Multi-Edit 9

By American Cybernetics, Inc.

Contents

Multi-Edit 9

What's New?

Enhanced HTML Support

Multi-Edit 9 contains **Full HTML 4 support**, including all HTML 4 tags and attributes. Also available is our newly updated HTML 4 online reference guide (now in HTML Help format).

- **Updated HTML tag database**, including all HTML 4 tags and attributes.
- **Miva tag database**
- **ColdFusion tag database**
- **JSP tag database**
- **New HTML online reference guide** (now in HTML Help format).
- **Enhanced tag editing** includes tabbed attribute dialogs.
- Integration with **CSE HTML Validator** for HTML Code Validation.
- Integration with **Topstyle** for editing cascading style sheets.
- **Improved tag highlighting.**
- **Improved embedded language handling.** Supports more languages and embedding methods, including ASP, PHP, Miva and COLDFUSION scripting. Other embedded scripting languages: SQL, TCL, REBOL, PYTHON, VHDL. Improved support for PERLScript, JavaScript and VBScript.

Hyperlink Support

- **Automatically parse URLS** like http://www.multiedit.com/, or ftp://, or mailto:, or file:, and identify it as a hyperlink by underlining it.
- **Launch URL** from within the file.
- Handles **exec:** to launch a program, and **macro:** to run a Multi-Edit macro.
- **Supports the following identifiers:** ftp, http, gopher, mailto, news, nntp, telnet, wais, file, prospero, exec, macro.

Added Tool Integrations

- **MS Visual Studio Integration:** syncs automatically to the Visual Studio editor. Contains Visual Studio browser and tagging support.
  - Function browsing and parameter tips using the BSC Browser add-on.
- **Oracle SQL Integration**
  - Compile SQL code using PL/SQL add-on
- **Macromedia's Cold Fusion Studio and Homesite Integration**
- **Added Version Control System support**
  - Support for CVS
  - Support for TLIB dll's
  - Auto adds subdirectories to associations

User Interface Improvements

- **History auto-completion** of text fields in dialog boxes.
- **New file open/save prompt.**
  - Complete "my computer" tree
  - File management features
  - Drag-and-drop
  - Configurable
  - Save and Restore of position
  - Drive Bar
- **Search improvements**
  - Tree list of all Searches/results in **two** selectable windows
  - Able to save Search history.
  - Separate search results lists.
  - Regular Expression drop-down list for insertion in Search field.
- **Tabbed vertical window (Navigation Pane) - similar to results window (Tools Pane).**
- **Improved Customize dialog - now uses tree format**
- **NEW Preview Pane supports file and tag previews.**

Improved Project Manager

- **New user interface** - Tree and list views for fast and easy navigation.
- **Improved FTP support** for sending project files to/from a remote system.
- **Project Quick List** for switching quickly between projects

Improvements to Multi-Tags

- Tags are now displayed on the Navigation Pane for fast access.
- "Show All Tags" option displays all tags contained in the tag file.
- Integration with the project manager and ability to scan all project files.
- View tags in other tag files using the drop down box.
- Scan tags using **wildcards.**

Power Editing Improvements

- **HTML from Code** allows you to create an HTML version of your source with syntax coloring!
- **Improved file compare with file merging**
- **Collapse on tags**
- **Multiple clipboards** - docked or floating buffer dialogs
- **Operator alignment**
  - Align all operators on multiple lines with a single keystroke
  - No more tabbing to line up equals "=" signs
- **New meta commands**
  - Generate Project List
  - Run Macro and substitute output

Added Language Support

- **Full support for:** SQL, PL/SQL, TCL, REBOL, COLDFUSION, PYTHON, VHDL, Verilog, C#, Matlab, Scilab, Progress, AutoLisp, HTMLScript.

More Features

- **NetCompile** will execute compiles on any remote system that supports a telnet interface. Map network drives with either SAMBA or NFS, or access them via FTP to create a complete remote editing environment.
- **Printing Improvements**
  - Color "Syntax" Printing
- **Desktop integration adds Multi-Edit to context sensitive menus**
- **Improved macro compiler**
  - Supports long filenames
  - Supports DLL imports up to 128 characters in length
- **COM Client/server support** for advanced integration of Multi-Edit to other products

New Updated Help

There is a new version of Multi-Edit's Help in HTML Help form. This will require IE 4.0 or greater to be installed and the HTML Help engine. The Help Engine will be automatically updated for you from the install. This is all pre-installed if you have Windows 98 or Windows 2000.

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 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 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

This topic provides reference information about where you can find specific components in the Multi-Edit setup, 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.

Feature	Default Install	Location in Setup	What's installed
Program Files	Yes	Program Files	Executable files, library's, and system macro binaries (*.mac, mew.mcl) necessary to run the Multi-Edit.
Program Help	Yes	Program Help	Main program help (Me.chm)
Macro Compiler	No	Macro Compiler, Headers, and Help	Cmacin.exe
Macro Headers	No	Macro Compiler, Headers,	System macro headers

		and Help	(*.sh)
Macro Help	No	Macro Compiler, Headers, and Help	Macro Help (Cmac.chm)
Macro Source	No	Macro Source	System macro source files (*.s)
Html Reference	No	HTML Reference Help	HtmlRef.chm
Cold Fusion/Homesite Integration	No	Macromedia Integration	Library and source files necessary to integrate. (Macromedia folder)
Delphi Integration	No	Borland Integration | Options	Library files and source files necessary to integrate. Dependent on version of Delphi selected (Borland folder)
C++ Builder Integration	No	Borland Integration | Options	Library files and source files necessary to integrate. Dependent on version of Delphi selected (Borland folder)
Visual Basic Integration	No	Microsoft Integration | Options	Library files and source files necessary to integrate. (Microsoft folder)
Visual C++ Integration	No	Microsoft Integration | Options	Library files and source files necessary to integrate. (Microsoft folder)
Visual C++ Browse Support	No	Microsoft Integration | Options	Library files and source files necessary to integrate. (Bsc Browser folder)
Watcom C++ Integration	No	Watcom Integration	Library files and source files necessary to integrate. (Watcom folder)
Oracle Integration	No	Oracle Integration	Library files and source files necessary to integrate. (PL_Sql folder)
TopStyle Integration	Yes	Program Files	Library files and source files necessary to integrate. (TopStyle folder)
CSE HTML Validator Integration	Yes, if CSE validator installed	Program Files	Library files and source files necessary to integrate. (CSEValidator folder)

Upgrading From Previous Versions

Upgrading Existing Multi-Edit Installations

All upgrade installations require Multi-Edit 9 to be installed in a different directory than the existing install. However, Multi-Edit 9 can keep your customizations from all Multi-Edit for Windows products by copying your configuration files to the new installation directory and updating them 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, your new Multi-Edit 9 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 8 Menus in Multi-Edit 9

File menu

Multi-Edit 8 command	Changes to this in Multi-Edit 9
Information	Renamed **Properties**

Block menu

Multi-Edit 8 command	Changes to this in Multi-Edit 9
Copy Block	Removed from menu. You can use Cut/Paste as default.
Move Block	Removed from menu You can use Cut/Paste as default.
Delete Block	Moved to the **Edit | Block Operations** menu.
Block Operations	Moved to the **Edit** menu.
Save Block to Disk	Moved to the **File** menu
Indent	Moved to the **Edit | Block Operations** menu
Undent	Moved to the **Edit | Block Operations** menu.
Math Operation	Moved to the **Edit** menu.
Window Copy	Removed. Use the Cut/Paste combination by default. Can still be used if enabled in command map.
Window Move	Removed. Use the Cut/Paste combination by default. Can still be used if enabled in the command map.
Mark Lines of Text	Moved to **Edit** menu.
Mark Columns of Text	Moved to **Edit** menu.
Mark Streams of Text	Moved to **Edit** menu.
End Marking	Moved to **Edit** menu.
Turn Marking Off	Moved to **Edit** menu.
Persistent Blocks	Moved to **Edit** menu.

Project menu

Multi-Edit 8 command	Changes to this in Multi-Edit 9
Set	Removed. Replaced by using either **Open Project** or **Select Project**
View	Moved to **View** menu, item **Project**
Options	Renamed to **Properties**

Macro menu

Multi-Edit 8 command	Changes to this in Multi-Edit 9
List all Globals	Consolidated in the **View | System** command.
List all Macros	Consolidated in the **View | System** command.

Tags menu

Multi-Edit 8 command	Changes to this in Multi-Edit 9
List tags	Consolidated in the **View | Tags** command.
Browse current file	Consolidated in the **View | Tags** command. Still can be accessed if setup in the command map.

New Menu Commands in Multi-Edit 9

The following summarizes the commands that are new in Multi-Edit 9 from Multi-Edit 8.

Edit menu

This Multi-Edit 9 command	Allows you to
Block Operations | Align Operators	Line up operators on multiple lines to the same indent level

View menu

This Multi-Edit 9 command	Allows you to
View Menu	Brings up the appropriate view

Project menu

This Multi-Edit 9 command	Allows you to
Open Project	Open a project file (*.mep)
Create Project	Create a new project file
Select Project	Select a project to open from a quick list
Close Project	Closes the current project
Notebook	Open the project specific notebook

Tools menu

This Multi-Edit 9 command	Allows you to
Create HTML from Code	Takes the current file and generates a syntax highlighted version in HTML

Help menu

This Multi-Edit 9 command	Allows you to
Command Map Report	Detailed report of all Keymap, Toolbar and menu entries.
Tip of the Day	A random tip about Multi-Edit

Installation Requirements (Hardware/Software)

Multi-Edit 9 requires Microsoft Windows 9x/2000/NT 4.0 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 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

Using a Network Version of Multi-Edit offers the ability to run from one installed copy of the product while allowing each user to maintain his own configurations, such as key mappings, menus, toolbars and options.

To install the network version of Multi-Edit you will need to enter the Network Code at the Get Registration Information setup screen when installing Multi-Edit. In the Select Components setup screen ensure that the Program Files is checked and the number of Users is correct. Multi-Edit will then install the needed components to establish a network setup.

The users are configured by the use of Environment Variables. These variables are normally set in some type of login script. However, they can also be setup in the AUTOEXEC.BAT under Windows 95/98 or in the Control Panel under Window NT/2000.

We will discuss how to set these in the next section.

