensions | Edit). Only one language template can be assigned to an extension, although other templates and global templates can still be used.

All global templates are stored in the file named GLOBAL.TPT, when enabled, are available to any Editing Window regardless of the file extension. These templates would be templates that contain text that you would enter into any type of file such as your name or company name, etc. To use global templates, you must enable them by checking Enable global templates in the Tools | Edit Templates dialog and Auto-Template expansion in the Extension Setup dialog (Tools | Customize | Filename Extensions | Edit) for each file extension for which global templates are to be used. When enabled, global templates will be searched for after the language template set is scanned. Thus, a template in a language set by the same name as a global template will be expanded instead of the global template.

Other templates are templates that may not be specific to any language such as templates for the Win32 APIs or templates that are specific to a vendor's language product such as Borland C++ runtime libraries. Any number of these template sets may be used and are set up by entering their names, separated by semicolons, in the Add-On templates field of any Extension Setup dialog.

Creating a Template

All templates are user-configurable and new ones can be easily added. To change or add templates, use Tools | Edit Templates to bring up the Edit Templates dialog where you can edit, add, delete and create new template sets.

Expanding a Template

Template expansion can be invoked by selecting Tools | Build Templates or by enabling Auto-Template expansion in the Extension Setup dialog (Tools | Customize | Filename Extensions | Edit). This causes the space key to trigger template expansion.

Template expansion can be performed at any time while editing text. When invoked, the template system checks to see if there are characters before the cursor that match a defined template. If a unique matching template is found, it is automatically expanded; otherwise, a list of possible templates is shown. From this list, select a template and press Expand to cause the highlighted template to be expanded at the cursor location. For example, you're working in C language and type "s" and then highlight and select the Build Template command. Multi-Edit will then build the following template:

```c
switch ( ){
    case :
}
```

The toolbar button method of template expansion can be demonstrated by using the HTML templates. When one of the HTML template toolbar buttons is selected, the template system expands the specific template for that button. These templates are special in that they have been designed to remove a block of marked text if it exists, insert the template text and paste the contents of the removed block into the correct place in the text. These templates can only be expanded from the toolbar button or selecting them from the Build Template list.

A new feature with 9.10 is the TmplPane Add-On. This provides convenient access to a set of templates appropriate for the currently selected file. After installation, a new template pane option will be available on the View menu. You may activate this feature by selecting it from the View menu.
By default the Template pane will 'follow' the current window. When you select a different window, the Template sets shown will match the file extension for the active window. For most languages, the top section will show language specific templates, while the bottom section will show the global templates.

For web related extensions (where the LANG buffer property equals HTML), you can choose ASP, Script, WebLair, or HTML templates for both the top and bottom sections previously mentioned. ASP and Script templates are found by looking at the ASP_DEF and SCRIPT_DEF buffer properties.

The template set choices can also be overridden by setting a buffer property TMLPANE_TOP or TMLPANE_BOTTOM in the extension or language setup. To use the template pane, select a template from the list and double click it to expand it at the current cursor position.

One feature built into templates that can be a great time saver is field strings or field markers. A field marker is text in the template between back tick characters. The purpose of these markers is twofold. First, they document what is expected at its location when entered in templates with many fields. Most importantly, the field markers provide a quick way to navigate through a template as it is being filled in.

In the default keymap, the Ctrl+I and Ctrl+U keys are assigned to functions that move the cursor to the next and previous template field marker. Using these keys allows you to move the cursor quickly to the next field marker and have the field marker block marked and deleted when text is typed in.

If ( expression )
   {
      'statement'
   }

Of course, not everyone likes the field markers being entered, so templates that do have field markers can be set to have the field markers stripped when the template is expanded. This is accomplished in the Tools | Edit Templates dialog by selecting Properties and then checking Strip field strings.

Adding a New Language

Overview

Multi-Edit ships with support for many of the most popular programming languages, C/C++, Pascal, Modula-2, DBase and many more but with a little work unsupported languages can easily be added. This document describes all of the language features Multi-Edit supports and how to go about adding these features for an unsupported language.

As Multi-Edit has evolved over the years, the features users want to change most frequently have been redesigned so that most of the changes could be made without resorting to writing macros and usually consist of filling in entries in dialogs. This idea has been applied to the language support also, but with the vast differences between programming languages there are still some areas where macros are required to provide the needed support.

A number of different language features are provided and can be added independent of each other. This means that when adding support for a new language, all of these features do not need to be implemented to reap the benefit of a particular feature. Thus the new language support can be added and tested in stages or not even support some features.
The following list of language features are provided in Multi-Edit and are shown from the easiest to implement to the hardest. Not all of these features will be covered in detail here since they are covered fairly well in the online help and the manual.

- Color syntax and keyword highlighting
- Code commenting and uncommenting
- Templates
- Compiler support and error processing
- Construct matching
- Smart indenting
- Function tagging
- Properties

The way Multi-Edit supports different languages is through the filename extension. When a file with a specific extension is loaded, a check is done to see if that extension has been defined, if it has, the language features associated with it are initialized. The Filename Extension Setup dialog is where the extension specific information is set up and the language associated with that extension is specified. It is also where the templates and compiler entries are associated with a given extension.

The language specific information is mostly defined in the Language Setup dialog accessed from the Tools | Customize dialog or the context menus. Templates are created and changed in the Edit Templates dialog accessed from the Tools | Customize dialog as well.

### Syntax and Keyword Highlighting

The ability to highlight keywords and syntax with color is a very powerful feature. This and the Code commenting feature is the easiest to add. All that is required to add this feature is to create a new language record by doing the following:

- From the main menu select Tools | Customize | Languages
- Select the Insert button and type in the name of the new language.

This will bring up the Language Setup dialog for the new language and the fields can then be filled in with the appropriate data. See the online help for more information about what each field should contain.

When the new language record has been defined, it must be associated with a set of file extensions. This is done as follows:

- From the main menu select Tools | Customize | Filename extension
- Select a defined extension and hit the Edit button or Select the Insert button and type a new set of file extensions
- Select the ... button by Language and Select the newly defined language from the list.

### Code Commenting and Uncommenting

This feature is automatically added as soon as the comment fields are filled out in the Language Setup dialog. See the online help manual for more information.

### Templates

The new template system that is provided with Multi-Edit allows adding abbreviations for common code fragments and language constructs that fully expand when the space key is pressed. These templates are
stored as *.TPT files under the /MULTI-EDIT/CONFIG subdirectory. A specific language template can be
defined and associated with an extension just like a language record. This is done as follows:

- From the main menu select **Tools | Customize | Filename extensions**
- Select the '...' button beside the Template field
- Select a defined template and click the Select button or select the Insert button and define a new
language template.

There can be only one main language template associated with a set of extensions, but by adding template set
names separated by semicolons in the Add-On template field, it is possible to allow additional templates to be
added to the extension. This is how the Windows API template set would be added.

Also, while in the Extension Setup dialog be sure to check the Auto template expansion option to enable
template expansion for the specified extension.

To edit or add new templates do the following:
- From the main menu select **Tools | Edit templates**

This will open the Edit Template dialog for the language specified in the Extension Setup dialog for the
currently loaded window. This is where new templates are added and older templates are modified. The
fields in this dialog allow each template to be modified with specifications as to what will be inserted and
when it will be expanded. See the online help manual for more details about all of the fields in the template
dialog.

### Construct Matching

Construct matching provides the ability to start with the cursor on a opening or closing construct, i.e. (), {},
begin end, and find (optionally highlighting) the matching construct. This feature can be implemented in two
ways, with a macro that does the searching or through the new general purpose LangDoMatch routine.

The macro method requires that a macro be written to locate and match all supported constructs. This is the
older method and will not be discussed in detail here.

The new method of adding construct matching is to define a set of global variables that define the patterns to
match and then using the general purpose LangDoMatch macro to do the matching. To implement this, the
following steps must be completed.

1. Set up the special pattern global variable.
2. Write a special _xGetMatchPat( ) macro.
3. Write an xMatch macro that calls the LangDoMatch macro passing the correct parameters depending
   upon the language Properties.
4. Set up the Init and Match macro fields in the Language Setup dialog to point to the correct macros.

To set up the special pattern global variables currently requires a xInit macro be written, where x is the
language prefix, usually the first three letters of the language support macro filename. Eventually, we hope to
move this into a dialog and save them in a DB file so that it could be more easily updated.

There are three global variables that must be set up and are shown below using C as the example. See the
CInit macro in C.S for more details.

!CMatchExtra - Characters used to specify the start and end of words

This global string variable consists of three parts starting with the \x7F character shown below as %

\%B=\textit{str} where \textit{str} is a list of all of the characters that can precede a construct word.

\%E=\textit{str} where \textit{str} is a list of all of the characters that can come after a construct word.

\%D=\textit{char} where \textit{char} is a character that is used to separate multiple construct patterns. The default is
to use the space character as the delimiter but should be changed to something else for languages such as Ada and Visual Basic that use double word constructs.

!CMatchBegPat - Beginning construct patterns to match

This global string variable consists of a series of records using '\x7F' as the delimiter character, shown below as '%', and each record can consist of up to seven fields.

%Str1%Str2... where Str# is the character string of a beginning construct, i.e. IF or (

%F= Flag where Flag is an optional bit flag used to control how the match routine will work for matching the current construct. See Language.sh for the _mfc_Xxxx flag values.

- _mfc_StrOnly: This flag when set, will cause the matching routine to exactly match the construct ignoring the characters before and after it.
- _mfc_SkipMid: This flag when set, changes the meaning of the %M= strings to cause them to be skipped when searching for a match. The %I= field should be used to specify patterns to ignore when a middle pattern is also required.
- _mfc_ContMatch: This flag when set, will cause matching to continue when the %C= expression is found after a %B= or %E= string was located. When this flag is reset, matching will end if the %C= expression is not found after a %B= or %E= string was located.

%B= Str1 Str2 ... where Str# is a %D=char delimited, before and after, list of beginning construct patterns, i.e. IF or ( 

%M= Str1 Str2 ... where Str# is a %D=char delimited, before and after, list of middle construct patterns, i.e. ELSE or blank for (matching. If the %F= flag has the _mfc_SkipMid bit set then the match routine will cause a found string matching one of the %M= strings to be skipped.

%E= Str1 Str2 ... where Str# is a %D=char delimited, before and after, list of ending construct patterns, i.e. ENDIF or ).

%X=XStr where XStr is a UNIX style regular expression that will match any of the B, M or E string, i.e. (IF)|(ELSE)|(ENDIF) or [\(\)].

%I=IStr where IStr is an optional %D=char delimited, before and after, list of construct patterns to ignore when doing a match. See VBasic.s or Ada.s for an example of how this would be used.

%C=CStr where CStr is an optional UNIX style regular expression to search for after a %B= or %E= match was found. What happens when a match is found is determined by the value of the _mfc_ContMatch bit of the %F= flag. See VBasic.s for an example of how this would be used.

% The ending delimiter is required

!CMatchEndPat - Ending construct patterns to match

This global string variable serves the same functions as the !CMatchBegPat except that it is used to specify ending patterns.

%Str1%Str2... where Str# is the character string of a ending construct, i.e. ENDIF or )

%F= Flag where Flag is an optional bit flag used to control how the match routine will work for matching the current construct. Used exactly as described above.
%B= **Str1 Str2 ...** where **Str#** is a %D=char delimited, before and after, list of ending construct patterns, i.e. ENDIF or ).

%M= **Str1 Str2 ...** where **Str#** is an optional %D=char delimited, before and after, list of middle construct patterns, i.e. ELSE or blank for (matching. If the %F= flag has the _mfc_SkipMid bit set then the match routine will cause a found string matching one of the %M= strings to be skipped.

%E= **Str1 Str2 ...** where **Str#** is a space delimited, before and after, list of ending construct patterns, i.e. IF or (.

%X= **XStr** where **XStr** is a UNIX style regular expression that will match any of the B or E string, i.e. (IF)|(ENDIF) or [\(\)].

%I= **IStr** where **IStr** is an optional %D=char delimited, before and after, list of construct patterns to ignore when doing a match. See VBasic.s or Ada.s for an example of how this would be used.

%C= **CStr** where **CStr** is an optional UNIX style regular expression to search for after a %B= or %E= match was found. What happens when a match is found is determined by the value of the _mfc_ContMatch bit of the %F= flag. See VBasic.s for an example of how this would be used.

% The ending delimiter is required

Before we go into more detail, let's explain how the matching feature works. When the match routine is run either from a key, toolbar or menu, the Match macro looks in the language record specified by the current file's extension for the macro in the Match field. If an entry is found, the specified macro is executed. This is usually a macro that calls the LangDoMatch routine with the language prefix specified /LP= and possibly some other parameters to specify if highlighting is to be done or not.

The first thing that LangDoMatch does is call a special macro for the specified language called _xGetMatchPat, x being the language prefix passed as the /LP= parameter. This macro is to provide special character processing for the specific language. This can be as simple as setting Return_Str to "" and returning (i.e. no special processing), or checking if the current character is one of the special characters and returning the character surrounded by the \x7F delimiters characters, or searching for special characters on the current line and then returning the delimited character. This macro is usually used to process single character and possible double characters (such as comment characters /*) but can also be used to reposition the cursor on another word that is supported. See Fortran.s or Ada.s for an example of the word-repositioning feature. The main LangDoMatch will then process whole words if the _xGetMatchPat returns "" or the special character cannot be matched.

If the _xGetMatchPat returns a pattern then the LangDoMatch routine searches the special MatchBegPat global variable for a beginning pattern match. If one is found, it will parse out the fields of that record. It will then use the specified regular expression to do a forward search. When a match is found, the found string is compared to the beginning, middle and end patterns of the specified record. What happens next depends upon which string the found string is found in. If it is found in the begin string a match count is incremented by one, since a nested construct is found and the search is continued. If the end string contains the found pattern then the count is decremented by one and will exit the search loop when it reaches 0, i.e. the ending construct was found. If the middle string contains the found string then the search is repeated unless the count is at 1, i.e. still inside the first construct and thus a middle construct match.

If the original pattern was not found in the MatchBegPat global variable, then the MatchEndPat global is "search" and the same process as above happens except for searching backwards for matching patterns.

If the _xGetMatchPat returns "" or the pattern cannot be matched, the word the cursor is sitting on is read using the beginning and end word string to delimit the word. The above process is then repeated with the
word instead of a character pattern. Thus if the new language is to only support words then the _xGetMatchpat can always return ""

When a matching construct is found it will be highlighted if the appropriate parameters are passed. See the LangDoMatch macro in Language.s for more details.

After the xInit, _xGetMatchPat and xMatch macros are written, the xInit macro needs to be specified in the Init macro field and the xMatch macro in the Match field of the Language Setup dialog.

**Smart Indenting**

Smart indenting is the ability to position the cursor in the correct column to continue typing code after the Enter key is pressed. Since this feature is very language dependent, a macro must be written which, when called by the CR system macro, should check the context of the cursor and reposition the cursor on the next line in the correct position.

We have developed two different types of routines that should handle most languages and they can be seen in the C.s and Pascal.s files. The C.s macro, CIndent, checks the line ending characters of previous lines to determine the indent level of the next line where the Pascal.s macro, PasIndent, checks the first word on previous lines to do likewise.

To implement the indent macro for the new language, determine which type of routine the new language is most like and start with a copy of one of the supplied macros and change it to implement the specific cases.

When you have the macro written and tested, it is installed by doing the following:

1. From the main menu select **Tools | Customize | Filename extension**
2. Select the specific extensions and hit the Edit button.
3. Set Indent style to Smart.
4. Exit dialog by hitting OK
5. From the main menu select **Tools | Customize | Languages**
6. Select the new language entry and hit the Edit button.
7. Enter the macro name in the Indent macros field, i.e. C^CIndent

---

*Select Indent style "Auto" if a language indent macro has not been written. This will cause the cursor to line up with the first word on the previous line.*

**Function Tagging**

This feature requires a macro be written to scan the source file for the function declaration and/or variable names and writes them to a tag file in a specific format. The MeTags macro must then be patched to support the new language. This will not be covered in this document since this is rather involved. We hope to eventually make this easier to add in the future.

**Properties**

The properties feature is not really a separate feature in itself, but is used to support some of the other features. The Set Properties button in the Language Setup dialog, when pressed, will run the macro specified in the Config macro field. This macro should display a dialog that presents the user with configuration options
that are supported in the new language support. Examples of this can be seen in the C and Pascal support where the indent style and auto highlighting of closing ')' can be enabled or changed.

If the new language is to support properties then a set of macros needs to be written to support them. These macros are described below using C as the example. Replace the leading C with the first three characters from the name of the new language macro file and make the needed changes for the specific properties.

```c
void CSetProperties(
    str GStr    = Parse_Str( "/GSTR=", MParm_Str ),
    str Parameter = MParm_Str
)

/************************************************************
Function: Set C and CMAC specific properties. Should only be run by the LangSetProperties macro.
Entry   : str GStr - Name of global string containing properties (/GSTR=)
       str ParmStr  - Misc parameters
       /L=str      - Language name to show on title bar
Exit    : None
*************************************************************/
```

This macro displays a dialog of all of the available properties that can be set for the specified language. This macro should save the changed properties data in the global variable specified by GStr before exiting so that the Language Setup dialog can update the db record for the specified language.

```c
int CGetProperties( struct tCProperties rCP )

/************************************************************
Function: Get the specific language properties for the current file.
Entry   : struct rCP   - A structure to fill with the properties.
Exit    : int
      True         - Properties fill from Db
      False        - Properties set to internal defaults
*************************************************************/
```

This macro is called by all the other macros when they need to query a specific property. A structure should be defined that contains entries for all of the supported properties.

```c
void CIndentTmp(
    str Set  = Parse_Str( "/S=", MParm_Str ),
    str Name = Parse_Str( "/N=", MParm_Str )
)

/************************************************************
Function: Insert the indent style template Name from the template set.
Entry   : str Set - The template set
       str Name - The template name
Exit    : None
*************************************************************/
```
This macro would only be written and used if the new language allows templates to adjust indent style based upon a property setting. When this is used the template must be set up to call this macro to determine the indent style and the name of a template that will expand to the selected indent style. See C.TPT for examples.

```c
#ifdef
#endif
```

If any user has developed support for a previously unsupported language and would like it to become a supported language, send us a copy of your language macro and support files, i.e. *TPT etc and we will try to get it added to a future version of Multi-Edit.

---

**Version Control**

### What is VCS?

VCS stands for **Version Control System**. One of the main purposes of VCS programs is to help maintain and track versions or revisions of a set of files.

Since you are probably using Multi-Edit to write programs, think about what it would take to maintain a single program after releasing three or four versions with a couple of maintenance releases along the way. Next add tracking of all the new feature requests and bug reports that you get and implement. After thinking about this, add the possibility of creating three or four other programs and since you haven't enough time to develop them yourself, plan on having five or six other programmers help in their development. How do you maintain this?

One possibility would be to keep a copy of every file of every program for every version that you produce, making sure you maintain a document that describes all of the changes that you make. This may not be too hard when it's just you doing all of the work. When you have others working with you and they also need to document their changes, it becomes more difficult. Then there is the issue of who gets what file when. As deadlines approach, documentation is often the first thing that suffers. As a result, maintaining control over who edits each file becomes increasingly difficult.

On top of all of this, there is the problem of storage space. Keeping complete copies of every file, even when it is compressed into an archive, still takes up quite a bit of space.

It is situations like the above that VCS programs were designed to handle. The first thing you should do is put all of your source files under VCS control. This is done by first creating and checking in a copy of each of your source files into a VCS archive and adding a description for each file. Once the VCS package has "control" of your source files, you can think of the VCS package as you would a public library. To get a book to read out of a library, you will need to check it out. In order to make changes to a source file, you need to check it out of the VCS library.

The VCS program tracks who has each file checked out and prevents another programmer from modifying a file already checked out. Thus, the same file is prevented from being changed by two programmers at once. After a file has been modified and tested, you will then check it back into the VCS library or archive and add a comment about what changes you made. This information is saved along with the new revision of the file. All changes are tracked and documented without much extra effort.

The way VCS programs help the storage space issue is two fold. First, all versions or revisions of a file are stored in one file or project instead of being split between different version archives. Second, most VCS programs only store the full text of only one version of the file and save a "delta" or a set of changes for each
of the other revisions. There are two methods used to implement this, forward and reverse delta. Both methods serve the same function, but implement it differently.

In the forward delta method that TLIB uses, the first version of a file is saved in the archive. Every new version is saved as a set of changes that create a newer version when applied to a previous version of the file. Thus with a TLIB archive contains 5 revisions, the original file is stored first, followed by four deltas. When these deltas are applied to the original file, a newer revision of the file is created.

The reverse delta method, as used by VCS and most of the other supported packages, does the same except in reverse. The latest version is saved and then deltas are saved that change the newer file into a previous revision.

What is SCC?

SCC is an API designed by Microsoft to allow programs to interface to their Visual SourceSafe version control program. After its release, many other companies that provide VCS programs added support to their programs to support this API.

The advantage of using the API is that Editor and IDE authors can develop a common interface that will work with a number of VCS programs. Also, since it is an API, Multi-Edit has direct access to the routines and doesn't need to run an external executable, capture the output and then parse the output when doing a VCS operation.

Why have VCS support in Multi-Edit?

All of the benefits of a VCS system are moot if the program is never used. Even in a Windows environment where multiple applications can be run simultaneously, a programmer must switch from his or her editor to the VCS program, perform a VCS action, then switch back to the editing environment. The Multi-Edit VCS support was developed in order to allow users to access their VCS system from within the Multi-Edit environment.

Early programming was mostly done from a command line. You ran a command to check a file out of its archive, another command to edit it, another to compile it, another to test. You would repeat the edit, compile and test cycle until the program ran as you wanted and then you would run a command to check it back into its archive.

Since Windows is becoming a popular programming environment, programmers are moving away from using command line and individual programs to using an Integrated Development Environment (IDE).

With the newer IDEs and programmer's editors like Multi-Edit, where the editing, compiling and testing can be done in one place, it makes sense to add the VCS stages of the cycle into this environment as well.

How Does the VCS Support Work?

Since every programmer works differently, the VCS support was designed to make it easy for programmers to work with the VCS programs. Currently there are four different interfaces provided and they are documented under the section heading “The VCS Interfaces”. By learning how to use these different interfaces you will be able to do most, if not all, of your VCS work from within Multi-Edit.

The first version of the VCS macros was designed to work with one VCS program. It allowed the end user to configure it to do most of the simple VCS operations. As new features and new VCS packages were added, the code went through some major restructuring, but was successfully integrated into one set of macros. With the previous release of the VCS macros in Multi-Edit for DOS version 7.0, the VCS support macros were
getting quite complicated. With this release, these macros were redesigned to allow a more flexible and powerful way of implementing and expanding the VCS support.

The new design, while still maintaining the capabilities of the previous versions, has been expanded from one macro file into a number of macro files. Instead of the "VCS.MAC" file containing all the VCS support code, it now only contains the high level API code and some common support code. The VCS package specific driver code has been moved into separate macro files, one for each of the supported VCS packages. This change to the code has made VCS integration cleaner and easier to test and update. It is now possible to add features that are specific to one VCS package without the possibility of "breaking" support for another package.

The VCS macros were designed in a layered structure, much like an operating system, with APIs and drivers. The VCS support's high level APIs, designed to provide a common interface to all of the specific VCS programs, consist of a set of macros called by the file open and close system macros, menus and toolbar commands. No matter which of these macros is run, the "VCSCmds" macro is eventually called. This macro determines the currently selected VCS package and causes the appropriate lower level driver macro, located in the one of the VCS specific macro files, to be run.

The low-level driver macro will convert the desired API command to a specific VCS command line, execute it and check to make sure there were no errors. This information is then passed back to the "VCSCmds" macro, which will display an error message or manage the file loading and closing, depending upon the command.

Once the VCS support is configured and set up properly, doing VCS operations is almost as easy as loading and closing files.

### Configuring the VCS System

The VCS menu, accelerator keys, and toolbar buttons can be fully customized to your liking.

Before the VCS system is used the first time, it must be configured for your working environment. To accomplish this there are two main configuration dialogs, VCS Configuration and VCS Package Setup.

The first dialog, VCS Configuration, can be started either from the Customize dialog in the Tools Menu or from the VCS Menu. It is used to set up the common configuration items, which determine the operating features of the VCS support. These options have the same effect regardless of the VCS package you use. There are options to enable the VCS support, which VCS package to use by default, the kinds of information to be saved and many more.

The second dialog, VCS Package Setup, enables you to set up items specific to the VCS program that you are using. This dialog box is started from the main Configure Dialog. It allows you to specify the archive path and extension, user I.D. and other VCS specific options. In addition, the command lines for each of the VCS functions can be changed here. However, the default commands should suffice for most users.

If you will be using the <LOOKUP> archive path alias, you will need to set up the <LOOKUP> database.

The last item related to configuring the VCS support is Setup Verification. Though rarely used, this is provided to help diagnose set up problems with the VCS support. When run, checks on the VCS system are executed and the results are shown in a list window dialog.
Using Associate Directories

This dialog gives a list of file directories with their associated archive directories. Adding or changing these association is done by selecting the Insert or Edit buttons. This brings up the Associate Directories Setup dialog to appear where the desired directories can be entered or edited.

Hitting Alt+I or the Insert key, or by selecting the Insert button, will cause the Create New Record dialog to appear asking for a New File Directory. Enter the full path name of a working directory here and hit ENTER or select OK to create a new entry into the associate directory database and bring up the Associate Directories Setup dialog.

The Associate Directories Setup dialog is where the file directory and the archive/lock directories are entered. This information is then stored in the VCS.DB file and is later used when checking for archive and lock files.

This information is only used when the <LOOKUP> alias is defined in the archive path option in the VCS Package Setup dialog or when the Work Dir option in the main VCS Configure dialog is set to Projects.

Files...
This field is used to enter the directory where your files are located or being edited. The VCS macros, when trying to find an archive for a specified file, will use the path of the specified file and search the "Associate Directories" database for an exact match to this field. When a match is found, the "Archive" entry would be returned. There can be multiple entries with the same path, thus allowing a single working directory to be associated with multiple archive directories. If multiple entries are found then the combined archive paths are returned with a ";" between each entry.

Archive
This field is used to enter the directory where the archives for the files in the directory specified in the above "Files" field are located. There can be a single path or a series of paths separated by ";". The preferred method is to use multiple records with a single path per field.

Work...
This field is used to enter the work directory where the extracted files will be edited. This field usually contains the same directory as the "Files" field and can be left blank. The main reason for entering a path here is to specify a directory to extract a file to for editing that is different from the directory the loaded read-only in located. Good for loading a "reference" file from a directory on a network and checking out the file to a local directory to change it.

Project
This field is used to enter the base project directory for a "named project". Since named project support does not exist yet, this field should be left blank.

LOK... (TLIB Only)
This field is used to enter the directory where TLIB Lock files are to be stored.
Base... (LCM Only)
This field is used to enter the LCM base directory for the project located in the "Files" directory. This would be the same as the base= entry in the LCMPROJ.CFG.

This information is only used when the <LOOKUP> alias is defined in the archive path option in the VCS Package Setup dialog or when the Work Dir option in the main VCS Configure dialog is Support.

Understanding Setup Verification

When Setup Verification is run from the VCS menu, a window is displayed that shows the results of a series of tests that check the VCS support setup dialog. By understanding the information shown in this window you will be able to troubleshoot problems that occur when setting up the VCS support.

The VCS support consists of a main interface macro file, "VCS.MAC", and a number of specific VCS support macro files. Since these support macros depend upon a consistent interface between them and the main macro, a form of version checking was implemented. Every VCS macro file has a version number and revision information embedded into them, which gets checked every time the VCS support is initialized.

The version number consists of two parts: a Multi-Edit version number an update letter, i.e. 9.10.03. The Multi-Edit version number, which is not checked, is the version/revision of Multi-Edit that is required for the VCS support macros to be fully functional. It is the VCS release number of the main interface macro "VCS.MAC" and the selected VCS package support macro that is checked during initialization.

The release number is changed whenever a major release of the VCS macros is released, i.e. a change to some interface where all macros needed updating, and this number must match exactly for the VCS support to be enabled. The update letter is changed when a minor update is released, i.e. changes to one or more of the support macros or main macro that doesn't affect another macro. The update letter does not need to match and is provided for tracking purposes only.

The top three lines in this window form a header that presents the information in columns or fields.

- **Macro:** The file name of a specific VCS support macro.
- **Mac Chk:** This field will have an "x" shown when the specific VCS macro has passed all checks and the support for that package would successfully be installed.
- **Ver Chk:** This field will have an "x" shown when the specific VCS macro version number matches the main VCS macro version number.
- **EXE Chk:** This field will have an "x" shown when the "GET" command executable for the specific VCS package is found.
- **Version:** This field contains the version number of the specific VCS macro.
- **Revision Info:** This field will show the internal revision string of the specific VCS macro.
- **Executable Path:** This field will show the path and filename of the specific VCS package "GET" command.

The fourth line shows the above information for the main VCS interface macro "VCS.MAC". The number of lines that follow depend upon the number of supported VCS packages. There will be three lines for each support macro, a blank line followed by the Macro to Revision Info on the next line and the EXE Chk and Executable Path field on the third line.
Using the VCS Interfaces

The VCS support actually contains four different interfaces to the selected third-party VCS programs. This was done to provide convenience and to help increase your productivity. With these interfaces most if not all of your VCS work can be done without exiting Multi-Edit.

The first interface is essentially the same interface used to open and close files. When the VCS support is enabled, the file open and close functions have the ability to check files out from VCS archives when loading non-existing files and to check files back into VCS archives when closing them or upon exiting Multi-Edit. For more detail on how these functions work, refer to Using The File Open And Close Interface.

The other three interfaces are all accessed through the VCS menu or toolbar buttons. Probably the most used interface would be Using The Current File Menu/Toolbar Interface. This allows you to do most of your VCS operations using the file that is loaded into the current window.

Using the "Directory Of Archives Dialog" and "Using Multi-Edit Files From Archives Dialog" dialogs allow you to pick and choose the files that you want the VCS operations to run on. These are very powerful ways to check files and functions in and out and are very similar to the file manager that comes with Windows.

The last interface involves selecting the "Run VCS Interactively" command from the VCS menu. This starts the "User Interface" that comes with your VCS program without exiting Multi-Edit. This would be used for features that you require that are not currently fully supported in the VCS support macros.

Using The File Open And Close Interface

This interface provides seamless integration of checking files into and out of VCS archives by "hooking" into the standard open and close file functions. Every time the VCS support is started, three global variables are initialized. These variables are set to contain macro command lines and are used by some of the Multi-Edit system macros to allow "hooking" or adding functionality.

The first variable, "@FNF_LOAD_MACRO@" is set to contain the string "VCSGetFile" which is a macro located in the main VCS.S file. When this variable is set and you try to load a file that isn't found, Multi-Edit will try to execute the string that is contained in this variable as a macro and pass the filename as a parameter. Since the VCS support has set this variable to point to the "VCSGetFile" macro, this macro will be executed. The "VCSGetFile" macro first checks to see if there exists a VCS archive for the file being loaded. If there isn't an archive for that file, then a return value of "False" is returned and Multi-Edit will assume you are loading a new file.

On the other hand, if there is an archive for this file, then what happens next will depend upon how the "Get Mode" option is configured (see "Get Mode" under Using the Configure Dialog). If the default setting of "PROMPT" is still active, a dialog box, shown below, will appear and prompt you for an action to take. If one of the other options was selected, then that action will be taken without a prompt being shown.

- **To Modify**: Selecting Enter or Alt+M or selecting this button will cause the specified file to be extracted by the VCS software for modification.
- **To View**: Selecting Alt+V or selecting this button will cause the specified file to be extracted by the VCS software for viewing only (read-only).
- **To Lock**: Selecting Alt+L or selecting this button will cause the specified file to be locked by the VCS software. The file will not be extracted, only locked, to prevent another person from checking it out to modify.
• **Cancel**: Selecting Esc or selecting this button will return to the editor without extracting a file. If this
dialog was caused by loading a new file (i.e., macro LOADFILE), then a new file window will be
created.

The other two global variables, "@DEL_VERIFY_MACRO@" and "@ME_EXIT_MACRO@", are used
to tell Multi-Edit what macros to call when closing a file and upon exiting. Both of these variables actually
point to the same VCS macro, "VCSCleanup", but produce different results depending upon which function it
was called from.

When the "VCSCleanup" macro is called from the close file function, it first checks the VCS checkout status
that is maintained by the VCS support to see if the file being closed was previously checked out of a VCS
archive. If the file does have a checkout history, then one of two dialogs will be shown prompting you to
select an action to take.

The first dialog that could appear is displayed when the file has been checked out of an archive for
modification. The dialog states that the file is checked out and asks if you want to check the file into its
archive.

  • **Yes**: Selecting Enter, Alt+Y or selecting this button will cause the current file to be checked into its
    VCS archive.
  • **No**: Selecting Alt+N or selecting this button will cause the window to be deleted, but the file will not be
    checked into its archive.
  • **Cancel**: Selecting Esc or selecting this button will abort the "close file" function and will cause the file
to remain checked out and load.
  • **Unlock/Abandon**: Pressing Alt+U or selecting this button will unlock the current file and delete it from
disk if the "Delete Source" option is checked in "Using The VCS Package Setup Dialog".

The other dialog that could appear is displayed when the file has been checked out of an archive for viewing
and has been loaded as read-only. It states that the file is checked out and asks if you want to delete the file.

> This dialog will only appear when a file has been checked out of a VCS archive for viewing (read-only)
> and the "Browse Saved" option is checked in Using the VCS Configure Dialog.

  • **Yes**: Selecting Enter, Alt+Y or selecting this button will cause the file to be closed as well as deleted
    from the disk.
  • **No**: Selecting Alt+N or selecting this button will cause the window to be closed but the file will not be
    deleted.
  • **Cancel**: Selecting Esc or selecting this button will abort the "close file" function.

When the "VCSCleanup" macro is called just before Multi-Edit shuts down, it also checks the VCS checkout
status to see if any files have been checked out of a VCS archive. If a file has been checked out of a VCS
archive then a dialog is shown that lists all of the files checked out. This gives you a chance to check them in
before exiting or leaving them checked out. This is the same dialog that is shown and documented under
"Using Multi-Edit Files From Archives Dialog".

> This dialog will only be shown on Exit if the "Exit Status" option is checked in Using The VCS Configure
> Dialog.
**Difference List**

The information shown in the Difference list is the output of either the VCMPR.EXE program or the VCS DIFF program specified under the DIFF command in the VCS Package Setup Dialog for the selected VCS program.

**Errors**

When the VCS macro finds an error condition after running a VCS program, a dialog will pop up that shows the results of the failing program. All VCS specific errors start at 9000 and are just the error number returned by the failing VCS program + 9000.

An error of 9100 with a blank list is an indicator that the VCS program did not actually run or that it did not generate any output.

**Supported VCS Aliases (or Metacommands)**

The supported Aliases (or Metacommands) are listed below and continued upon the next page.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;LOOKUP&gt;</td>
<td>Substitutes the archive path looked up in the associate directory database.</td>
</tr>
<tr>
<td>&lt;LPATHX&gt;</td>
<td>Substitutes the derived library path minus an ending &quot;.&quot;.</td>
</tr>
<tr>
<td>&lt;LPATH&gt;</td>
<td>Substitutes the derived library path.</td>
</tr>
<tr>
<td>&lt;OPATH&gt;</td>
<td>Substitutes the current window's path minus drive letter.</td>
</tr>
<tr>
<td>&lt;COMNT&gt;</td>
<td>Substitutes the filename of the file containing the VCS comment.</td>
</tr>
<tr>
<td>&lt;RCSDIR&gt;</td>
<td>Substitutes the value of the DOS environment variable &quot;RCSDIR&quot;.</td>
</tr>
<tr>
<td>&lt;SAPCMT&gt;</td>
<td>Substitutes the Sourcerer's Apprentice command &quot;-c comment&quot;.</td>
</tr>
<tr>
<td>&lt;%TMP&gt;</td>
<td>Substitutes the value of DOS environment variable &quot;TMP&quot; with an ending &quot;.&quot;.</td>
</tr>
<tr>
<td>&lt;TMP_PATH&gt;</td>
<td>Substitutes the value of Multi-Edit temporary file path variable &quot;@TMP_FILE_PATH&quot;.</td>
</tr>
<tr>
<td>&lt;TMP_PATHX&gt;</td>
<td>Substitutes the value of Multi-Edit temporary file path variable &quot;@TMP_FILE_PATH&quot; minus an ending &quot;.&quot;.</td>
</tr>
<tr>
<td>&lt;FILE&gt;</td>
<td>Substitutes the current window's file name minus extension.</td>
</tr>
<tr>
<td>&lt;EXT&gt;</td>
<td>Substitutes the current window's file extension.</td>
</tr>
<tr>
<td>&lt;NAME&gt;</td>
<td>Substitutes the current window's file name minus extension path.</td>
</tr>
</tbody>
</table>
### Multi-Edit 9.10 Working in Multi-Edit

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;LOOKUP&gt;</td>
<td>Substitutes the archive path looked up in the associate directory database.</td>
</tr>
<tr>
<td>&lt;PATH&gt;</td>
<td>Substitutes the current window's path.</td>
</tr>
<tr>
<td>&lt;ME_PATH&gt;</td>
<td>Substitutes the path where Multi-Edit resides.</td>
</tr>
<tr>
<td>&lt;COMSPEC&gt;</td>
<td>Substitutes the current command processor (command.com).</td>
</tr>
<tr>
<td>&lt;USER_PATH&gt;</td>
<td>Substitutes the path where the user specific configuration files reside.</td>
</tr>
<tr>
<td></td>
<td>Only applicable to the Network version.</td>
</tr>
<tr>
<td>&lt;USER_ID&gt;</td>
<td>Substitutes the current User ID.</td>
</tr>
<tr>
<td></td>
<td>Only applicable to the Network version.</td>
</tr>
<tr>
<td>&lt;%str&gt;</td>
<td>Substitutes the value of the DOS environment variable &quot;str&quot;.</td>
</tr>
<tr>
<td>&lt;-str&gt;</td>
<td>Substitutes the value of the global string &quot;str&quot;.</td>
</tr>
</tbody>
</table>

---

### Supported Version Control Systems

**Burton Systems Software - TLIB**

TLIB by Burton System Software's stands for Text LIBrarian. This package is currently available in text based DOS and OS/2 versions. The TLIB support macros were originally developed and tested using TLIB version 4.12g but all of the newest development and testing has been done using the latest version 5.01m. See the TLIB manual for configuration information.

**CVS**

CVS support uses the WinCVS 1.2 command line programs to access the source archives. It will work with local archives or archives on the Internet.

**TLIB 5.x DLL support**

Support for running the TLIB 5.x VCS program via their DLL has been added. It is enabled by selecting the TLIBD entry in the VCS Packages dialog. It is configured the same as if you were using the command line version of TLIB.

**MERANT – PVCS**

PVCS is MERANT Version Control Software that allows the user to keep a history of changes made to text files such as source code and help files. This program has been around the longest and is available for many different platforms. The PVCS support macros should work with any Personal, Corporate, or Network version of PVCS version 2.1c or later. The original development and testing was done using a copy of the Personal Version of PVCS version 2.1c but current development and testing is being done using a copy of PVCS version 5.1. There is an API for PVCS but the VCS support macros are not using this. Instead, they are using the dual mode DOS command line programs.

**Mortice Kern Software (MKS) – RCS**

RCS is Mortice Kern, Inc.’s Revision Control System that allows the user to keep a history of changes made to text files such as source code and help files. The RCS support macros where originally written and tested using version 4.3a but current development and testing is being done using version 6.2. Some DOS versions of the GNU RCS programs also work with the existing RCS support macros.
**Sourcerer's Apprentice – SAP**
Sourcerer's Apprentice originally sold by Solution Software and now owned by Borland International, is a Version Control Systems package that allows the user to keep a history of changes made to a text file such as source code and help files. The SAP support was originally written and tested using version 1.02. There has been a newer release but we haven't ordered a copy. Since Borland is not supporting this program it is very unlikely that we will support the updated version.

**SourceSafe – SSAFE**
SourceSafe, originally developed by One Tree Software but now owned by Microsoft, is a project oriented version control software that allows the user to keep a history of changes made to a text file such as source code and help files. The original support for SSAFE was written and tested using version 2.00. The current development and testing is being done using version 4.0 of Visual SourceSafe. The current support macros are using the new SSWCL.EXE and SSWLOGIN.EXE programs that come with the 3.0 and later versions of the Windows version of SourceSafe and are not compatible with older versions of SourceSafe. Older versions can be used but there needs to be some code changes made and the commands will need to be modified to do this.

**Subversion – Tigris**
Subversion by Tigris is meant to be a better CVS, so it has most of CVS's features. Generally, Subversion's interface to a particular feature is similar to CVS's, except where there's a compelling reason to do otherwise.

**PAN/LCM – LCM**
PAN/LCM by Computer Associates is version control software that allows the user to keep a history of changes made to a text file such as source code and help files. The original support for LCM was written and tested using version 3.3.

### Using Projects

#### What are Projects
Projects are the organizational unit that groups files together for processing. A project consists of a tree list of files cornered around a base directory. Once files are added, there are many functions that can be performed on the list of files.

Files can be easily accessed from the Project View. A file list can be generated for use with the compiler. Tags can be generated for all files in the project.

Each project is a file with the extension (.mep), this file stores information about the files and where they are located and their properties. Each file in the project can have properties such as, don't include in file list, etc.

FTP support is directly tied to projects, and at the moment can only work as part of one.

You can have several sessions associated with a project, including, if you wish, a privileged one with the same name as the project, which will load when the project does.

A project is a collection of files and some configuration settings, while a session is a snapshot of the current editor state.

You can tag by project, instead of file-by-file or wildcard.

#### Creating Projects
Projects can be created by selecting **Project** | **Create Project** and filling out the dialog. Name of the project will be the name of the project file with a (.mep) extension.
Adding files to a project are fixed to those directories. If you move the file, you must delete and re-add the file to the list.

Using the Project View

Buttons

- Create Project: This in project operations
- Open Project: This is in project operations
- Project Properties: This is in project operations
- Project Notebook: This is in project operations

Project Name

- Name of Current project: Right Clicking on name will bring up a context sensitive menu.

Menu

- Open all files in Project: This does just what it says so be careful when executing this on large files.
- Add files: Brings up a dialog where files can be marked to be added to the project
- Properties: Brings up the project properties dialog.

Project Folder Tree

The Project folder tree represents only the folders and sub folders for your current project. The Root folder represents all files within the root directory of your project. Navigation uses standard windows tree navigation. Selecting a folder will show its corresponding files in the Project File List. You may also right click on a folder to perform context sensitive actions.

Right click menu – Standard folder

- Open files in folder: This will open all files in the folder, note this does not open files in subfolders.
- Send files in folder: Send files via FTP to the remote site. This sends files in all subfolders.
- Receive files in folder: Receive files via FTP from the remote site. This does receive files in subfolders.
- Add files: Opens files selections prompt for selection of files to add to project. Directory will be positioned at the current folder.
- Remove Folder: Remove the folder and all sub-folders and files from the project.
- Properties: Edit the folder properties.
Project File List

Lists all files in the currently selected folder. Navigation is via keyboard and mouse and multiple entries are selectable. In addition, the list supports a right click context menu. You may also navigate from the keyboard or the folder list while focus is in the file list by using the Ctrl+Arrow keys.

File List Right Click Menu

- **Open files**: Open the files selected
- **Send files**: Send by ftp the files selected
- **Receive files**: Receive the selected files by ftp
- **Add files**: Add files to the project
- **Remove files**: Remove the selected files from the project

FTP

Setting up FTP Support

Files can be easily transferred to and from a server using the built-in FTP functions in Multi-Edit. FTP only works in combination with the Project Manager. Once you have configured a Project with the files you will be editing, select Project | View to view the Project Tools Pane. From here, select the Project | Properties and choose the FTP tab.

Also set up Project | Properties | Directories and Project | Properties | WebLair.

Associate Directories

Associate Directories are used to map directories that are external, i.e. outside of the main project tree. A local directory can then be mapped to a directory on a remote ftp site. For example, if we have a project with the following set up:

```
Root
  htmfiles
  images
External
  d:\cgifiles
```

If our local root directory id mapped to C:\website, and our remote root directory is mapped to usr/local, then files sent from the cgifiles directory do not know where to go when received or sent. So we would set up a local directory of `d:\cgifiles` and a remote directory of `usr/bin` and now all files sent/received from the cgifiles directory will go to the `usr/bin` directory on the remote system.
Sending / Receiving Files

The project manager controls sending and receiving of files in Multi-Edit. Every project may have an ftp site associated with it. This is set up in the Project Options. Once set up, any local file may be individually sent or sent in groups by selecting these files from the project list and then accessing the Right -Click context menu to send or receive those files. Since the project manager controls all access to the web site, currently, all files must first be located locally on the system. When marking files, if a sub-folder is selected, no files in that sub-folder will be processed.

In addition, all files in a folder can be sent/received by right clicking on the folder and selecting send/receive the folder. Note: this will process all files in any sub-folders in the tree. If it is desired to not send a particular file when multiple files are sent, you can mark a file as "Exclude from Group FTP operations", which will exclude it unless it is the only file being processed.

Sessions

What are Sessions?

Sessions are used to restore a previous state within Multi-Edit. This includes restoring opened files, the Navigation or Tools Panes, search history, window location, cursor location (per window), and the last opened project.

A Session is saved based on the "Restore Status Method" option located on the Tools | Customize | Sessions dialog. To restore a session, use the command line parameter /SR when starting Multi-Edit.

Multiple Sessions: Multi-Edit also provides the ability to have Multiple Sessions. This is useful for having different states for a single project or perhaps different states for each language used. Multiple Sessions can be created and configured using the Session Manager but prior to using the Session Manager, the "Encoded status files for each dir" option must be enabled. This allows Multi-Edit to save session information in a .mew file and allows multiple sessions to be created and maintained. Once this is enabled, the Session Manager can be displayed by clicking on File | Session Manager.

Creating a new Session: To create a new session, click on the "create" button within the Session Manager dialog. Specify the name and working directory for this new session by filling in the "Name" and "Directory" text fields and then click "Accept". Clicking on the "Select" button will then start this session.

Selecting a Session: To change to a previously saved Session, open the Session Manager, select the Session you wish to use and then click on the "Select" button.
Protecting Sessions
Multi-Edit's Sessions Manager provides a "Protect" option, which will prevent the state of the selected Session from being updated. For example, if a session with two files open is marked as protected, then restoring this Session will open only the two files even if other files were opened prior to exiting this session. To protect a Session, open the Session Manager and select the session to protect, and then click on the "Protect" button.

Session Manager
Session Manager allows you to organize different projects you may be working on into sessions. Multi-Edit saves the complete editor status for each session. This includes all the files loaded in that session, history lists, and non-temporary global variables. Each session has a name and a working directory. You may have multiple sessions in a single directory, provided they have different session names.

Descriptive names for each environment, such as "Database Project", store all editor environments related to that editing session. Multiple projects can each have different layouts and settings. This allows you to configure work environments for each project or session. The session list can be sorted and sessions can be protected so that environment settings will not change. For example, Session X has two files open. If it is protected and another file is opened, the next time the session is started only the two original files would be opened. If the session were not protected, all three files would be opened. Encoded status files are files stored with a Multi-Edit extension.

The Session Manager uses an encoded status file restore method. In order for the Session Manager to be activated, "Encoded status files for each dir" option must be selected under Session Manager Setup.

Command line options are available to allow you to start a named session, start the last session (which is the default), or to bring up the Session Manager on startup to allow you to pick from the list of sessions.

WebLair – HTML Editing

Internet Programming Support
Internet programming language support allows for editing of HTML and embedded languages. This also includes standard support for languages like Java and Perl.

When editing an HTML file, Internet programming commands are accessible from the toolbars displayed below the Editing Window or by right clicking on an Editing Window to pop-up the Context Menu. As with others in Multi-Edit, these toolbars and menus are fully configurable via Tools | Customize. The Context Menu includes options for editing tags, obtaining Help on the selected tag, selection of new HTML tags, access to HTML Tools and FTP transfer of files.

Embedded Language Support
In addition to standard HTML editing, Multi-Edit 9.10 offers full support for embedded scripts in an HTML file. This includes JavaScript, VBScript and ASP scripting syntax. Multi-Edit will automatically detect the
language change and provide syntax highlighting, language templates, construct matching, and any other language-sensitive operations (excluding Tag support).

**What are Markup Language Tag Databases?**

Tags for markup languages such as HTML and ColdFusion are contained in databases located under Tools | Customize | WebLair. When enabled, these databases are referenced from the right-click context menu for editing the tag currently under the cursor.

For example, if the HTML4 database is enabled and an HTML file containing a `<TABLE>` tag is currently open, right clicking on the `<TABLE>` tag and selecting "Edit `<TABLE>` tag..." would produce a dialog with all of the table tags style, attributes, and event attributes. This dialog is dynamically created based on the setting for the table tag within the HTML4 database.

**Adding an ML Database**

To create a new markup language database, click on the add button, enter a name for the new database then click ok.

**Adding Tags to a Database**

Once a database has been added, tags can then be inserted into the database by selecting the database in the Tag Database list and clicking the edit button. This will display a detailed dialog containing all of the modifiable attributes and settings for the selected tag. Once all of the desirable fields are completed, click on ok.

**Modifying an ML Database**

A markup language database can be modified by selecting it in the tag database list and then clicking on the edit button. This will display a dialog containing a list of all the tags contained in the selected database. Selecting any one of these tags and clicking on the edit button will display a detailed dialog containing all of the modifiable attributes and settings for the selected tag. After making the desired modifications, click the ok button.

**Enabling an ML Database**

Until a markup language database is enabled, it will NOT be referenced by the right-click context menu. To enable a database, select it in the Tag Database list and click the 'Use' button.

**Deleting an ML Database**

To permanently delete a markup language database, select the database from the Tag Database list and click on the delete button.

**Using the HTML Formatter**

The HTML formatter is executed by right clicking within a markup language file and selecting Tools | HTML Formatter from the context menu. The formatter will attempt to format the markup language code based on the following settings.

**Indenting**

- **Indenting:** Properly indents tags when enabled.

**Right Margin Wrap**

- **Wrap enabled:** Allows wrapping of text.
- **Break tag:** Allows entering an end of line character to successfully wrap text.
- **Capitalization:** Sets the case of the tags/attributes based on the following options.
Capitalization

- **Ignore**: Leaves the tag/attribute as it is.
- **Lowercase**: Ensures that the tag/attribute is lowercase.
- **Uppercase**: Ensures that the tag/attribute is uppercase.
- **Propercase**: Init-capitalizes the tag/attribute. (<Table> for example).

*The capitalization settings are also set in the HTML formatting dialog.*

---

Common Code Manager

When combined with the Project Manager, Internet programming language support includes many features for managing an entire web site. One of the most useful timesaving features is the Common Code Manager, which allows you to duplicate a piece of HTML code across an entire Project list.

To use the Common Code Manager, mark a block of code that you will want to use in a number of places and select Manage Common Code from the toolbar or **HTML Tools | Common code manager** from the Context Menu. Select Add and provide a name for the block of code. This will create a marker in the HTML code and create a separate file to contain this code for later duplication and editing.

Once a common code block has been created, use the Common Code Manager to insert that block of text into each file in which you wish it to appear. This offers a great advantage over cutting and pasting or searching and replacing. When changing the common code, you can run the Common Code Manager again, edit the common code block, and update it across all files in the Project containing that common code block. This saves a significant amount of time when updating common footers and headers on your web site.

---

Editing HTML Tags

When editing an HTML file, Multi-Edit displays the HTML text with syntax-highlighted tags. SCRIPT tags are recognized automatically, showing the correct script (i.e., JavaScript or VBScript). Tags for ASP and PHP are recognized even when embedded in a HTML tag. Each HTML tag can be edited individually by right clicking on the tag and selecting **Edit <xxx> Tag** (where xxx is the type of tag) from the pop-up Context Menu.

---

Configuring a Browser

Files can be viewed in WYSIWYG mode by launching your Browser from within Multi-Edit. Up to four browsers can be configured and updated at once. Configure browsers by going to **Tools | Customize | WebLair | Browser Manager** or select **HTML Tools | Browser Manager** from the Context Menu in an Editing Window.

Browsers can be set to update automatically when switching to the browser or switching HTML files. To update browsers manually, select **HTML Tools | Update Browser** from the right-click menu or Update Web Browser from the toolbar.
Automating Tasks Using Macros

Recorded Macros

See the section titled Recording Keystroke Macros for details on recording a keystroke macro.

Selecting Macro | Recorded Macros will open the Recorded Macro Manager which is a powerful dialog box that allows you to launch a keystroke macro that you previously recorded, create a keystroke macro, edit one, delete one, or copy one. You can even use this command to run a keystroke macro without first assigning any key to it (this feature is very helpful if you forget what key is assigned to a keystroke macro or if you run out of key assignments).

While you may use the Create command button to create a keystroke macro from scratch, you will usually want to create a keystroke macro by recording. See the section titled Recording Keystroke Macros for details.

When you activate the Keystroke Macro Manager, a standard Multi-Edit list box will appear with a list of your currently recorded macros. Both permanent and temporary macros recorded during this session will be displayed. To play a macro, select it from the list and press the Playback button.

Press the Edit or Insert command button to display the Editing Recorded Macro dialog box:

**Description**
Enter a description of a Keystroke Macro's function so you'll remember what it does.

**Key Assignment**
Shows you the current key assignment for the macro, and allows you to change that key assignment. To change the key assignment, select the button to the right of the field. Another dialog box will ask you to "Press Key to Assign." At this point, assign the macro by pressing the desired key or key combination.
Multi-Edit has a means to resolve conflicts that arise from giving a key assignment to a keystroke macro that's already being used. If this should happen, you'll see a series of dialog boxes that will help you correct the conflict.

![Edit Keystrokes dialog box]

**Edit Keystrokes**

This dialog box allows you to alter the keystrokes that make up a keystroke macro. Multi-Edit monitors both key pressed and key released states, so your keystroke macro listings will reflect this. For example, entering an exclamation point in your keystroke macro (Shift 1) will appear as three separate lines in the edit keystrokes dialog:

- Key down  Shift
- Key down  Shift + 1
- Key up    Shift

You need to keep this in mind when you edit the keystrokes in your macros.

**Record**

**Keystroke Recording** allows you to record a series of key-presses that will accomplish a specific task. The recorded keystrokes are then saved as a keystroke macro, which can be "played back" with a single key assignment.

As of Version 9, you can now control Caps, Numlock, and Insert for each keystroke. In addition, there is more control over the timing of the keystrokes. Support for repeated keystrokes is not included, so if you hold the cursor movement keys down while recording a macro, this will not play back correctly.

**An Example Keystroke Macro**

**Keystroke macros** are designed to make your work easier and faster by allowing you to cut down on repetitive actions. Let's consider the following example:

Suppose you write letters often and every time you begin one you enter the same basic information such as your company name, company address and a salutation. One way to save time is to create a file that contains this header information and "splice" it into each new letter you create. To do this you would create a file containing the header information, save it to disk, and then use the Merge file from disk option from the File Menu. This makes things a bit faster, but creating a keystroke macro would allow you to complete the whole process in one keystroke.

To begin keystroke recording, press the Record key **Alt+F10** or **Macro | Record**. You may also click on the REC portion of the status bar. Notice the REC on the status bar appears red. This alerts you that all keystrokes will be recorded until you press the Record key again to stop. The following keystrokes would be required to accomplish the file splicing:

- `<ESC>` brings up the Main Menu
- `<F>` selects the File Menu
- `<M>` selects Merge
- `<header>` type in the name of file to load
- `<ENTER>`
To end keystroke recording, press the Record key again. Once recording is terminated, a dialog box will pop up asking you whether this keystroke macro is temporary (Current session only) or permanent (Save permanent). If you select temporary, you will simply be prompted for a keystroke to assign to that macro. A temporary macro is only active during the current session and is NOT saved when you exit Multi-Edit. If you select permanent, the Keystroke Macro Manager is automatically invoked and the Editing Recorded Macro dialog box will appear on screen.

**Repeat**

**Edit | Repeat** or **Ctrl+R** allows you to repeat a single keystroke (whether assigned to a command or not) many times in succession. Here is how the Repeat command works:

- Select Repeat. A message appears on the Status/Message Line prompting you to enter a number.
- Type the number of times you want the keystroke to repeat.
- Type the desired keystroke. The keystroke will be performed the number of times you have indicated.

If you type 0 (zero) for the number of times you would like the keystroke to repeat, Multi-Edit will repeat that keystroke until the cursor reaches the End of File marker. You should ensure that the keystroke you enter would cause the cursor to reach the End of File marker. If, by mistake, the Repeat command becomes stuck in an infinite loop (will never reach the end of the file), use **Ctrl+Break** to terminate the macro.

**A Brief intro to CMac**

Have you encountered a situation where a regular expression search and replace just won't cut it, or user input is required, or perhaps keystroke macros are just not offering the complexity you need? Well, then it's time to look at Multi-Edit's CMac language.

CMac is a powerful C like structured language that offers local and global variables, preprocessing directives, screen, keyboard, mouse and hardware access, DLL import capabilities and much more. You can see it's power and flexibility by using Multi-Edit, as the majority of Multi-Edit's features and functionality were created using the CMac language.

For an in depth view into the CMac language, please refer to the online CMac reference guide which is provided with Multi-Edit and can be viewed by selecting **Help | CMac Language** from the main menu. The reference guide may alternatively be displayed by using the Ctrl+F1 key command while editing a CMac file (.s or .sh extension).

> If the cursor is under a CMac function when the Ctrl+F1 key command is used, the reference guide will automatically display help on that function.

You may also want to periodically visit our website (http://www.multieditsoftware.com) for the latest CMac information and tutorials.
Other Useful Tools

Spell Check

The Spell Checker allows you to respond immediately to misspelled words as they’re found. The spell checker is designed for memos and letters, as well as for spell checking string constants in your source code.

When Spell Check is invoked, a Spelling dialog box appears with a number of spell check options. Select the desired options and press OK to begin checking.

From

Depending upon which option button you have selected, the spell checker will start checking for misspelled words at your cursor position (Cursor) or the top of the file (Top) and continue to the end of the file.

Options

- **Selected text only**: Restricts the spell check to a marked block.
- **Text in Quotes**: Include - Include Quoted text in spell check. Ignore - Ignore Quoted text in spell check. Only - Only check text in Quotes.
- **Text in Comments**: Include - Include Commented text in spell check. Ignore - Ignore Commented text in spell check. Only - Only check text in Comments.
- **Lookup word under cursor**: Checks the spelling of the word the cursor is on.

Setup…: Allows you to set the defaults for the spell checker. See the dialog image below:

The following spell checking options occur during the spell check operation.

- **Delete**: When a double word (like "Fire! Fire!") is encountered, a Delete button replaces the Change button in the above dialog box. Press the Delete button to delete the second occurrence of the word.
- **Ignore**: Ignore this word and go on to the next word.
- **Ignore All**: Skips over this word for the remainder of the spell check.

- **Change**: Change to word listed in the Change To text box. You may change what appears in the Change To box either manually or by selecting one of the suggested spellings underneath it.
- **Change All**: Change this and all subsequent occurrences of this word to the word listed in the Change To text box.
- **Add to Aux1**: Allows you to enter the word in the first auxiliary dictionary. Once a word is added to this dictionary, it will no longer be considered a misspelled word.
• **Add to Aux2:** Allows you to enter the word in the second auxiliary dictionary and have the spell checker no longer consider it a misspelling.

Difference between Aux1 and Aux2 word lists is if MULTI-EDIT encounters a misspelled word it will check the Aux1 list for a possible correct word but will not check the Aux2 list.

Click the left mouse button in the Word not in dictionary field to copy the word in that field to the Change To field. Click the left mouse button in the Change To text box to manually edit that word. Select other words from the Suggestions list to enter in the Change To field.

---

**ASCII Table**

Tools | ASCII Table or Alt+A will display a dialog of all the ASCII codes from 0 to 255. Both Hex and Decimal codes are shown for two different fonts, labeled OEM and ANSI. The ASCII table dialog is modeless, meaning you can continue to edit text while the table still appears on your screen. The following describes the fields in the dialog:

**Paste:** This button pastes the ASCII character corresponding to the currently highlighted ASCII code into the cursor position in the currently active file window.

Multi-Edit does not insert the character shown in either the OEM or ANSI columns when the paste command is selected. It simply inserts the ASCII code into the file. What actually appears on your screen will depend on which font you have selected in Multi-Edit. If you have an ANSI font selected and wish to insert a special OEM font character, you will most likely need to switch to that OEM font for that character to display properly. For more information on OEM/ANSI fonts and using them in Multi-Edit, see OEM Translation or Fonts.

**OEM / ANSI:** In Windows there are generally two different types of fonts, commonly referred to as OEM and ANSI. Both have the same ASCII codes representing the most common characters (decimal numbers, capital and lowercase letters, etc), but often have differing "non-typeable" characters (characters above ASCII 122). To accommodate this we have given the ASCII table the ability to display two different fonts. By clicking on the OEM or ANSI button, you can select a different OEM or ANSI font to display in the ASCII table. This is extremely useful for comparing two fonts.

---

**Create HTML from Code**

Create HTML From Code is a very convenient and timesaving feature that allows you to quickly create an HTML file from the current source file and maintains the source formatting and colored syntax highlighting!

This feature is located under the **Tools** menu on the main menu bar and offers two options for creating the HTML output.

The first will generate an HTML island and copy it to the Multi-Edit buffer so that it may be pasted into an existing HTML file. This proves useful when posting messages on a web forum. Note that using this option
does not provide the style sheet, which is used for syntax highlighting, but does maintain
the bolded keywords.

- An HTML island contains only the HTML tags needed for formatting the source code. An HTML island
  would not contain tags such as HTML, HEAD, or BODY.

The second option uses a template file that contains a style island that specifies the syntax colors to use.
Selecting this option generates the HTML (using the specified template) within a new file.

- A default template file is provided and located in the Defaults subdirectory.

**Calculator**

Selecting **Tools | Calculator** brings up a full function, programmer's
calculator that performs decimal, hexadecimal, octal, and binary calculations.
It also has an on-screen tape. The calculator is modeless, so you can keep it
up and available for easy use while you work. The calculator functions are as
follows:

---

**Memory Operations**

- **Min - Memory In:** This button stores the current result into memory, allowing you to recall it at a
  future time.
- **M+ - Memory Plus:** This button takes the current result and adds it to the value already stored in
  memory.
- **Mrec - Memory Recall:** This button recalls the value in memory. The memory is not cleared or reset.
- **Mclr - Memory Clear:** This button clears the memory, changing its value to zero.

**Base Operations**

You can use these option buttons to select the number base you want to work in. You can select Binary,
Decimal, Octal or Hexadecimal. You can switch modes to easily convert the value in the register from one
base to another.

**On Screen Tape**

The on-screen tape shows your previous calculator operations. You can use the scroll bar to move backward
through all your previous operations as needed.

**Register**

The register appears below the tape and above the buttons that cover the bottom half of the calculator. This is
where your current result or entered value is held before you perform an operation on it.
Binary Operations
These buttons appear in the lower left of the calculator and correspond to various logical operations typically performed on binary numbers:

- **And**: Will And two numbers.
- **Or**: Will Or two numbers.
- **Xor**: Will Exclusive Or two numbers.
- **<Shl**: Will perform a Shift Left function.
- **>Shr**: Will perform a Shift Right function.
- **Mod**: Performs a Modulus operation on the number.
- **!Not**: Provides the Not of the number.

Hexadecimal Buttons
The buttons A through F will enter hexadecimal values A through F into the register. The number base must be set to Hex for these values to be active.

Decimal Buttons
The buttons 0 through 9 enter the decimal numbers 0 through 9 into the register. The "." key places a decimal point on the register. The "+-" key will toggle between positive and negative values of the same number.

Clear
The clear button clears the register and indicates that the register was cleared on the on-screen tape. In addition, it cancels the current operation. For example, if you entered the number 10 and then pressed the '+=' button, then pressed the Clear button, the register would be cleared and the addition operation you were about to perform would be cancelled.

Operand Buttons
These buttons allow you to add, subtract, multiply and divide numbers, as well as gain the result of an operation.

Clr Tape
This button clears the on-screen tape. It does not affect the current register entry.

Paste
This button pastes the current register entry into the current cursor position of your current file.

Setup
This brings up the Calculator Setup dialog.

Calculator Setup
- **Word Length**: This allows you to define a word as 8, 16, or 32 bits. Since internal word lengths in Multi-Edit are 32 bits, this allows proper conversion between bases when using word lengths other than 32 bits.
- **Enable Tape**: When this is checked, the on-screen tape is enabled.
- **Unsigned**: When this is checked, all operations are considered unsigned. Otherwise, they are interpreted as signed.
- **Force Numlock**: This checkbox automatically turns the numlock on whenever the calculator is enabled and has focus.
Notebook

Tools | Notebook or Alt+N will open the Multi-Edit Notebook, a utility that allows you to compose and organize notes in a very convenient manner. When the Notebook is activated, a dialog box will appear with several buttons and two list fields.

The first list field, labeled 'Categories', contains a list of your current notebook categories. This allows you to organize your notes by subject, rather than having them jumbled together in one large list. A check mark is displayed next to the currently activated category.

The second list field is labeled 'Notes' and contains a list of all the note titles for the currently selected category, along with the date and time they were created. Icons will appear next to the notes according to whether they are marked 'To do', 'Completed', or neither.