Option One (Network Configurations)

This is the least secure but easiest to maintain method. Each user will be assigned a directory beneath the directory where the *Multi-Edit 9.0* install is located with the name of XXX.usr (XXX represents the three letter ID that the user set up on his workstation). This is great since all the directories are automatically created by program, although the users must have full access rights on the network for the directory that contains the *Multi-Edit 9.0* install.

Multi-Edit 9.0 Network Install Directory (Full Access Rights)

Xx1.usr
Xx2.usr
Xx3.usr

Environment variables

ME9_ID - set to the unique three letter ID for each user.

ME9_LOG – Set to one if you wish to have the user prompted for the unique id every time.

If the ME9_LOG variable is used with ME9_ID then the user will be prompted, however if they cancel prompt the ME9_ID will be used.

Option Two (Network configurations with separate user directory)

This option is structurally the same as Option 1 except the user configurations are kept in a different directory than the *Multi-Edit 9.0* Network Install Directory. This way the *Multi-Edit 9.0* Network Install Directory can be set up with READ ONLY rights to protect the program from accidental deletions, etc. The user directories are then automatically created under a separate directory that will have full access rights.

If you are a previous user of Multi-Edit and you wish for your users to get updated configurations, then you have two choices. One, point the User Directory Path at the same directory used for Multi-Edit user directories. If you do this, the previous install configurations will no longer be usable. Or two, setup a new directory and copy the previous install configurations to this new directory before the users run their client setups.

Multi-Edit 9.0 Network Install Directory (READ ONLY access rights)

User Directories (Full access rights)

Xx1.usr
Xx2.usr
Xx3.usr

Environment variables

ME9_ID - Set to the unique three letter ID for each user.

ME9_USERS_DIR – Set to the user directory.

ME9_LOG – Set to 1 if you wish to have the user prompted for the unique id every time.

If the ME9_LOG variable is used with ME9_ID then the user will be prompted, however if they cancel prompt the ME9_ID will be used.

Option Three (User selected configuration directories)

This option keeps each user's configurations anywhere he specifies. One user could select to keep them on a local drive in "C:\me9" while another could put them on a Network directory he has access to. It does not require an ID, just the directory name. This is the most configurable method for the user, but does not allow easy management of the configurations, as they may not be accessible to the administrator.

Environment variables

ME9_CFG_DIR – contains the directory for the each user.

Once the option selection is made, the server installation will be copied along with a Client Setup Program called NETSETUP.EXE. Each user must run the program in order to install and set up the workstation he is working from.

Setting Environment Variables

The environment variables can be set in several different ways. The easiest and most flexible is to set them in a network login script. This allows the user to login on any client and have access to their configuration. It is also possible to set them up in the AUTOEXEC.BAT file. In Windows 9x/2000, this is the only method if you are not using the login script. Under Windows NT there are several methods one is to use the System Control Panel Environment tab. Put them in the User variables section if you wish the setting to be based on each logged in user. Put them in the System variables section if you wish them to be System wide.

You will need Administrator access to set the System variables.

In addition, you may set them in the AUTOEXEC.BAT for system wide settings, although this is not the preferred method.

Client Setup

Each client, before using the Network version of *Multi-Edit 9.0*, must run the NETSETUP.EXE Client Install Program. This install can be found in the *Multi-Edit 9.0* Server Install Directory. Before running the NETSETUP program, make sure that no version of *Multi-Edit 9.0* is currently running. The install will not complete correctly if *Multi-Edit 9.0* is running. The setup will verify this with you in the first dialog. Next, you will be asked to give the directory where the server install for *Multi-Edit 9.0* is located. Once this is selected, you can select the options for the client.

Before running the client setup the Environment variables will need to be setup for the user since the client setup will start Multi-Edit to finish the install. In addition, if you wish to update your previous Multi-Edit configuration, the files that were in the previous install configuration directory will need to be copied to the directory where the 9.0 configuration files will be kept before running the client setup.

Year 2000 Compliance

Multi-Edit (all versions) does not internally use dates as a function of the program. Multi-Edit does contain functions that output dates into text files. In this context, the following information applies:

Multi-Edit: Outputs date text based on operating system settings; fully year 2000 compliant.

License Agreement

Multi-Edit is protected under United States copyright law. Copies may be made for the purpose of backup ONLY. If this software was purchased as a single user copy, the software may be moved from one computer to another and may be used by more than one person. However, THIS SOFTWARE MAY NOT BE USED ON MORE THAN ONE COMPUTER, OR BY MORE THAN ONE PERSON, AT THE SAME TIME unless covered under a separate License Agreement with American Cybernetics, Inc., the terms of which Agreement supersede this Single-User License Agreement. To obtain a multiple-user site or network license, contact American Cybernetics, Inc.

Limited Warranty

American Cybernetics warrants the software media in this package for a period of 30 days regardless of reason. If, in this period, the software media is found to be defective, it may be returned to us for free replacement. American Cybernetics makes no representations or warranties as to the merchantability or fitness of this product to a particular purpose. There is no other warranty, express or implied.

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 American Cybernetics web site 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, fax or telephone.

One of the easiest ways of contacting ACI'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.

This feature does require an Internet connection.

We sincerely hope you never have to call for technical support on Multi-Edit. 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 and letter that you're using (found under **Help | About**).

Phone Support

ACI Voice Number: (480) 968-1945
Monday-Friday 7:00 AM to 4:00 PM (MST, Arizona)

By registering your copy of Multi-Edit, you are entitled to 90 days of free phone support from your first support call. Registered users within their 90 day phone support window may call the number above to receive free phone support. If you are outside the 90 day support period you can **email** or **fax** ACI for technical assistance. Please have your serial number and revision number (found under **Help | About**) ready when you call.

FAX Support

ACI FAX Number: (480) 966-1654

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: 8.0j). Revision letter and 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.

Online Web Support

E-mail: tech@multiedit.com
Anonymous FTP: ftp.multiedit.com
World Wide Web: http://www.multiedit.com/

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

ACI 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 ACI'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.*

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 setup to allow access to any language specific help provided by the language vendor by configuring the filename extension setup. Within this setup, 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 " to use 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 files 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

The way a file is displayed in Multi-Edit can be modified by using one or more of the following modes.

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 the 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

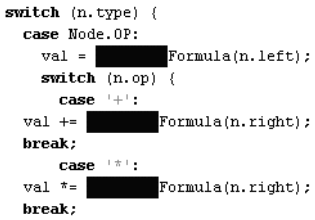
Searching Within Multi-Edit

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 to 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** 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; this will enable the 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 positions 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.

Templates -

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 FC 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 are all files that are contained in or are children of the project root folder. The Project Root folder also, will 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.

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 include things such as 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 files 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 the 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 languages 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), and the configuration of **helper applications**.

Multi-Edit now 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.

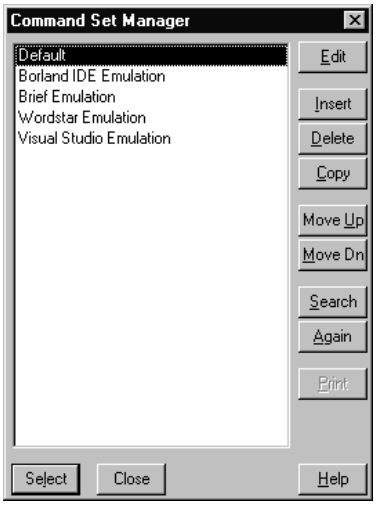
When you load a Command Set, you load all of the above-mentioned items for the Command Set. If you create a new Command Set, you will have to create Keys/Commands, Toolbars and Menus. Therefore, we suggest that you copy an existing Command Set and modify it to fit your needs rather than create your own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

*A Command Set is more than just keys; it contains menus and toolbars. 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. It will be copied to the Config directory if any changes are made.

Init Macro 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.

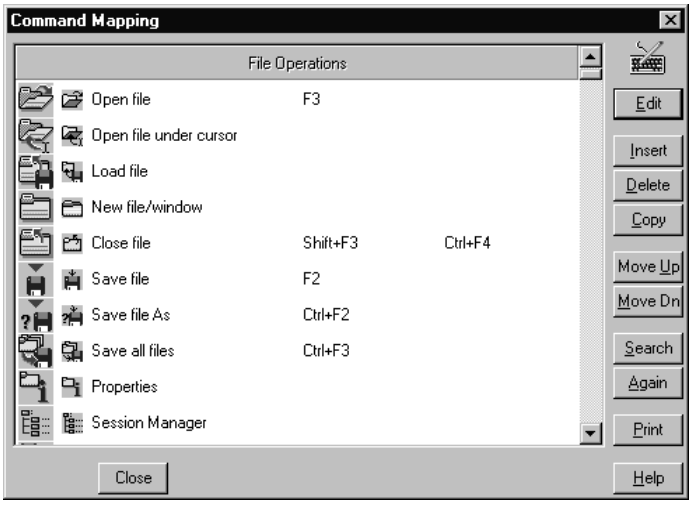
*To create a key command report for the current command set being used, click on **Help | Command Map Report**.*

*To see what command is associated to 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 in a title (descriptive enough for you) 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 they type of 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, perhaps. 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 command 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 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
The help search string to use if help is requested for this command or icon is listed here. This does not refer to creating a command entry to access a help file. Help search strings are entered in this form: [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 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:

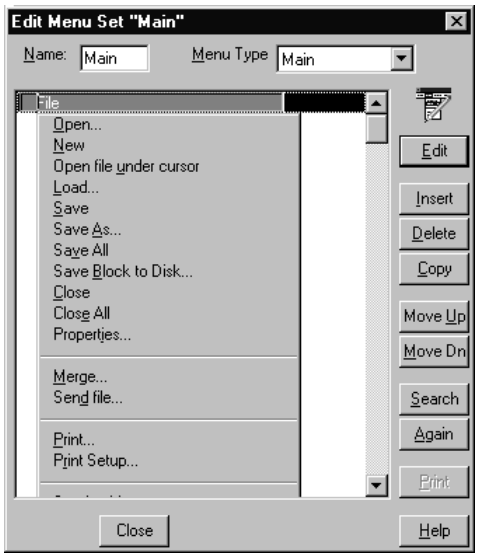
```
WLIST_WCMD_ID    =  Window List
CLOSE_WCMD_ID    =  Close Window
MIN_WCMD_ID      =  Minimize Window
MAX_WCMD_ID      =  Maximize Window
RESTORE_WCMD_ID  =  Restore Window
MOVE_WCMD_ID     =  Move Window
SIZE_WCMD_ID     =  Size Window
```