Activate
This button allows you to activate the category currently highlighted. Alternatively, you can activate a category by double clicking on the category title. The check box will move to the active category, and note titles for that category will be displayed in the Notes list box.

Complete
This button allows you to quickly mark notes as completed, whether they were initially marked 'To do' or not. Completed notes are marked with a lightning bolt and move to the bottom of the Notes list.

Insert
The insert button creates either a new category or a new note, depending on what list field you currently have active. See Edit Note.

Delete
The delete button simply deletes either the currently highlighted category or note. You will always be prompted to confirm your delete.

Edit
By selecting the edit button, you are able to edit the currently highlighted category or note.

Change
Multi-Edit's notebook feature allows you to keep separate notebook files. Thus, it is possible to have a notebook file for your personal notes and another file for your work notes. Within each notebook file, you can have different categories and notes.

When you select the change button, you will be presented with a 'Select Notebook' dialog box that prompts you for the filename of the new notebook file. Notebook files can have any extension, but by default, Multi-Edit gives notebook files an extension of .TXT.
Print
This button brings up the Notebook Print dialog box with several printing options:
- **Print Category**: This prints all of the notes in the selected category.
- **Print Single Note**: This prints the currently highlighted note.
- **Print Summary**: When checked and Print category is invoked, a summary of all notes precedes the printout of the notes themselves.
- **"To Do" Items Only**: When checked and Print category is invoked, only notes with the 'To Do' check box turned on will be printed.

Edit Note
The Edit Note Dialog box allows you to create new notes or modify the content of existing notes in the Notebook. The following describes the fields in the dialog and their use:

**Subject**
You may enter any subject appropriate for your note in this field.

**To Do**
The 'To Do' check box has been provided to allow you to organize your notes better. When an item is marked 'To Do', it will appear at the top of the list of notes. If several notes are marked 'To Do', they all appear at the top, sorted with the newest notes on top.

**Complete**
To complement the 'To Do' check box, a 'Complete' check box has been added to allow you to designate which notes are completed and which aren't. When a note is marked 'Complete' it moves to the bottom of the list of notes. This allows you to keep notes on file even after they've been completed without having them clutter up your workspace.

**Text**
The text field is large enough to accommodate a note of virtually any size. This is where you can keep the bulk of your note's detail.

**Created**
The 'Created' field holds the time and date that the note was initially created. If you wish, these fields can be edited manually.

**Last Mod**
Short for 'Last Modification'. This field holds the time and date this note was last modified. As with the 'Created' field it can be manually edited.

**Linedraw**
To use the line drawing feature:
1. Select Linedraw from **Tools | Line Draw**.
2. Select the type of line you want (none, single, double).
3. Use the mouse to return focus to the editing window (click in the editing window).
4. Move your caret (cursor) to where you want your lines to start.
5. Press and hold the SHIFT key while using the arrow keys to draw the lines.

---

⚠️ If you do not select a font within Multi-Edit that supports line drawing characters, your line drawing characters may appear incorrectly. Set up specific fonts for each extension from the Filename Extensions dialog box, and you can change the font globally from the Fonts dialog box.
System View

View | System will open the System View pane in Multi-Edit which is used for viewing global variables, window/buffer properties and loaded macros. This view is most often used when debugging macros or verifying window properties.

Globals and Window/Buffer Properties
The root tree items Globals, Window Properties, and Buffer Properties all contain a String and Integer sub item for quick reference. These sub items can be sorted by right clicking on them and then selecting sort and the ascending/descending option.

When a value is assigned to a Global, Window Property, or Buffer Property variable, it is placed in its corresponding tree sub item depending on type (String or Integer). Selecting one of these variables will display its value in a text box at the top of the tree where it can then be modified.

Loaded Macros
Expanding the Loaded Macros tree item will display all macros currently in memory. Selecting a macro will display it in the textbox at the top of the tree where parameters can then be specified.

Set/Run Button
Depending on the type of item selected (variable or macro) a "set" or "run" button is displayed at the top of the tree. If a variable is selected, the "set" button is displayed which will save the value specified in the adjoining text field. If a macro is selected, then the "run" button is displayed allowing the current macro to be executed.

Refreshing the tree
Clicking on the tree button above the tree control will refresh the tree adding and/or removing variables and macros as necessary.
Metacommands

What are Metacommands?

Multi-Edit Metacommands are special character sequences that allow the user to express certain data in command lines, prompts, and configuration options that would otherwise be difficult, if not impossible to enter literally. One of the most common uses for Metacommands is embedding them in a compiler command line. Below is a list of all of the supported Metacommands classified into groups by where they can be accessed.

In addition to the general command line Metacommands, which can be used anywhere Metacommands are supported, there are special Metacommands for printing, templates, help file support, VCS support, program execution and the search dialogs. There are also error-parsing routines that use special metacommands called Regular Expression Aliases.

Long Filename Metacommands

Two new metacommands have been added for long filename support. Neither of these metacommands actually expands to anything (unlike the <FILE> metacommand). Instead, they act as "flags" to the various parts of Multi-Edit that call them. When using these metacommands, you should ensure that any external applications you are calling supports long filenames (or short filenames, as may be the case). If, for some reason, you want to change filename schemes in the middle of your command line, you can do so by specifying a different filename metacommand in the middle of the command line. The Long Filename Support Table:

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SFN&gt;</td>
<td>Stands for Short File Name. When this metacommand is inserted, all file and pathname metacommands following it will be expanded in SHORT name format. It is also the default metacommand (i.e. filenames expand in short form by default).</td>
</tr>
</tbody>
</table>
| <LFN>       | Stands for Long File Name. The <LFN> metacommand causes all file and pathname metacommands following it to expand in LONG format. Example:  

\[ <LFN>D:\PROGRAMS\SOMEPROG.EXE -F<FILE>.<EXT> -A -B -C... etc \]
General Command Line Metacommands

The general command line Metacommands are expanded by the McTools`TranslateCmdline macro and any Metacommand that it does not recognize will remain intact on exit. This macro is usually called by the other Metacommand translate macros after they have expanded the ones they recognize.

General Command Line Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>@$macro$</td>
<td>Substitutes the string in Return_Str after running the macro specified by &quot;macro&quot;.</td>
</tr>
<tr>
<td>$%var$</td>
<td>Substitutes the value of the DOS environment variable &quot;var&quot;.</td>
</tr>
<tr>
<td>$&lt;str$</td>
<td>Substitutes the value of the global variable &quot;str&quot;. The global variable can be either a string or integer variable where the integer value will be written as a string. For example, $&lt;!HELP_PATH&gt;$ would insert the value of Global_Str(&quot;!HELP_PATH&quot;) into the prompt or option.</td>
</tr>
<tr>
<td>$&lt;FILE&gt;$</td>
<td>Substitutes the current window's file name minus an extension.</td>
</tr>
<tr>
<td>$&lt;EXT&gt;$</td>
<td>Substitutes the current window's file extension.</td>
</tr>
<tr>
<td>$&lt;NAME&gt;$</td>
<td>Substitutes the current window's file name minus the extension and path.</td>
</tr>
<tr>
<td>$&lt;PATH&gt;$</td>
<td>Substitutes the current window's path.</td>
</tr>
<tr>
<td>$&lt;ME_PATH&gt;$</td>
<td>Substitutes the path where Multi-Edit resides.</td>
</tr>
<tr>
<td>$&lt;USER_PATH&gt;$</td>
<td>Substitutes the path where the user specific configuration files reside. If not network version, defaults to the &quot;config directory&quot;.</td>
</tr>
<tr>
<td>$&lt;MAC_PATH&gt;$</td>
<td>Substitutes the path to the current user's directory.</td>
</tr>
<tr>
<td>$&lt;DEFAULTS_PATH&gt;$</td>
<td>Substitutes the path where default templates and configuration items are kept.</td>
</tr>
<tr>
<td>$&lt;USER_ID&gt;$</td>
<td>Substitutes the current User Id. Only applicable to the Network version. See NETWORK SUPPORT for more details.</td>
</tr>
<tr>
<td>$&lt;COMSPEC&gt;$</td>
<td>Substitutes the current command processor (command.com).</td>
</tr>
</tbody>
</table>
## General Command Line Support continued…

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;CTIME&gt;</code></td>
<td>Substitutes the current time. The <code>&lt;CTime&gt;</code> Metacommands allow for passing an optional format string. To specify a format string, use the following format.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;CTime format string&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;CTIME HH':'mm&gt;</code> would insert the time as 15:07</td>
</tr>
<tr>
<td></td>
<td>See table below for all of the valid format string values.</td>
</tr>
<tr>
<td><code>&lt;CDATE&gt;</code></td>
<td>Substitutes the current date. The <code>&lt;CDate&gt;</code> Metacommands allow for passing an optional format string. To specify a format string, use the following format.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;CDate format string&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;CDATE dd'-'MMM'-'yy&gt;</code> would insert the date as 27-Jan-99</td>
</tr>
<tr>
<td></td>
<td>See table below for all of the valid format string values.</td>
</tr>
<tr>
<td><code>&lt;FTIME&gt;</code></td>
<td>Substitutes the last file save time.</td>
</tr>
<tr>
<td><code>&lt;FDATE&gt;</code></td>
<td>Substitutes the last file save date.</td>
</tr>
<tr>
<td><code>&lt;PROJECT&gt;</code></td>
<td>Substitutes the current project file's name minus the extension and path.</td>
</tr>
<tr>
<td><code>&lt;PROJECT_FILE&gt;</code></td>
<td>Substitutes the current project file's name.</td>
</tr>
<tr>
<td><code>&lt;PROJECT_FILEPATH&gt;</code></td>
<td>Substitutes the current project file's path.</td>
</tr>
<tr>
<td><code>&lt;PROJECT_ROOT&gt;</code></td>
<td>Substitutes the root path for the current project.</td>
</tr>
<tr>
<td><code>&lt;PROJECT_LIST&gt;</code></td>
<td>Substitutes the name of a file that contains a list of the filenames in the current project.</td>
</tr>
</tbody>
</table>
### <CDate> and <CTime> Format String Values

<table>
<thead>
<tr>
<th>Format String</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>d</strong></td>
<td>Day of month as digits with no leading zero for single-digit days.</td>
</tr>
<tr>
<td><strong>dd</strong></td>
<td>Day of month as digits with leading zero for single-digit days.</td>
</tr>
<tr>
<td><strong>ddd</strong></td>
<td>Day of week as a three-letter abbreviation. The function uses the LOCALE_SABBREVDAYNAME value associated with the specified locale.</td>
</tr>
<tr>
<td><strong>dddd</strong></td>
<td>Day of week as its full name. The function uses the LOCALE_SDAYNAME value associated with the specified locale.</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>Month as digits with no leading zero for single-digit months.</td>
</tr>
<tr>
<td><strong>MM</strong></td>
<td>Month as digits with leading zero for single-digit months.</td>
</tr>
<tr>
<td><strong>MMM</strong></td>
<td>Month as a three-letter abbreviation. The function uses the LOCALE_SABBREVMONTHNAME value associated with the specified locale.</td>
</tr>
<tr>
<td><strong>MMMM</strong></td>
<td>Month as its full name. The function uses the LOCALE_SMONTHNAME value associated with the specified locale.</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>Year as last two digits, but with no leading zero for years less than 10.</td>
</tr>
<tr>
<td><strong>yy</strong></td>
<td>Year as last two digits, but with leading zero for years less than 10.</td>
</tr>
<tr>
<td><strong>yyyy</strong></td>
<td>Year represented by full four digits.</td>
</tr>
<tr>
<td><strong>gg</strong></td>
<td>Period/era string. The function uses the CAL_SERASTRING value associated with the specified locale. This element is ignored if the date to be formatted does not have an associated era or period string.</td>
</tr>
<tr>
<td><strong>h</strong></td>
<td>Hours with no leading zero for single-digit hours; 12-hour clock.</td>
</tr>
<tr>
<td><strong>hh</strong></td>
<td>Hours with leading zero for single-digit hours; 12-hour clock.</td>
</tr>
<tr>
<td><strong>H</strong></td>
<td>Hours with no leading zero for single-digit hours; 24-hour clock.</td>
</tr>
<tr>
<td><strong>HH</strong></td>
<td>Hours with leading zero for single-digit hours; 24-hour clock.</td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>Minutes with no leading zero for single-digit minutes.</td>
</tr>
<tr>
<td><strong>mm</strong></td>
<td>Minutes with leading zero for single-digit minutes.</td>
</tr>
</tbody>
</table>

( Date and Time string values table continues…)
<CDate> and <CTime> Format String Values continued…

<table>
<thead>
<tr>
<th>Format String</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Seconds with no leading zero for single-digit seconds</td>
</tr>
<tr>
<td>ss</td>
<td>Seconds with leading zero for single-digit seconds</td>
</tr>
<tr>
<td>T</td>
<td>One character time marker string, such as A or P</td>
</tr>
<tr>
<td>tt</td>
<td>Multicharacter time marker string, such as AM or PM</td>
</tr>
</tbody>
</table>

**Execute Program Metacommands**

The following Metacommands allow specifying the exe type of a program on the command line so that the ExecProg macro can be notified how the program is to be launched. These can be placed anywhere in the command line but the best place to use these would be just before the first parameter of the command line.

**Execute Program Support**

<table>
<thead>
<tr>
<th>Metacommend</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;XDOS&gt;</td>
<td>Force command line to be run as a Dos program (Uses DosExec)</td>
</tr>
<tr>
<td>&lt;XOS2&gt;</td>
<td>Force command line to be run as an OS/2 program (Uses Os2Exec)</td>
</tr>
<tr>
<td>&lt;XW16&gt;</td>
<td>Force command line to be run as a 16-bit Windows program. (Run directly)</td>
</tr>
<tr>
<td>&lt;XW32&gt;</td>
<td>Force command line to be run as a 32-bit Windows program (Run directly)</td>
</tr>
<tr>
<td>&lt;XW32C&gt;</td>
<td>Force command line to be run as a Win32 Console program (Uses W32Exec)</td>
</tr>
<tr>
<td>&lt;XMAC&gt;</td>
<td>Force command line to be run as a macro.</td>
</tr>
</tbody>
</table>

Example:

Get command for PVCS
GET.EXE <XDOS>-L <LPATHX>(<NAME>.<EXT>)
Printing Metacommands

The following are valid only for printing. Used for headers and footers set up.

### Printing Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;LM&gt;</td>
<td>Pushes all following text to left margin</td>
</tr>
<tr>
<td>&lt;CM&gt;</td>
<td>Centers the following text</td>
</tr>
<tr>
<td>&lt;RM&gt;</td>
<td>Right justifies following text</td>
</tr>
<tr>
<td>&lt;PAGE&gt;</td>
<td>Substitutes current page number</td>
</tr>
</tbody>
</table>

Compiler Metacommands

These Metacommands are only valid in a compiler command line via the compiler run dialog and are expanded directly by the Compile^CompileEx macro.

### Compiler Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;MEERR&gt;</td>
<td>This will substitute the name of the file used to capture the compiler screen output (stdout/stderr), which is a unique temporary filename. If you are using a compiler command line or a batch file that has MEERR.TMP in it, you should convert over to using &lt;MEERR&gt; instead. In the case of batch files, you'll probably have to add &lt;MEERR&gt; to the end of the command line that runs the batch file, then change the occurrences of MEERR.TMP in the file to %x (x being the next available parameter number).</td>
</tr>
<tr>
<td>&lt;NR&gt;</td>
<td>This does not do a substitution but will set the options to not capture any output (stdout/stderr).</td>
</tr>
<tr>
<td>&lt;TESTERR&gt;</td>
<td>This is a special Metacommand that is used for testing purposes. When this Metacommand appears on the command line the compile macro will not launch any program but rather open a CmpError.db file which contains error samples, search it for a page matching the program type, mark and paste this into the output window and then start the error processing macro. We use this to test the error processing regular expression data for compilers that we do not have access to.</td>
</tr>
</tbody>
</table>
# VCS Metacommands

These Metacommands are specific to the VCS support and a few are for specific VCS programs.

## VCS Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;LOOKUP&gt;</td>
<td>Substitutes the archive path looked up in the associate directory database.</td>
</tr>
<tr>
<td>&lt;CFGFILE&gt;</td>
<td>Substitutes the archive path looked up in the specific VCS package config file. Currently the same as &lt;LOOKUP&gt;</td>
</tr>
<tr>
<td>&lt;LPATHX&gt;</td>
<td>Substitutes the derived library path minus an ending &quot;/&quot;.</td>
</tr>
<tr>
<td>&lt;LPATH&gt;</td>
<td>Substitutes the derived library path.</td>
</tr>
<tr>
<td>&lt;OPATH&gt;</td>
<td>Substitutes the current window's path minus drive letter.</td>
</tr>
<tr>
<td>&lt;COMNT&gt;</td>
<td>Substitutes the filename of the file containing the VCS comment.</td>
</tr>
<tr>
<td>&lt;%TMP&gt;</td>
<td>Substitutes the value of DOS environment variable &quot;TMP&quot; with an ending &quot;/&quot;.</td>
</tr>
<tr>
<td>&lt;TMP_PATH&gt;</td>
<td>Substitutes the value of Multi-Edit temporary file path variable &quot;@TMP_FILE_PATH&quot;.</td>
</tr>
<tr>
<td>&lt;TMP_PATHX&gt;</td>
<td>Substitutes the value of Multi-Edit temporary file path variable &quot;@TMP_FILE_PATH&quot; minus an ending &quot;/&quot;.</td>
</tr>
</tbody>
</table>

## CVS Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CVSDIR&gt;</td>
<td>Substitute the value of the environment variable &quot;CVSDIR&quot;</td>
</tr>
</tbody>
</table>

## RCS Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;RCSDIR&gt;</td>
<td>Substitutes the value of the DOS environment variable &quot;RCSDIR&quot;.</td>
</tr>
</tbody>
</table>
### SAP Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;SAPCMT&gt;</code></td>
<td>Substitutes the Sourcerer’s Apprentice command &quot;-c comment&quot;.</td>
</tr>
</tbody>
</table>

### Help File Metacommands

These Metacommands are used in the Help Viewers Setup dialog to specify help files and options to be passed to the Help macro.

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;HFILE&gt;</code></td>
<td>Substitutes the name of the specified Help file without extension.</td>
</tr>
<tr>
<td><code>&lt;HEXT&gt;</code></td>
<td>Substitutes the extension of the specified Help file.</td>
</tr>
<tr>
<td><code>&lt;HNAME&gt;</code></td>
<td>Substitutes the name of the specified Help file without path or extension.</td>
</tr>
<tr>
<td><code>&lt;HPATH&gt;</code></td>
<td>Substitutes the path of the specified Help file without extension.</td>
</tr>
<tr>
<td><code>&lt;HOPTS&gt;</code></td>
<td>Substitutes the options, usually a help index, for the current context sensitive help call.</td>
</tr>
<tr>
<td><code>&lt;HTMLHELP&gt;</code></td>
<td>Substitutes the command line to run to invoke the HtmlHelp program.</td>
</tr>
<tr>
<td><code>&lt;WINHELP&gt;</code></td>
<td>Substitutes the command line to run to invoke the WinHelp program.</td>
</tr>
</tbody>
</table>
## Template Metacommands

These Metacommands can only be used in templates.

### Template Support

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;-?&gt;</td>
<td>This metacommand expands to a matching begin/end comment, with a ? in the middle that you can fill in later.</td>
</tr>
<tr>
<td>&lt;-SPACE&gt;</td>
<td>Inserts a space for use at end of line.</td>
</tr>
<tr>
<td>&lt;-TAB&gt;</td>
<td>This will have the effect of pressing the tab key.</td>
</tr>
<tr>
<td>&lt;-UP&gt;</td>
<td>Move the cursor up.</td>
</tr>
<tr>
<td>&lt;-CUR&gt;</td>
<td>This will indicate where you want the cursor to end up after expansion is completed.</td>
</tr>
<tr>
<td>&lt;-CR&gt;</td>
<td>Line break. This is only needed if you wish to insert a carriage return at the end of your template result. Otherwise you can simply break a line in the result field within your template.</td>
</tr>
<tr>
<td>&lt;-SMARTIND+&gt;</td>
<td>Any line breaks in the result that follow this command will invoke the smart indent macro rather than simple carriage returns.</td>
</tr>
<tr>
<td>&lt;-SMARTIND-&gt;</td>
<td>This turns off the above metacommand.</td>
</tr>
<tr>
<td>&lt;-SCUR&gt;</td>
<td>This saves the cursor position at the point of insertion for later restoring.</td>
</tr>
<tr>
<td>&lt;-RCUR&gt;</td>
<td>If used after a &lt;-SCUR&gt; metacommand, will restore the cursor position.</td>
</tr>
<tr>
<td>&lt;-XCUR&gt;</td>
<td>Exchange the current cursor position with the last &lt;-SCUR&gt; saved position and then move the cursor to the original saved position.</td>
</tr>
<tr>
<td>&lt;-SIND&gt;</td>
<td>Save the old indent level and then set the indent level to the first word on the current line. (Used for nested code templates).</td>
</tr>
<tr>
<td>&lt;-SCIND&gt;</td>
<td>Save the old indent level and set the indent level to the current cursor position. (Used for nested code templates).</td>
</tr>
<tr>
<td>&lt;-RIND&gt;</td>
<td>Restore the indent level saved by &lt;-SIND&gt; and &lt;-SCIND&gt;.</td>
</tr>
</tbody>
</table>

(Template Support Table continues…)
Template Support continued…

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
</table>
| <-PROMPT>     | Will invoke a prompt during a template expansion whose result will be inserted in the result in this position. Multiple prompts are supported as well as specifying a default value. The format is as follows:  
  
  <-Prompt~str message default>  
  
  where:  
  
  ~str   (Optional) string specifying prompt global for <-LastPrompt>  
  
  message   (Optional) message that will be show in the prompt  
  
  default   (Optional) string that will be the default value in the edit field  
  
  Examples:  
  
  <-Prompt>  
  
  <-Prompt~1 Enter name’Dan> |
| <-LASTPROMPT> | If used after a <-Prompt> metacommand, it can be used to redundantly insert the value of the prompt without having to be prompted again. Supports multiple prompts with the following format:  
  
  <-LastPrompt~str>  
  
  where  
  
  ~str – (optional) string specifying the prompt global set in <-Prompt>  
  
  Examples:  
  
  <-LastPrompt>  
  
  <-LastPrompt~1> |
| <-GETBLOCK>   | This metacommand will cut a marked block to buffer 52 if the cursor is in the block. |
| <-PUTBLOCK>   | This metacommand will paste a block cut by the <-GetBlock> metacommand. |
| <-TEXT str>   | Insert "str" exactly as entered. |

(Template Support Table continues…)

178 • Metacommmands

Multi-Edit 9.10
### Template Support continued…

<table>
<thead>
<tr>
<th>Metacommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;^template&gt;</code></td>
<td>This metacommand will invoke another template in the set. The text between the ^ and the &gt; must be a valid template name in either the current set or the Global set.</td>
</tr>
<tr>
<td><code>&lt;nnn&gt;</code></td>
<td>Character in decimal (0-255). This is useful for entering &quot;non-typeable&quot; characters.</td>
</tr>
<tr>
<td><code>&lt;0xHH&gt;</code></td>
<td>Character in hex (0-FF). Same as above but uses hex numbers instead of decimal numbers.</td>
</tr>
<tr>
<td><code>&lt;!macro&gt;</code></td>
<td>Runs the macro specified. For example, to run the macro Sp_Setup in the macro file mymacs, the metacommand would be <code>&lt;!Mymacs^Sp_Setup&gt;</code>. Regular parameters can be passed using this method, i.e. <code>&lt;!Mymacs^Sp_Setup /PARM1=something/PARM2=something else&gt;</code></td>
</tr>
</tbody>
</table>

Not all prompts and configuration options, etc. support Metacommands. Check the documentation for each prompt, etc. to see if it supports Metacommands.
Modifying Multi-Edit Startup

Command Line Switches

What are Command Line Switches?
Multi Edit offers the ability to configure almost all of its options via the Tools | Customize dialog. This dialog box allows the user to set up default configurations that remain constant from one Multi-Edit session to another. Sometimes, the user needs to change a default setting temporarily, or there are some options that can't be configured due to hardware constraints. This is where command line options can be used.

No Restore - /NR
This is used if you have the Restore feature turned ON (via the Session Manager Setup dialog box) and would like to invoke Multi-Edit without restoring the previous status of the editor. This will not disable Multi-Edit from saving the status as you exit.

To use the No Restore command line switch, type the following at the command prompt:
MEW32 /NR

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Read Only - /RO
The Read Only switch is available so a writable file can be loaded read-only into Multi-Edit. To use the Read Only command line switch, type the following at the command prompt:
MEW32 /RO

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.
Read/Write - /RW

This will reverse the /RO option and will cause files to be loaded based upon their attributes. To use the Read/Write command line switch, type the following at the command prompt:
MEW32 /RW

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

A file flagged as read-only will never be loaded writable even when using the /RW command line switch.

No Splash - /NoSplash

Use this command line switch for removing the Splash Screen when starting Multi-Edit.
To use the No Splash command line switch, type the following at the command prompt:
MEW32 /NoSplash

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

No Restore, No Save - /NS

This option works similarly to /NR, except this also instructs Multi-Edit not to save the status upon exiting, in addition to not restoring the editors previous status.
To use the No Restore, No Save command line switch, type the following at the command prompt:
MEW32 /NS

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Run A Macro - /R[macro_name]

Runs the macro: macro_name immediately upon startup. A space between the /R and the macro_name is optional.

A macro executed in this manner is run immediately upon startup. If you want to run a macro from the command line, you should take into consideration that not everything in Multi-Edit has been initialized when the macro executes.
To use the Run a Macro command line switch, type the following at the command prompt:

MEW32 /R[macro name]

_A space between MEW32 and the / is optional, but you must separate multiple command line options with a space._

**Display Session Manager - /SM**

This option will start the session manager after starting Multi-Edit. This allows you to immediately pick a session from the list displayed or create a new session to work in.

To use the Display Session Manager command line switch, type the following at the command prompt:

MEW32 /SM

_A space between MEW32 and the / is optional, but you must separate multiple command line options with a space._

**Start A Session - /SN[session_name]**

Similar to /SM, this option starts a specific session which you define in the command line. Unlike the /R option, the session_name must immediately follow the /SN command line option. The session_name refers to the actual session name, not the working directory. If you are specifying a session containing space characters, you must substitute the underscore for each space. For example a session called MY NEW SESSION would be specified /SNMY_NEW_SESSION.

To use the Start A Session command line switch, type the following at the command prompt:

MEW32 /SN[session name]

_A space between MEW32 and the / is optional, but you must separate multiple command line options with a space._

**Start The Last Session - /SR**

Another option dealing with the session manager, this one will start Multi-Edit with the session used at the last time Multi-Edit was run.

To use the Start The Last Session command line switch, type the following at the command prompt:

MEW32 /SR

_The /SR switch is set active by default with installation of Multi-Edit._

_A space between MEW32 and the / is optional, but you must separate multiple command line options with a space._
Goto Line n - /L[n]
This will cause Multi-Edit to move to line n in the first file loaded.
To use the Goto Line n command line switch, type the following at the command prompt:
MEW32 /L[n]

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Goto Column n - /C[n]
This will cause Multi-Edit to move to column n in the first file loaded.
To use the Goto Column n command line switch, type the following at the command prompt:
MEW32 /C[n]

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Bypass VCS - /NV
Using this command line switch allows you to bypass the Version Control initialization.
To use the Bypass VCS command line switch, type the following at the command prompt:
MEW32 /NV

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Load A File – [filename]
You can load a file into Multi-Edit by specifying its filename as a command line option. Alternatively, you can specify a "file type" for this file to use upon startup.

When loading files in Multi-Edit, take care to specify the full path and filename to the file, especially when loading files not in the working directory. It is also recommended that you use quotes around the filename.

To use the Load A File command line switch, type the following at the command prompt:
MEW32 [filename]

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.
Load A DOS File - /*[filename]

Loads the file in MSDOS format.

When loading files in Multi-Edit, take care to specify the full path and filename to the file, especially when loading files not in the working directory.

To use the Load A DOS File command line switch, type the following at the command prompt:
MEW32 /*[filename]

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Load A Unix File - /!/filename

Loads the file in Unix format.

When loading files in Multi-Edit, take care to specify the full path and filename to the file, especially when loading files not in the working directory.

To use the Load A UNIX File command line switch, type the following at the command prompt:
MEW32 /!/filename

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Load A Binary File - /&[filename]

Loads the file in Binary format.

When loading files in Multi-Edit, take care to specify the full path and filename to the file, especially when loading files not in the working directory.

To use the Load A Binary File command line switch, type the following at the command prompt:
MEW32 /&[filename]

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.
Load a List of Files - /@[filelist]

This command line option instructs Multi-Edit to open the list of files defined in the filelist. The file list must be an MSDOS text file that contains only one file per line.

When loading files in Multi-Edit, take care to specify the full path and filename to the file, especially when loading files not in the working directory.

To use the Load A List of Files command line switch, type the following at the command prompt:

MEW32 /@[filelist]

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Multiple Instances - /NI

This tells Multi-Edit to invoke another instance if one is already running. In other words, you can have multiple copies of Multi-Edit running at once.

To use the Multiple Sessions command line switch, type the following at the command prompt:

MEW32 /NI

A space between MEW32 and the / is optional, but you must separate multiple command line options with a space.

Startup.cfg and Startup2.cfg Files

With the introduction of the Update script and Add-On package support in Multi-Edit 8, a method to use these features to replace the Startup.s file was added to Multi-Edit 9.10. Using the Update scripting language, a Startup.cfg file is created that will allow almost everything that can be done with the Startup.s file to be done but without needing to compile a macro. To support this an entry was added to the Add-On package list called Startup, which will run the Startup.cfg and Startup2.cfg update scripts when Multi-Edit start or switches session just like any other Add-On package.

The advantage of using the Startup.cfg file over the Startup.s method is that being set up as an Add-On package it can be enabled/disabled as the user pleases just by selecting a button in the Add-On Package Manager dialog.

Startup.cfg is run before all of the command line options are processed and before the session data is restored. Startup2.cfg serves the same purpose except it is run after the command line is processed and the session data is restored which allows for doing things such as overriding global variable values that would be restored in a session restore. Both are run during startup and session switches.
Below are some of the things that can be done in the Startup.cfg file.

- Setting a global integer variable
  GInt variable_name value
- Setting a global string variable
  GStr variable_name value
- Load a macro file
  LoadMac macro_file
- Run a macro
  RunMac macro_command_line

For actual examples of these, see the Startup.cfg file in the Multi-Edit 9.10 Defaults sub-directory.

---

**The Update Macro Processor**

The purpose of the Update macro is to provide a simple script driven method of updating command maps, menus, toolboxes and other db files. It can also be used for configuring and installing Add-On packages. The following will describe the script language defined in the Update macro.

A script file is a text file that contains lines of commands that perform desired functions. The extensions for these script files can be anything, but using one of the four defined extensions is desired. These are

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CFG</td>
<td>contain scripts to configure Multi-Edit, a good way to replace STARTUP.S.</td>
</tr>
<tr>
<td>.DAT</td>
<td>contain scripts to install Add-On packages. Used by the Install macro.</td>
</tr>
<tr>
<td>.DEL</td>
<td>contains scripts to uninstall Add-On packages. Used by the Install macro.</td>
</tr>
<tr>
<td>.UPD</td>
<td>contain scripts to update db files.</td>
</tr>
</tbody>
</table>

The Update macro starts at the top of the script file and reads each line, checking to be sure the first non white space string on each line is a valid command, and then performs the desired function. Since the Update macro reads and executes statements on a line-by-line basis, statements must not cross line boundaries. A statement is classified into four groups:

- Line comments
- Block comments
- Conditionals
- Commands

(topic continues…)
Commands are further divided into
- General commands
- Data commands
- Set commands

Statements are not case sensitive and can be entered however the user desires.

In this document the following have special meaning:
- Text enclosed in `{ }` can be used zero or more times.
- Text enclosed in `[]` is optional.
- Commands are shown in Mixed Case and start with a capital letter.
- User entered data is shown in italics.

---

**Update Script Reference**

**Line Comments**

`;` A line starting with a `;` is interpreted as a line comment, where all text between the `;` and end of line character are ignored.

⚠️ Blank lines are also treated as a line comment and are ignored.

**Block Comments**

/* Beginning of block comment

*/ End of block comment

A line starting with `/*` will be interpreted as the start of a block comment, and the rest of the line is ignored. Each following line will be checked and ignored until a line starting with `*/` is found. When this is found, the block comment ends and the following line will be interpreted as a new statement to be executed.

⚠️ Block comments can be nested.

---

**Conditionals**

**IfRC expression**

Every statement, when run, sets the Result variable, which can be tested with the `IfRC` statement. A line starting with `IfRC`, meaning If Result Code, will enable statements to be conditionally executed depending upon the evaluated result of an expression. An expression that evaluates to True will cause all the following lines to be executed until a line starting with either `Else` or `EndIf` is encountered, otherwise it will ignore the following lines until either `Else` or `EndIf` is found.
Expression is defined as an operator followed by a number. All valid expressions are listed below, where number is a user-defined number.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>== number</td>
<td>evaluates True when Result is equal to number</td>
</tr>
<tr>
<td>&lt; number</td>
<td>evaluates True when Result is less than number</td>
</tr>
<tr>
<td>&gt; number</td>
<td>evaluates True when Result is greater than number</td>
</tr>
<tr>
<td>&lt;= number</td>
<td>evaluates True when Result is less than or equal to number</td>
</tr>
<tr>
<td>&gt;= number</td>
<td>evaluates True when Result is greater than or equal to number</td>
</tr>
<tr>
<td>!= number</td>
<td>evaluates True when Result is not equal to number</td>
</tr>
</tbody>
</table>

If expression

A line starting with If will enable statements to be conditionally executed depending upon the evaluated result of expression. An expression that evaluates to True will cause all of the following lines to be executed until a line starting with Else or EndIf is encountered, otherwise all following lines until either Else or EndIf are found are ignored.

Expression is defined as statement1 operator statement2, where statement can be any of the following:

- a string
- a number
- a global_str_var, i.e. <--Str_Var>
- a global_int_var, i.e. <--Int_Var>

Operator can be any of the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>evaluates True when statement1 is equal to statement2</td>
</tr>
<tr>
<td>&lt;</td>
<td>evaluates True when statement1 is less than statement2</td>
</tr>
<tr>
<td>&gt;</td>
<td>evaluates True when statement1 is greater than statement2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>evaluates True when statement1 is less than or equal to statement2</td>
</tr>
<tr>
<td>&gt;=</td>
<td>evaluates True when statement1 is greater than or equal to statement2</td>
</tr>
<tr>
<td>!=</td>
<td>evaluates True when statement1 is not equal to statement2</td>
</tr>
</tbody>
</table>

Multi-Edit 9.10 Modifying Multi-Edit Startup ● 189
Else
When a line starting with the Else statement is found, the following lines will NOT be executed if the preceding IfRC expression evaluated to True. Otherwise they will be executed until a line starting with EndIf is located.

Endif
A line starting with EndIf will end the conditional execution for the innermost encountered IfRC.

Conditional statements may be nested up to 16 levels deep.

Loop
A line starting with the Loop statement will save the line number it is on so that the matching EndLoop statement can reset the next line to be executed to be the Loop statement.

EndLoop
A line starting with the EndLoop statement will cause the matching Loop statement to be the next statement executed.

Break
When a line starting with the Break statement is found, all lines up to and including the next EndLoop statement are ignored. The next statement to be executed will be the line following the EndLoop.

Loop statements may be nested up to 10 levels deep.

Quit
When this command is encountered, processing of the script file is stopped and the rest of the file is ignored. This allows a way to exit processing of a script file.

General Commands
The commands in this section are used to create/set global variables, manipulate macros, files, directories, and execute programs. There is also a command to allow registering a resource DLL with Multi-Edit.

GInt global_int_var [ number ]
GStr global_str_var [ string ]

These commands allow setting global integer and string variables, respectively. If number or string is not provided, then values of 0 and "" are used which will erase the global variable. Otherwise the variable is created and set to the passed value.

The result code is always set to _noError (See DbTools.sh for the definition of the error codes).

Global variables may be referred to in some of the other script commands by using the metacommand <<var_name>>, where var_name is the name of a string global variable.
**LoadMac macro_file_name**

This command loads the macro file, *macro_file_name*, into memory. All metacommands are translated in *macro_file_name* before the Load_Macro_File() function is called.

The result code is set to the Error_Level resulting from running the Load_Macro_File( ) function unless *macro_file_name* is blank, and is then set to _errorNoFile.

**RunMac macro_name { parameter }**

This command causes the macro *macro_name* to be run using the *parameter* list. This command translates all metacommands contained in the macro command line.

The result code is set to whatever Return_Int contains upon exiting the macro unless *macro_name* is not specified, and is then set to _errorNoFile.

**Compile macro_name**

This command compiles the macro *macro_name*. All metacommands are translated. If *macro_name* starts with the "@" character, then the file *macro_name* is read as a list of macros to be compiled.

The result code is the result of the WinExecAndWait macro used to execute the compiler unless *macro_name* is blank, and is then set to _errorNoFile.

---

This will fail if you did not choose to install the CMAC compiler during installation. The CMAC compiler is not installed by default.

**Md directory_name**

**MkDir directory_name**

These commands try to make the directory *directory_name*. All metacommands contained in *directory_name* are translated.

The result code is set to the result of the MkDir( ) function unless *directory_name* is not specified, and is then set to _errorNoFile.

**Rd directory_name**

**RmDir directory_name**

These commands try to remove the directory *directory_name*. All metacommands contained in *directory_name* are translated.

The result code is set to the result of the RmDir( ) function unless *directory_name* is not specified, and is then set to _errorNoFile.

**CopyFile src_file_name dest_file_name**

This command copies a file *src_file_name* to *dest_file_name*. All metacommands are translated.

The result code is set to the result of the Copy_File( ) function unless *src_file_name* and *dest_file_name* are blank, and is then set to _errorNoFile.
**DelFile file_name**

This command causes the file *file_name* to be deleted. All metacommands are translated.
The result code is set to the Error_Level resulting from the Del_File( ) function unless *file_name* is blank, in which case it is set to _errorNoFile.

**RenFile old_file_name new_file_name**

This command renames the file *old_file_name* to *new_file_name*. All metacommands are translated.
The result code is set to the result of the Rename_File( ) function unless *old_file_name* and *new_file_name* are blank, and is then set to _errorNoFile.

**LoadFile file_name**

This command loads *file_name* into the editor. All metacommands are translated.
The result code is set to the Error_Level resulting from LdFiles macro unless *file_name* is blank, and is then set to _errorNoFile.

**Exec program { parameter }**

This command executes *program*, passing the *parameter* list on the command line. All metacommands are translated.
The following Set command affects the operation of the Exec command.

**Set WrkDir**

This sets the working directory that the ExecProg( ) macro changes to when executing the program.
The result code is set to the result of the ExecProg macro used to run *program* unless *program* is not specified, and is then set to _errorNoFile.

**RegDll dll_name**

This command registers the resources in the DLL *dll_name* with Multi-Edit.
The result code is set to _NoError unless *dll_name* is blank or the DLL file is not found, and is then set to _errorNoFile.

**Data Commands**

The following commands are used to add, delete, replace, and update data records in dB files. There are commands to operate on general data, and other commands that work with data of a specific type, such as CmdMaps, Toolbars, and Menus.

**DelPage page_name**

This command is used to delete the page header *page_name*, as well as all of the data contained under *page_name*. All metacommands are translated.
The result code is set to the result of DbDelPage( ) macro unless *page_name* is blank, and then it will be set to _errorNoDbPage

**Dat data_record**

This command is used to add, delete, replace, and update general data records in dB files.
There are a number of **Set** commands that need to be run before using this command.

The following **Set** commands affect this **Dat** command. Commands marked with * must, at a minimum, be run before a **Dat** command is processed.

- **Set File**: Sets the dB file the following Dat commands use.
- **Set Page**: Sets the page in the dB file the following Dat commands use.
- **Set Func**: Sets the operation the following Dat commands perform.
- **Set Create**: Specifies if the dB File or Page is to be created.
- **Set Dups**: Specifies whether duplicate data records are allowed.
- **Set Insert**: Specifies where data_record is inserted when added.
- **Set Dat**: Specifies a data record that the next Dat command tries to locate to insert data_record before or after.
- **Set RecNo**: Specifies the record number the next Dat command tries to delete or the record number data_record is inserted before or after.

---

**This command overrides Set Dat.**

- **Set Replace Dat**: Specifies the data record that the next Dat command tries to replace.
- **Set Replace RecNo**: Specifies the record number that the next Dat command tries to replace.
- **Set Update Dat**: Specifies the data record that the next Dat command tries to update.
- **Set Update RecNo**: Specifies the record number that the next Dat command tries to update.
- **Set Fields**: Specifies the fields that will be included or excluded on the next Dat command after a Set Replace Dat, Set Replace RecNo, Set Update Dat or Set Update RecNo command.

data_record is the complete data record that is to be added or deleted, or the data record used to replace or update a located record.

The result code is set to the result of the DbPutRecord( ) or DbDelRecord( ) macros.

**FirstMatch field_id**

This command is used to locate the first of multiple data records in dB files matching a regular expression in a specified field. If a record is found, the global string variable !UpdMatchStr is set to the data from the found record specified by the parameter field_id or the complete record if field_id is not specified. It also sets up some internal variables that the **NextMatch** command uses for locating the next matching record. Use the **Loop … EndLoop** statements with the **NextMatch** command to process all matching data records. The **CloseMatch** command should be used to clean up the internal variable data used by the **FirstMatch/NextMatch** commands before ending the **Loop**.

There are a number of **Set** commands that need to be run before using this command.

The following **Set** commands affect the **FirstMatch** command. Commands marked with * must, at a minimum, be run before a **FirstMatch** command.

- **Set File**: Sets the dB file the following FirstMatch command uses.
- **Set Page**: Sets the page in the dB file the following FirstMatch command uses.
- **Set Dat**: Specifies a regular expression the next FirstMatch command will try to find.
- **Set Fields I**: Specifies the field that the next FirstMatch command will search to try to find a pattern matching the regular expression specified by the Set Dat command.

field_id is the field identifier of the data from the found record that is returned in the global string variable, !UpdMatchStr.

(topic continues….)
The result code is set to _noError if at least one matching record was found, otherwise _errorNoDbRecord is returned.

Example:

```plaintext
;--- Add HTML commands ---
Set Page EXT=DB
Set Fields I  EXT=
Set Dat (HTM)|(ASP)|(CDF)|(JSP)
FirstMatch  EXT=
Loop
  IfRc == 0
    Set Page =~!UpdMatchStr>.PGM
    Set Create True
    Set Update Dat PN=HtmlTidy
    Dat PN=HtmlTidy CL=d:\Mew32\HtmlTidy -wrap 79 -ium
    <FILE>.<EXT> WORKLOC=1 PT=HtmlTidy EXEID=1 SHOW=0
  Else
    CloseMatch
    Break
  EndIf
NextMatch  EXT=
EndLoop
```

**NextMatch** field_id

This command is used to locate the next data record in dB files set up by the FirstMatch command. If a record is found, the global string variable !UpdMatchStr is set to the data from the found record specified by the parameter field_id, or the complete record is returned if field_id is not specified.

The data to match is set up by the FirstMatch command and must be run before using this command.

field_id is the field identifier of the data from the found record that is returned in the global string variable, !UpdMatchStr.

The result code is set to the value returned by the DbFindNext macro.

**CloseMatch**

This command is used to clean up the internal variable data the FirstMatch command sets up.

The result code is set to the value returned by the DbFindClose macro if a previous FirstMatch was run, otherwise it will return _errorNoDbRecord.

**Sec section_name**

This command is used to add, delete, replace, and update sections in command map files.

There are a number of Set commands that need to be run before using this command.

The following Set commands affect the Sec command. Commands marked with * must, at a minimum, be run before a Sec command.

- *Set File: Sets the dB file the following Sec commands use.

*Set File blank to use the user's default cmdmap file.*
- **Set Func**: Sets the operation the following Sec commands perform.
- **Set Insert**: Specifies where section_name is inserted when added.
- **Set Sec**: Specifies the section that the next Sec command tries to locate to insert the new section before or after.
- **Set Replace Sec**: Specifies the section that the next Sec command tries to replace.
- **Set Update Sec**: Specifies the section that the next Sec command tries to update.

section_record is the name of a command section to add or delete, or to use to replace or update the located section.

The result code is set to the results of the DbPutCmdSection( ) or DbDelCmdSection( ) macros.

### Cmd cmd_record

This command is used to add, delete, replace, and update CmdMap record entries in CmdMap dB files.

A number of **Set** commands must be run before this command will function properly.

Following is the list of **Set** commands that affect the **Cmd** command. Commands marked with * must, at a minimum, be run before a **Cmd** command is executed.

- *Set File*: Sets the dB file the following Cmd commands use.

- **Set Func**: Sets the operation the following Cmd commands perform.
- **Set Dups**: Specifies whether duplicate CmdMap records are allowed.
- **Set Insert**: Specifies where cmd_record is inserted when added.
- **Set Sec**: Specifies the CmdMap section under which the following Cmd commands locate and place CmdMap records.
- **Set Cmd**: Specifies the CmdMap record the next Cmd command tries to locate to insert cmd_record before or after.
- **Set Replace Cmd**: Specifies the CmdMap record that the next Cmd command tries to replace.
- **Set Update Cmd**: Specifies the CmdMap record that the next Cmd command tries to update.
- **Set Fields**: Specifies the fields to included or excluded on the next Cmd command after a Set Replace Cmd or Set Update Cmd command.

cmd_record is the complete CmdMap record to add or delete or the record to replace or update the located CmdMap record.

The result code is the result of running the DbPutCmd( ) or DbDelCmd( ) macros.

### Mnu menu_record

This command is used to add, delete, replace, and update menu entries under a specified menu page in CmdMap dB files.

A number of **Set** commands must be run before this command will function properly.
Following is the list of **Set** command that affect the **Mnu** command. Commands marked with * must, at a minimum, be run before a **Mnu** command is executed.

- ***Set File:** Sets the dB file the following Mnu commands use.

---

**Set File blank to use the user's default CmdMap file.**

- ***Set Page:** Sets the page in the dB file the following Mnu commands use, i.e. CONTEXT.MNU.
- **Set Func:** Sets the operation the following Mnu commands perform.
- **Set Dups:** Specifies whether duplicate Menu entries are allowed.

---

The separator command is always duplicated.

- **Set Insert:** Specifies where menu_record is inserted.
- **Set Mnu:** Specifies the menu entry that the next Mnu command tries to locate to insert menu_record before or after.
- **Set Replace Mnu:** Specifies the menu entry that the next Mnu command tries to replace.
- **Set Update Mnu:** Specifies the menu entry that the next Mnu command tries to update.

**menu_record** is the complete menu record to add or delete, used to replace or update the located menu record. The result code is the result of running the **DbPutMenu( )** or **DbDelMenu( )** macros.

---

**Tbx toolbox_record**

This command is used to add, delete, and replace/update toolbox records in CmdMap dB files and to set up the environment to allow the **Btn** command to add, delete, and replace/update buttons for the specified toolbox.

A number of **Set** commands can be used to alter this command.

The following **Set** commands affect the operation of the **Tbx** command. Commands marked with * must be run before this command will function properly.

- ***Set File:** Sets the dB file the following Tbx commands use.

---

**Set File blank to use the user's default CmdMap file.**

- **Set Func:** Sets the operation the following Tbx commands perform.
- **Set Dups:** Specifies whether duplicate toolboxes are allowed.
- **Set Insert:** Specifies where toolbox_record is inserted when added.
- **Set Tbx:** Specifies the toolbox entry that the next Tbx command tries to locate to insert toolbox_record before or after.
- **Set Replace Tbx:** Specifies the toolbox record that the next Tbx command tries to replace.
- **Set Update Tbx:** Specifies the toolbox record that the next Tbx command tries to update.
- **Set Fields:** Specifies the fields to included or excluded on the next Tbx command after a Set Replace Tbx or Set Update Tbx command.

**toolbox_record** is the complete toolbox record to add, delete, replace, or update the located toolbox record. The result code is the result of running the **DbPutToolBox( )** or **DbDelToolBox( )** macros.
**Btn button_record**

This command is used to add, delete, and replace/update toolbox button records in CmdMap dB files for a specified toolbox.

A number of **Set** commands can be used to alter this command.

The following **Set** commands affect the operation of the **Btn** command. Commands marked with * must be run before this command will function properly.

- **Set File:** Sets the dB file the following Btn commands use.
  - *Set Tbx:* Specifies the toolbox page under which the following Btn commands locate and place buttons.
- **Set Func:** Sets the operation the following Btn commands perform.
- **Set Dups:** Specifies whether duplicate buttons are allowed.
- **Set Insert:** Specifies where button_record is inserted when added.
- **Set Tbx:** Specifies the toolbox page under which the following Btn commands locate and place buttons.
- **Set Btn:** Specifies the button that the next Btn command tries to locate to insert button_record before or after.
- **Set BtnNo:** Specifies the button position number the next Btn command inserts button_record before or after.

- **Set Replace Btn:** Specifies the button the next Btn command tries to replace.
- **Set Replace BtnNo:** Specifies the button number the next Btn command tries to replace.
- **Set Update Btn:** Specifies the button the next Btn command tries to update.
- **Set Update BtnNo:** Specifies the button number the next Btn command tries to update.

button_record is the complete button record to add or delete, or to use to replace, or update the located button record.

The result code is the result of running the DbPutButton( ) or DbDelButton( ) macros.

**Set Commands**

The set commands are used to set up the operating environment for the data and a few of the general commands. There are quite a few set commands. Some affect most commands while others only affect a few commands. All set commands are discussed below.

**Set Log Off, On, Reset, Msg [text]**

The **Set Log** command is used to enable and disable the logging of messages to a log file. It can also be used to write messages to the screen and log file from script files.

Whether a log file is created and written to or not depends upon the state of the **Log** flag.

(topic continues…)
The following table shows the meaning of each state.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>Commands will not write progress and error messages to the log file.</td>
</tr>
<tr>
<td>On</td>
<td>Commands will write progress and error messages to the log file.</td>
</tr>
<tr>
<td>Reset</td>
<td>Will reset the Log status to what it was when the Update macro was first run. The initial Log status is set by the Install macro and is user configurable.</td>
</tr>
<tr>
<td>Msg text</td>
<td>This option will not affect the Log status but is used to write text to the status line and to the log file when logging is On.</td>
</tr>
</tbody>
</table>

When a log file is saved it is located in the user's Config directory and has the same name as the script file except with an extension of .LOG.

**Set NoMsgBox Off, On, Reset**

The **Set NoMsgBox** command is used to enable and disable the use of a modeless message box dialog to display the logging and error messages while the script is being executed.

Whether the message box dialog is used or not depends upon the state of the **NoMsgBox** flag. The following table shows the meaning of each state.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>The message box will not be used to display progress and error messages.</td>
</tr>
<tr>
<td>On</td>
<td>Progress and error messages will be shown in a modeless message box dialog.</td>
</tr>
<tr>
<td>Reset</td>
<td>Will reset the NoMsgBox status to what it was when the Update macro was first run. The initial NoMsgBox status is set by the Install macro and is user configurable.</td>
</tr>
</tbody>
</table>

**Set Verify Off, On, Reset**

The **Set Verify** command is used to enable and disable a verify dialog before a data command is executed asking if the user wants to execute the current data command.
Whether the verify dialog is shown or not, depends upon the state of the Verify flag. The following table shows the meaning of each state.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>The user will not be prompted before a data command is executed.</td>
</tr>
<tr>
<td>On</td>
<td>The user will be prompted to verify their desire to proceed with the current data command.</td>
</tr>
<tr>
<td>Reset</td>
<td>Will reset the Verify status to what it was when the Update macro was started. The initial Verify status is set by the Install macro and is user configurable.</td>
</tr>
</tbody>
</table>

**Set WrkDir [ work_directory ]**

The Set WrkDir command specifies the working directory that is used when the Exec command is run. The last directory set is used until another Set WrkDir is executed.

**Set EndMac macro_name { parameter } {; macro_name { parameter } }**

The Set EndMac command is used to specify a list of macros that are to be executed just before the Install macro exits. A global variable is set to contain the list of macros that the Install macro runs and then deletes when all macros have been executed.

This command should only be used in scripts that are run by the Install macro and will have no effect when running the Update macro alone.

**Set Create False ( Default ), True**

The Set Create command specifies whether a File or Page is created when it does not already exist. The File or Page will be created if it does not exist when Create is set True.

A Set Create must be done BEFORE a Set File for it to affect the creation of Files, and remains in the last set state until another Set Create command is run.

A Set Create must be done AFTER Set Page to have a page created when a data command is performed. This is needed because the Set Page command causes the flags used by the data commands to be set to the defaults (Create False).
**Set File [ file_name ]**

The **Set File** command loads the specified file (*file_name*) or the user's default WCmdMap file when *file_name* is blank, into a window. The file will be created if it does not exist when the "**Create**" option is set to True. If *file_name* contains no path information, then the user's path is used unless no user id is specified, and then the Multi-Edit directory is used.

Every time a new file is loaded, the previously loaded file is saved.

---

*A Set File file_name command should always be run before any data commands that make changes to files. The only exception is when the user's default WCmdMap file is being used.*

---

**Set Func Add ( Default ), Delete**

The **Set Func** command is use to specify what operation will be done when a data command is executed. This flag remains in the last set state until changed by another **Set Func** or a **Set Page** command is run which will reset it to the default (Add).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (Default)</td>
<td>Causes a data command to run the appropriate DbPutXxx macro in DbTools to add, replace or update the specified data. The actual operation that is performed, normally Add, is determined by flags set by the Set Replace or Set Update commands. When one of these commands is run the next data operation will be a Replace or Update but the following data operation will then revert back to Add unless another Set Replace/Update command is run.</td>
</tr>
<tr>
<td>Delete</td>
<td>The Delete state will cause a data command to run the appropriate DbDelXxx macro in DbTools to delete the specified data.</td>
</tr>
</tbody>
</table>

---

*The Set Update and Set Replace commands also sets the Func state to Add but sets flags to indicate the next data operation will be an Update or Replace.*

---

**Set Page [ db_page_name ]**

The **Set Page** command sets the page that the **Dat** and **Mnu** commands use when updating data or menus. This option will reset most of the data options to their defaults.
Set Insert After ( Default ), Before

The Set Insert command is used to specify where the data for data commands will be inserted into the file. This option will remain in the last set state until another Set Insert changes it or a Set Page is executed.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>After (Default)</td>
<td>Will cause the data to be inserted after the specified data record or at the end of the page when no data is found.</td>
</tr>
<tr>
<td>Before</td>
<td>Will cause the data to be inserted before a found data record or as the first entry in the page if no data record was found.</td>
</tr>
</tbody>
</table>

Set Dups False ( Default ), True

The Set Dups command determines whether duplicate records are allowed when a data command is run. This option will remain in the last set state until it is set by another Set Dups or a Set Page is executed, in which case it will be reset to the default (False).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>False (Default)</td>
<td>Will not allow a duplicate record to be added to the file when a data command is executed.</td>
</tr>
<tr>
<td>True</td>
<td>Will allow duplicate records to be added to the file when a data command is executed</td>
</tr>
</tbody>
</table>

Set CmdTrans Off ( Default ), On

The Set CmdTrans command determines whether the data commands will have the command line metacommands translated or not before the command is run. This option will remain in the last set state until it is set by another Set CmdTrans or a Set page is executed, in which case it will be reset to the default (Off) state.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off (Default)</td>
<td>Will cause the command line to be used exactly as it appears.</td>
</tr>
<tr>
<td>On</td>
<td>Will cause all metacommands in the command line to be translated before the data command is executed</td>
</tr>
</tbody>
</table>
**Set RecNo [record_no]**

The *Set RecNo* command sets the record number the next *Dat* command will use when searching for a data record. This value will only be used for the next *Dat* command and will be updated to the current record number after the *Dat* command is finished. This option should be used when you desire to replace or delete a specific record by number or specify a record number to insert a new record before or after.

The *Set Page* and *Set Func* commands causes this to be set to 0 which is also the same as doing a *Set RecNo* with no record_no specified.

**Set BtnNo [button_no]**

The *Set BtnNo* command is used to set the button position number the *Btn* command will use when searching for a button in a Toolbox. This value will be updated after every *Btn* command, thus causing the added buttons to be inserted before/after the previous button unless another *Set BtnNo* or *Set Btn* command is run.

The *Set Page*, *Set Func* and *Set Tbx* commands cause *BtnNo* to be set to 0. Also when button_no is not specified *BtnNo* will be set to 0.

**Set Dat find_data**

The *Set Dat* command is used to set the data record the following *Dat* command tries to locate to insert the new data record before or after depending upon the state of the *Insert* mode.

*find_data* contains a field name followed by the data for that field that is to be searched for, i.e. *NAME=Test*. Only one field and its data should be specified.

**Set Sec find_section_name**

The *Set Sec* command sets the CmdMap section. The following *Sec* command tries to locate and insert the new section before or after. It also sets the CmdMap section the following *Cmd* command places its command records under...

*find_section_name* is the text for the section name, i.e. File Operations.

**Set Cmd find_cmd**

The *Set Cmd* command sets the command record that the following *Cmd* command tries to locate to insert the new command record before or after.

*find_cmd* contains the field name and data to search for. There are three fields that can be specified. See the following page for the Command Description Table.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME=name</td>
<td>Used to locate the CmdMap record by name, where name is a text string specifying the command name, i.e. Delete Window.</td>
</tr>
<tr>
<td>KEY=keystr</td>
<td>Used to locate the CmdMap record by assigned key value, where keystr is a text string of the following form.</td>
</tr>
<tr>
<td></td>
<td>/KL=F3/K1=114/K2=1</td>
</tr>
<tr>
<td></td>
<td>Both primary and secondary keys will be searched.</td>
</tr>
<tr>
<td>WCMD=number</td>
<td>Used to locate the CmdMap record by WCmd number, where number is a WCmd number</td>
</tr>
</tbody>
</table>

### Set Mnu find_menu

The `Set Mnu` command sets the menu entry the next `Mnu` command tries to locate to insert a new menu entry before or after.

`find_menu` contains a string that specifies the entire menu entry. This string is made up of one of the following:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/WCMD=number</td>
<td>Specifies the WCmd number used to locate the menu by WCmd, where number is a WCmd number.</td>
</tr>
<tr>
<td>/IN=level</td>
<td>Specifies the indent level of the menu to locate, where level is the level number.</td>
</tr>
<tr>
<td></td>
<td>/#= entries only need to be specified up to # equal to level</td>
</tr>
<tr>
<td>/1=menu_str</td>
<td>Specifies the text of a level 1 menu entry, where menu_str is the text string, i.e. &amp;File.</td>
</tr>
<tr>
<td>/2=menu_str</td>
<td>Specifies the level 2 menu entry, where menu_str is the text of a menu entry, i.e. &amp;Open.</td>
</tr>
<tr>
<td>/3=menu_str</td>
<td>Specifies the text of the level 3 menu entry.</td>
</tr>
<tr>
<td>/4=menu_str</td>
<td>Specifies the text of the level 4 menu entry.</td>
</tr>
</tbody>
</table>

### Set Tbx find_toolbox

The `Set Tbx` command sets the toolbox entry the next `Tbx` command tries to locate to insert a new toolbox record before or after and it also sets the toolbox page the following `Btn` commands use to place their buttons under.

`find_toolbox` contains a string that specifies the toolbox name, i.e. Search. This string will be used by the `Btn` command to locate the page under which buttons are added, replaced, updated and removed.
Set Btn button_wcmd

The Set Btn command sets the button WCmd the next Btn command tries to insert a new button before or after. This option is overridden by the Set BtnNo so only one or the other should be used.

button_wcmd contains a number that represents the WCmd for that button to locate.

Set Replace Dat find_data, RecNo record_no, Sec find_section_name, Cmd find_cmd, Mnu find_menu, Tbx find_toolbox, Btn button_wcmd, BtnNo button_no

These commands set the appropriate values so the next command of the same type can replace the located item. See the appropriate Set Xxx section for the meaning of each parameter.

These set the replace flag and only affect the next command because the replace flag is cleared when the next command is run.

Set Update Dat find_data, RecNo record_no, Sec find_section_name, Cmd find_cmd, Mnu find_menu, Tbx find_toolbox, Btn button_wcmd, BtnNo button_no

These commands set the appropriate values so the next command of the same type can update the located item. See the appropriate Set Xxx section for the meaning of each parameter.

These set the update flag and only affect the next command because the update flag is cleared when the next command is run.

An update will replace the data if it was located but will add the new data when no data was located.

Set Fields I /F=field_name { field_name }, X /F=field_name { field_name }

The Set Field command sets the fields that are either included or excluded from a replace or update operation. This command should appear after a Set Replace or Set Update command but before the Xxx data command.

When no parameters are specified after the Set Field command, the next data operation will replace or update the whole record.

A parameter of "I /F=field_name { field_name }" causes only the fields specified by field_name in the located data record to be replaced with the data from the specified fields of the new data record, i.e. included.

A parameter of "X /F=field_name { field_name }" causes the fields specified by field_name in the located data record to NOT be replaced with the data from the specified fields of the new data record, i.e. excluded.

field_name is a list of space delimited fields names, i.e. NAME= KEY=.

Add-On Packages Installation Files

Installation Overview

Instead of having each Add-On package developer design and code their own installation and management macro, MESI has provided a standardized way of installing Add-On packages to Multi-Edit. This is done through a couple of entries in the Tools menu and a few script files that the Add-On developer creates. These script files are written in the script language defined in the Update Macro document. This script language is
powerful enough that we use it ourselves to update the command map file whenever we release a Multi-Edit update.

More information on setting up your own Add-On Packages or about the Update macro can be downloaded from our web site.

**Install.lst (Required)**

This file contains a list of information that the Install macro uses to install a package. It contains a single Multi-Edit db record that can contain the following fields:

---

*Please note that the 0x7F character is represented by the symbol below.*

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P=</td>
<td>The package name (Required)</td>
</tr>
<tr>
<td>V=</td>
<td>The version of the package (Required)</td>
</tr>
<tr>
<td>F=</td>
<td>A script file to copy files etc &quot;MyAddOn.dat&quot; (Required)</td>
</tr>
<tr>
<td>CF=</td>
<td>A script file to add package commands &quot;MyAddOn.upd&quot;</td>
</tr>
<tr>
<td>D=</td>
<td>The default install directory</td>
</tr>
<tr>
<td>R=</td>
<td>A readme file</td>
</tr>
<tr>
<td>U=</td>
<td>A script file to delete package &quot;MyAddOn.del&quot;</td>
</tr>
<tr>
<td>I=</td>
<td>A macro command to initialize the package upon startup and session switches</td>
</tr>
<tr>
<td>POST=</td>
<td>A macro command that is run after a session is restored.</td>
</tr>
<tr>
<td>EM=</td>
<td>A macro command that is run when the Enable button in the Manage Add-On Package dialog is selected.</td>
</tr>
<tr>
<td>DM=</td>
<td>A macro command that is run when the Disable button in the Manage Add-On Package dialog is selected.</td>
</tr>
<tr>
<td>DLL=</td>
<td>A .DLL file that could contain bitmaps etc</td>
</tr>
<tr>
<td>NDC=</td>
<td>No DLL Copy flag, 0-Copy DLL to Multi-Edit dir, 1-Do not copy</td>
</tr>
</tbody>
</table>
.dat file (Required)

This file contains an update script that copies any needed files into the proper directories. If an external installation program such as Wise Installation System or Install Shield is used, which installs all of the files into the correct directories, then this file would be blank.

*This file must always exist but can be blank.*

.upd file

This file contains an update script that adds data to the command map file allowing you to modify commands, key mapping, menus and toolbar information.

.del file

This file contains an update script that removes the Add-On package. It removes all of the data added to the configuration files by the MyAddOn.upd script. It can also remove all of the files added by the MyAddOn.dat script if an external install program was not used to install the package.

Integrating External Applications

DDE

DDE Server Name

The Multi-Edit server name = "Multi-Edit"

EXECUTE Command Syntax

*In the table below, the + sign denotes one or more occurrences of the item, * denotes zero or more occurrences of the item. Also the table continues on the following page.*

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute_string</td>
<td>command+</td>
<td>An execute request consists of one or more commands</td>
</tr>
<tr>
<td>command</td>
<td>[command_name args*]</td>
<td>Each command is enclosed in square brackets. A command may have an optional argument</td>
</tr>
<tr>
<td>Name</td>
<td>Definition</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>command_name</td>
<td>grchar+</td>
<td>A valid command name consists of one or more alphanumeric characters.</td>
</tr>
<tr>
<td>Grchar</td>
<td>[a-zA-Z0-9!#$%^&amp;()-_{}~]</td>
<td>The set of legal MS-DOS® file system characters.</td>
</tr>
<tr>
<td>Args</td>
<td>(arglist)</td>
<td>Command arguments are contained within parentheses.</td>
</tr>
<tr>
<td>Arglist</td>
<td>Arg or any arg, arglist</td>
<td>An argument list consists of zero or more arguments separated by commas.</td>
</tr>
<tr>
<td>Arg</td>
<td>&quot;string&quot; or string</td>
<td>An argument consists of a string or a string enclosed in double quotation marks. The argument may be empty.</td>
</tr>
<tr>
<td>String</td>
<td>grchar*</td>
<td>A series of valid string characters.</td>
</tr>
</tbody>
</table>

Examples

The following set of examples are all valid command strings:

[Quit]
[Pos(5,8)]
[Text("Insert Text")]

When the command has no arguments (the Quit example above), it is not valid to use command name with an empty set of parentheses. The following example shows a command that takes three arguments, and the second argument has been omitted:

[ThreeArgCommand("arg 1",,"arg 3")]

Commands may be concatenated to form a single execute-request string. The commands will be executed in the order they occur in the request. For example, the following sequence of commands might be used to make a selection and copy it to the Clipboard:

[Select(all)][Copy]

**Including Special Characters in String Arguments**

Characters that are not alphanumeric—quotation marks, tabs, and so on—can be embedded in the argument strings by using the backslash character (\) as an escape code. Quotation marks can also be included by inserting them as pairs. The backslash escape mechanism (as used in CMAC strings) gives complete flexibility. Turn the page for an example.
The examples below show the command string and resulting text.