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 **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 dialogs 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 (you may wish to widen this Help window for a better view):

```
#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

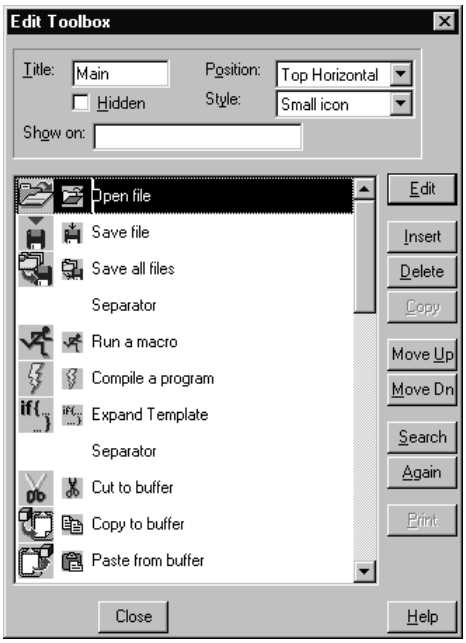
Customizing Toolbars

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 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 **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 to 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

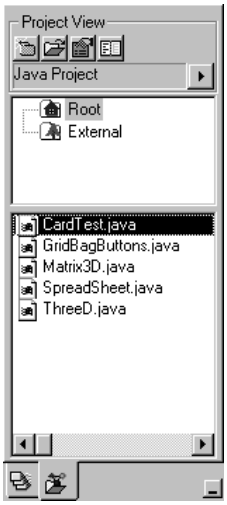
The Tools Pane is a tabbed pane at the bottom of the Multi-Edit screen that holds tabs for Bookmarks, FTP Results, Compiler Results, Collapse Mode, Find Results and Background Task List.

The Tools Pane displays when first used. 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.

Status Bar

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

From right to left, 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.

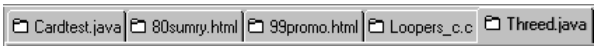
Num
Displays the Num Lock status. Pressing this button (or the <Num Lock> key) toggles the keyboard Num Lock status.

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.



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 with the following options:

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.

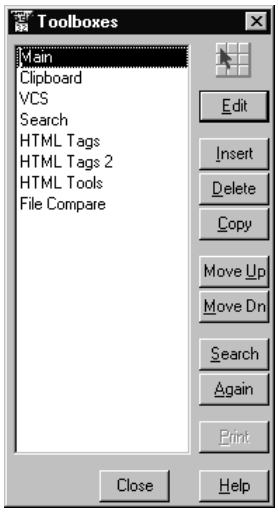
Fonts, font colors, and tab colors can be customized as well.

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.



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 up, 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

Multi-Edit supports "drag and drop". 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 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**.**

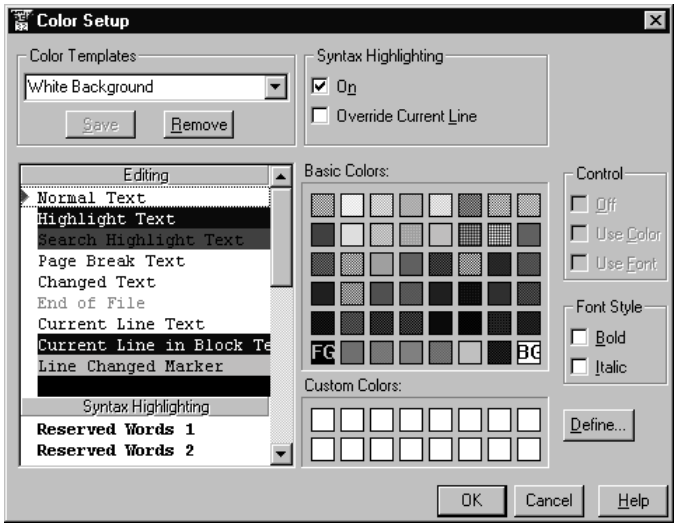
The editing window colors within Multi-Edit are also user configurable and can be saved as a **color template**. Color templates are extension specific and are used to define colors such as reserved words, changed text, and search highlight text. Each color setting includes a foreground and background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

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

1. Select Customize from the Tools menu and click on the Customize tab. Press the Colors button to display the Colors Setup dialog box.
2. Select the screen display item whose color you want to change. FG (for foreground) and BG (for 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:



Color Templates
This area allows you to define several color templates that you can select and use at any time, depending on your editing needs. You can **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. Otherwise, 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.

Control
Multi-Edit allows you to set some special options when defining your window colors:

Use Color: Allows you to instruct a Syntax Highlighted item to use its currently defined color or use the normal text color instead.

Use Font: Allows you to use the Font Style option button field to set a bold or italic font for this Syntax Highlighted item.

Off: 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 to 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, you will find yourself needing to use the ASCII character set. This is especially true when you are using DOS compilers and using high ASCII value characters (characters above ASCII 127 - like special International characters or linedrawing 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, if 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's 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.

Working in Multi-Edit

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 by only a linefeed character (0x0A- LF) and Apple machines use only the carriage return character (0x0D - CR). Some files use a combination of these line terminators, or even use a different character altogether. The current version of Multi-Edit can only deal with DOS and UNIX style text files and treats the other files as binary files, i.e. a file with no line terminating character(s).

When loading a file in Multi-Edit the file type must first be determined. 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 setup data for the extension or the *File type* option is set to *Auto Detect* then Multi-Edit will load the first 4096 characters and count all CRLF 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 file type is defined by the user, 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 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 already shown windows, 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

Split allows you to divide your 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 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 resize 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

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

Previous

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

Adjacent Window

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

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

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

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

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; see the Window Behavior Setup dialog for more details.

Clicking on a tab is all that is required to switch to the selected window. This is much easier that 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 **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 currently 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.

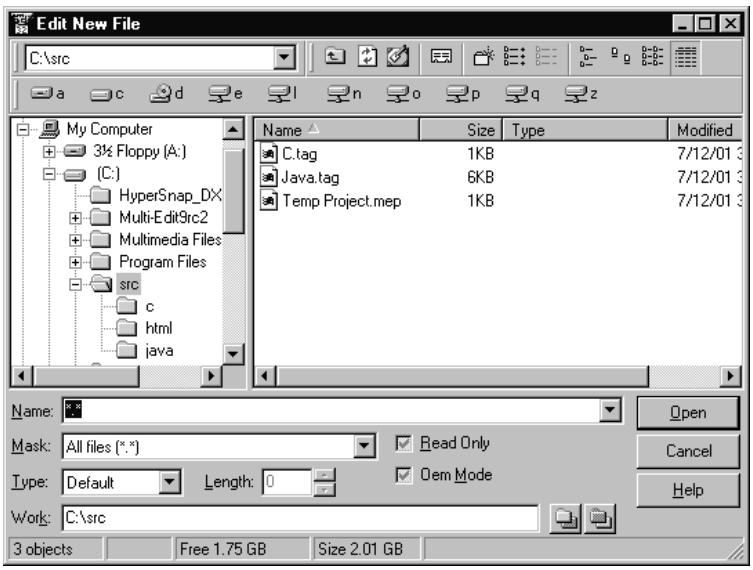
Key Commands

Use the **Help | Command Map Report** command to generate a list of all defined command 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 layout as well.

Files

Opening and Saving Files

A new File Prompt has been implemented for Multi-Edit 9, 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 – Created 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 in 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 and 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.

Network button
Press the Network button to display the Map Network Drive dialog box, in which you can select a path and map it to a specific drive. Opt to Connect As and Reconnect at Logon, among others.

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 *does* the read/write status of that file.

Changing the Locked/Read Only setting for a file located on a network server can yield undesirable results.

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.

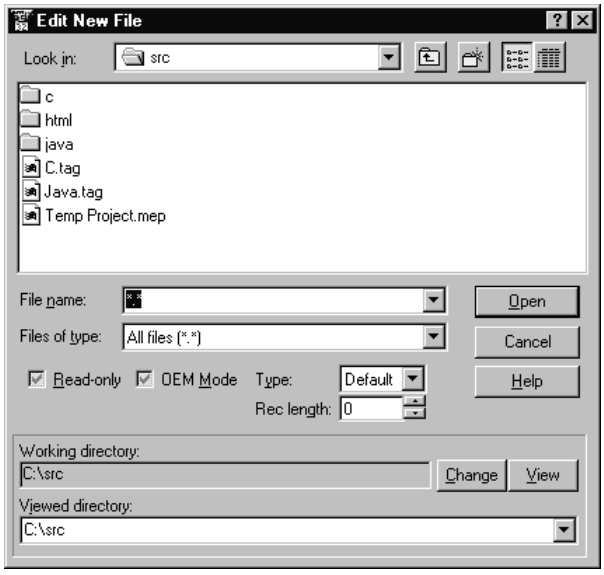
Binary
Binary files have no line terminators-they are continuous streams of data; however, for easier reading, binary files use a Binary Record Length that can be specified on the fly or for a particular extension in Filename Extension Setup.

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 files 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.

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.