### Examples of Embedding Special Characters in Argument Strings

<table>
<thead>
<tr>
<th>Argument</th>
<th>Resultant string</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;&quot;&quot;Woof&quot;&quot;&quot;, said the big dog.&quot;</td>
<td>&quot;Woof&quot;, said the big dog.</td>
</tr>
<tr>
<td>&quot;&quot;Woof&quot;&quot;, said the big dog.&quot;</td>
<td>&quot;Woof&quot;, said the big dog.</td>
</tr>
</tbody>
</table>

Because the escape characters are stripped from the strings, you cannot include the \0 (null) character in a string. Doing so effectively truncates the string.

### Returning Result Information

Returning result information from an Execute command is not provided by the DDE protocol, so Multi-Edit uses the standard used by Microsoft Word and Excel, the "Execute Control 1" Protocol. When a client wants to be able to retrieve result information from Multi-Edit, it tells the Multi-Edit that it would like to have this information saved in a named DDE item. It does this by sending a special execute command naming the Result item, followed by the command for which it wants to retrieve the result information. For example, let's say we are about to make a request to open a file and want to know if it succeeds or, if not, why it failed. To do this, we would send the following DDE execute command:

```
[Result(OpenResults)][Open(bogus.dat)]
```

The Result command tells Multi-Edit to create a special temporary DDE item, called OpenResults, under the current conversation. When the Open command is executed, its result information is copied to the OpenResults item. When the command request is complete, the client can ask Multi-Edit for the contents of the OpenResults item. The server will return the string returned by the Open command, and then delete the special temporary OpenResults item.

```
[Result]
-or-
[Result()]
```

If the name of the item given in the Result command is the same as an existing item for the topic, the existing item will be suspended while the Result item is active. When result reporting is turned off, the original item is reactivated. Suspending an existing item is a choice that the client makes. The client can enumerate the current list of items for the topic, so there is no reason for it to override an existing item without being aware that it is doing so.

Once a Result command is active, it remains active until the conversation is ended, a new Result command is issued or Result info is requested.

Whenever Multi-Edit needs to return data in response to an execute command, it concatenates the new data onto the end of any existing Result item data. Each item is terminated by a termination sequence, \r\n (carriage return, line feed).

Result strings are dependent on the function called.
Supported Execute Command Set

Example: [RunMacro("Macro_File^Macro_Name /Parm1=foo/Parm2=other foo")]

<table>
<thead>
<tr>
<th>Command</th>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunMacro</td>
<td>System</td>
<td>Takes one parameter, which is the name of the macro and any string parameters</td>
</tr>
</tbody>
</table>

Example: [RM('Macro_File^Macro_Name /Parm1=foo/Parm2=other foo ')]

<table>
<thead>
<tr>
<th>Command</th>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM</td>
<td>System</td>
<td>Takes one parameter, which is the name of the macro and any string parameters</td>
</tr>
</tbody>
</table>

Example: [PrepareExit]

<table>
<thead>
<tr>
<th>Command</th>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrepareExit</td>
<td>System</td>
<td>Does all necessary work to close Multi-Edit. However, does not close. This allows for checking if user stops the exit. Returns: a 1 if successful, 0 otherwise.</td>
</tr>
</tbody>
</table>

Example: [Quit]

<table>
<thead>
<tr>
<th>Command</th>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quit</td>
<td>System</td>
<td>Shuts down Multi-Edit.</td>
</tr>
</tbody>
</table>

Example: [LineCol(4,6)]

<table>
<thead>
<tr>
<th>Command</th>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineCol</td>
<td>Current Window</td>
<td>Sets current line and column. Accepts two parameters, where first is the line and second is the column.</td>
</tr>
</tbody>
</table>

Topics and Items Supported

<table>
<thead>
<tr>
<th>Topic</th>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentWindow</td>
<td>Name</td>
<td>Complete filename of the current window.</td>
</tr>
<tr>
<td>CurrentWindow</td>
<td>LineCol</td>
<td>Line and Column position of the current window.</td>
</tr>
</tbody>
</table>
COM

Multi-Edit 9.10 contains a **COM server** that will allow programs to control Multi-Edit via the COM interface. We use this interface for all of our IDE integration packages except for the Watcom integration, which still uses DDE.

When Multi-Edit 9.10 is installed, it registers its COM server type library with Windows so that other programs can access it. The following is the basic information needed to get started.

- **Type Library:** MEW32 Library (MEW32.tlb in MEW32.exe)
- **Interface:** Iapp
- **Methods:** Quit, RunMac, GlobalInt, GlobalStr

Once a Multi-Edit COM object is opened, any Multi-Edit global integer or string can be read or written via the GlobalInt and GlobalStr methods.

To do anything else the RunMacro command is used to run a Multi-Edit macro that can be user written or one of the system macros. We provide a set of macros in the file Remote.s that we use in our IDE integration Add-On's which allow most things to be done.

On the following pages is a sample Visual Basic Script that shows how to control Multi-Edit via a script. Additional *wrappers* can be found on the Multi-Edit website. This is the code that is used in most of our IDE integration packages.

```vbs
' ==================================================================
' Script to illustrate use of automating Multi-Edit
' ==================================================================
' Call the main subroutine
' ------------------------------------------------------------------
call Main( )
' ------------------------------------------------------------------
' We are finishing now
' ------------------------------------------------------------------
WScript.Quit( 0 )
' ------------------------------------------------------------------
' The main subroutine
' Sub Main
' ' Get an object of the params passed in to this script
' Set objArgs = Wscript.Arguments
' ' Check the arguments passed, only have 1 param in this script
' If objArgs.Count <> 1 Then
' WScript.Echo "Please pass only 1 param, for Multi-Edit to open"
' WScript.Quit ( 1 )
End If
```

(code example continued on next page…)
Create an instance of Multi-Edit automation server

Set objMEW32 = CreateObject( "MEW32.App" )

Open the specified

objMEW32.RunMac( "Remote^RemActivate" )
objMEW32.RunMac( "Remote^RemReloadFile /L=2/FLAGS=1/F=" + objArgs( 0 ) )
objMEW32.RunMac( "Remote^RemReloadFile /L=2/FLAGS=0/F=d:\src\scripts\Multi-EditObj.vbs" )
objMEW32.RunMac( "Remote^RemShowMsg Hello World!" )

Clear globals
objMEW32.GlobalInt( "!ComTest" ) = 0
objMEW32.GlobalStr( "!ComTestStr" ) = ""

Close Multi-Edit and Clean up
objMEW32.Quit
Set objMEW32 = Nothing

End Sub

End of script
External Application Integration

Borland Integration

Borland

The Borland IDE integration package allows you to switch back and forth between Multi-Edit and a Borland IDE, and have both environments reflect any editing changes. It also makes a Borland menu available in Multi-Edit, which allows you to launch Borland IDE commands (like Compile) directly from Multi-Edit. Currently Delphi 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 and C++Builder 1.0, 3.0, 4.0 and 5.0 are supported. For Delphi 2.0 or greater and all C++Builder versions, we allow you to synchronize IDE projects to Multi-Edit sessions.

Delphi 1.0

The Delphi 1.0 integration is much more primitive than the Delphi 2.0 (and above) and C++ Builder integrations. This is due to the very limited API that was exposed with Delphi 1.0. It works by saving all changed files whenever Delphi is deactivated and then, upon reactivation, checking for changes to the time/date stamps of those files and reloading the ones that have changed.

A special DLL expert (MEWEXPT.DLL) is installed in Delphi 1.0 that causes it to save files when task switching, and to check for files that need to be refreshed upon switching back. The DLL also lets you set a hotkey to switch quickly back and forth. If you use the hotkey, or the Tools | Multi-Edit menu entry (in the Delphi IDE) to switch from Delphi to Multi-Edit, then Multi-Edit will automatically go to the correct file and cursor position. This synchronization process allows full compatibility with the Delphi "two-way tool" concept.

The install process will modify the DELPHI.INI file to load MEWEXPT.DLL (the integration DLL) and create the Multi-Edit menu entry in the Delphi Tools menu.

You can configure the Delphi 1.0 autosave as follows:

**Off:** No auto saving when switching tasks. This means that you will have to MANUALLY save and reload your files.

**Whole Project:** Every file that has been changed will be saved when switching out of the Delphi IDE. This includes units, forms, and other non-unit files.
Current file only: Only the currently active unit will be saved. The exception to this is if the currently active unit is the project file, in which case the ENTIRE project will be saved.

No auto-saving will occur on a new file, i.e. a file that has never been saved to disk.

Limitations

Multi-Edit can synch its cursor position to Delphi's, but Delphi cannot synch its cursor to Multi-Edit. This is a limitation of the Delphi 1.0 IDE interface. In other words, when switching from Delphi to Multi-Edit, you can match Multi-Edit's cursor to Delphi's; however, when switching from Multi-Edit to Delphi, you cannot set Delphi's cursor the same as Multi-Edit's. This is only valid for Delphi 1.0.

Under Windows NT, if either Multi-Edit or Delphi is configured to run in "separate address space", then you MUST start Multi-Edit from within Delphi (via the tools menu), or start Delphi from within Multi-Edit. If you start them both from the program manager, then the integration will not work, as neither program will be able to see the existence of the other.

If you use the Delphi IDE to create a new event, Delphi automatically adds a function prototype to the appropriate unit. However, if you do not add anything to that prototype (in other words, you leave it blank), then it will be stripped when the IDE saves the changes to the file. This is an attempt by the IDE to be "smart". So far we have not found anyway around it except to just add something, even a single character, to the new function before switching to Multi-Edit.

Configuration Requirements

1. You MUST have "Save files when switching tasks" and "Reload when switching back" on in Multi-Edit. This is done from the Tools | Customize | Files | Backups | Temp files | Autosave dialog. You must also at least have "Determine changed file by" set to "Date/Time". See Tools | Customize | Files | Backups | Temp Files | Autosave dialog box for information on these settings.

2. You must have "Network file locking" Off!!! This is because both Multi-Edit and Delphi must have full read/write access to the source files. See Tools | Customize | Files | Backups | Temp Files | Autosave dialog box for information.

3. If you are running under Windows NT, and you have either Multi-Edit or Delphi configured to "Run in separate memory space", then you must start Multi-Edit from within Delphi, or start Delphi from within Multi-Edit. If you do not do this, than Multi-Edit and Delphi will not be able to communicate.

There are a few configuration requirements in order for the Delphi 1.0 integration to work:

For Delphi 1.0 integration to work:

You MUST have "Save files when switching tasks" and "Reload when switching back" ON in Multi-Edit. This is done from the Tools | Customize | Files | Backups | Temp files | Autosave dialog. You must also at least have "Determine changed file by" set to "Date/Time".

You MUST have "Network file locking" OFF!!! This is because both Multi-Edit and Delphi must have full read/write access to the source files.

If you are running under Windows NT, and you have either Multi-Edit or Delphi configured to "Run in separate memory space", then you MUST start Multi-Edit from within Delphi, or start Delphi from within Multi-Edit. If you do not do this, than Multi-Edit and Delphi will not be able to communicate.
**Delphi and C++ Builder**

The integration with **Borland's 32-bit IDEs** has become more sophisticated. Most of the limitations that exist with the 1.0 integration are gone. When switching back and forth between Multi-Edit and the IDE, the actual file is not modified on disk. Instead, a temp file is written, in which modifications are made. In Multi-Edit, the file you are editing will appear to have the same path and name as the file in the IDE, but it will actually be accessed through the temporary file.

> It is the IDE's responsibility to save the file, overwriting the original. If you have the **"Safety Autosave"** feature turned on (in Multi-Edit's Borland->Configure dialog), then all modified files will be saved every time you switch back to the IDE. If this option is not turned on, then you will have to save the files from the IDE (File->Save). Although this sounds a little scary ("Can I lose my files?"), there is actually very little risk. If the IDE were to crash before you saved the files, then the temp files will still exist (Multi-Edit will not remove them). The temp files are normally stored in the Windows TEMP directory, but you can configure this by adding a new key string to the following registry entry:  HKEY_CURRENT_USER\Software\Multi Edit Software\Multi-Edit\9.1\Addon\Borland

*All of the settings are stored in the Registry under the  "Software\Multi Edit Software\Multi-Edit\9.1\Addon" key.*

The install process will modify the Delphi registry entries to load a specific copy of MewEx32.dll and the C++Builder registry to load MewEx32C.DLL, and it will build a Multi-Edit menu entry in the IDE Tools menu.

**Limitations:**

Synching will not be active when debugging/running your application from the IDE.

**Configuration Requirements**

You **must** have "Network file locking" **OFF**! This is because both Multi-Edit and Delphi must have full read/write access to the source files.

> More detailed information about each Add-On package can be found in the txt file contained in the package subdirectory.

**Microsoft Integration**

**Microsoft**

What the **Microsoft IDE integration package** does is allow you to switch back and forth between Multi-Edit and a Microsoft IDE, and have both environments reflect any editing changes. It also makes available a Microsoft menu in Multi-Edit, which allows you to launch Microsoft IDE commands (like Compile) directly from Multi-Edit. Currently DevStudio 6.0 and Visual Basic 6.0 are supported. The rest of this document contains information specific to each of the supported Microsoft products.

> More detailed information about each Add-On package can be found in the txt file contained in the package subdirectory.
DevStudio 6.0

When installed, a Microsoft subdirectory under Multi-Edit is created and all of the needed files are copied there. A Microsoft menu is also added in the Add-On menu for switching to one of the IDEs.

All of the integration settings are being stored in the Registry under:

\[HKEY_CURRENT_USER\Software\Multi Edit Software\Multi-Edit\9.1\Addon\]

The specific Microsoft settings are stored under the 'Microsoft' subkey of the above and the DevStudio specific settings are stored under the subkey "Microsoft | Activate DevStudio" of the above key.

The install process will modify.dll, and will then create a Multi-Edit menu entry for the Microsoft tools.

The "Enable" option in the Multi-Edit "Microsoft | Configure" dialog just enables/disables the loading of the AddIn when DevStudio the DevStudio registry entries to install the MeDsSync.dll AddIn, register the server in the MeDsSync starts. To remove the AddIn from DevStudio and unregister the server, do the following:

1. Bring up the Manage Add-On Package dialog in Multi-Edit, Tools | Manage Add-On Packages...
2. Select Microsoft Integration in the package list.
3. Select the Disable button.

To reinstall and reregister the server, do the following:
1. Bring up the Manage Add-On Package dialog in Multi-Edit, Tools | Manage Add-On Packages...
2. Select Microsoft Integration in the package list.
3. Select the Enable button.

More detailed information about each Add-On package can be found in the txt file contained in the package subdirectory.

Visual Basic 6.0

Visual Basic is rather limited in what can be done with the built-in editor, therefore the only thing that our integration does is to save and reload files when switching from/to VBasic. The cursor position will not be tracked between switching.

All of the integration settings are being stored in the Registry under:

\[HKEY_CURRENT_USER\Software\Multi Edit Software\Multi-Edit\9.1\Addon\]

The specific Microsoft settings are stored under the 'Microsoft' subkey of the above and the VBasic specific settings are stored under the subkey "Microsoft | Activate VBasic" of the above key.

The install process will modify the VBasic registry entries to install the MeVbSync.dll AddIn and register the server in the MeVbSync.dll.
The "Enable" option in the Multi-Edit "Microsoft | Configure" dialog just enables/disables the loading of the AddIn when VBasic starts. To remove the AddIn from VBasic and unregister the server, do the following:
1. Bring up the Manage Add-On Package dialog in Multi-Edit, Tools | Manage Add-On Packages...
2. Select Microsoft Integration in the package list.
3. Select the Disable button.

To re-install and reregister the server, do the following:
1. Bring up the Manage Add-On Package dialog in Multi-Edit, Tools | Manage Add-On Packages...
2. Select Microsoft Integration in the package list.
3. Select the Enable button.

**Important requirements:**
You MUST have "Network file locking" OFF!!! This is because both Multi-Edit and DevStudio must have full read/write access to the source files.

More detailed information about each Add-On package can be found in the txt file contained in the package subdirectory.

---

**BradSoft**

**Top Style**

When the TopStyle Add-On is installed and enabled, it will allow Multi-Edit to use the TopStyle CSS editor for editing external style sheet files, `<style>...</style>` blocks and `<... style="...">` attributes from within HTML files. This Add-on requires a copy of TopStyle Lite to be installed on your computer, which is freely available from http://www.bradsoft.com/topstyle.

The TopStyle Lite program needs to be installed even if a registered copy of TopStyle Pro is also installed, since the editing of embedded style requires the Lite program. The Pro version will be used when editing style sheet files if it is installed.

When editing HTML files in Multi-Edit, right clicking on a tag will bring up a context menu that has entries to show a dialog to enter data specific to the tag under the cursor. When the TopStyle Add-On is enabled, it will add a few more entries for the appropriate tags.

Right clicking on a `<link ... rel="stylesheet" ...>` tag, will add an "Edit Linked Stylesheet..." entry to the context menu. When this entry is selected, Multi-Edit will launch the TopStyle editor with the specified style sheet file.

Right clicking on a `<style type="text/css">` tag will add an "Edit Style Block..." entry. Selecting this entry will cause Multi-Edit to block mark the text between the `<style ...></style>` tags, start TopStyle Lite and pass the block of text to it. When the "Done" button in TopStyle is selected, the edited text will replace the marked text in Multi-Edit. When TopStyle is exited without hitting the Done button, the text in Multi-Edit will not be changed. (continued....)
All tags that support the `<... style="...">` attribute will have a Style tab in the "Edit `<tag>` Tag..." dialog which contains a Style entry. Selecting the "..." button following the Style text field will launch TopStyle Lite, passing it the contexts of the Style text field. Hitting the "Done" button in TopStyle will cause the contexts of the Style text field to be replaced with the edited text.

More detailed information about each Add-On package can be found in the txt file contained in the package subdirectory.

---

**Scooter Software’s Beyond Compare**

With Multi-Edit 9.10 you can easily integrate a variety of external compare utilities. Beyond Compare, by Scooter Software is a powerful time-saving utility for comparing files and folders in your development projects. With the steps below you can install any compare utility, but for this example we will use Beyond Compare as our example.

The Beyond Compare demo or licensed copies can be configured as the compare utility within Multi-Edit by following the steps below:

1. Click on **Tools | Customize | File Compare**
2. Select the option “Use external diff program”
3. Click the “…” button to browse to the location of the BC2.exe file
4. Select the BC2.exe
5. Click on OPEN

The path to Beyond Compare should now be noted in your entry field, click on OK to exit the customizations dialog.

To invoke the file compare within Multi-Edit, click on **Text | Compare Files**, and select the file to be compared against the currently viewed file. Your file compare utility will then be loaded and pass the file to be compared.

More detailed information about each Add-On package can be found in the txt file contained in the package subdirectory.
AI Internet Solutions

CSE HTML Validator

**CSE HTML Validator integration** - Support for CSE HTML Validator has been added as a Multi-Edit Add-On package. HTML Validator Pro can be found at [http://www.htmlvalidator.com/](http://www.htmlvalidator.com/).

This package allows HTML files to be validated using a user-installed copy of CSE HTML Validator.

When installed, the support macros are copied into a CSEValidator subdirectory under Multi-Edit. Three new toolbar buttons are added to the "HTML Tools" toolbar, a compiler entry for HTML files is added, and an error processing record is added that will handle finding errors when validate is run.

The toolbar buttons are:

- **CseValidate**: Configure
  - This button brings up the CSE HTML Validator Configuration dialog.

- **CseValidate**: SetOptions
  - This button brings up the CSE HTML Validator Engine Options dialog.

- **CseValidate**: Validate
  - This button runs the CSE HTML Validator on the selected HTML file.

When the Validate button or the Validate compiler entry is selected, the Validator is run on the currently selected HTML file. All output from the validator will be shown in the Compiler Tools Pane and the normal error processing support is used to locate and display found errors.

> **HTML Validator Pro is required, as the Lite version will not work with Multi-Edit.**

More detailed information about each Add-On package can be found in the txt file contained in the package subdirectory.
Add-Ons

Polystyle

About Polystyle

Multi-Edit ships with a fully-integrated copy of the Polystyle source-code beautifier, which installs as an Add-On, giving you the ability to format source files directly within Multi-Edit. It understands a number of the languages Multi-Edit supports, with more being added from time to time. You can accept one of its predefined styles, modify one to your liking, or simply create your own from scratch.

This section covers the basics of installing the bundled copy of Polystyle and using it in Multi-Edit. For more specific usage information, see Polystyle's help, or visit its web site, http://www.polystyle.com.

Installing and Updating Polystyle

You install Polystyle during Multi-Edit installation, by simply choosing it from the checklist of options (see the Installation Guide for more on installation).

Since Polystyle is an actively-evolving product, you should check the Polystyle web site occasionally for new versions (http://www.polystyle.com). The version to download is the Windows Upgrade version (the full version will also work, but is larger to download). Unzip it directly into your Multi-Edit install directory (not the Polystyle subdirectory, or anywhere else). Obviously, you shouldn't be running Multi-Edit when you do this. If you've run Polystyle since booting your machine, you may have to reboot to get a successful unpack; in some instances a Polystyle-related shell DLL may still be running, making it impossible to overwrite.

Customizing Polystyle to Suit Your Coding Style

There are two kinds of customization information Polystyle uses: example files, which affect specific languages; and global preferences, which affect all languages Polystyle can handle.

(topic continues…)
Example Files

Example files are basically collections of "dummy" samples that tell Polystyle how to format a file of a given type. You can choose one of the supplied example file sets and simply use it as-is, or you can modify one of them to suit yourself by several methods. One thing to note is that Polystyle organizes collections of example files into "suites," which include not only a set of example files for each language, but a version of the global settings; if you don't ever create a new suite, any changes you make to example files affect the default suite that's created on installation. If you want different suites, create them.

If none of the supplied styles is exactly what you want, there are two methods by which you can modify the example files. First, Polystyle's Graphical Styler supplies a dialog-based interface for each language; by setting the dialog controls the way you want, you modify the example files without having to edit them. Second, you can open example files and edit them directly. While you can achieve some finer control this way, it's obviously a method for experts. If you do it this way, there are some things to observe.

If you've created new suites, they'll appear in separate, named directories beneath the main example directory; be sure you know which file you're editing.

You shouldn't ever edit any of the supplied styles. Within example files, comments are null, and affect nothing; all that matters is spacing, both inserted whitespace and blank lines. That's all you should edit. If you want to experiment, be sure to do it on a copy, and run test-formats on unimportant files.

Global Preferences

The global preferences are simply a group of settings that are applied to all languages in a suite, such as tab settings and margins. Where you've selected a suite based on a particular tab length, the global tab length must be the same.

Formatting Files

To format a file, open it in Multi-Edit and either select Tools | Format Code from the main menu or press Alt+Shift+F (in the default keymap). Upon successful completion, your file will be modified in place and saved, and a backup created in the same location with ".polystyle.backup" appended to its filename.

If for any reason Polystyle is unable to parse your file, it will display a dialog showing a portion of the code around the area it was unable to parse, with a dashed line and a caret beneath the problem line to show exactly where it hit the problem, as well as the line and column number. As with any language parser, you should be aware that the actual cause of the misparse may be earlier in the file, as Polystyle will continue to parse so long as some interpretation is possible.

Fix the problem code and rerun Polystyle; as with all debugging, you may have to iterate a few times to get it right. It's not billed as a feature, but you can actually use Polystyle as a simple debugger in this fashion.

Getting Support for Polystyle

Polystyle is a third-party product, written and owned by Flashbulb Studios, so you should direct any support issues other than ones involving Multi-Edit's Polystyle integration to their support staff (http://www.polystyle.com).

Direct questions concerning the Multi-Edit integration (i.e., "Nothing happens when I select Format Code," "Polystyle says it's a trial copy," etc.) to our team at support@multieditsoftware.com.
Evolve

Introduction

EVOLVE is a wrap-around shell for Multi-Edit that works with the XBase language you program in. It watches as you work, adjusting itself to your style where appropriate. EVOLVE automates many of the tedious programming chores that take up so much of your time. Frequently used source code statements are automatically expanded; source files are located for you without hesitation; database structures are viewed with the touch of a key.

Language Dialect Support

Evolve supports many XBase as well as a few non-XBase dialects that have been diverging from the original roots of dBASE since the mid 1980's. Much of the dialect specific support contained in Evolve's macros are identical between XBase languages. However, there are instances where it is necessary to make certain exceptions and these are generally noted in this help guide. The following dialects are supported in Evolve. Additional [and more detailed] information can be found in the following topics: Installation, Configuration, and Passive Productivity and Template Editing.

<table>
<thead>
<tr>
<th>Language Style</th>
<th>Dialect</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>Clipper Summer ’87/5.0 and later.</td>
</tr>
<tr>
<td>ARA.</td>
<td>Arago For DOS</td>
</tr>
<tr>
<td>DBA.</td>
<td>dBASE III Plus and IV</td>
</tr>
<tr>
<td>DBF.</td>
<td>dBASE V For DOS</td>
</tr>
<tr>
<td>DBW.</td>
<td>dBASE For Windows</td>
</tr>
<tr>
<td>FXP.</td>
<td>FoxPro For DOS</td>
</tr>
<tr>
<td>FXW.</td>
<td>FoxPro For Windows</td>
</tr>
<tr>
<td>VIS.</td>
<td>CA-Visual Objects</td>
</tr>
<tr>
<td>CAF.</td>
<td>dBFast For Windows</td>
</tr>
</tbody>
</table>
**Configuration**

The primary configuration of Evolve includes the main configuration dialog screen accessed by selecting the Configuration tool bar option, selecting Configuration from the menu or by running the EVCONFIG macro. Since Evolve is highly dependent on Multi-Edit, it's a good idea to totally familiarize yourself with installation and set up issues as well. Other areas that may help you achieve more productivity gains can be found in the manipulation of preprocessor constants located in EVOLVE.SH. While it is likely that all the set up and configuration options in Evolve's configuration dialog will handle most of the customization desired, you may want to browse through EVOLVE.SH to see if there are other items that could be modified to help improve your development productivity.

**Multi-Edit's Language Style**

Within the **Filename Extension Setup** dialog, Multi-Edit allows you to attach "language styles". The language style provides configuration fields that enable you to enter keywords, comment styles, and punctuation marks for syntax highlighting purposes. It is important to Evolve that a language style be selected if (and only if) you want Multi-Edit to highlight syntax key words for you. In this case, the language style in Multi-Edit need not be configured with key words. Evolve itself will handle this for you automatically and will adjust the keywords based upon the dialect you select in the Evolve configuration dialog screen. This will make it possible to see key words such as SCAN...ENDSCAN highlighted when the dBASE dialect is selected, and commands such as WHILE...END highlighted when Clipper is selected. Evolve will create and select the proper language styles depending on the dialect you configure Evolve for.

<table>
<thead>
<tr>
<th>Language Style</th>
<th>Dialect</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI.</td>
<td>CA-Clipper</td>
</tr>
<tr>
<td>ARA.</td>
<td>Arago For DOS</td>
</tr>
<tr>
<td>DBA.</td>
<td>dBASE III Plus and IV</td>
</tr>
<tr>
<td>DBF.</td>
<td>dBASE V For DOS</td>
</tr>
<tr>
<td>DBW.</td>
<td>dBASE For Windows</td>
</tr>
<tr>
<td>FXP.</td>
<td>FoxPro For DOS</td>
</tr>
<tr>
<td>FXW.</td>
<td>FoxPro For Windows</td>
</tr>
<tr>
<td>VIS.</td>
<td>CA-Visual Objects</td>
</tr>
<tr>
<td>CAF.</td>
<td>CA-dBFast For Windows</td>
</tr>
<tr>
<td>FIV.</td>
<td>FiveWin</td>
</tr>
</tbody>
</table>

If you desire your own syntax highlighting scheme, you may use any of the above listed language types for custom syntax highlighting configured with your own list of preferred keywords and comment styles. Evolve will automatically add language styles that don't exist and will also fill in the reserved words and comment
characters appropriate for the dialect. Once a language style has been added by Evolve, you may then customize the reserved words and other information for the language style using Multi-Edit's installation menu. Evolve will not write over your changes to language styles.

If you ever have the need to reset an Evolve-based language style back to the default settings, simply use Multi-Edit's Languages dialog and delete the style. The next time that style is called for by Evolve, it will be re-created automatically.

Setup

Since Multi-Edit includes a complete file extension support system, Evolve has been designed to embrace these features and depend on the resources that provide support for syntax color highlighting and other resident capabilities. The ability to enable Evolve's specialized features only while editing file extensions you identify is founded in the setup and configuration capabilities of Multi-Edit.

Evolve Is Private To Each Window

Evolve utilizes Multi-Edit's internal file extension configuration so that its features are active only in editing windows that have file extensions configured in Multi-Edit. Because of this, it is necessary to identify all of the file extensions you intend to edit with Evolve capabilities, under one extension entry (e.g., for example - PRG;WFM, etc). This is done in Multi-Edit's Tools | Customize | Filename Extensions menu.

Multi-Edit's Language Style

Within the filename extension setup, Multi-Edit allows you to attach "language styles". The language style provides configuration fields that enable you to enter keywords, comment styles, and punctuation marks for syntax highlighting purposes. Evolve requires that a language style be selected if (and only if) you want Multi-Edit to highlight syntax key words for you and set the assignment operator [e.g., the equals sign]. In this case, the language style in Multi-Edit need not be configured with key words. Evolve itself will handle this for you automatically and will adjust the keywords based upon the dialect you select in the Evolve configuration dialog screen. This will make it possible to see key words such as SCAN...ENDSCAN highlighted when the dBASE dialect is selected, and commands such as WHILE...END highlighted when Clipper is selected. Evolve will create and select the proper language styles depending on the dialect you select in Evolve's configuration screen.

Please note this table continues on the following page...

<table>
<thead>
<tr>
<th>Language Style</th>
<th>Dialect</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI.</td>
<td>Clipper Summer '87/5.0 and later.</td>
</tr>
<tr>
<td>ARA.</td>
<td>Arago For DOS</td>
</tr>
<tr>
<td>DBA.</td>
<td>dBASE III Plus and IV</td>
</tr>
<tr>
<td>DBF.</td>
<td>dBASE V For DOS</td>
</tr>
<tr>
<td>DBW.</td>
<td>dBASE For Windows</td>
</tr>
<tr>
<td>FXP.</td>
<td>FoxPro For DOS</td>
</tr>
</tbody>
</table>
If you desire your own syntax highlighting scheme, you may use any of the above listed language types for custom syntax highlighting configured with your own list of preferred keywords and comment styles. Evolve will automatically add language styles that don't exist and will also fill in the reserved words and comment characters appropriate for the dialect. Once a language style has been added by Evolve, you may then customize the reserved words and other information for the language style using Multi-Edit's installation menu. Evolve will not write over your changes once these have been customized.

**EV.CFG Configuration File**

The EV.CFG file is one of two Evolve configuration files and its purpose is to store and maintain commenting templates and source code report formats. It contains various elements that provide customization to suit your particular commenting and reporting style. Important to the goal of Evolve is that there may be multiple EV.CFG files. Perhaps a different one for each directory you write applications in. Multiple CFG files provide the ability for you to have project sensitive configurations for comment headers, copyright notices, and other information that may vary from project to project.

How Evolve Finds The Right EV.CFG File

When Evolve looks in the current directory for EV.CFG, it looks to see if EV.CFG exists and if not found it does a second scan for EVxxx.CFG (where xxx is the multi-user network ID) also in the current directory. The second scan enables you to create multiple CFG files in the same directory that are dependent on the network user ID for utilization.

The EV.CFG file is installed, by default, to the ME directory where it is used by Evolve as the default CFG file. If you want different CFG formats (e.g., perhaps different default application titles for program headers), you need to make a copy of EV.CFG in the directory where you are working on a specific project. Evolve will use the EV.CFG file found in the current directory first.

Network users will find that as a last resort, Evolve looks for the default EV.CFG file in the USER_ID path before looking to the ME path. This allows you to have different default CFG files for each user on a network in addition to different directory level configurations for each user.

A sample of the format of the CFG file is shown below and may be configured to support your own style of header comment blocks for programs, procedures, and functions. Additionally, the format used for source code printing is established in this file. The CFG File is a standard text file so it can be easily modified at any time with Multi-Edit. The design of Evolve permits modification to this file during an edit session. All changes to this file will become active in Evolve immediately. See the default EV.CFG file that is shipped with Evolve for more header possibilities.

A Sample EV.CFG is shown on the following page…
Sample EV.CFG

[Author] := John Doe
[Copyright] := Multi Edit Software
[AppTitle] := Billing Information System
[Cursor] := {Entry}

[Program Header]
Program
Application: {AppTitle}
File Name: {File}
Author: {Author}
Date created: {DateCreated} Date updated: {DateUpdated}
Time created: {TimeCreated} Time updated: {TimeUpdated}
Copyright: {Yr} by {Copyright}

[Include Header]
Program
Application: {AppTitle}
Description: {Entry}
File Name: {File}
Author: {Author}
Date created: {DateCreated} Date updated: {DateUpdated}
Time created: {TimeCreated} Time updated: {TimeUpdated}
Copyright: {Yr} by {Copyright}

[Procedure Header]
Procedure
Description: {Entry}
Author: {Author}
Date created: {DateCreated} Date updated: {DateUpdated}
Time created: {TimeCreated} Time updated: {TimeUpdated}
Copyright: {Copyright}
Arguments: {Argument}
See Also: {Field}

[Function Header]
Function
Description: {Entry}
Author: {Author}
Date created: {DateCreated} Date updated: {DateUpdated}
Time created: {TimeCreated} Time updated: {TimeUpdated}
Copyright: {Copyright}
Arguments: {Argument}
Return Value: {ReturnValue}
See Also: {Field}

[Print Format]
{h} := Application: {AppTitle}  Page: {Page}
{h} := Source File: {FileName}  Print Date: {DateCreated}
{h} := ----------------------------------------------------------
{h} := Line #  Source
{h} := ----------------------------------------------------------
{b} := {Ln#} - {Source}
{f} := Author: {Author}  Copyright: {Copyright}
[eof]
In addition to configuration elements available in the Evolve configuration dialog, there are quite a few "hard coded" preprocessor constants defined in EVOLVE.SH. It is rare that these variables will need to be changed, but they're documented should the need arise.

EVOLVE.SH can be found in the EVOLVE directory and additional details describing changes to this file are discussed in the topic titled Primary Manifest Constants.

Configuration of Evolve is simple and accomplished completely via dialog screens. Much of the control and behavior in Evolve is accomplished through global variables, which are defined in EV_.S, EVOLVE.S, and some are contained in the dialect driver files [such as ev_clirpr.s]. Most of the variables may be changed through the Evolve Configuration dialog.

The configuration dialog box can be accessed from the Evolve menu, tool bar, or by using the EVCONFIG command. The variables that are controlled from this dialog provide configuration elements such as the type of comment characters used, the tab level of same-line-comments, certain coding styles, and template expansion controls. The configuration system of Evolve will change the global variables and recompile EV_.S based on the selections made in this dialog.

Configuration support through the EVCONFIG dialog box handles both default and current session changes. Using the OK button applies the changes only to the current session. The Save Configuration button changes the current session and establishes the changes as the start-up default settings.

The configuration format of Evolve enables you to have different configurations for different users on a network. The single-user mode will use the EV_ macro as Evolve's source of configuration for both the key and the global variables that affect the way Evolve behaves. In the case of multi-user environments, the EV_ macro will be replaced with EV_ plus the 3 character User ID. In either case, Evolve will expect the configuration macro to be in the <ME_PATH> directory.

There are a number of primary manifest constants in EVOLVE.SH that you may want to alter to achieve custom performance. Most of these items have to do with areas of Evolve that will generally conform to your needs as shipped. However, tools work best if they work for you, so they have been documented and arranged for easy modification.

The following topics provide additional information concerning modification and performance tuning of Evolve.

In addition to configuration elements available from the Evolve configuration dialog, there are quite a few "hard coded" preprocessor constants defined in EVOLVE.SH. It is rare that you will have to alter the values in these constants, but they're documented in the header file should the need arise. For the most part EVOLVE.SH contains default configuration settings that apply to most XBase development situations. However, we recognize the importance of adaptation [by the tool rather than the programmer] when
situations vary. As a result, EVOLVE.SH has been created with various constants so that changes can be made to the behavior of the product without significant understanding of the macro programming language or compiler.

To make a manual configuration change to Evolve, simply edit the EVOLVE.SH file and make the desired modifications to the preprocessor constants. The constants are fairly well commented in the header file, and experimentation with various settings is encouraged. Once the header file has been modified and saved to disk, you will need to recompile the Evolve macros using the EVOLVE.LST recompile file. This file can be found in the Evolve directory and is easily attached to a Windows icon as shown below.

cmacwin.exe -Ievolve;src -Pmac -L evolve\evolve.lst

Use of this recompilation specification is important if you change any of the manifest constants in EVOLVE.SH.

**Overview Of EVOLVE.SH**

While most of the constants in EVOLVE.SH are self explanatory and well commented, the following items are a bit more complex and have been documented here with additional information.

Some Features May Be Public

Since Evolve is window-dependent with regard to its activeness, we have provided a few constants in EVOLVE.SH that allow you to enable (or disable) some of the popular Evolve key assignments on a public basis (e.g., all windows regardless of file extension). These include the following constants and key assignments which are enabled [by default] in EVOLVE.SH. If you don't want the associated key assignments for these features to be active in all windows regardless of file extension, change the values to zero [0] and recompile the macros.

<table>
<thead>
<tr>
<th>Content</th>
<th>Key Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVc_PUBLICASSIST</td>
<td>Main menu assignment</td>
</tr>
<tr>
<td>EVc_PUBLICQUOTEEXP</td>
<td>Quotation keys expansion</td>
</tr>
<tr>
<td>EVc_PUBLICSCANNER</td>
<td>Scan keys assignment</td>
</tr>
<tr>
<td>EVc_PUBLICCOMMENT</td>
<td>Commenting keys assignment</td>
</tr>
<tr>
<td>EVc_PUBLICALIGN</td>
<td>Alignment keys assignment</td>
</tr>
</tbody>
</table>

**Performance Tuning**

There are a number of configuration constants in EVOLVE.SH that you may want to change, although it is unlikely. Most of these items have to do with areas of Evolve that will generally conform to your needs as shipped. However, tools work best if they work for you, so they have been documented and arranged in a way that makes it possible to achieve further customization.

To make a manual configuration change to Evolve, simply edit the EVOLVE.SH file and make the desired modifications to the preprocessor constants. Once changed and saved to disk, you will need to recompile the
Evolve system using EVUPDATE.BAT. This file can be found in the Evolve directory. EVOLVE.SH includes many different types of constants that control various elements.

If you are going to modify Evolve at this level, please restrict changes to the constants listed in each submenu below. Modifying other constants in EVOLVE.SH can sometimes cause problems.

**EVc_GENMEMCASE**

This manifest constant, when set to zero [0] (the default setting), causes all generated memory variable names derived from field names in a database structure to be generated in proper case format (e.g., cAddress). Other settings for this constant include 1 (all lower case variable names), and 2 (all upper case variable name).

**EVc_ENDTEXT**

This variable enables the indenting systems to ignore TEXT...ENDTEXT constructs (e.g., text within the construct will never be indented). This constant is set to one [1] by default (ignores the construct). This configuration integer affects real-time and ad hoc indenting features.

**EVc_DBFFUNCTIONS**

A manifest constant named EVc_DBFFUNCTIONS is included in EVOLVE.SH that enables the addition of third party library function name specifications for purpose of database structure display by pointing and clicking on the function call. This constant simply contains a list of the functions and the number of the argument that holds the file name that you use to open or close databases. Default values for the new constants are shown below.

```
EVc_DBFFUNCTIONS  ' DBUSEAREA(3) NET_USE(1) NETUSE(1) ' 
```

**EVc_NDXFUNCTIONS**

A manifest constant named EVc_NDXFUNCTIONS is included in EVOLVE.SH that enables the addition of third party library function name specifications for purpose of index structure display by pointing and clicking on the function call. This constant simply contains a list of the functions and the number of the argument that holds the file name that you use to open or close your index files. Default values for the new constants are shown below.

```
EVc_NDXFUNCTIONS  ' DBSETINDEX(1) ' 
```

**EVc_PARENSPACES and EVc_PARENCURSOR**

Preprocessor constants exist in EVOLVE.SH that allow you to configure the number of spaces inserted in expanded braces such as parens (), and brackets []. The constants EVc_PARENSPACES, and EVc_PARENCURSOR allow you to control the spaces and cursor position following the expansion of the matching character. Values and the generated results of each value are shown below for configuration of the parenthesis expansion constants.
Preprocessor constants are provided for brackets [] and curly braces {} and behave in the same manner as that shown for the parenthesis.

<table>
<thead>
<tr>
<th>EVc_PARENSPACES</th>
<th>EVc_PARENCURSOR</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>()</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>(_</td>
</tr>
<tr>
<td>2 [default]</td>
<td>2</td>
<td>( _</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>( _ )</td>
</tr>
</tbody>
</table>

Note: The underscore character indicates the position of the cursor following insertion of the closing brace.

The position of the cursor following the expansion of the closing brace is determined by a second preprocessor constant named EVc_PARENCURSOR. There are also a set of duplicate constants in the header that pertain to bracket "[]" expansion, however, these constants are set to zero by default.

**EVC_RETURNSPACE**

This manifest constant provides you with a mechanism to pad additional spaces following the automatic creation of RETURN statements and default return arguments (if any) by the template expansion system. By default, this constant is set to a single space thereby producing "RETURN (...)". If you remove the space in the #DEFINE statement for EVc_RETURNSPACE (and recompile with EVUPDATE.BAT), the code will be generated without the spaces [i.e., RETURN(...)].

**EVC_QTYDOCASE and EVC_QTYELSEIF**

One of the great aspects of Evolve is it's ability to be configured to your liking. There are numerous manifest constants contained in EVOLVE.SH, each one providing specific configuration and customization possibilities. One such constant named EVc_QTYDOCASE controls the number of CASE statements generated when you type DOC<space>.

By default, the number is 6 which generates a structure like this:

```
DO CASE
CASE
CASE
CASE
CASE
CASE
OTHERWISE
ENDCASE
```

If you would like more (or fewer) CASE statements you simply need to change the value of EVc_QTYDOCASE to the number of CASE statements desired. Once the value has been changed in the Evolve header file [EVOLVE.SH], you'll need to recompile EV_EXPND.S, or you can simply run EVUPDATE.BAT file. This concept also applies to the quantity of ELSEIF statements generated from the IFI template expansion. The manifest constant that controls the quantity generated is EVc_QTYELSEIF.
Altering The Behavior Of The Dialect Driver Macro

In each dialect driver macro, you'll find the supported language definitions stored to a series of global variables. The dialect number, the default index file extension, and other information specific to a given XBase language, are all based on manifest constants in EVOLVE.SH. An example dialect driver for Clipper is shown below.

```
Dialect Driver Constants
// -----------------------------------------------------------------------
// Dialect Specific Constants
// -----------------------------------------------------------------------
// CA-Clipper/DOS Support
#define EVC_CLIPR_INDEXEXT 'NTX' // default index file extension
#define EVC_CLIPR_DATATYPES 'abcnldmox' // default notation data types
#define EVC_CLIPR_OBJSEARCH '{% *}|{{|9}{static proc}}||{proc}||{static func}||{func}||{static meth}||{|meth}??'
#define EVC_CLIPR_RESERVED ''
#define EVC_CLIPR_EOLCOMMENT '//' // default eol comment style
#define EVC_CLIPR_HEADEREXT '.CH' // default header file extension
#define EVC_CLIPR_OPERATOR ':=' // variable assignment operator
```

These constants exist for each dialect supported by Evolve and they represent an easy way to make subtle changes that are language specific within the supported file extensions of Evolve. The names are self explanatory as well as their purpose. However, caution should be exercised when making modifications to the OBJSEARCH constant. This symbol contains very specific search expressions which could drastically alter the behavior of Evolve. A thorough understanding of Multi-Edit's regular expression syntax is recommended before altering this constant.

Template Expansion API

Introduction

Of the most productive features of Multi-Edit and Evolve, template expansion, is used most by programmers. Template expansion makes it possible to repetitively generate source fragments by typing just a few characters.

To make the template expansion system work better for you, a new macro file was added to release 6.1 named EV_MPINT.S. It provides a series of programming interface macros that cater mostly to the aspect of template expansion customization and addition of new expansion keywords. The macros for template expansion support include mpiInit() and mpiTemplate().

Template Expansion Process

When Evolve is first loaded, the mpiInit() macro is launched, which simply runs mpiTemplate() with an argument of /I [initialization call]. Thereafter, mpiTemplate() is called every time you press the space bar following the entry of the first word [or character sequence] on a new line. Because this macro is called every time a new "first word" is entered on a line, it provides for a perfect API to the template system.

This mpiTemplate() macro is called from the template expansion macro to test for custom templates that you may have implemented in EV_MPINT.S. As an instructional aid, it includes a sample template named ABC that expands to a code construct in similar fashion to a DO WHILE...ENDDO structure. It demonstrates the ease of creating and integrating a custom template into Evolve's space-triggered template expansion environment.
Adding A New Template

To add a new template to Evolve, you needn't make any modifications to EV_EXPND.S or any other macro file. Simply duplicate one of the template code segments shown below and modify it to meet the desired results. Note that the characters of the template need to be listed in the string variable named "cTemplates" at the top of the mpiTemplate macro.

Step One: Create The Triggering Character Sequence

The macro code fragment shown below is the first statement in the mpiTemplate() macro. It simply defines a string variable and makes an assignment of a character sequence named ABC. This is intended as an example and may be removed or replaced if desired. The first step in adding a new custom template is to add the triggering character sequence to the string variable cGenText.

Triggering Character Sequences Before New Addition

```
str  cGenText,
cTemplates = 'ABC ';
```

Suppose we want to add a new trigger called MYFUN. The string variable would be modified as shown below. Note the delimiting spaces within the text string.

Triggering Character Sequences After New Addition

```
str  cGenText,
cTemplates = 'ABC MYFUN ';
```

Step Two: Create Code Generation Segment

In this step, we duplicate [or manually enter] a new ELSE IF segment of the mpiTemplate() macro to effectively look for, and respond to the template sequence [in this case, MYFUN<space>]. The ABC example is useful as a starting point, but note how [and why] it differs. In this particular example, we are adding an option that generates a function call, rather than a looping construct as the ABC example provides.

New Macro Segment To Generate Code

```
// MYFUN... ( myFunction(arg1, arg2, arg3) )
} else if( caps(global_str('!EVgPrevWord')) == 'MYFUN ' ) {
    return_int = 1;
    goto_col(global_int('!EVgCurCol') - length(global_str('!EVgPrevWord')));
    del_chars(length(global_str('!EVgPrevWord')) - 1);
    cGenText = 'myFunction(arg1, arg2, arg3)';
    call WriteLine;
    eol;
```

The addition of this new ELSE IF segment results in the generation of the following code whenever you type MYFUN<space> at the beginning of a new line. Furthermore, the original key phrase [myfun] is replaced with the generated code.

```
myFunction(arg1, arg2, arg3)
```
**Step 3: Recompile EV_MPINT.S**

Following any changes to a macro file, the macro must be recompiled before they become activated. From the menu, or from the keyboard, recompile EVMPINT.S, making certain that there are no compiler errors [they'll appear on the message line, depending on the setup attributes of Multi-Edit].

**Step 4: Test The New Template Feature**

While editing in a buffer window that is supported by Evolve [PRG is most preferred], try typing the new triggering sequence and then press the space bar.

**Important Aspects Of The Macro Code**

A few of the source code lines in the ELSE IF segment are noteworthy and a thorough understanding of them will help you create superior and custom support for your development environment.

Indicate Successful Expansion - First and foremost, the return_int = 1 statement is important because it informs Evolve's template expansion system that an expansion has taken place in the API, therefore no further search of the internal expansion possibilities are necessary. The result is faster processing of the space character.

**Indication Of Successful Expansion**

```c
// MYFUN... ( myFunction(arg1, arg2, arg3) )
} else if( caps(global_str('!EVgPrevWord')) == 'MYFUN ' ) {
    return_int = 1;
    goto_col(global_int('!EVgCurCol') - length(global_str('!EVgPrevWord')));
    del_chars(length(global_str('!EVgPrevWord')) - 1);
    cGenText = 'myFunction(arg1, arg2, arg3)';
    call WriteLine;
    eol;
```

**Generating The Template Code** - Another interesting macro code fragment are the lines responsible for generating code into your program file. First, the code to be generated is assigned to a string variable named cGenText. Then, the WriteLine procedure is called, which takes the pre-assigned memory variable and outputs its contents to the program file buffer.

**Generating The Template Code**

```c
// MYFUN... ( myFunction(arg1, arg2, arg3) )
} else if( caps(global_str('!EVgPrevWord')) == 'MYFUN ' ) {
    return_int = 1;
    goto_col(global_int('!EVgCurCol') - length(global_str('!EVgPrevWord')));
    del_chars(length(global_str('!EVgPrevWord')) - 1);
    cGenText = 'myFunction(arg1, arg2, arg3)';
    call WriteLine;
    eol;
```

**Generating Multiple Lines Of Source Code** - If you had the need to generate more than one line of text to the program file buffer, simply re-assign cGenText again, and then call the WriteLine procedure, replicating the same pair of lines until the complete code construct is complete. A good example of this is found in the sample ELSE IF segment that creates the ABC...ENDABC code construct.
// ABC... ( abc...endabc )
}

} else if( caps(global_str('!EVgPrevWord')) == 'ABC ' ) {
    return_int = 1;
    goto_col(global_int('!EVgCurCol') l;
    length(global_str('!EVgPrevWord')));
    del_chars(length(global_str('!EVgPrevWord')) - 1);
    cGenText = 'Abc ';
    call WriteLine;
    cr;
    goto_col(global_int('!EVgFirstWordCol')));
    cGenText = 'Endabc';
    call WriteLine;
    up;
    eol;

Inserting Blank Lines - Sometimes, it's necessary to create a blank line in the generated source code. While it's possible to simply assign a null string to cGenText and call WriteLine, the quickest and most efficient approach is to use the Multi-Edit primitive called CR.

// ABC... ( abc...endabc )
}

} else if( caps(global_str('!EVgPrevWord')) == 'ABC ' ) {
    return_int = 1;
    goto_col(global_int('!EVgCurCol') l;
    length(global_str('!EVgPrevWord')));
    del_chars(length(global_str('!EVgPrevWord')) - 1);
    cGenText = 'Abc ';
    call WriteLine;
    cr;
    goto_col(global_int('!EVgFirstWordCol')));
    cGenText = 'Endabc';
    call WriteLine;
    up;
    eol;

Moving The Cursor - There are also times when cursor movement is required during the code generation process, and it's relatively straightforward. Multi-Edit includes primitive functions that can be used to change cursor position as demonstrated by the boldfaced macro statements.

// ABC... ( abc...endabc )
}

} else if( caps(global_str('!EVgPrevWord')) == 'ABC ' ) {
    return_int = 1;
    goto_col(global_int('!EVgCurCol') l;
    length(global_str('!EVgPrevWord')));
    del_chars(length(global_str('!EVgPrevWord')) - 1);
    cGenText = 'Abc ';
    call WriteLine;
    cr;
    goto_col(global_int('!EVgFirstWordCol')));
    cGenText = 'Endabc';
    call WriteLine;
    up;
    eol;
Commenting Support

Adding comments and marking or un-marking source code with comment characters can be extremely time consuming. EVOLVE assists you in this aspect of development by providing you with just a single tool bar button to handle eight different commenting features. The different behaviors of this single tool bar button are accomplished by applying various block marks to the source code before clicking the button. For example, the commenting support macro behaves a certain way when there is a column mark, and a completely different behavior is derived with a line mark. Explore the features listed below for more details.

Same-Line-Comment

Place a same-line-comment at a designated comment column by simply clicking the comment button with the cursor resting on a line of source code. The comment column is defined in the Evolve configuration dialog and by default is set to 50. In the event that the source code on the line being commented exceeds 50 characters, then the commenting system will position the same-line-comment to the end of the line plus two characters.

```c
FUNCTION __SetGraphics
parameters mode
if mode
  sethires(0)
else
  settext()
endif
RETURN(.T.)
```

If the comment button is utilized while the cursor is positioned on a blank line, the position of the comment will be anchored to the location of the cursor at the time the comment button was clicked. If the comment key is pressed on a line that is already commented with a same-line comment, the cursor will move to the beginning of the comment text.

Block Style Comments

**Comment A Block Of Text**

Using the column mark, begin a block mark with the first line of text to be included in the commented area. Make sure the column mark is located at the column you want the comment characters to be written. Continue the column mark until the last line in the text to be commented is reached. The column mark need only be a single character in width. Then click the comment button. The block mark will disappear and each line in the block will be commented.

**Uncomment A Block Of Text**

Using a column mark, begin a block mark with the first comment character to be uncommented. Continue the column mark until the last line in the commented area. The column mark need only be a single character in width but must start in the same column as the current comment characters. Then click the comment button. The block mark will disappear and each line in the block will be uncommented.
Comment Out A Block Of Text (C Style)

Using the stream mark, begin a block mark with the first line of text to be included in the commented area. Continue the block mark until the last character in the text to be commented is reached. Then click the comment button. The block will disappear and the beginning and ending of the block will be enclosed in the C style comment.

Uncomment A Block Of Text (C Style)

Using the stream mark, begin a block mark with the beginning C style comment characters. Then continue the block mark until the ending C style character sequence has been marked. Click the comment button. The block will disappear and the marked block will be uncommented. This commenting style is not available in all XBase dialects.

It is important that you make sure that the cursor is within the boundaries of the block mark when attempting to achieve a particular commenting behavior. If the cursor is outside the block mark when you click the comment button, the commenting macro will assume operation as if there were no block mark, and will perform a same-line-comment.

Function Header Comment

Place the cursor on a source line containing a FUNCTION statement and click the comment button. A comment header will be displayed. The header design and layout is programmable and defined in EV.CFG. More details concerning headers can be found in the Commenting Support section of this guide. Information concerning the layout and programmability of headers can be found in the EV.CFG Configuration File topic.

Procedure Header Comment

Place the cursor on a source line containing a PROCEDURE statement and click the comment button. A comment header will be displayed. The header design and layout is programmable and defined in EV.CFG. More details concerning headers can be found in the Commenting Support section of this guide. Information concerning the layout and programmability of headers can be found in the EV.CFG Configuration File topic.

Class Header Comment

Place the cursor on a source line containing a CLASS or DEFINE CLASS statement and click the comment button. A comment header will be displayed. The header design and layout is programmable and defined in EV.CFG. More details concerning headers can be found in the Commenting Support section of this guide. Information concerning the layout and programmability of headers can be found in the EV.CFG Configuration File topic.

Program Header Comment

Place the cursor on line one and press the comment key. A comment header will be displayed. The header design and layout is programmable and defined in EV.CFG. More details concerning headers can be found in the Commenting Support section of this guide. Information concerning the layout and programmability of headers can be found in the EV.CFG Configuration File topic.
#INCLUDE Header Comment

Most XBase dialects support #INCLUDE capabilities that allow the inclusion of header files into program source. This feature makes it possible for you to pre-configure comment headers for these types of files in a custom format and insert them into your source files in an instant.

To see this feature in operation, place the cursor on the first line of a header file [.H or .CH file extension] and click the comment tool bar item. A default comment header will be displayed instantly. Like Evolve's program headers, #INCLUDE headers are defined in EV.CFG. More details concerning comment headers can be found in the Commenting Support section of this guide. Additional information concerning configuration of comment headers can be found in the topic EV.CFG Configuration File.

Using Programmable Comment Headers

The Comment Headers feature makes it possible to create standardized headers that serve as comment blocks for header files, program files, procedures, and functions. The headers are configurable in ordinary text files so they're easy to create and modify while using Multi-Edit.

Additionally, Evolve can support multiple header configurations without re-installing each time you need a different header format. This ability is easily accomplished because Evolve first looks for the configuration file in the current directory, and then looks in the Multi-Edit path. This enables you to create different EV.CFG files in all the directories you develop applications.

When the current directory is scanned for EV.CFG, Evolve looks to see if EV.CFG exists and if not found it does a second scan for EVxxx.CFG (where xxx is the multi-user ID). The second scan enables you to create multiple headers for multi-user environments in the same directory.

The header templates also support meta command words that are linked to information in the configuration file. For example, the template command word {Author} will expand to the author name as set forth in the EV.CFG file. See the example EV.CFG file later in this chapter.

Headers are automatically inserted whenever you create a function or procedure with the template expansion features of Evolve. Additionally, the headers can be inserted manually by positioning the cursor at line 1 in a program file, and pressing Ctrl+F7. Evolve assumes that you want a program header comment since the cursor is at line one, and there are no column or block marks present to indicate a normal comment operation is desired. Additionally, this assumption is made when the cursor is on a FUNCTION or PROCEDURE statement.

The format of the CFG file is shown below and may be configured to your own header commenting layout and style. Note that the larger the header structure, the longer it will take Evolve to read it in and parse the meta command words.

User Definable Meta Commands

You can also create your own template keywords (meta commands) by simply using a new bracketed word in the static definition area (i.e., the top of the CFG file), the value assigned at the top will be carried to the body of the header and inserted wherever your custom meta command appears. In fact, all commands in the sample CFG file we provided, are meta commands (i.e., they're not hard coded in Evolve) with the exception of the following commands which are system value dependent or source code dependent (e.g., they are automatically created if Evolve can discern their values from your source code).
<table>
<thead>
<tr>
<th>Template Keyword</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>{File}</td>
<td>The file name (no extension)</td>
</tr>
<tr>
<td>{Yr}</td>
<td>The current year</td>
</tr>
<tr>
<td>{DateCreated}</td>
<td>The date the header was created</td>
</tr>
<tr>
<td>{TimeCreated}</td>
<td>The time the header was created</td>
</tr>
<tr>
<td>{Page}</td>
<td>The page number (custom reports)</td>
</tr>
<tr>
<td>{Ln#}</td>
<td>The line number (custom reports)</td>
</tr>
<tr>
<td>{Source}</td>
<td>The source code (custom reports)</td>
</tr>
<tr>
<td><em>{ModuleName}</em></td>
<td>The function/procedure name</td>
</tr>
<tr>
<td><em>{ArgumentList}</em></td>
<td>The parameter list of the module</td>
</tr>
<tr>
<td><em>{Argument}</em></td>
<td>Each parameter on a separate line</td>
</tr>
<tr>
<td><em>{ReturnValue}</em></td>
<td>The return value of the module</td>
</tr>
<tr>
<td>{ModuleName}</td>
<td>The function/procedure name</td>
</tr>
<tr>
<td>{ModuleDesc}</td>
<td>The routine's description (CodeWriter)</td>
</tr>
<tr>
<td>{ArgumentList}</td>
<td>The parameter list of the module</td>
</tr>
<tr>
<td>{Argument}</td>
<td>Each parameter on a separate line</td>
</tr>
<tr>
<td>{Arguments}</td>
<td>Parameters listed on the separate line</td>
</tr>
<tr>
<td>{SuperClass}</td>
<td>Each parent class on a separate line</td>
</tr>
<tr>
<td>{ReturnValue}</td>
<td>The return value of the module</td>
</tr>
<tr>
<td>{Entry}</td>
<td>The default cursor position after insertion</td>
</tr>
<tr>
<td>{Field}</td>
<td>An entry field for cursor positioning</td>
</tr>
<tr>
<td>{DateUpdated}</td>
<td>Update field for the current date</td>
</tr>
<tr>
<td>{TimeUpdated}</td>
<td>Update field for the current time</td>
</tr>
</tbody>
</table>

**IMPORTANT**: Note that these template meta commands are only utilized when creating a header comment block for existing programs, functions, and procedures.

**No Information For A Meta Command**

Meta commands in your header configuration file (EV.CFG) will be ignored if the information they require cannot be discerned from the source code within the body of the construct. Additionally, default entries for {Argument} and {ReturnValue} are entered if these are not detected in the source code. Each will be entered as "None" if not found. The default values are also definable in EVOLVE.SH using a manifest constant.
Multiple Function Arguments

An additional configuration option allows you to specify a different format for parsed arguments in addition to the one where the meta command \{Argument\} is utilized. This feature makes it possible to create comment headers that do not include the first argument line continuously replaced as many times as there are arguments for the function. Instead, an alternative argument line can be inserted for arguments number [2] through [n].

By defining a new configuration item in EV.CFG as shown below, the output of the function or procedure header will utilize the defined format whenever the number of arguments parsed exceeds 1.

\[
\text{[Multiple Argument Line]} \\
* \{Argument\}
\]

The default EV.CFG file provided with Evolve includes this exact definition of the [Multiple Argument Line]. When you create a different header format, it is important that you modify this section of EV.CFG to correspond with your new header format, otherwise multiple argument lines may look awkward. If you prefer multiple argument lines to be the same format as that found in the header definitions, simply delete the [Multiple Argument Line] section from the configuration file.

Cursor Positioning With \{Entry\}

A meta command exists that enables you to position the cursor to any of the variables defined in the comment header templates. The name of the static command is [CursorPos] and is set to a value in the same manner that all other static commands are set such as [Author], etc. The assignment in EV.CFG looks like this:

\[
\text{[CursorPos]} := \{Entry\}
\]

Wherever the first \{Entry\} appears in a comment header, the cursor will be repositioned there once the header is inserted. Additionally, Multi-Edit will be switched into overtype mode if characters are detected to the right of the entry point AND the preprocessor constant "EVc_COMMENTOVERTYPE" is set to one [1] in EVOLVE.SH. By default, this is the behavior of Evolve unless you change EVc_COMMENTOVERTYPE to a zero [0] and recompile your macros.

Cursor Positioning With \{Field\}

The meta command \{Field\} enables you to create a "form" based on the header comment templates. Insert this meta command wherever you want the cursor to stop following after the insertion of any of the comment headers available in the EV.CFG file. When this feature is utilized, the cursor is positioned to the first field and Multi-Edit is placed in overtype mode if enabled by setting EVc_COMMENTOVERTYPE to one [1].

With each press of the TAB key, Evolve will continue to re-position the cursor to each \{Field\} discovered in the comment header template. The order of the entry positions is determined by the same order as discovered in the header during insertion into the source code.

For example, the following excerpt from the default EV.CFG will create a procedure style comment header and move the cursor to the line and column where \{Field\} appears in the header template, thereby allowing you to enter the various prompted items.
Following insertion of a header with {Field}'s in it, the TAB key will continue to reposition the cursor to the next line. When the last field has been reached, the very next TAB key press will attempt to position the cursor to the first line of the function or procedure being commented. In the case of Program and Include file comment headers, the cursor will finally come to rest at line one of the source buffer.

NOTE: Comment headers will only expand when the commenting key is pressed and there is no comment header present at the point of insertion. Furthermore, if a function, procedure, or program header exists, re-commenting will cause update-style date and time fields to be updated.

Prototype Comment Table

**Macro Command: PROTOTYPE**

The Prototype command provides a way to extract a summary listing of all functions and procedures in a source file and instantly convert the listing into a commented summary at the top of the source file.

Programmers familiar with "C" development requirements recognize this as a "prototype" of each function so that it can easily be known what functions are in the source file, and what arguments are required. This is also known as a form of forward declaration of the function which is not required in any XBase dialect [yet].

When you run the Prototype command, a menu listing similar to the Scan menu will appear allowing you to browse through the function listings. From this menu, select "Update Prototype Comment Table" to add the listing as a comment block. If a prototype table already exists in the source file, it will be replaced with a new one, otherwise, the listing will be dropped at the cursor location prior to the execution of the prototype feature.

**Prototype Sorting**

The prototyping system allows you to display and insert the prototype tables sorted by routine type (e.g., procedure/function/method) and by routine name. This feature is enabled by default but may be disabled by changing the controlling preprocessor constant in EVOLVE.SH named EVc_SORTPROTOTYPE.
Prototype Table Format

The output format from the prototyping system provides an informative header with columns for the routine name and the return value of functions contained in the listing. If you prefer a simpler format found in earlier versions of Evolve, simply change the preprocessor constant named EVc_FANCYPROTOTYPE to a zero [0] and recompile the macros.

Active Productivity and Template Editing

In this category of features, active productivity enhancements are discussed. This class of productivity features requires some interaction on your part.

Construct Completion

This feature allows you to complete the currently open conditional construct and is accessed through the toolbar or from the menu. It is used at a point where you need to type the ending statement of an indentable code construct.

This macro simply back-dents the cursor one level, and then proceeds to scan up through the source code at the back-dented indent level, looking for the beginning construct statement. It does not parse the source code in reverse, so if your source code is not indented properly, it may fail to find the proper beginning construct resulting in unpredictable behavior.

In the case of the automatic completion of RETURN statements for functions, the return value inserted into the statement is derived from the return value specified in Evolve's configuration. Please refer to the Configuration topic for additional details.