```
Seconds of keyboard inactivity
0 = OFF
Save status
Absolute interval in seconds
0 = OFF
Save status
```

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.

Bookmarks	Show the list of bookmarks.
Collapse	Show the collapse dialog.
Find List 0	Show the first Find list.
Find List 1	Show the second Find list.
Ftp Results	Show the results of the last Ftp session.
Output	Show the results of the last compile.
Paste Buffers	Show the paste buffers.
Preview Pane	Show the preview pane.
Tasks	Show the list of background tasks.

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. 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 Navigation pane entries.

Code Manager	Show the Weblair code manager dialog.
Project	Show the Project manager dialog.
System	Show the System dialog, which show the list of global variables and loaded macros.
Tags	Show the Tags dialog.

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 setup different kind 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. This is done by selecting one of the Type radio buttons.

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 is only find simple text it can do so at a much faster rate than the other types of searches.

Using the Regular expressions type is useful for locating variable length text and/or multiple words. When this option is checked, a number of characters take on special meaning. See the section on Regular Expressions for more details about what each of the supported characters do. If the string entered into the Search for field does not contain any of the special characters then the search engine will revert to doing a literal search. The time it takes to do a search with regular expressions depends upon the regular expression. To make it easier for using regular expressions, an Alias button is available which allows associating commonly entered regular expressions to a name and quickly entering them into the Search for field. In addition, there is a button which shows the complete list of each of the special regular expression characters with a short description and clicking on one will insert it into the Search for field at the 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 between specific columns by enabling the Columns option and setting the Start and Stop column numbers. Usually a search only search the current window and when it finds a match it stops and highlight the matching text. If the Search all windows options is checked then 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 is needed 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 support two and is selected by picking one from the File list option. The reason for multiple Find lists is so that a File find could be done and then allowing searching each found file for some other text and not needing to navigate through the tree every time to load the next found file. See the Find List topic for how to use it.

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 replace 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 search, 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 Seach 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 are sophisticated wildcards. They provide the ability to deal with unprintable characters like 'beginning of line' or 'end of line'; specific sets or 'classes' of characters, strings of unknown characters in a single search or replace pattern, and the ability to search across multiple lines.

Multi-Edit provides support for two styles of Regular Expressions. The Multi-Edit **Classic style**, and a **Unix compatible style**, based on the X/Open standard for EXTENDED expressions. Maximal and minimal matching is supported in the UNIX mode.

In addition, regular expressions may now span lines. So, if given the following text in a document:

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

For a regular expression search string of "Test?*$Wow?*$What", a match will be found.

Regular Expressions provide a very useful and powerful way of locating text in files but it comes at a cost. Unless you have used regular expressions for awhile and/or use them regularly, you usually have to stop and think about what the different characters mean or remember that great regular expression you developed to find all the local variables in your program. Also regular expression strings can get quite long and leaving one character out or in the wrong place will cause the search to fail. In addition, Multi-Edit supports two different style of regular expressions, Classic and Unix styles that you need to remember. There is now a solution to the above problems and that is the addition of regular expression aliases.

Regular expression aliases provide a way for you to assign an expression to an easy to remember name and the next time you wish to use that expression just type in the new name and it will be converted over to the defined expression when the search is done. There are a number of predefined aliases that are used by some of the system macros but you can add your own to this list. All regular expression aliases are defined as <alias> where “alias” is the name of the defined alias. Filename and path aliases support long filenames with spaces.

Regular expression aliases have been incorporated into some of the system macro the search macros being one of them. Users that write their own macros can also incorporate the use of regular expression aliases in their macros, see the REGEXP.SH file for the needed information. Basically, all that you need to do is to call the ReTranslate macro to translate or expand the regular expression aliases before you do your search. Also calling ReSelectAlias will display a Regular Expression Alias List Dialog that will present a list of currently defined aliases that the user can select and will be returned to you.

Classic Regular Expressions

Search String Expressions

*****	match 0 or more occurrences of the previous character or expression
+	match 1 or more occurrences of the previous character or expression
?	match any character
[]	match class of characters
[~]	match any character except the enclosed class
%	match beginning of line
$	match end of line
{ }	encloses a group of 1 or more expressions
|	match previous OR next character or expression
@	match (or replace) next character literally
@a	0x07 BEL
@b	0x08 BS
@f	0x0C FF
@n	0x0A LF
@rb	0x0D CR
@t	0x09 HT
@v	0x0B VT
@xHH	HH = string of 2 hex digits

Replace String Expressions

$	Insert a carriage return
%	Delete a character
&	Insert the original found text
^	Place cursor at this position in the replaced text
#n	Insert text matched by group number 'n'

Unix Regular Expressions

Search String Expressions

^	Beginning of line
$	End of line
*****	Maximal match zero or more of previous char
+	Maximal match one or more of previous char
@	Minimal match zero or more of prev char
#	Minimal match one or more of prev char
.	Match any char
[set]	Match any char in set
[^set]	Match any char NOT in set
|	"Or" char
( )	Encloses a group of expressions
\a	0x07 BEL
\b	0x08 BS
\f	0x0C FF
\n	0x0A LF
\r	0x0D CR
\t	0x09 HT
\v	0x0B VT
\xHH	HH = string of 2 hex digits
****	Match next char literally

Replace String Expressions

$	Insert a carriage return
%	Delete a character
&	Insert the original found text
^	Place cursor at this position in the replaced text
\#	Insert text matched by group number 'n'
\a	0x07 BEL
\b	0x08 BS
\f	0x0C FF
\n	0x0A LF
\r	0x0D CR
\t	0x09 HT
\v	0x0B VT
\xHH	HH = string of 2 hex digits

Minimal vs. Maximal Closure

Minimal matching (@ or #) matches the LEAST number of characters required to match the expression. Maximal matching (* or +) matches the MOST # of characters it can BEFORE completing the expression.

For example, using the following code fragment:

```
 if ( ( wrap_stat && doc_mode ) ||
format_stat && indent_style) ) {
          use_format_line = True;
   }
```

If the search string specified is "\(.*\)", using maximal matching, then the expression found would be:

```
( (wrap_stat && doc_mode) ||
  (format_stat && indent_style) )
```

If the search string specified is "\(.@\)", using minimal matching, then the expression found would be:

```
( ( wrap_stat && doc_mode)
```

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 <Ctrl+Alt+Up> or <Ctrl+Alt+Dn> keys available in most keymaps or by using <Shift> or <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 new search is done. There can be up to three levels for each search depending upon the options selected. The top-level entries, Find nodes, contain the information about the search such as the string being search for, the file mask and the total found count. The second level entries, File nodes, contain the name and path of the file where a match was found. It can also contain the number of time the string was found in the file. The third level entries, 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 either use 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 the 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 setup 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 and 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, that allow doing all of the above with the mouse.

A special menu button is provided that 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 **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:

Esc:	Cancels the search and returns the cursor its original position before the search began.
Alt+P:	Searches backward for the next occurrence of the current search string.
Alt+N:	Searches forward for the next occurrence of the current search string.
Backspace:	Removes the last character from the search string and returns the cursor to the previously found string.
Alt+W:	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 "ear" in the middle of the word "search").
Alt+C:	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.

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.

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.

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

Find
  Search for:

Replace
  Search for:
  Replace with:

File find
  Search for:

File replace
  Search for:
  Replace with:

Hilite
  Highlight:

Examples

Search for the '$' character

EXAMPLE 1:

Search for the '$' character.

SEARCH EXPRESSION:

```
(Classic Style)      @$

(Unix Style)         \$
```

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

```
(Classic Style)
   @ Find literal.
   $ Search for '$' char


(Unix 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)         ^ $ * + @ # . [ ] | ( ) \
```

Find a parenthesis set

EXAMPLE 2:
Find a parenthesis set.

SEARCH EXPRESSION:

```
(Classic Style)      (?*)


(Unix Style)         /(.*/)
```

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

```
(Classic Style)
(   Search for an open parenthesis
?* Followed by any # of occurrences (>=0) of any character
)   Followed by a close parenthesis


(Unix Style)
/( Search for an open parenthesis
.* Followed by maximal match of any # of occurrences (>=0) of any character
/) Followed by a close parenthesis
```

ENGLISH TRANSLATION:

Finds parenthesis sets by searching for an open parenthesis followed by any number of characters, of any kind, followed by a close parenthesis.

For UNIX 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:

(Classic Style) `{"?*"}|{'?*'}`

(Unix Style) `(".*")|('.*')`

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)		
	{	Begin group
	"	Search for double quote
Group 0	**?***	Followed by any # of occurrences (>=0) of any character
	"	Search for double quote
	}	End group
	|	-OR-
	{	Begin group
	'	Search for single quote
Group 1	**?***	Followed by any # of occurrences (>=0) of any character
	'	Search for single quote
	}	End group
(Unix Style)		
	(	Begin group
	"	Search for double quote
Group 0	**.@**	Followed by minimal match of any # of occurrences (>=0) of any character
	"	Search for double quote
	)	End group
	|	-OR-
	(	Begin group
	'	Search for single quote
Group 1	**.@**	Followed by minimal match of any # of occurrences (>=0) of any character
	'	Search for single quote
	)	End group

ENGLISH TRANSLATION:
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 Style users, take care to select the proper minimal/maximal matching character. See Understanding Minimal vs. Maximal Closure for more information.

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_]
```

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

```
(Classic Style)
   %              Search for beginning of line
   |                        -OR-
   [~a-z0-9_]      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-
   [^a-z0-9_]      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
```

ENGLISH TRANSLATION: Search for the beginning of the line OR search for the first occurrence of a blank followed by a letter, digit, or underscore.

Searches for the following operators

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

SEARCH EXPRESSION:

```
(Classic Style)     [=><!@|@&]


(Unix Style)        [=<>!\|&]
```

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

```
(Classic Style)
[     Begin set.
=     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 set.



(Unix Style)
[     Begin set
=     Search for '=' character
<     Search for '>' character
>     Search for '<' character
!     Search for '!' character

\|    Search for '|' character literally.  Since the | is a wildcard character
      a \ 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 set
```

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 the '&' as a replace expression wildcard character.

CURRENT STRING: Multi-Edit
DESIRED RESULT: (Multi-Edit)

SEARCH EXPRESSION:
(Classic & UNIX Style) `Multi-Edit`
REPLACE EXPRESSION:
(Classic & UNIX Style) `(&)`

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENT (S):

(Classic and Unix Style)

(	`Replace with open parenthesis`
&	`Followed by the original found string`
)	`Followed by a close parenthesis`

Search for a blank line and delete it

EXAMPLE 7:
Searches for a blank line and deletes it.

SEARCH EXPRESSION:

```
(Classic Style)     %$


(Unix Style)        ^$
```

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

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


(Unix Style)
   ^     Search for the beginning of a line
   $     Followed by the end of a line
```

REPLACE EXPRESSION:
(Classic & Unix Style) %

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENT(S):
(Classic & Unix Style) % Delete a character

ENGLISH TRANSLATION:
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 ({ }) to define groups within a search expression adds power and flexibility to Multi-Edit's Search and Replace operation. 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.

EXAMPLE 8:
Searches for the first occurrence of a ';' and deletes the rest of the line.

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

DESIRED RESULT: `goto_line(1);`

SEARCH EXPRESSION:

(Classic Style) `{?*;}{?*}`

(Unix Style) `(.@;)(.*$)`

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)		
Group 0	**{**	Begin group
	?*	Search for any # of occurrences (>=0) of any character
	;	Followed by a semi-colon
	}	End group
Group 1	**{**	Begin group
	?*	Followed by any # of occurrences (>=0) of any character
	}	End group
(Unix Style)		
Group 0	**(**	Begin group
	.@	Search for minimal match of any # of occurrences (>=0) of any character
	;	Followed by a semi-colon
	)	End group
Group 1	**(**	Begin group
	.*	Followed by maximal match of any # of occurrences (>=0) of any character
	$	Followed by the end of line
	)	End group

ENGLISH TRANSLATION:
Find any character that occurs any number of times, (including 0 times) , that is followed by a semicolon. This constitutes group 0. Following the previous match, find any character that occurs any number of times. This will constitute Group 1.

REPLACE EXPRESSION:

(Classic Style) `#0`

(Unix Style) `/0`

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENT (S):

(Classic Style)

#0 replace found string with group 0 from search string.

(Unix Style)

\0 replace found string with group 0 from search string.

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\((.#),(.#)\);
```

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)

	gotoxy(	**Find the string 'gotoxy**
	⌈ **{**	Begin group
Group 0	**?+**	Followed by one or more occurrences of any char
	⌊ **}**	End group
	,	Followed by a comma
	⌈ **{**	Begin group
Group 1	**?+**	Followed by one or more occurrences of any char
	⌊ **}**	End group
	)	Followed by a close parenthesis
	;	Followed by a semi-colon

(Unix Style)

	gotoxy	**Find the string 'gotoxy'**
	\(	Followed by a open parenthesis
	⌈ **(**	Begin group
Group 0	**.#**	Followed by minimal match of one or more
	|	occurrences of any character
	⌊ **)**	End group
	,	Followed by a comma
	⌈ **(**	Begin group
Group 1	**.#**	Followed by minimal match of one or more
	|	occurrences of any character
	⌊ **)**	End group
	\)	Followed by a close parenthesis
	;	Followed by a semi-colon

ENGLISH TRANSLATION:
Finds 'gotoxy(', followed by one or more occurrences of any character (This would be the first parameter or group 0) followed by a comma (This is the comma that separates the two parameters), followed by the second parameter, (or group 1) which consists of 1 or more occurrences of any character, followed by a close parenthesis and a semi-colon.

REPLACE EXPRESSION:

(Classic Style)
```
gotoxy(#1,#0);
```

(Unix Style)
```
gotoxy(\1,\0);
```

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENT (S):

(Classic Style)

gotoxy(Replace with 'gotoxy('

#1 Followed by group 1 from search expression

, Followed by a comma

#0 Followed by group 0 from search expression

); Followed by ');'

(Unix Style)

gotoxy(Replace with 'gotoxy('

\1 Followed by group 1 from search expression

, Followed by a comma

\0 Followed by group 0 from search expression

); Followed by ');'

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) `{;} *{[~ ]?+;}`

(Unix Style) `(;) @([^ ].#;)`

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:

(Classic Style)		
Group 0	**{**	Begin group
	;	Search for semi-colon
	}	End group
	' *'	Followed by any # of occurrences (>=0) of a space
Group 1	**{**	Begin group
	[	Begin class
	~	Match any char except those that follow in this class
		(A space)
	]	End class
	?+	Followed by one or more occurrences of any character
	;	Followed by a semi-colon
	}	End group
(Unix Style)		
Group 0	**(**	Begin group
	;	Search for semi-colon
	)	End group
	'@'	Followed by minimal match of zero or more occurrences of a space
Group 1	**(**	Begin group
	[	Begin class
	^	Match any char except those that follow in this class
		(A space)
	]	End class
	.#	Followed by minimal match of one or more occurrences of any character
	;	Followed by a semi-colon
	)	End group

ENGLISH TRANSLATION:
Find the first occurrence of a semi-colon on a line. This will constitute Group 0. Following this, find any number of occurrences of a space. Then find Group 1, which consists of one or more occurrences of any character, (excluding a space) followed by a semi-colon.

REPLACE EXPRESSION:

(Classic Style) `#0$#1`

(Unix Style) `\0$\1`

DETAILED EXPLANATION OF REPLACE EXPRESSION COMPONENTS:

(Classic Style)

#0	Replace found string with group 0
$	Followed by a carriage return
#1	Followed by group 1

(Unix Style)

\0	Replace found string with group 0
$	Followed by carriage return
\1	Followed by group 1

SUMMARY:
In this example, two groups are defined. The first consists of the first occurrence of a semi-colon 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

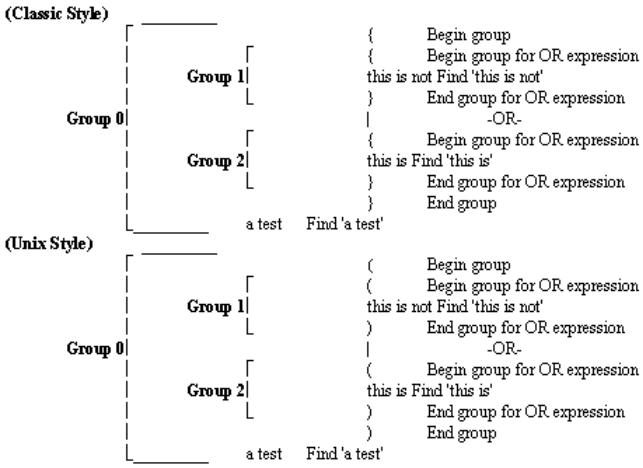
EXAMPLE 11:
Searches for the phrases 'this is a test' OR 'this is not a test' and replaces 'a test' with 'a pizza'

```
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 is not}|{this is}} a test


(Unix Style)        ((this is not)|(this is)) a test
```

DETAILED EXPLANATION OF SEARCH EXPRESSION COMPONENTS:



ENGLISH TRANSLATION:
Find either 'this is not' OR 'this is' (which constitutes group #0), followed by 'a test'.

REPLACE EXPRESSION:

```
(Classic Style)     #0 a pizza


(Unix Style)        \0 a pizza
```

ENGLISH TRANSLATION:
Replace the found string with group 0 from the search expression followed by 'a pizza'.

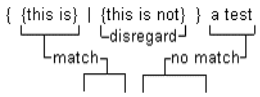
SUMMARY:
There is a very important detail to be noted in the search expression of this example:

'this is not' came before 'this is'.

This was done intentionally. Since 'OR' expressions are evaluated from left to right, close attention must be paid to the order in which its components appear. Once any component of the expression is found, none of the remaining components will be evaluated for a match.

If 'this is' had been placed before 'this is not' the following would result when encountering the line 'this is not a test':

SEARCH EXPRESSION:



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

The 'this is' portion of the line would be recognized as a match, terminating evaluation of the rest of the OR expression. The remaining component 'this is not' would never be evaluated. Following this, the next part of the expression is evaluated. This would be 'a test'. Since the found text contains 'not a test' it would not form a match.

You should 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+v 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 which 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 it's 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 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 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 on the desired ending point and 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 over 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.

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 in 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 in 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 pre-determined increment value.

Case C:

If at least two lines are marked.

There are numbers contained within in 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 in 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.

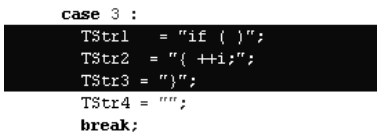
ConvertBlock
The ConvertBlock macro will prompt to convert a stream block to a line or column style block. All the Fill features listed above use this macro.

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.

Take, for example, the following code snippet:



Using Align Operators would produce the following result:

```
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:

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

will produce the following when Align Operators is used:

```
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 you comment at the point specified in your stream clock (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** to display line numbers along the left margin of the Editing Window.

```
003 #include <limits.h>
004 #include <float.h>
005
006 void multTables(void);
007 void floatingPointLoop(void);
008 void squares(void);
009 void numberSystems(void);
010
```

Center Line

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

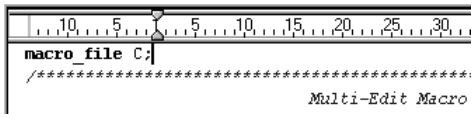
The Center Line command does not dynamically center text as you type it. To use the Center Line command correctly, first type the text you want to center, and then center the text. If you add to that text later, you may want to re-center it to correct for unbalanced space on either side of the centered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

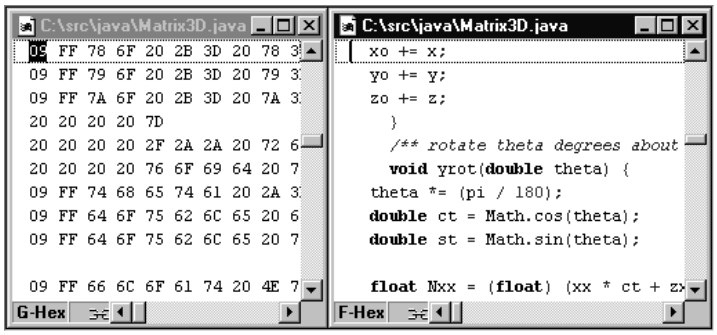
Formatting Lines

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.

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 of 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 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 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.

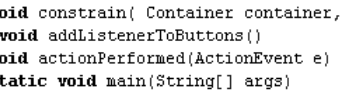
To view the collapsed text, click on the plus button. The text will be expanded, and a minus button will replace the plus button.

Example:

```
+  public void constrain( Container container,
+  private void addListenerToButtons()
-  public void actionPerformed(ActionEvent e)
   {
      if (e.getSource() instanceof Button)
      {
         String buttonLabel = e.getActionCommand();

         this.setTitle("You pressed the " + buttonLabel
```

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.

The collapse mode Tools Pane has the following fields:

Collapse Modes

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 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.

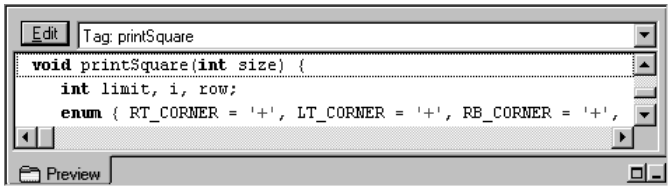
Exclude collapsed text on

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, Fortan, 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 select **Tags | 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 **Tags | Find again** will locate the next occurrence.