Template Files

Template files are simply text files that contain the code fragments for each control type. There is one template file for each language supported by CommonTools. The template file naming convention is based on the file extension of the source code files. For example, when writing code in PAS files, the template file name utilized by CommonTools is CTPAS.TEM. For C++, it files, it will likely be CTCPP.TEM or CTC.TEM. For XBase developers, the template file is CTPRG.TEM.

Note: If you develop code in more than one file extension for the same language, copy the original template for that language so that it exists for the additional file extension. As an example, if you customarily program in PAS files and you would like to have CommonTools support source files with extensions of P, copy CTPAS.TEM to CTP.TEM.

Template files are similar to Windows INI files and include template sections identifying each control type. A sample control type and associated template code is shown below.

```
[MenuObject]
DEFINE MENU .?. OF this..?.
PROPERTY ;
  text  "..?"," ;
  helped "..?"," ;
  onclick {..?}
```

As indicated, each control type is identified with its name in brackets (i.e., [MenuObject]). The control type names should not be changed unless you have reviewed the CommonTools API topic and understand the impact of such a change. The source code following the control type name is the code template that actually expands when the control is selected from the tool bar.
Optionally, the beginning of template files may include certain configuration items such as enabling upper and lower case expansion, or disabling argument prompting.

Template files are searched for by CommonTools in the following manner. If the search turns up nothing for the current file extension, an error message is displayed.

1st Current [working] directory as CTxxxnnn.TEM where xxx is the file extension and nnn is the user id.
2nd me_path + user_id + .USR\ + CTxxx.TEM.
3rd Current [working] directory as CTxxx.TEM.
4th me_path + CTxxx.TEM.

Control Types

Control Types refers to the different controls [sometimes called objects] supported by CommonTools. One type of control is a text object [e.g., a string displayed on a form or in a window]. Another type is a spin box [an integer entry field with arrows for incrementing and decrementing the value].

Control types, in most object oriented languages, are universal to some degree, however, the source code necessary to create such objects varies from language to language. The control types supported in CommonTools are representative of many popular languages as well as configurable in the event the support is not specific enough for your development style.

Template Expansion

To make a particular control expand to its code equivalent, certain conditions must exist.

First, there must exist a CommonTools template file for the file extension of the current buffer. If you are editing a Pascal source file with a PAS file extension, the template file CTPAS.TEM must exist in the current working directory, the Multi-Edit directory, or a user directory [for Multi-Edit network installations].

Second, the specified template file must contain a code fragment definition for the desired control type. For some languages, there may not be a way to represent certain control types common to other languages. In these infrequent cases, the template file will contain an entry as shown below.

[MenuObject]
Not Supported...

And third, the line that the cursor is resting on must be a blank line. CommonTools will not expand if the current line already contains text.

Once the code has expanded, CommonTools will position the cursor in an attempt to prompt you for required [or optional] arguments in the source statements. Prompts are identified in a template with ?. character sequences so they're easy to identify. When the first argument is typed, simply press TAB to advance to the next prompt. CommonTools will erase the ? sequence for you and position the cursor accordingly.

Some aspects of expansion are configurable. See the topic entitled Expansion Configuration.

Expansion Configuration

It is possible to configure certain behavioral aspects of CommonTools through expansion configuration settings. The topics on the following page provide complete details.
Argument Prompting

This refers to the automatic prompting of arguments once a template has expanded. By default, CommonTools will automatically position the cursor to specific locations in the template to allow completion of required [or optional] parameters in the code fragment. The following example shows a template where prompting has been disabled.

```
Prompts - on
***********************************************************
|CA-Clipper|
***********************************************************
```

Case Expansion

Templates are expanded in Proper Case format which is how they exist in the template file. Case expansion can be controlled by adding a template configuration line at the beginning of the template file. For example, to force expansion in upper case, the following template configuration line accomplishes this.

```
Prompts = proper
***********************************************************
|CA-Clipper|
***********************************************************
```

Template Galleries

CommonTools has a fixed set of control pushbuttons, however, you may attach many different types of code fragments to each button. If more than one code fragment is present in the template file for the same control type, CommonTools will present each of the fragment names in a pick list so that you can decide which one to choose. Here's an example of a TEM file programmed for multiple code fragments for the Window control type.

```
[Window - MDI Child]
DEFINE WINDOW .?. MDICHILD FROM .?., .?. TO .?., .?. TITLE .?. ;
  BRUSH .?. CURSOR .?. MENU .?. ICON .?. OF .?. ;
  VSCROLL HSCROLL COLOR .?., .?.. PIXEL STYLE .?.

[Window - MDI Parent]
DEFINE WINDOW .?. FROM .?., .?. TO .?., .?. TITLE .?. ;
  STYLE .?. MENU .?. BRUSH .?. ICON .?. MDI ;
  COLOR .?.. .?. VSCROLL HSCROLL BORDER .?. OF .?.

[Window - Regular]
DEFINE WINDOW .?. FROM .?., .?. TO .?., .?. TITLE .?. ;
  COLOR .?.. .?. OF .?. BRUSH .?. CURSOR .?. ;
  ICON .?. MENU .?. STYLE .?. BORDER .?. NOSYSMENU ;
  NOCAPTION NOICONIZE NOZOOM VSCROLL HSCROLL
```

The Window type identifies that all of these fragments are of the same class of template. The text to the right of the dash [-] is a description and may include up to 64 characters of text.
Passive Productivity and Template Editing

An enormous amount of time is spent entering the same pieces of text, time and time again. EVOLVE helps cut down on this repetition by providing automatic expansion of "templates". To put it simply, a template is an abbreviation that explodes into more complete code fragments.

Template expansion is designed to eliminate keystrokes and to that end, Evolve continually watches for specific character sequences in an attempt to spot words, phrases, and even entire code snippets that can be expanded for you. For example, when you type two "I" characters back-to-back, Evolve assumes you are typing an immediate IF function "IIF()" function.

**Brace Closure**

Key Assignment: ",', {, [ and (

Some commands in XBase languages require the presence of a terminating command such as the ENDIF statement for each IF statements. The same principal applies to certain character sequences as well. For example, unless you were writing a compiler or something similar, you would rarely need a double quote character by itself.

Evolve contains five specific template expanders that automatically add the second character when the first is typed. By default, the key assignments for these expanders are enabled. You may disable these expansion templates in the Evolve's Configuration dialog. The configuration section is entitled Passive Expansion Items.

<table>
<thead>
<tr>
<th>If you type</th>
<th>Evolve will expand to</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>(</td>
<td>()</td>
</tr>
<tr>
<td>[</td>
<td>[]</td>
</tr>
<tr>
<td>{</td>
<td>{}</td>
</tr>
</tbody>
</table>

**Complete Brace**

The Brace Completion system helps you by closing the next logical brace enclosure in a line of source code. For example, once a series of braces have been entered into an expression or command line, Clicking this tool bar item will search back to determine which level of brace enclosure is required to complete the expression. This is very handy when developing complex, multi-line/multi-brace enclosures. For example, given this line of source code:

```
Func1(* MyArray[n, * 100.5 (12 / 144), iif(n == 1, 1, 0)
```

With the cursor positioned at the end of the expression, each click of this tool bar button will complete the next required brace closure yielding a completed expression as shown below.

```
Func1(* MyArray[n, * 100.5 (12 / 144), iif(n == 1, 1, 0)])
```

Note that this feature works across multiple lines of a comment statement. When the Complete Brace button is clicked and all braces have been completed, no further brace insertions will be made. Also, bear in mind that this feature uses Multi-Edit's block marking system, so it is temporarily suspended if a block mark is present in the current window.
**Key Code Insertion**

Macro Command: InsertKeyCode

This feature enables you to enter your languages' keyboard control codes or keyboard preprocessor constants by simply pressing the key combination desired. For example, let's assume that you needed to create an argument involving Alt+U but you didn't know the key code. Simply press Ctrl+K followed by Alt+U and Evolve will insert the proper code for the currently configured dialect directly into your source code.

The key code that is entered will vary depending on the dialect that you are using. Dialects that support preprocessor constants will automatically generate the key code constant, whereas, those that do not, will insert the actual key code.

Clipper Developers

To enter a key code preprocessor constant such as those defined in INKEY.CH, simply click the key code tool bar button (which activates the key code dialog) and then press the desired key or key stroke combination. Evolve will insert a key constant at the cursor location in the current PRG file.

Other Dialects

To enter a key control code, simply click on the key code tool bar button (activates the key control expansion dialog) and then press the desired key combination. Evolve will insert a number or standard constant representation at the cursor location in the current PRG file.

Control over what is generated into the program file via the keyboard is stored in the dialect macros (i.e., EV_CLIPR.S, EV_DBASE.S...) and may be modified easily. Make sure you don't create more than 100 keyboard translation elements in each of the two global variables that store the control codes. These variables are named !EVgKeyCodes1 and !EVgKeyCodes2.

**Sticky Tab**

Assignment Character Insertion And Formatting

The assignment character insertion and formatting feature in StickyTab makes it easier to enter memory variable or field assignments in a formatted manner. Each time the TAB key is utilized on a line immediately following an assignment statement, Sticky Tab will intercede causing a single TAB press to align beneath the previous equals sign and enter the next equals sign. The following code fragment demonstrates this productivity enhancement.

```
cVar = 1
nVar = 1
```

By pressing the TAB key immediately after entering "nVar", the Sticky Tab feature inserts the equals sign directly beneath the equals sign on the previous line and positions the cursor to the appropriate entry point. This feature is only triggered when the line immediately above contains an assignment operator such as "=" or ":=". In any other case, the normal TAB character is inserted.

Local Variable Creation

The StickyTab button and the Tab key can now be used to create local variable declarations for new memory variables in a function, method, procedure or class. Simply type the new variable just as though you normally would, then create a stream block mark around the memory variable [an easy way to do this is to simply double click on the memory variable]. Then press the Tab key or click the StickyTab button [var=].
Evolve will test to see if the local variable highlighted is already declared in the current function. If the variable has not been declared, a LOCAL statement will be inserted at the top of the function for this memory variable. If other LOCAL statements are already present, the new declaration will be created after the last LOCAL statement in the current function.

**Assignment Operator**

Key Assignment  
• <SpaceBar>

The expansion of the assignment operator is automatic when Evolve detects a memory variable convention that expands the data type of the variable with a leading character followed by an upper case character [i.e., cAddress]. This practice of using the leading character of a memory variable to identify it's type is known as dHung notation and is considered somewhat of a standard in many languages including C and C++.  

For example:

If you type... It will expand to

PUBLIC xValue<br>PRIVATE xValue<br>LOCAL xValue<br>xValue<br>cName<br>1Married<br>dInvDate<br>nAge<br>mComments<br>aMyArray<br=oMyObject<br>

PUBLIC xValue =
PRIVATE xValue =
Local xValue =
xValue =
cName =
1Married =
dInvDate =
nAge =
mComments =
aMyArray =
oMyObject =

The variable naming convention known as dHung [described above and utilized by Evolve] is shown in the following chart.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Leading Character</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any Type</td>
<td>x</td>
<td>xTempVar</td>
</tr>
<tr>
<td>Character</td>
<td>c</td>
<td>cAddress</td>
</tr>
<tr>
<td>Numeric</td>
<td>n</td>
<td>nAge</td>
</tr>
<tr>
<td>Date</td>
<td>d</td>
<td>dReptDate</td>
</tr>
<tr>
<td>Memo</td>
<td>m</td>
<td>mComment</td>
</tr>
<tr>
<td>Array</td>
<td>a</td>
<td>aPayRates</td>
</tr>
<tr>
<td>Logical</td>
<td>l</td>
<td>lMarried</td>
</tr>
<tr>
<td>Object</td>
<td>o</td>
<td>oMainMenu</td>
</tr>
</tbody>
</table>
Preprocessor Templates

Key Assignment: Various [#in, #ifdef<sp>, #ifndef<sp>, #el, #en, #u, #d, #c, #t, #xc, #xt]

Hash (#) expansion for preprocessor directives and conditionals includes support for the following code constructs and vary depending on the dialect in use.

If you type... Evolve will expand
#in   #include "ch" or "h"
#d   #define
#u   #undef
#ifd<sp>   #ifdef ... #endif
#ifn<sp>   #ifndef ... #endif
#el   #else
#en   #endif
#c   #command =>
#xc   #xcommand =>
#t   #translate =>
#xt   #xtranslate =>

If you type any of the above identifiers, Evolve will instantly expand to the full construct and position the cursor to the appropriate entry point. Case consistency is preserved in the expansion unless it has been configured differently. For example, if you type #IN, the resulting expansion will be fully upper cased. Hash (#) directives will expand only once on a given line, at the beginning of the line.

Immediate IF() Template

Key Assignment: ii and II

The immediate IIF() expansion template is automatically created when Evolve detects two consecutive letter "i" characters typed. The cursor will be repositioned after the expansion to enable entry of the arguments. Most XBase languages support this function capability.

<table>
<thead>
<tr>
<th>If you type</th>
<th>Evolve will expand to</th>
</tr>
</thead>
<tbody>
<tr>
<td>li</td>
<td>iiif( , , )</td>
</tr>
<tr>
<td>li</td>
<td>IIF( , , )</td>
</tr>
<tr>
<td>II</td>
<td>IIF( , , )</td>
</tr>
</tbody>
</table>

Boolean Templates

Key Assignment: <SpaceBar>

Boolean templates are automatic code expansions for the words .OR., .NOT., and .AND..

Evolve is always on the lookout for a dot[.] followed by the letters "O", "N", or "A". If it detects a period followed by one of these letters, either upper or lower case, it will automatically expand to the appropriate boolean word and one additional space character. This is known as a Passive Productivity feature.
If you type | Evolve will expand to
---|---
.o | .or.
.N | .NOT.
.a | .and.
.t | .t.
.F | .F.

Further configuration is possible by enabling True and False expansion for .T and .F triggers. This is recommended only if the dialect in use supports constants such as True and False. If your dialect supports #DEFINE preprocessor capabilities, you can create constants to support the use of the words TRUE and FALSE.

**Code Construct Templates**

Key Assignment: `<Spacebar>`

Whenever you begin a command line with text that would indicate a code construct is being typed, Evolve will automatically expand the rest of the code necessary to complete a valid statement or series of statements.

The expansion takes place when you press the space bar key after typing specific characters that represent one of the expansion templates. For example, when you type IF <space> on a new line, Evolve will enter ENDIF on a new line below, and return the cursor to the line where the IF statement was typed. This is called template editing.

As with all code generation features in Evolve, this one will produce case sensitive source based upon the way you type. For example, if you type the template characters in lower case, the expanded code will also appear in lower case, and so forth for proper and upper case typing. This behavior can be modified to always produce upper case template statements.

To suppress construct template expansion, simply de-select the desired features in the Evolve configuration dialog screen and save the new configuration. See the chapter entitled Configuration for more details.

Depending on the dialect in use, some or all of the templates listed below will be available to you. See the chart below for a list of the various automatic templates available in most dialects.

<table>
<thead>
<tr>
<th>This...</th>
<th>Expands To This...</th>
</tr>
</thead>
<tbody>
<tr>
<td>If <code>&lt;sp&gt;</code></td>
<td>If Endif</td>
</tr>
<tr>
<td>Ife <code>&lt;sp&gt;</code></td>
<td>If Else Endif</td>
</tr>
<tr>
<td>Ifi <code>&lt;sp&gt;</code></td>
<td>If Elseif ElseIf Elseif Endif</td>
</tr>
<tr>
<td>Pro <code>&lt;sp&gt;</code></td>
<td>* Header... (if automatic comment headers are enabled) Procedure Return (example continues…)</td>
</tr>
</tbody>
</table>
This... Expands To This...
Fun <sp> * Header...
Function
Return(Nil)
Doc <sp> Do Case
Case
Case
Case
Otherwise
Endcase
Dow <sp> Do While
Enddo
Tex <sp> Text
Endtext
#Ifd <sp> #Ifdef
#Endif
#Endtext
FoxPro Only:
This... Expands To This...
For<sp> For = 1 to
Endfor
Beg <sp> Begin Transaction
End Transaction
Sca <sp> Scan
Endscan
Pri <sp> Printjob
Endprintjob
dBASE Only:
This... Expands To This...
Beg <sp> Begin Transaction
End Transaction
Sca <sp> Scan
Endscan
Pri <sp> Printjob
Endprintjob
For <sp> For = 1 to
Next
Clipper Only:

This... Expands To This...
For <sp> For := 1 to
Next
Beg <sp> Begin Sequence
End Sequence
Spro <sp> * Header... (if automatic comment headers are enabled)
Static Procedure
Return
Ipro <sp> * Header... (if automatic comment headers are enabled)
Init Procedure
Return
Epro <sp> * Header... (if automatic comment headers are enabled)
Exit Procedure
Return
Sfun <sp> * Header... (if automatic comment headers are enabled)
Static Function
Return(Nil)
Met <sp> * Header... (if automatic comment headers are enabled)
Method
Return(Self)
Smet <sp> * Header... (if automatic comment headers are enabled)
Static Method
Return(Self)
Whi <sp> While
End
Cla <sp> * Header... (if automatic comment headers are enabled)
Class
End Class
Cre <sp> * Header... (if automatic comment headers are enabled)
Create Class
Endclass
Con <sp> * Header... (if automatic comment headers are enabled)
Constructor
Return

Once the space bar has been pressed on a line, the template expander evaluates the entry and attempts to make a proper expansion on the current line only one time. This eliminates the possibility of inadvertent multiple expansions.
Code Formatting And Beautification

Code Formatting And Beautification Overview

This class of features provides support that helps you develop and maintain clean looking and easy to read source code.

Align Operators

Macro Command: ALIGN

This macro will align assignment operators and their values.

Using a column mark in Multi-Edit, mark the lines of source code containing the assignment operator statements [i.e., statements such as nVar = 0]. When marking the area, position the cursor to the column that you want all of the "=" operators to indent to and mark a single character column down the screen. Then click the alignment tool on the Evolve tool bar or run the ALIGN macro.

Using this feature, source code that originally looks ragged, becomes much more readable.

This feature also supports the processing of +=, -=, *=, /=, and /= operators for those language dialects that support them. Furthermore, continuation lines within one of these structures are indented to the level of the first expression following the operator. See the example re-alignment below.

Before...                                                      After…
MyMemVar += aArray[1, 1] =;  MyMemVar += aArray[1, 1] +;
    aArray[2, 1] +;    aArray[2, 1] +;
    aArray[3, 1] +;    aArray[3, 1] +;
    aArray[4, 1] +;    aArray[4, 1] +;
    aArray[5, 1] +;    aArray[5, 1] +;

Source Code Indenter

Macro Command: INDENT

Evolve's indenting capabilities are supported by a high-speed macro that scans your source code and adjusts all command and comment lines to proper indenting format. To execute this macro, you may run it from the Assist menu, the tool bar, or directly as a Multi-Edit macro command.

Control Structures

Many of the control structures supported by Evolve's indenting system are listed below. Note, however, that there are some differences between the various XBase dialects supported by Evolve. A complete list of the indented control structures are presented in the Code Construct Templates topic.

For .. Do While .. While .. If .. #Ifdef .. Begin ..
    ...    ...      ...    ...       ...
Loop     Loop   Loop      Else  #Else   Loop
    ...    ...      ...    ...            ...
Exit     Exit   Exit      ...    ...      Exit
    ...    ...      ...    elseif  #Endif Recover
    ...    ...      ...    ...            ...
Next     Enddo   End      Endif    #Endif   End


**Procedures, Methods, and Functions**

The following table identifies the indenting style for procedures, functions, and other routine types in all dialects supported by Evolve.

<table>
<thead>
<tr>
<th>No RETURN Indent</th>
<th>Backdent On RETURN</th>
<th>Undent After RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure ...</td>
<td>Procedure ...</td>
<td>Procedure ...</td>
</tr>
<tr>
<td>Return (untouched)</td>
<td>Return</td>
<td>Return</td>
</tr>
</tbody>
</table>

⚠️ *The shortened form of RETURN (i.e., RETU) is never back-dented making it suitable for use in an embedded fashion however, use of nested RETURN statements is certainly not discouraged.*

**Case Constructs**

<table>
<thead>
<tr>
<th>No CASE Indent</th>
<th>CASE Indent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Case</td>
<td>Do Case</td>
</tr>
<tr>
<td>Case</td>
<td>Case ...</td>
</tr>
<tr>
<td>Case</td>
<td>Case ...</td>
</tr>
<tr>
<td>Otherwise</td>
<td>Otherwise</td>
</tr>
<tr>
<td>Endcase</td>
<td>Endcase</td>
</tr>
</tbody>
</table>

In some cases, you will want to leave certain sections of code indented the way that you established them when they were typed. To accomplish this, the extended comments `{NoIndent}` and `{Indent}` have been created. By placing a same-line-comment with the extended comment `{NoIndent}`, indentation will be suspended during the INDENT process thus ignoring all source code lines thereafter until the extended comment `{Indent}` is encountered. For example:

```plaintext
if ...
replace name with cName,;  // {NoIndent}
    address with cAddress,;
    city with cCity,;
    state with cState,;
    zip with cZip  // {Indent}
select ...
endif
```

Support for continuation line indenting is available in both the real-time indenter and ad-hoc indenting facilities. Source lines with a continuation character [;] at the end of the line are indented a second level if this capability is enabled in Evolve's Configuration dialog.

**Nested RETURN Statements**

Nested RETURN statements have always been a difficult situation with Evolve. In this release, there are some improvements on this topic and this discussion may help you understand the reasons the product behaves as it does.

With the evolution of XBase languages and object oriented programming abilities, we now have to deal [not only with nested RETURNS] but also with nested functions and procedures, sometimes called methods. The indenting possibilities and preferences are enormous, given this new set of parameters. Since it's difficult for Evolve to be "all knowing" and understand exactly what indenting format you want under varying situations, it must follow a specific set of rules.

The rules also vary depending on whether you're discussing the real-time indenting facility, or the ad-hoc indenting system. Each set of features behave differently and there are specific reasons. The ad-hoc [or reindent] macro looks at each line of code in sequential order before making an indenting decision. As a result, it can make better decisions within a specific context. The real-time indenter has less information to
work with. To make it smarter would require a significant amount of code parsing each time you press the enter key and this would adversely affect performance.

The rules are relatively simple.

**Inside A Function Or Procedure:**

If a RETURN statement occurs greater than one indent level from the left margin, and it's inside a FUNCTION, back-denting is disabled. This makes it possible to maintain back-denting for the last RETURN statement in functions and procedures while disabling RETURN statement back-denting at any higher indent level. Any RETURN statements that occur beyond indent level one are considered nested RETURN's and will not require back-denting.

**Inside A Class Construct:**

If a RETURN statement occurs greater than two indent levels from the left margin, and it's inside a CLASS construct, back-denting is disabled. This makes it possible to maintain back-denting on the last RETURN statement in a nested method within the CLASS construct. This rule is necessary because the method's RETURN statement will be at indent level 2 or less. Any RETURN statements beyond indent level two will obviously be nested RETURN's within a method and will not require back-denting.

Here is the general format that Evolve follows when indenting code with the re-indent feature as well as real-time indenting.

```
class myClass of yourClass
    Local mVar
    [code]...
    FUNCTION myMethod()
    [code]...
    IF [code]...
        return (0)
    ENDIF
    return (0)
    [code]...
endclass
function myFunction(cVar, nVar4, dVar, cVar)
    [code]...
    IF [code]...
        return (0)
    else
        IF [code]...
            return (1)
        ENDIF
    ENDIF
    return (0)
```

**A Word About Preprocessor Conditionals**

In the past, if indenting was on, Evolve indented preprocessor conditionals just like normal conditional statements. However, in Evolve For Windows, there is a new configuration item that enables/disables conditional preprocessor statements. If you nest functions, procedure or classes inside conditional preprocessor statements, we strongly recommend that this new configuration item be disabled [unchecked].

**Reformat Multi-Line Statement**

This feature provides an easy way to reformat a multi-line XBase statement. Designed primarily for object oriented statements, it can rearrange [and properly reindent] a multi-line OOP command in an instant.
Simply place a column block mark at the desired indent level for all lines involved in the command and click the reformat icon.

Evolve will align the source code to the selected column and will indent the object oriented statement appropriately. Keywords such as FIELDS, and PROPERTIES trigger yet another indent level during the reformatting process.

In addition to the reformatting of the source code itself, all same-line comments will be repositioned as well.

Report Writer

The default report format is shown below, and may be changed by simply changing EV.CFG. This makes it possible to have different formats for each directory you develop applications in.

```plaintext
[Print Format]
{h} := Application: {AppTitle}   Page: {Page}
{h} := Source File: {FileName}   Print Date: {DateCreated}
{h} := ----------------------------------------------------------
{h} := Line #   Source
{h} := ----------------------------------------------------------
{b} := {Ln#}  - {Source}
{f} := Author: {Author}    Copyright: {Copyright}
```

The report form may contain three separate form blocks. The HEADER, BODY, and FOOTER. Also note that the format may also contain references to meta commands defined in the static area of the CFG file. For example, you can use the {Author} meta command anywhere in the report form provided it has also been established at the top of the CFG file.

The HEADER and FOOTER may have numerous lines, although from a practical standpoint it is not suggested that you have more than a handful for either headers or footers. The BODY of the report is limited to one line, and since the intent is to print textual information in a sequential order, there is not a great deal of room for creativity in this regard.

Source Code Navigation

More time is spent moving around files, databases, and source libraries than any other process in application development. Evolve addresses this problem by providing a number of ways of navigating as quickly as possible.

Display Current Function Name

Match Construct

This feature allows you to jump forward or backward to the end or beginning of a conditional or looping construct. Using the Ctrl+M key with the cursor placed on an IF statement will cause the cursor to scan forward to the next line at the same indent level. Invoking this feature on anything but the beginning of the construct will cause the search to move up the file.

Object Editing

By simply pointing at a function, procedure or #INCLUDE file name, EVOLVE can find the targeted object, and display it for viewing or editing. Simply place the cursor on the first character of the object and click the object icon. If the object can be found, EVOLVE will present it in a window. If the object is already in memory, such as an #INCLUDE file, EVOLVE will switch to that window.
Source Code Scanner

The Local Scan and Global Scan keys will launch a local and global scan of the current buffer and all buffers respectively. EVOLVE will scan for the word at the current cursor location. This can be a function call, a procedure name, a database name, or any other word found in your source code. The scanner is considered a navigation aid because the resulting scan list can be used as a menu to select an occurrence and have immediate access to that occurrence.

Database Structures

Again, the objects icon will locate and display the database structure of a DBF file by positioning the cursor at a USE or SELECT statement. Additionally, the DBUSEAREA() function is supported for structure display.

Preprocessor Support

Preprocessors have become a helpful and sometimes required aspect of XBase development. If you don't currently use their features, we suggest you become acquainted with the concepts and features in a preprocessor before investigating these unique Evolve features.

Create New Constant

Macro Command: CREATECONSTANT

This feature is a preprocessor constant generator that creates manifest constants while utilizing the text at the cursor position as the basis for transference to a header file in the form of a #DEFINE statement.

To utilize this macro simply place the cursor on a string or integer value that you want to convert to a preprocessor constant, and click the Create New Constant button. Before clicking the button, the cursor should be positioned on the first quote mark of a string to be converted to a manifest constant, or on the first number of an integer value. Optionally, a stream-type block mark can be used to identify the string or number to be changed into a constant, however, in the case of a string, the block mark must include both quotation marks surrounding the text.

A dialog box will display prompts for the constant value that is under the cursor, the constant name that you want to assign, a comment entry (attached as a same-line-comment to the generated #DEFINE statement), and the destination file for the insertion of the #DEFINE. The dialog box also includes a pick list of all #INCLUDE'd files found in the buffer and their paths based on the DOS environment variable INCLUDE. Evolve will set the default destination of the new manifest constant to the current program file. In any case, all prompts in the dialog box can be overridden.

Show Constant Value

Macro Command: SHOWCONSTANT

This feature provides a useful viewing facility for preprocessor constants that may be stored in immediate source files or external header files. This allows you to see the value of a preprocessor constant without actually navigating to the header file or #DEFINE statement that establishes the constant.

For example, with the cursor placed on a known constant that is #DEFINE'd in a header file, click the Show Constant Value tool bar button. Evolve will search the current buffer for the #DEFINE statement, and if unsuccessful, will begin a search of all header files #INCLUDE'd in the current file starting with the last header file first. This feature also searches all current buffers for the #DEFINE statement in the event that the sought after header file is in memory and being altered. The search through all buffers in memory can be disabled in EVOLVE.SH.
List Available Constants

List Available Constants provides a fast way to select and insert a previously defined preprocessor constant. When invoked, this macro looks backwards through the current PRG file and loads each header file identified in a #INCLUDE statement and builds a list of all #DEFINE statements. You may then choose from the list, inserting the constant at the cursor position.

Database Structures

Macro Command: DISPLAY

This feature is a gateway to many capabilities of Evolve. The DISPLAY command allows you to instantly view the file structure of any DBF file in a scrollable window. Additionally, options to perform other tasks related to database file structures are available once the file structure is on screen. Limited index structure viewing is also supported through the DISPLAY command although the file structure information available differs depending on the type of index being viewed. The index structures currently supported include:

- NTX - Clipper indexes
- NDX - dBASE Indexes
- IDX - FoxPro non-compact indexes
- CDX - FoxPro multiple compact indexes
- MDX - dBASE multiple indexes

MEM file displays are also possible, however, only the standard MEM file structure utilized in dBASE III Plus and dBASE IV are supported. The various macro command formats supported for the DISPLAY command are shown below.

<table>
<thead>
<tr>
<th>Macro Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY</td>
<td>Displays a pick list prompt</td>
</tr>
<tr>
<td>DISPLAY [*.DBF]</td>
<td>Displays DBF pick list</td>
</tr>
<tr>
<td>DISPLAY [*.MEM]</td>
<td>Displays MEM pick list</td>
</tr>
<tr>
<td>DISPLAY [*.IDX]</td>
<td>Displays IDX pick list</td>
</tr>
</tbody>
</table>

Once the database structure is displayed, it is also possible to generate FIELD, LOCAL, PRIVATE, and scatter/gather array structures based on the database structure.

Display DBF File Structures

Accessing a DBF structure is done by using the DISPLAY command, or the objects icon. When using the objects icon, you must position the cursor on a statement that references a DBF file. The following access rules apply and require that the cursor be positioned within these statements if using the objects icon.

```
DBUSEAREA(.T., "DBFNTX", "SAMPLE.DBF")
USE Sample ALIAS Sample
NET_USE("SAMPLE.DBF")
SAMPLE->field
SELECT Sample
```

Position the cursor anywhere on the words shown in bold and the file structure for SAMPLE.DBF will be displayed when you click the objects icon.

Optionally, Evolve automatically displays a database file structure if the alias name is entered into a program file followed by "->" and a space bar. This gateway will also allow you to select the field name from the structure display and insert it directly into the source code following the alias directive statement.
Display Options
The DISPLAY options include various utilities to create text files from the structure, instant comment blocks, STORE and REPLACE statements, and hard copy output.

If you attempt to access a database file in this manner, and the file cannot be found in the current directory or the specified path in the DBF environment variable, a dialog box will prompt you for a manual selection of the desired database file.

DBF Environment Variable
To facilitate a more complete search for the database file, Evolve first looks in the current directory for the file, and then looks through all paths of the DBF environment variable for databases and index files. For example, the following DOS command establishes a relative directory and one specific directory for database and index file searching.

C> set dbf=data;c:\database

In this case, regardless of the current directory, Evolve will always search the DATA directory for DBF and IDX files.

Build Comment Block
In the DISPLAY window, select this option to instantly create a commented block representing the file structure. The comment block will be inserted at the point where the cursor was last positioned before the DISPLAY command was invoked and the indent level will also assume the last cursor position. Optionally this feature can create a text file of the DBF file structure in a new Multi-Edit window.

Fancy Structure Format
The DBF structure feature that generates comment blocks, printer, and file output provides a fancy format. The output of DBF structures in the above format is enabled through a preprocessor constant named EVc_FANCYSTRUCTURE. This constant is located in EVOLVE.SH. You may disable the fancier output of DBF structures by simply changing this constant to a zero [0] and recompiling the Evolve macros.

Field -> MemVar

This option will build source code statements that will STORE the fields of the DBF file into memory variables. The source code will be created where the cursor was last positioned before the DISPLAY command was invoked. The indent level of the generated code will be established at the cursor column. The alias name used is based, by default, on the file name of the database. However, you may specify a different alias name in the Alias: field of the file display dialog box. A sample output is shown below.