Tags | 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:

```
if( jx == 10 )            // @METAGS wow
{
  fprint("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 where 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.

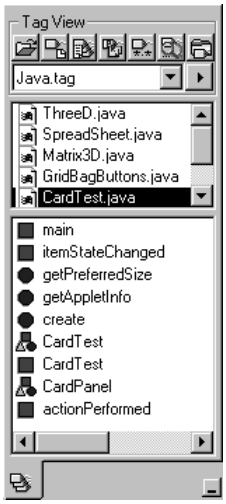
Project Scan
Scanning a project for tags will scan ALL files contained in the project. If a project has been created, click on **Tags | Scan Project**. Alternatively, if the Tag View is displayed, the **Scan Project** button may be used.

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 pressing the delete key or by right clicking on a 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-Tags Database (Scanning files)

Multi-Tags databases 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.

To return to a marked position, select **Search | Get Position from Marker Stack**. 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 check then only 10 mark for all 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 book marked
- 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 difference 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. Remove 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 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 setup for .DOC files. When invoked, a list box appears containing a list of compilers available for your currently 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 a 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 string field contains a description of the compiler or program that is being setup. This is what is shown in the Compiler/Program Type List Dialog.

Defaults

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 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 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 use of a macro or regular expressions, which is the suggested method.

Version 9 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 in 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 Setup 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 with out 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 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 is used when not file name was located.

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 then parsed for one or more of the following string 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.

F	File Name
L	Line number
C	Column number or position
M	Message
M+	Append to Message

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 would be to find another line that contains the information specified. In these strings x is replace with one of the type letters shown above.

/xP=str
The value of str would be one of the following 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 replace 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.

D#	Search forward # lines starting with the next line down
U#	Search backward # lines starting with the previous line up
F#	Search forward # lines
B#	Search backward # lines

/xS=str
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.

/xR=str
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 setup 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.

This is match group 3 because the <p> alias uses two match groups in its definition.

, - Match a comma.

<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 :.

.#) - 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:

/F=\0	Replace group 0 with /F=filename.
/L=\3	Replace group 3 with /L=lin num.
/C=\4	Replace group 4 with /C=col num.
/M=\5	Replace group 6 with /M=Error #: message.

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/M=\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 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:

/F=\0	Replace group 0 with /F=filename.
/L=\3	Replace group 3 with /L=lin num.
/M=\4	Replace group 6 with /M=Error #: message.

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. You can add your own compiler support; if you have trouble, contact Technical Support by email mailto:tech@multiedit.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. 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 | Options | Tools*** *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 is 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 9.0 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 setup these types of programs.

Examples:

```
(Compiler/Program Type Setup)
Type: Filter
Compile Macros
Pre: Compile^PreCmpFilter
Post: Compile^PostCmpFilter


Type: Filter (Col Block)
Compiler Macros
Pre:
Post: Compile^PostCmpFilter /BT=2


Type: Filter (Line Block)
Compiler Macros
Pre:
Post: Compile^PostCmpFilter /BT=1

Type: HtmlTidy Filter
Compiler 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 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 work 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 line will affect the ability to locate the next difference. To overcome this limitation, Multi-Edit 9.0 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 the 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 the line 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 again 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 number 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 which 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 a 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 there login name. Usually, "gin:" the last 4 chars of "Login:"

Password Prompt:
This is the text that precedes a user typing in there password. Usually, "word:" the last 5 chars of "Password:". Note, 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" 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. Common 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.

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 use 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 sting 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 navigate 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 setup 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 pattern data global 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 pattern data global. 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 setup, 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 to 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 support multiple indent styles. These are setup, 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 it 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 has 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 setup after selecting them but the Smart Indent Style requires additional setup. 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 for set these features. Languages that use this dialog are setup 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 controls out denting 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. 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: Ignore Format Line: 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:

Keystroke	Function
Tab	Inserts (or overwrites) a tab stop
R *or* r	Sets the right margin at the column where it's entered
Del	Deletes the character under the cursor
Backspace	Deletes the character to the left of the cursor
End	Moves the cursor to the end of the Format Line
Home	Moves the cursor to the beginning of the Format Line
Left Arrow	Moves you to the left on the Format Line
Right Arrow	Moves you to the right on the Format Line
Enter	Exits Format Line editing and saves the changes
Esc	Exits Format Line editing without saving changes

Expand to spaces: When pressing a tab, spaces are entered to fill the space. This overrides the default in **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 setup extension specific window colors. Selecting this will bring up the Window Colors dialog. See Color Setup.

Font
You may set up a specific font for each extension setup, 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 **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. 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.

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.

Addon 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 Setup
Allows you to specify and customize compiler setups for the extension you have selected.

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.

```
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, 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 fielgd 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 and, 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 **Addon 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:

```
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.

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 consists 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 and if it has the language features associated with it are initialized. The Filename Extension Setup dialog is where the extension specific information is setup 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 main or the context menus. Template are created and changed in the Edit Template dialog accessed from the Tools menu 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:

1) From the main menu select **Tools | Customize | Languages**
2) 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:

1) From the main menu select **Tools | Customize | Filename extension**
2) Select a defined extension and hit the Edit button or Select the Insert button and type a new set of file extensions
3) 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 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 /MEW/CONFIG subdirectory. A specific language template can be defined and associated with an extension just like a language record. This is done as follows:

1) From the main menu select **Tools | Customize | Filename extensions**
2) Select the '**...**' button beside the Template field
3) 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 ;'s in the Addon 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 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 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. (), {} or 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 that must be done.

1) Setup 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) Setup the Init and Match macro fields in the Language Setup dialog to point to the correct macros.

To setup the special pattern global variables currently requires an 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 setup 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=*str* where *str* is a list of all of the characters that can precede a construct word.

%E=*str* where str is a list of all of the characters that can come after a construct word.

%D=*char* where ***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 consists 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 flags 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. The %I= field is available in Mew80b+ releases.
- **_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. (Available in Mew80b+ releases)

%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. (Available in Mew80b+ releases)

%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, lets 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 begin 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 type 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.

```
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.

```
*****************************************************************( ldh )***/
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 define that contains entries for all of the supported properties.

```
*****************************************************************( ldh )***/
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 setup 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.

```
*****************************************************************( ldh )***/
```

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 that contains 5 revisions, the original file is stored first followed by four deltas. When these deltas are applied to original file a newer revision of the file is created.

The reverse delta method, as used by PVCS 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 puser 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 setup 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 setup 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 setup 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.

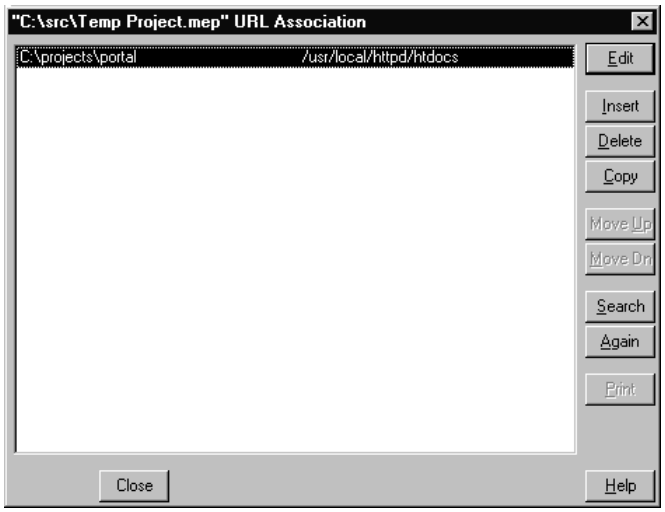
If you will be using the <LOOKUP> archive path alias, you will need to set up the <LOOKUP> database.

The last item related to configuring the VCS support is Setup Verification. Though rarely used, this is provided to help diagnose setup 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 allow 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. 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. 8.0j. 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 function 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 Directory Of Archives Dialog and Using Multi-Edit Files From Archives Dialog use dialogs that allow you to pick and choose the files that you what the VCS operations run on. These are very powerful ways to check in and out files and functions 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 add 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 then a dialog box, shown below, will be shown 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 result 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 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 had been previously 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 was not actually run or that it did not generate any output.

Supported VCS Aliases (or Metacommands)

The supported Aliases (or Metacommands) are listed below.

<LOOKUP>	Substitutes the archive path looked up in the associate directory database.
<LPATHX>	Substitutes the derived library path minus an ending "\".
<LPATH>	Substitutes the derived library path.
<OPATH>	Substitutes the current window's path minus drive letter.
<COMNT>	Substitutes the filename of the file containing the VCS comment.
<RCSDIR>	Substitutes the value of the DOS environment variable "RCSDIR".
<SAPCMT>	Substitutes the Sourcerer's Apprentice command "-c comment".
<%TMP>	Substitutes the value of DOS environment variable "TMP" with an ending "\".
<TMP_PATH>	Substitutes the value of Multi-Edit temporary file path variable "@TMP_FILE_PATH".
<TMP_PATHX>	Substitutes the value of Multi-Edit temporary file path variable "@TMP_FILE_PATH" minus an ending "\".
<FILE>	Substitutes the current window's file name minus extension.
<EXT>	Substitutes the current window's file extension.
<NAME>	Substitutes the current window's file name minus extension path.
<PATH>	Substitutes the current window's path.

<ME_PATH>	Substitutes the path where Multi-Edit resides.
<COMSPEC>	Substitutes the current command processor (command.com).
<USER_PATH>	Substitutes the path where the user specific configuration files reside. Only applicable to the Network version.
<USER_ID>	Substitutes the current User ID. Only applicable to the Network version.
<%str>	Substitutes the value of the DOS environment variable "str".
<~str>	Substitutes the value of the global string "str".

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. 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 that gets checked every time the VCS support is initialized.

The version number consists of two parts, a Multi-Edit version number followed by an update letter, i.e. 8.0j. 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 filename of a specific VCS support macro.

Mac Chk - This field will have a "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 numbers 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.

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 configure 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.

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 file in the project.

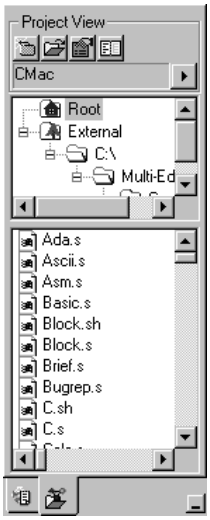
Each project is a file with the extension (.mep), this file stores information about the files and where they are located and there properties. Each file in the project can have properties such as, don't include in file list, etc.

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 does send files in all subfolders
Receive files in folder – Receive files via FTP from the remote site. This does receive files in sub-folders
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 file 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 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 – send by ftp the files selected
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 | Options** and choose the FTP tab.

*You will also want to set up **Project | Options | Directories** and **Project | Options | WebLair**.*

Associate Directories

Associate Directories are used to map directories that are external, i.e. outside of the main project tree can be found. 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 setup

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 setup 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

---

All files in the root project map to the same directories under the root remote directory unless overridden by and associated directory.

Sending / Receiving Files

Sending and recieving files in Multi-Edit is controlled by the project manager. Every project may have a ftp site associated with it. This is setup in the Project Options. Once setup 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/recieve 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 is processed.

Files line terminators are translated according to the mode set in the setup and the that files filename extension.

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**.

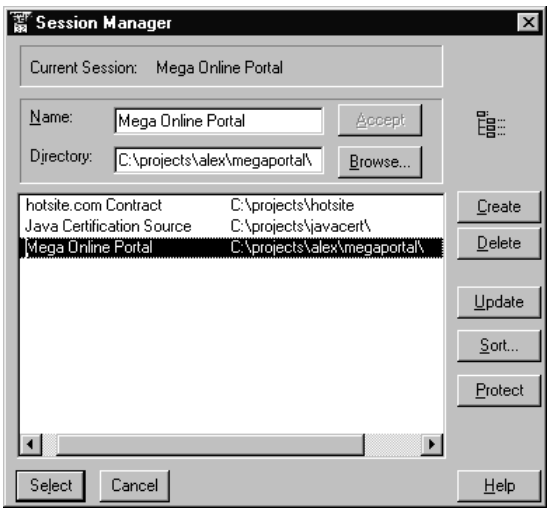
Creating a new Session:
To create a new session, click on the "create" button within the Session Manager dialog. Specify the name and working directory for this new session by filling in the "Name" and "Directory" text fields and then click "Accept". Clicking on the "Select" button will then start this 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, 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 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 (excepting 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 a 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 insert 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 a 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 a ML Database
Until a markup language database is enabled, it will NOT be reference by the right-click context menu. To enable a database, select it in the Tag Database list and click the use button.

Deleting a 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 HTML Tools, HTML Formatter from the context menu. The formatter will attempt to format the markup language code based on the following settings.

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.
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 **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

Keyboard Macros

Recorded Macros

See the section titled Recording Keystroke Macros for details on recording a keystroke macro.

The **Recorded Macro Manager** 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, press the key you'd like to assign the macro to.

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
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 8, 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 repeat 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 select **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

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

When a situation is encountered 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.

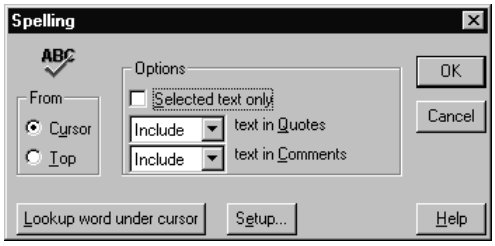
You may also want to periodically visit our website (http://www.multiedit.com) for the latest CMac information and tutorials.

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.

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.

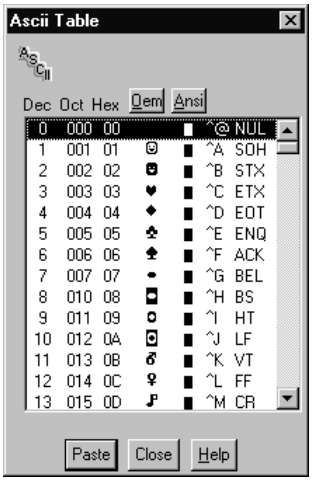
Add to Aux1: Allows you to enter the word in the first auxiliary dictionary. Once a word is added to this dictionary, it will no longer be considered a misspelled word.

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.

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 Chart

The **ASCII Table** command displays 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-type able" 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 out of 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 is proves useful when posting messages on a web forum. Note that using this option does not provided 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 **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 type. 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 art 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 enabled and has focus.

Notebook

The **Notebook** is 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 jumble 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.

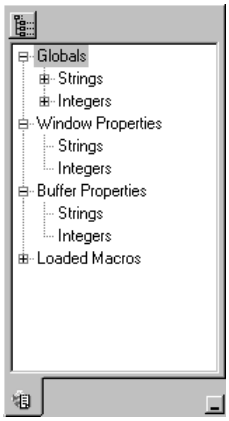
Linedraw

To use the line drawing feature:

1. Select **Linedraw** from the Multi-Edit main menu.
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. You can 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



The System View in Multi-Edit 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.

Long Filename Support

Metacommand	Function
<SFN>	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).
<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 MeTools^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