MemVar -> Field

This option will build source code statements that will REPLACE the fields of the DBF file with memory variables. The source code will be created where the cursor was last positioned before the DISPLAY command was invoked. The alias name used is based, by default, on the file name of the database. However, you may specify a different alias name in the Alias: field of the Display dialog box. A sample output is shown on the following page:
Declare FIELDS
This selection will generate FIELD declarations based on the database structure display for dialects that support this type of syntax.

```
FIELD Name,;
Address,;
City,;
State,;
Zip
```

Declare LOCALs
This selection will generate LOCAL declarations based on the database structure display for dialects that support this type of syntax.

```
LOCAL cName,;
cAddress,;
cCity,;
cState,;
cZip
```

Declare PRIVATEs
This selection will generate PRIVATE declarations based on the database structure display for dialects that support this type of syntax.

```
PRIVATE cName,;
cAddress,;
cCity,;
cState,;
cZip
```

Array Scatter
This selection will generate code suitable for the configured dialect that will scatter database fields to an array with an alias name that matches the database. An example using the Clipper dialect is shown below.

```
aSample := ARRAY(FCOUNT())
AEVAL(aSample,{|xExpression, nField| aSample[nField] := FIELDGET(nField)})
FoxPro:
SCATTER TO aSample
dBASE:
COPY TO aSample
```

Array Gather
This selection will generate code suitable for the configured dialect that will replace database fields from an array with an alias name that matches the database. An example using the Clipper dialect is shown below.

```
AEVAL(aSample,{|xExpression,nField|FIELDPUT(nField,xExpression)})
FoxPro:
GATHER FROM aSample
dBASE:
REPLACE FROM aSample
```

Print Structure
The print option simply outputs a text version of the DBF file structure to the printer.
Function Editing And Creation

There are various ways to create functions and procedures. Review the features listed below for more details.

**Code Writer**

Macro Command: **CWRITE**

The CodeWriter is, by far, one of the greatest time-savers for programmers that are flying by the seat of their pants. In terms of keystrokes, imagine what is required when you discover that a portion of one function really should have been another function. CodeWriter takes a line-mark style block of source code and converts it into a routine by itself.

The options of this feature allow you to specify the location of the resulting function, the case format of the commands, a comment for the function call, and whether a comment header should be created. For example, assume that the highlighted area in the function named reOpenProject() needs to be moved to a function by itself. Using a Multi-Edit line mark, highlight the source code lines that are to be established as a separate function, and then click the Code Writer icon on the Evolve tool bar or run the Code Writer menu option from the Assist menu.

Marking a line block to convert to a function

```plaintext
FUNCTION reOpenProject

    select Project
    nTemp = recno( |
    use
    use cbproj alias Project
    select Project
    go nTemp

RETURN (Void)
```

EVOLVE will respond with a dialog allowing you to select different upper and lower case formats for the new function as well as a function name, and other pertinent information concerning the creation of the function, its position, and possible argument list. The argument list is typed in just as it would appear in your source code, and will be automatically inserted in the function call as well as the function itself. A check box has been provided so that the arguments can be passed to LOCAL variables (i.e., prototyped in the function statement) or as PRIVATE variables in a PARAMETERS statement.

OOP support is also available in the dialog menu so that you can create a "method" and corresponding object message from the marked area. The format used, generally conforms to OOP paradigms and assumes the object message is being sent to itself.

When the prompts have been completed to your satisfaction, pressing **Enter** or clicking on OK will complete the task and the resulting code will appear.
Object Editing

Evolve features many types of object editing capabilities. Note that an object is a loose term we use to cover items that are referenced in source files that can be isolated for purposes of viewing or editing. The goal of object editing is to increase speed when accessing other files, structures, functions and procedures, without forcing the programmer to leave the point where the object is referenced. It's a hyper-navigator that takes you to specific places in your source code instantly.

This capability makes it possible to simply point at a procedure name [among other things], and immediately bring it up for viewing or editing. This applies to user defined functions as well as #INCLUDE files and even database structures and index files.

Evolve uses a specific search criteria for finding these various resources. For example, when you edit a #INCLUDE file, Evolve looks in all other buffers for the file requested. If it isn't currently being edited in Multi-Edit, it then looks in the current directory. Lastly, it looks in the DOS environment for the INCLUDE environment variable. The table on the following page shows the search logic for each resource type.

INCLUDE Environment Variable

This environment variable, which is usually established for the language or compiler you use, is generally employed to point to a directory where all header files are stored. When attempting to edit a file [using the object editing feature] identified by #INCLUDE, Evolve will first attempt to locate the include file in memory. If not found there, it will then search for the file on disk in the current directory, and lastly, the search will culminate in the directory [or directories] specified in the INCLUDE environment variable.

```
SET INCLUDE=C:\CLIPPER\INCLUDE;C:\TC\INC
```

For example, the SET INCLUDE statement shown above configures Evolve to search in Clipper's INCLUDE directory as well as Turbo C's INC directory.

DBF Environment Variable

The DBF environment variable works in similar fashion to the INCLUDE variable in that it provides a search path for Evolve to locate database and index files if they cannot be found in the current directory.

```
SET DBF=DATA;C:\DATADICT
```

LIBR Configuration Command

This meta command enables search of a source code file list and it is specified in EV.CFG. The [LIBR] meta command provides a source file list that Evolve will use to search for functions and procedures after it has exhausted the search of all current windows in Multi-Edit. A [LIBR] specification sample is shown below and may include an unlimited number of PRG files, up to 2048 characters.

```
[Copyright] := MESI
[AppTitle] := Billing Information System
[Libr] := source1.prg, source2.prg, source3.prg
```

The search for routines will progress through each of the listed files in the CFG file in the order that they are listed. Once the routine is located in one of the listed files, the object will be available for editing. Upon completion of the object edit, the file that contains the routine will appear in a new window.

This feature is useful on a project basis since Evolve always looks in the current directory for an EV.CFG file, then it looks to the ME directory and/or the users directory in the case of a network installation of Multi-Edit. This design allows you to have [LIBR] lists that vary from project to project.

To activate Evolve with regard to any of these features, you must point at a code structure that applies to the desired feature, and click the objects icon.
**Universal Object Access**

As indicated, this button assignment is sensitive to the code structure found at the point where the cursor presently rests. A list of the various supported code structures for Object Editing are shown below. Some of the code structures supported may not be supported by the XBase language dialect that you are using. However, as in the case of #INCLUDE support, it is possible, and recommended that third-party preprocessors like Pre/DB be used with XBase dialects that don't include these features such as dBASE for DOS and FoxPro.

```plaintext
#INCLUDE "keycodes.h"
```

Position the cursor anywhere on this line, and the file KEYCODES.H will pop-up in a window.

```plaintext
DBUSEAREA(.T., "DBFNTX", "SAMPLE.DBF")
USE Sample ALIAS Sample
SELECT Sample
```

Position the cursor anywhere on the words shown above in bold and the file structure for SAMPLE.DBF will be displayed.

```plaintext
SET INDEX TO Sample
USE Sample INDEX Sample
```

Position the cursor anywhere on the words shown above in bold, and the index expression for the SAMPLE index file will be displayed. If more than one index exists in the list, a pick list will be displayed allowing you to select the desired index file.

```plaintext
MyWin()
```

Position the cursor anywhere on the function name shown above, and the function MyWin() will be displayed in a window.

```plaintext
DO MyProc
```

Position the cursor anywhere on the procedure name shown above and the procedure MyProc will be displayed in a window.

**Objects That Can't Be Found**

Utilizing the object editor on functions and procedures that cannot be found will simply produce an error message. The scope of possibilities is just too wide to assume that the missing routine needs to be created.

In the case of #INCLUDE editing, Evolve will prompt for the creation of new #INCLUDE files and even provide a history list of all possible supported INCLUDE DOS paths for the missing file. This feature also creates a program header comment if Automatic Comment Headers is enabled in the Evolve configuration dialog.

**Search Criteria**

The search criteria used [when you attempt to edit a UDF or procedure] includes a search for the routine in the current window, followed by a search of all other windows open in Multi-Edit. As a last resort, the search culminates by looking through all source files listed in the [LIBR] meta command in EV.CFG.
Assist Menu

Macro Command: ASSIST

The ASSIST Menu presents most of the EVOLVE macros in an easy-to-use pop-up menu.

By default, the Assist Menu can be accessed from Multi-Edit's context menu [by clicking the right mouse button], or by running the macro named "ASSIST". To execute any of the menu options, simply point to the desired feature and press [enter] or use the mouse to click on the menu option.

Additionally, the Assist Menu can be displayed from the Evolve Tool Bar. The Evolve Tool Bar may be displayed or hidden [as you may require from time-to-time] via the Multi-Edit context menu.

PL/SQL

Overview

Contained in this Online Help are some important notes, installation procedures, enhancements, and a list of recent bug fixes. Please read this entire Online Help before using the PL/SQL Compile Add-On.

PL/SQL Compile Add-On adds ORACLE PL/SQL support to Multi-Edit 9.10. This includes:

- Start of compilation of PL/SQL within Multi-Edit (SQL Plus and an ORACLE Database are needed)
- Start of SQL-Scripts within Multi-Edit (SQL Plus and an ORACLE Database are needed)
- Compilation error handling for PL/SQL (jump to the error line and column delivered from SQL Plus and consideration of blank lines - Oracle delivers error line numbers without the consideration of blank lines (ORACLE 7))
- ORACLE Connection Administration

Installation and Requirements

Requirements

PL/SQL Add-On for Multi-Edit 9.10 has been tested with:

- ORACLE SQL Plus 3.3.4.0.0 - Production
- ORACLE SQL Plus 8.0.4.0.0 - Production
- ORACLE SQL Plus 8.1.6.0.0 - Production
- ORACLE Server Release 7.3.4.0.0 - Production
- ORACLE Server Release 8.0.4.0.0 - Production
- ORACLE Server Release 8.1.5.0.0 - Production
- ORACLE PL/SQL Release 2.3.4.0.0 - Production
- ORACLE PL/SQL Release 8.0.4.0.0 - Production
- ORACLE PL/SQL Release 8.1.5.0.0 – Production

(topic continues…)
Since PL/SQL Add-On for Multi-Edit Version 2.4 requires Multi-Edit v9.10, it is assumed that you have installed and/or upgraded your copy of Multi-Edit and are somewhat familiar with its operation.

**Installation**

The setup program creates the subdirectory PL_SQL_Add-In in with the PL/SQL Add-In support files in your Multi-Edit directory. This directory is necessary for the installation process (see step 3).

For the installation use the following steps.

1. Start Multi-Edit
2. From the main menu, select Tools -> Install Add-On Packages.
3. When the Add-On Installation screen appears, select Browse and choose the directory with the PL/SQL Add-In support files.
4. Highlight "PL/SQL Add-In for Multi-Edit 9.10" and press the "Install" button. Follow the instructions and PL/SQL Add-In will be installed on your computer.
5. In the main menu select Tools->ORACLE Connections->Connection Administration...
6. You can create/add and edit your ORACLE Connections in the dialog which appears.
7. The compilation of PL/SQL Code and the execution of SQL scripts are only successful when your ORACLE BIN path is set in the "PATH" environment variable of your operating system. Otherwise SQL Plus will not be found.
8. There is a problem with ORACLE SQL Plus in the Version 8 together with the ORACLE Server Release 7.X and the error handling. This version delivers a "Package created" message even if the file contains errors. The "show errors" command delivers a no error message.

The following "show errors" command has to be used "show errors [module type] [module name]" (for example: show errors package body examplepackage). This command delivers the error message with the needed line and column numbers.

The result is, if you want to use SQL Plus 8 together with an ORACLE Server Release 7.X then you should use the option "Use SQL Plus 8 for ORACLE 7 connections" in the "Language Properties Setup" dialog for PL/SQL in Multi-Edit. That option generates the correct show errors command for the compilation with SQL Plus 8.

9. Exit Multi-Edit and start it again.
10. You can type in a default name for the author of a module for the module templates. Choose "Edit templates". Choose in the "template set" "GLOBAL" and then in the "templates column" the "NAME" template. In "Result:" replace the displayed name with your name. In the "Keyword" field type in your first name or last name. In the "Min length" field choose the length of the name which you have typed in the "Keyword" field.

---

**Warning:** Do not change (add, delete or move) the filename extensions of the SQL- or PL/SQL language otherwise the ORACLE connection administration will not work and problems will occur during the next update of the PL/SQL Add-On.

**Limitations**

It is necessary to put each type of module of PL/SQL in a separate file, so that some functions work properly (Error handling and function tagging).

For example: Separate a Package Specification from a Package Body. Only one type in a file (for example: only a Package Specification). Combinations of module types in one file are not allowed (for example: not a
Stored Procedure with a Package Body). This is not really a limitation, because it is always better to put each module in a different file.

No error handling for SQL scripts.

ORACLE runtime errors cannot be handled.

Tagging for overloaded functions/procedures doesn't work in some cases. That occurs if the lines, which contain the name of the functions/procedures, have no difference. That can happen if the arguments of the functions are placed in different lines. This limitation is not based on the Add-On but on a limitation of Multi-Edit.

You should not run SQL Plus 3.X against an ORACLE 8 database. Because there are some incompatibilities, for example if you want to compile a PL/SQL source file and a compilation error occurs, you will receive a buffer overflow error. The problem also is not based on the Add-On but on a Multi-Edit limitation.

If a procedure or function prototype is declared in a package specification and the definition is forgotten in the package body, the attempt to compile the body causes a problem. ORACLE delivers an error message and the line number in that error message relies on the package specification source code and not on the package body source code. This cannot be handled by the PL/SQL Add-On.

Do not use the & character in comments of the source code otherwise ORACLE SQL Plus stops the compilation and expects an input value.

SQL Plus which is used for compilation can only handle file names inclusive the path with a maximum length of 80 characters.

---

**Warning**: Do not change (add, delete or move) the filename extensions of the SQL- or PL/SQL language, otherwise the ORACLE connection administration will not work and problems will occur during the next update of the PL/SQL Add-On.

Do not change the commandline setting in the "Compiler/Program Setup" dialog otherwise the PL/SQL Add-On does not work properly.

---

**BSC Browser**

**What is the BSC Browser?**

BSC Browser is a Multi-Edit Add-On, which allows you to use Microsoft Visual C++ browse information (.BSC files) within Multi-Edit.

The BSC Browser Add-On supports .BSC files created with Microsoft Visual C++ 4.2-6.0

BSC Browser allows you to browse the symbol under cursor or prompt for symbol name and locate its definition in source files or the declaration in header files. It is able to locate symbols even in changed files using heuristics and original line numbers. Browsing allows wildcards and the browser caches all possible information so next access is quick even for big lists. Results of previous functions are available from the BSC Browser menu. The BSC Browser saves the position where it was called from and it is able to return back. A one level undo function is also available for mistake corrections or for quickly switching back and
Browsing can cooperate with Multi-Tags and the same hotkeys being used for both the BSC Browser and Multi-Tags browse, backtracking and browse current file functions.

Every browse dialog contains set of filters based on symbol type and also advanced filters based on symbols usage which can be used for analysis purpose. It is also possible to display a list of callers of the symbol and a list of symbols used by the symbol. List all globals displays all globally visible symbols.

The BSC Browser can display and locate symbol references. Using original line numbers it can locate references even in files changed after compilation. References can be used for code analysis, quick navigation and troubleshooting because the BSC database contains exactly what the compiler sees after preprocessing.

BSC Browser is project aware and integrated to the project manager.

Browse modules dialog can be used for quick navigation in the project especially if local files feature is used. It is possible to synchronize BSC modules with Multi-Tags and display detailed information about every module. For BSC databases created on different computers’ paths stored in the BSC database can be substituted to match different local drives and directory mapping. Last but not least BSC Browser is able to display parameters info for the function under cursor using BSC database information and real source files. This info can be configured to display automatically when open parenthesis key is pressed after function or macro name.

### Installation

BSC Browser is installed just like any other Multi-Edit Add-On. Copy all files from the archive to a directory, run Multi-Edit, go to **Tools | Install Add-On Packages**, select the proper Source Directory and press the Install key. Files should be copied to proper locations and a new BSC Browser submenu should appear in the menu bar.

---

*You must restart Multi-Edit after installing the new add-on package.*

---

### Updating from previous versions

To update from a previous version click on **Tools | Install Add-On Packages** and point to the proper Source Directory where BSC Browser is installed. Multi-Edit will find the previous version and asks what to do, select "Full install" to begin the update process.

---

*You must restart Multi-Edit after installing the new add-on package.*
Using the BSC Browser Add-On

BSC Browser is Multi-Edit project-aware and is recommended for use with projects.

Recommended project set up is as follows:

1. Create a new project or set some existing project
2. Go to Project | Properties | Directories | Root Directory and set it to the directory to where the project source files are stored.
3. Create a .BSC file for this project in Microsoft Visual C++ (more about this).
4. Go to BSC project tab and select .BSC database file and set both directories if different from the project root directory. Try the Information button to see if BSC database file was set correctly, especially if metacommand are used.

Now you can List all modules, Browse modules, symbols etc.
Glossary of Terms

Ada
Ada was designed to be a general-purpose language for everything from business applications to rocket guidance systems. One of its principal features is that it supports real-time applications.

Aliases
An alternative name for an object, such as a variable, file, or device.

ANSI
American National Standards Institute.

ASCII
American Standard Code for Information Interchange. Pronounced ask-ee, ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127.

ASP
Active Server Pages

Batch file
A file that contains a sequence or batch of commands. Batch files are useful for storing sets of commands that are always executed together because you can simply enter the name of the batch file instead of entering each command individually.

CUA
Common User Access, a set of standards for user interfaces developed by IBM. The CUA standards deal with interface appearance, programming conventions, and communications.
Dbase
The dBASE format for storing data has become a de facto standard, and is supported by nearly all database management and spreadsheet systems. Even systems that do not use the dBASE format internally are able to import and export data in dBASE format.

Delphi
A Rapid Application Development (RAD) system developed by Borland International, Inc. Delphi is similar to Visual Basic from Microsoft, but whereas Visual Basic is based on the BASIC programming language, Delphi is based on Pascal.

FORTRAN-77
FORTRAN 77, Developed in 1977, includes a number of features not available in older versions of FORTRAN.

FTP
File Transfer Protocol

HTML
Hyper Text Markup Language

IDE
Integrated Development Environment

Java
Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web.

JavaScript
A scripting language developed by Netscape to enable Web authors to design interactive sites.

Macros
A symbol, name, or key that represents a list of commands, actions, or keystrokes.

MDI
Short for Multiple Document Interface, a Windows API that features multiple document windows within the parent application’s frame window
Modeless Dialog
Modeless dialog boxes are dialogs which stay on the screen and are available for use at any time but permit other user activities.

Modula-2
Modula-2 addresses Pascal’s lack of support for separate compilation of modules and multitasking. Although Modula-2 found support in academia, it is not often used for applications.

OEM
Original Equipment Manufacturer

Parse
In linguistics, to divide language into small components that can be analyzed. For example, parsing this sentence would involve dividing it into words and phrases and identifying the type of each component (e.g., verb, adjective, or noun).

Pascal
Pascal is best known for its affinity to structured programming techniques. The nature of the language forces programmers to design programs methodically and carefully.

Perl
Short for Practical Extraction and Report Language, Perl is a programming language developed by Larry Wall, especially designed for processing text.

PHP
PHP Hypertext Preprocessor is a server-side, HTML embedded scripting language used to create dynamic Web pages.

Regular Expressions
Regular expressions ("regex's" for short) are sets of symbols and syntactic elements used to match patterns of text.

Reserved words
A special word reserved by a programming language or by a program. You are not allowed to use reserved words as variable names. For example, in BASIC and COBOL, the word IF is reserved because it has a special meaning.

Results Window
The Results Window is a tabbed pane at the bottom of the Multi-Edit screen.
**SQL**
Structured Query Language, pronounced either see-kwell or as separate letters. SQL is a standardized query language for requesting information from a database.

**URL**
Uniform Resource Locator, the global address of documents and other resources on the World Wide Web.

**VBScript**
Visual Basic Scripting Edition, a scripting language developed by Microsoft and supported by Microsoft's Internet Explorer Web browser.

**Wildcard**
A special symbol that stands for one or more characters.

**WYSIWYG**
Pronounced wizzy-wig, stands for what you see is what you get. A WYSIWYG application is one that enables you to see on the display screen exactly what will appear when the document is printed.
Index

A
Adding a new Compiler 109
Adding a New Language 136
Adding Error Processing 112
Adding features to Multi-Edit 204
Add-On Integration 10
Addon templates 131
AI Internet Solutions 220
Aligning Operators 94
Alternative Mouse functionality 49
ASCII Chart 163
Assist Menu 263
Auto Detect 131
Automatic Function Templates 132
Automating Tasks Using Macros 159
Auto-Template Expansion 129

B
Backup and Autosave 62
Binary 131
Block manipulation 91
Block Operations 93
Block Options 49
Block Types 92
Blocks and Block Marking 27
Book marking Cursor Position 28
Bookmark View 105
Bookmarks 105
Borland 214
BradSoft 218
Bsc Browser 265
Burton Systems Software 151
Bypass VCS 184

C
C/C++ 21
Calculator 164
Calculator Setup 165
CDATE 171
Center Line 96
CFGFILE 175
Choosing a Command Set 35
Classic Regular Expressions 70
CloseMatch 194
CM 174
CMACWIN compiler 116
Code Commenting and Uncommenting 137
Collapse 100
collapse mode 100
Collapse Mode 27
Collapse Modes 101
Colors 51
Columnar blocking 49
COM 210
Command line prompt 111
Command Line Switches 181
Command Sets 35
Commenting 96
Common Code Manager 158
COMNT 175
Comparing files 106
Comparing Files 29
Compile Macros 113
Compiler Metacommands 174
Compiler Support 174
Compiling 29
Compiling on a remote computer 121
Composite Diff 109
COMSPEC 170
Configuring a Browser 158
Configuring the VCS System 145
Construct Matching 138
Copy 91
CR 177
Create HTML from Code 163
Creating a Template 135
Creating Projects 152
Creating Tags 103
CSE HTML Validator 220
CTIME 171
CUR 177
Current File Only 215
Custom Toolbar Buttons 45
Customize this files settings 50
Customizing Menus 41
Customizing Toolbars 43
Cut 91
CVS 151
CVS Support 175

D
Data Commands 192
Date Stamp 97
DDE 206
DDE Server Name 206
Default directory 131
Default Help file 131
DEFAULTS_PATH 170
Delphi 1.0 214
Delphi and C++ Builder 216
Develop Parms 112
DevStudio 6.0 217
Difference List 150
Difference Report 108
Display Default Parameters 133
Display Session Manager 183
Drag and Drop 51
Drag and Drop Files 51

E
Edit Mode 129
Edit this menu 50
Editing and Formatting 27
Editing HTML Tags 158
Embedded Language Support 156
Enabling Templates 134
Error Parsing Expressions 114
Error Processing 113
Errors 150
Evolve 223
Evolve Configuration 228
Evolve.sh Configuration Header 228
Example Keystroke Macro 160
Execute Command Syntax 206
Execute Program Support 173
Expand Style 133
Expand to spaces 130
Expanding a Template 135
Explicit Tags 102
Explorer-style dialog 61
Expression Highlight 77
EXT 170
Extension-Specific Help 50

F
Fax Support 22
FDATE 171
FILE 170
File Compare Toolbar 108
File Find and Replace 67
File List Right Click Menu 154
File Open and Close Interface 148
File Properties 61
File Replace 68
File Search 67
file sharing 62
File Type 130
Filename Extensions 128
FilePane Add-On 46, 65
Files 59
Files Buffers and Windows 55
Filtering Output 119
Find and Replace 66
Find Next Compiler Error 125
Find Previous Compiler Error 125
Find tag under cursor 50
Find Word Under Cursor 76
Finding Multi-Edit 9 menus in Multi-Edit 9.1 7
Floating 44
Fonts 53
Formatting Lines 98
Forum Support 21
FTIME 171
FTP Support 154
Function Editing and Creation 260
Function Tagging 141
Function Templates 133

G
General Command Line Metacommands 170
General Commands 190
GETBLOCK 178
Global Expression Highlighting 28
Goto Column 99
Goto Column n 184
Goto Line 99
Goto Line / Column 29
Goto Line n 184

H
Help File Metacommands 176
Help File Support 176
Hex Mode 27, 99
HEXT 176
HFILE 176
HNAME 176
HOPTS 176
How Does the VCS Support Work? 144
How to get Technical Support 21
HPATH 176
HTML Formatter 157
HTMLHELP 176

I
Incremental Search 76
Incremental Searching 28
Indent Style 129
Indenting 127
Indenting Blocks 92
Install.lst 205
Installation Requirements 10
Intellimouse Support 51
Internal Buffer vs Windows Clipboard 92
Internet Programming Support 156
Intro to CMac 161
Introduction to Multi-Edit 25

J
Justify Paragraph 98

K
Key / Command Assignments 11
Key Commands 59
Keystroke Macros 32

L
Language Specific Help 23
Language Support 29
LASTPROMPT 178
LCM 152
LFN 169
License Agreement 18
Line Blocking 49
Line Numbering 27
Line numbers 96
Linedraw 167
List Boxes 48
LM 174
Load A Binary File 185
Load A DOS File 185
Load A File 184
Load a List of Files 186
Load A Unix File 185
Load error file only 112
Long Filename Metacommands 169
LOOKUP 175
LPATH 175
LPATHX 175

M
MAC_PATH 170
macro 179
Macro Call 100
Macro Language Reference Guide 22
Marking A Block 49
Mask List 60
Match Constructs 99
Matching 126
ME Term 3
ME_PATH 170
MEERR 174
Memory Operations 164
MERANT 151
Metacommand - <CDate> and <CTime> Format
  String Values
D 173
dd 173
dddd 173
gg 173
h 173
H 173
hh 173
HH 173
m 173
M 173
mm 173
MM 173
MMMM 173
s 173
ss 173
T 173
tt 173
y 173
yy 173
yyyy 173
Metacommand - Compiler Support
MEERR 174
NR 174
TESTERR 174
metacommand - CVSDIR 175
Metacommand – Execute Program
XDOS 173
XMAC 173
XOS2 173
XW16 173
XW32 173
XW32C 173
Metacommand - General Command Line Support
- str 170, 171
%var 170, 171
@macro 170, 171
CDATE 170, 171
COMSPEC 170, 171
CTIME 170, 171
DEFAULTS_PATH 170, 171
EXT 170, 171
FDATE 170, 171
FILE 170, 171
FTIME 170, 171
MAC_PATH 170, 171
ME_PATH 170, 171
NAME 170, 171
PATH 170, 171
PROJECT 170, 171
PROJECT_FILE 170, 171
PROJECT_FILEPATH 170, 171
PROJECT_LIST 170, 171
PROJECT_ROOT 170, 171
USER_ID 170, 171
USER_PATH 170, 171
Metacommand – Help File Metacommands
HEXT 176
HFILE 176
HNAME 176
HOPTS 176
HPATH 176
HTMLHELP 176
WINHELP 176
metacommand - RCSDIR 175
metacommand - SAPCMT 176
Metacommand – Template Metacommands
<.?> 177
<.*template> 177
<0xHH> 177
<.CR> 177
<.CUR> 177
<GETBLOCK> 177
<LASTPROMPT> 177
<mnn> 177
<PROMPT> 177
<PUTBLOCK> 177
<RCUR> 177
<RIND> 177
<SCIND> 177
<SCUR> 177
<SIND> 177
<SMARTIND-> 177
<SPACE> 177
<TAB> 177
<TEXTstr> 177
<UP> 177
<XCUR> 177
Metacommand – VCS Metacommands
%TMP 175
CFGFILE 175
COMNT 175
LOOKUP 175
LPATH 175
LPATHX 175
OPATH 175
TMP_PATH 175
TMP_PATHX 175
Metacommands 169
Metacommands – Execute Program 173
Metacommands - Long File Names 169
Metacommands - Printing Support
  CM 174
  LM 174
  PAGE 174
  RM 174
Microsoft 216
Minimal vs Maximal Closure 75
Modifying a Command Map 36
Mortice Kern Software 151
Moving and Copying Blocks 49
MSDOS text 131
Multi User Configurations 11
Multi-Edit 5
Multiple Instances 186
Multi-Tags 30, 102

N
NAME 170
Navigating Windows 26, 58
Navigation Pane 46
Network Configuration 11
New menu commands 9
New Updated Help 3
Next Difference / Prev Difference 108
nnn 179
No error processing 111
No Restore 181
No Restore, No Save 182
No Splash 182
No stderr capture 111
No stdout capture 111
Notebook 166
NR 174

O
Object Editing 261
OEM Mode checkbox 60
OEM Translation 54
OPATH 175
Open File Under Cursor 50
Opening and Saving Files 59
Opening Files 25
Ordering Information 18
Organizing Windows 56
Override Current Line 52

P
PAGE 174
Page Break 97
Page Break String 97
PAN/LCM 152
Paste 91
Paste Buffers View 91
PATH 170
PL/SQL 263
Pop 105
Post-load macro 131
Pre-save macro 131
Preview Pane 102
Printing Metacommands 174
PROJECT 171
Project File List 154
Project Folder Tree 153
Project Management 30
PROJECT_FILE 171
PROJECT_FILEPATH 171
PROJECT_LIST 171
PROJECT_ROOT 171
PROMPT 178
Properties 141
Property Strings 125, 131
Protecting Sessions 156
Push 105
PUTBLOCK 178
PVCS 151

Q
Quick Pick Window List 58

R
Random 105
RCS 151
RCS Support 175
RCUR 177
Read Only 181
Read Only checkbox 60
Read/Write 182
Receiving Files 155
Record 160
Record Length 131
Record Length field 61
Record Macros 159
Redo 98
Reformat Multi-Line Statement 254
Reformat Paragraph 97
Regular Expression Example - Find the next word 81
Regular Expression Example – Operator Search 82
Regular Expression Example - Search and Replacing Phrases 89
Regular Expression Example - Search for a blank line and delete it 83
Regular Expression Example - Search for the 'S' character 78
Regular Expression Example – String group search and replace 82
Regular Expression Insert 78
Regular Expressions 68, 113
ReleaseParms 112
Reload File 111
Repeat 161
Report Writer 255
Returning Result Information 208
Right click menu 153
Right Margin 129
RIND 177
RM 174
Roving 44
Ruler 97
Run A Macro 182
Run in background 111
Running your compiler 118

S
SAP 152
SAP Support 176
Save All Files 111
Saving Files 26
SCC 144
SCIND 177
Scrolling 100
SCUR 177
searching 66
Searchinf Within Multi-Edit 28
Sending Files 155
Session Management 31
Session Manager 31, 156
Sessions 155
Set Commands 197
Setting up FTP Support 154
Setup Dialog 121
SFN 169
SIND 177
Smart Indenting 141
SMARTIND 177

Sourcerer's Apprentice 152
SourceSafe 152
SPACE 177
Spell Check 162
SSAFE 152
Start A Session 183
Start The Last Session 183
Startup.cfg 186
Startup2.cfg 186
Status Bar 47
Stream Blocking 49
Supported Executed Command Set 209
Supported VCS Aliases (or Metacommands) 150
Supported Version Control Systems 151
Syntax and Keyword Highlighting 137
System View 168

T
TAB 177
Tab Bar 47
Tab settings mode 129
Tab spacing 129
Tabs vs Spaces 98
Tags and Projects 104
template 179
Template Metacommands 177
Template Support 177
Templates 29, 134, 137
TESTERR 174
TEXT str 178
Time Stamp 97
Tip of the Day 23
TipWin 132
TipWin Configuration 133
TLIB 151
TLIB 5.x DLL Support 151
TMP 175
TMP_PATH 175
TMP_PATHX 175
TmplPane Add-on 46
TmplPane Add-On 64
Tools Pane 45
Top Style 218
Tracking Line Numbers 120
Trita 221
Type field 60

U
Uncomment 96
Understanding Setup Verification 147
Understanding Templates 134
Undo 98
Unhide 57
UNIX text 131
Unjustify Paragraph 98
UP 177
Update Macro Processor 187
Update Script Block Comments 188
Update Script Conditionals 188
Update Script Expressions 188
Update Script Line Comments 188
Upgrading 7
Upgrading existing installations 7
Use cmd processor 111
Use format line 129
Use tab and margin settings 129
USER_ID 170
USER_PATH 170
Using a Find View 76
Using Associate Directories 146
Using Regular Expressions 68
Using the Project View 153
Using the Tab Bar 58
Using the Window List 58

V

VCS 143
VCS Configuration 145
VCS Interfaces 148
VCS Metacommands 175
VCS Support 175
Version Control Support 31
Version Control System 143
Viewing Files 26
Viewing Modes 27
virtual screen 56
Visual Basic 6.0 217

W

WCMD Identifiers 40
Web Support 22
Weblair 33
What are Macros 32
What are Markup Language Tag Databases? 157
What are Metacommands 169
What are Projects 152
What are Sessions? 155, 156
What is a Command Map 36

What is Multi-Tags 102
What is VCS? 143
What's Installed 5
What's New? 1
Whole Project 214
Why have VCS Support in Multi-Edit? 144
Win32 Console 111
Window List Add-on 47
Window List Add-On 58, 65
WINHELP 176
Working directory - file prompt 61
Working with the Tool and Navigation Panes 63

X

XCUR 177
XDOS 173
XMAC 173
XOS2 173
XW16 173
XW32 173
XW32C 173