Metacommand	Function
<@macro>	Substitute the string in Return_Str after running the macro specified by "macro".
<%var>	Substitutes the value of the DOS environment variable "var".
<~str>	Substitutes the value of the global variable "str". The global variable can be either a string or integer variable where the integer value will be written as a string. For example, <~!HELP_PATH> would insert the value of Global_Str('!HELP_PATH') into the prompt or option.
<FILE>	Substitutes the current window's file name minus an extension.
<EXT>	Substitutes the current window's file extension.
<NAME>	Substitutes the current window's file name minus the extension and path.
<PATH>	Substitutes the current window's path.
<ME_PATH>	Substitutes the path where Multi-Edit resides.
<USER_PATH>	Substitutes the path where the user specific configuration files reside. If not network version defaults to the "config directory".
<MAC_PATH>	Substitutes the path to the current user's directory.
<DEFAULTS_PATH>	Substitutes the path where default templates and configuration items are kept.
<USER_ID>	Substitutes the current User Id. Only applicable to the Network version. See NETWORK SUPPORT for more details.
<COMSPEC>	Substitutes the current command processor (command.com).
<CTIME>	Substitutes the current time. The <CTime> Metacommands allow for passing an optional format string. To specify a format string, use the following format. <CTime format string>

	Examples: <CTIME HH':'mm> would insert the time as 15:07 See table below for all of the valid format string values.
<CDATE>	Substitutes the current date. The <CDate> Metacommands allow for passing an optional format string. To specify a format string, use the following format. <CDate format string> Examples: <CDATE dd'-'MMM'-'yy> would insert the date as 27-Jan-99 See table below for all of the valid format string values.
<FTIME>	Substitutes the last file save time.
<FDATE>	Substitutes the last file save date.
<PROJECT>	Substitutes the current project file's name minus the extension and path.
<PROJECT_FILE>	Substitutes the current project file's name.
<PROJECT_FILEPATH>	Substitutes the current project file's path.
<PROJECT_ROOT>	Substitutes the root path for the current project.
<PROJECT_LIST>	Substitutes the name of a file that contains a list of the filenames in the current project.

<CDate> and <CTime> Format String Values

Format String	Function
D	Day of month as digits with no leading zero for single-digit days.
dd	Day of month as digits with leading zero for single-digit days
ddd	Day of week as a three-letter abbreviation. The function uses the LOCALE_SABBREVDAYNAME value associated with the specified locale.
dddd	Day of week as its full name. The function uses the LOCALE_SDAYNAME value associated with the specified locale.
M	Month as digits with no leading zero for single-digit months.

MM	Month as digits with leading zero for single-digit months.
MMM	Month as a three-letter abbreviation. The function uses the LOCALE_SABBREVMONTHNAME value associated with the specified locale.
MMMM	Month as its full name. The function uses the LOCALE_SMONTHNAME value associated with the specified locale.
y	Year as last two digits, but with no leading zero for years less than 10.
yy	Year as last two digits, but with leading zero for years less than 10.
yyyy	Year represented by full four digits
gg	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.
h	Hours with no leading zero for single-digit hours; 12-hour clock
hh	Hours with leading zero for single-digit hours; 12-hour clock
H	Hours with no leading zero for single-digit hours; 24-hour clock
HH	Hours with leading zero for single-digit hours; 24-hour clock
m	Minutes with no leading zero for single-digit minutes
mm	Minutes with leading zero for single-digit minutes
s	Seconds with no leading zero for single-digit seconds
ss	Seconds with leading zero for single-digit seconds
T	One character time marker string, such as A or P
tt	Multicharacter time marker string, such as AM or PM

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

Metacommand	Function
<XDOS>	Force command line to be run as a Dos program (Uses DosExec)
<XOS2>	Force command line to be run as an OS/2 program (Uses Os2Exec)
<XW16>	Force command line to be run as a 16-bit Windows program. (Run directly)
<XW32>	Force command line to be run as a 32-bit Windows program (Run directly)
<XW32C>	Force command line to be run as a Win32 Console program (Uses W32Exec)
<XMAC>	Force command line to be run as a macro.

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 setup.

Printing Support

Metacommand	Function
<LM>	Pushes all following text to left margin
<CM>	Centers the following text
<RM>	Right justifies following text
<PAGE>	Substitutes current page number

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

Metacommand	Function
<MEERR>	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 <MEERR> instead. In the case of batch files, you'll probably have to add <MEERR> 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).
<NR>	This does not do a substitution but will set the options to not capture any output (stdout/stderr).
<TESTERR>	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.

VCS Metacommands

These Metacommands are specific to the VCS support and a few are for specific VCS programs.

VCS Support

Metacommand	Function
<LOOKUP>	Substitutes the archive path looked up in the associate directory database.
<CFGFILE>	Substitutes the archive path looked up in the specific VCS package config file. Currently the same as <LOOKUP>
<LPATHX>	Substitutes the derived library path minus an ending "\".
<LPATH>	Substitutes the derived library path.
<OPATH>	Substitutes the current window's path minus drive letter.
<COMNT>	Substitutes the filename of the file containing the VCS comment.
<%TMP>	Substitutes the value of DOS environment variable "TMP" with an ending "\".
<TMP_PATH>	Substitutes the value of Multi-Edit temporary file path variable "@TMP_FILE_PATH".
<TMP_PATHX>	Substitutes the value of Multi-Edit temporary file path variable "@TMP_FILE_PATH" minus an ending "\".

CVS Support

Metacommand	Function
<CVSDIR>	Substitute the value of the environment variable "CVSDIR"

RCS Support

Metacommand	Function
<RCSDIR>	Substitutes the value of the DOS environment variable "RCSDIR".

SAP Support

Metacommand	Function
<SAPCMT>	Substitutes the Sourcerer's Apprentice command "-c comment".

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.

Help File Support

Metacommand	Function
<HFILE>	Substitutes the name of the specified Help file without extension.
<HEXT>	Substitutes the extension of the specified Help file.
<HNAME>	Substitutes the name of the specified Help file without path or extension.
<HPATH>	Substitutes the path of the specified Help file without extension.
<HOPTS>	Substitutes the options, usually a help index, for the current context sensitive help call.
<HTMLHELP>	Substitutes the command line to run to invoke the HtmlHelp program.
<WINHELP>	Substitutes the command line to run to invoke the WinHelp program.

Template Metacommands

These Metacommands can only be used in templates.

Template Support

Metacommand	Function
<-?>	This metacommand expands to a matching begin/end comment, with a ? in the middle that you can fill in later.
<-SPACE>	Inserts a space for use at end of line.
<-TAB>	This will have the effect of pressing the tab key.
<-UP>	Move the cursor up.
<-CUR>	This will indicate where you want the cursor to end up after expansion is completed.
<-CR>	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.
<-SMARTIND+>	Any line breaks in the result that follow this command will invoke the smart indent macro rather than simple carriage returns.

<-SMARTIND->	This turns off the above metacommand.
<-SCUR>	This saves the cursor position at the point of insertion for later restoring.
<-RCUR>	If used after a <-SCUR> metacommand, will restore the cursor position.
<-XCUR>	Exchange the current cursor position with the last <-SCUR> saved position and then move the cursor to the original saved position.
<-SIND>	Save the old indent level and then set the indent level to the first word on the current line. (Used for nested code templates).
<-SCIND>	Save the old indent level and set the indent level to the current cursor position. (Used for nested code templates).
<-RIND>	Restore the indent level saved by <-SIND> and <-SCIND>.
<-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 the block.

<-PUTBLOCK>	This metacommand will paste a block cut by the <-GetBlock> metacommand.
<-TEXT str>	Insert “str” exactly as entered.
<^template>	This metacommand will invoke another template in the set. the text between the ^ and the > must be a valid template name in either the current set or the Global set.
<-nnn>	Character in decimal (0-255). This is useful for entering "non-type able" characters.
<-0xHH>	Character in hex (0-FF). Same as above but uses hex numbers instead of decimal numbers.
<!macro>	Runs the macro specified. For example, to run the macro Sp_Setup in the macro file mymacs, the metacommand would be <!MyMacs^Sp_Setup>. Regular parameters can be passed using this method, i.e. <!MyMacs^Sp_Setup /PARM1=something/PARM2=something else>

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

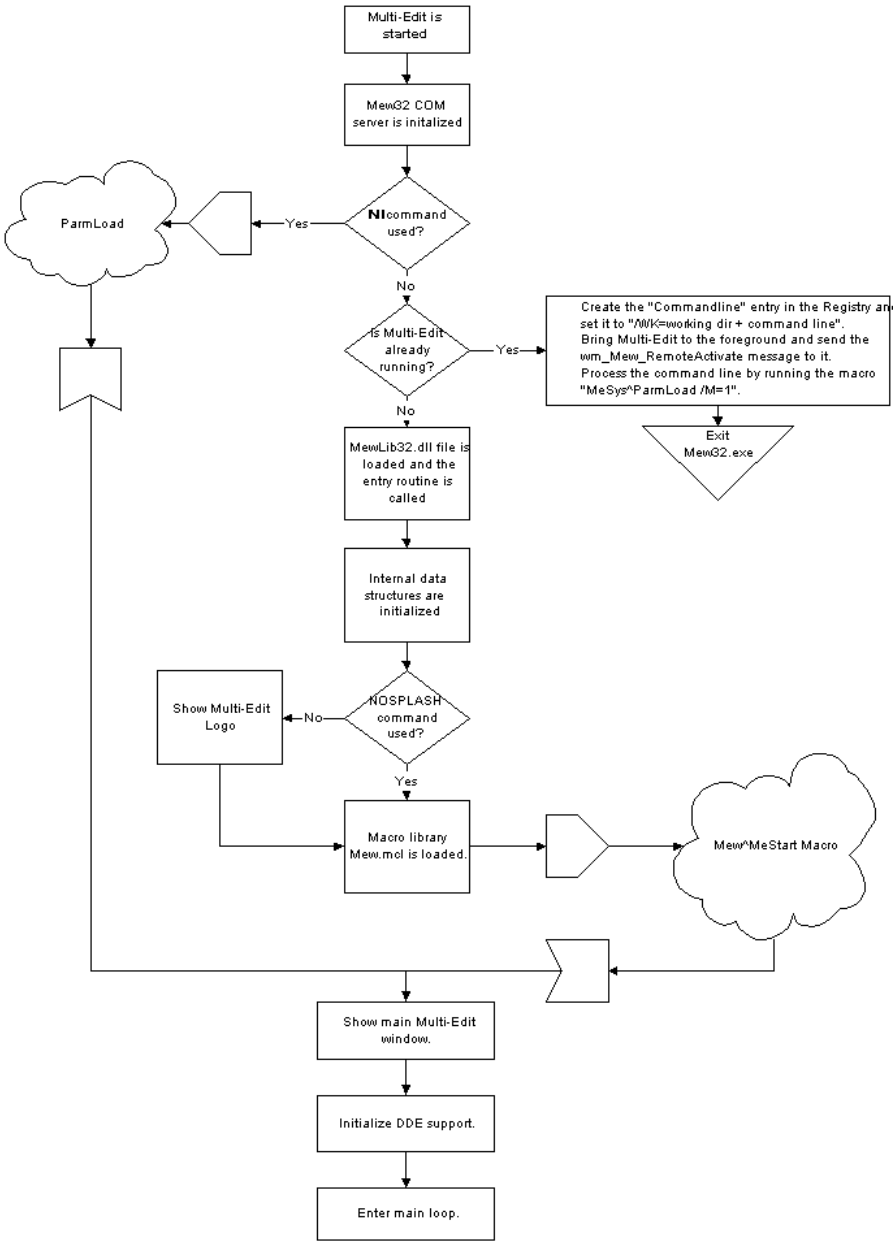
Overview

When Multi-Edit is started, the macro file STARTUP.MAC is searched for, first in the current directory, and (if not found) in the MAC subdirectory under the Multi-Edit directory. If STARTUP.MAC is found, then it is executed.

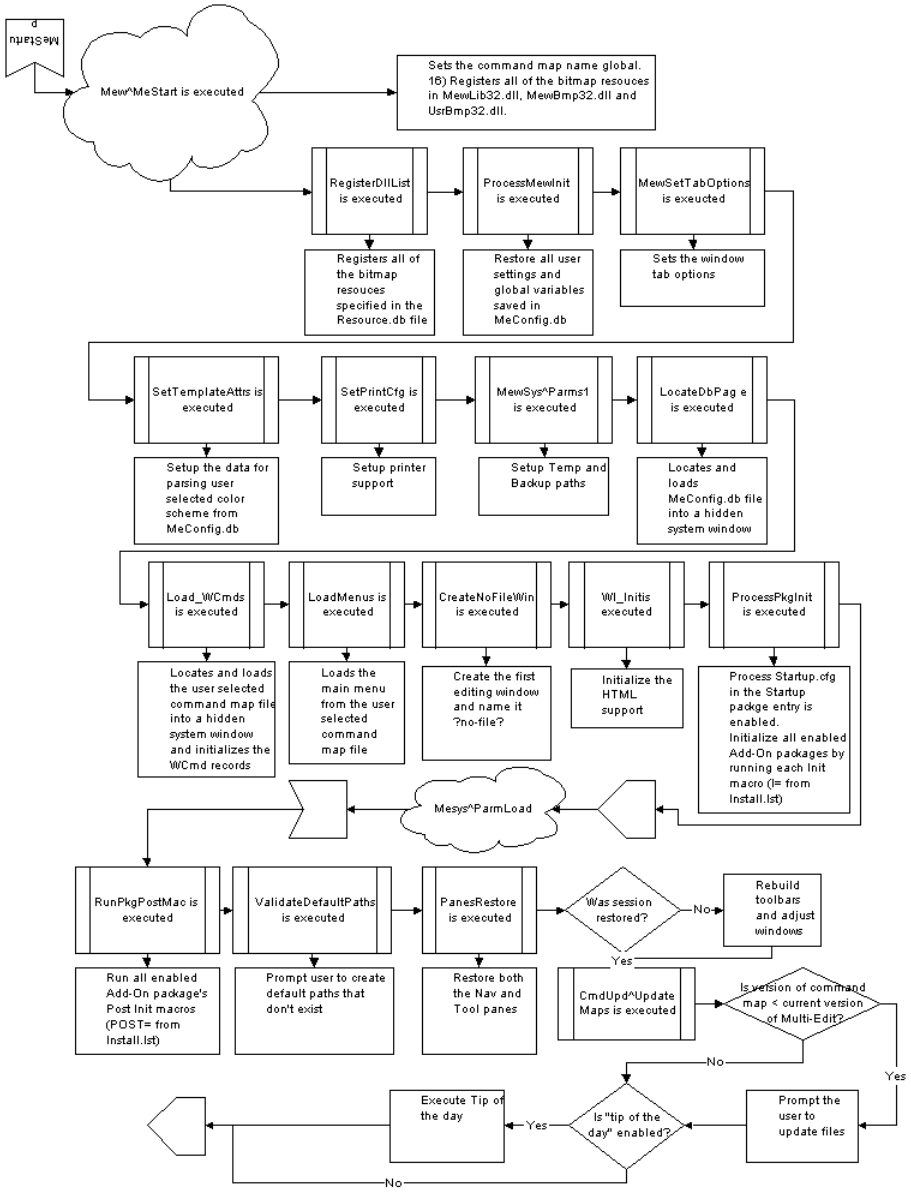
Since the Startup macro is run every time Multi-Edit is invoked, it provides a perfect place to put any custom initialization code you may need. This also allows you to load or run macros. An Example STARTUP.S macro source file is found in the SRC subdirectory under the Multi-Edit directory.

Startup Sequence

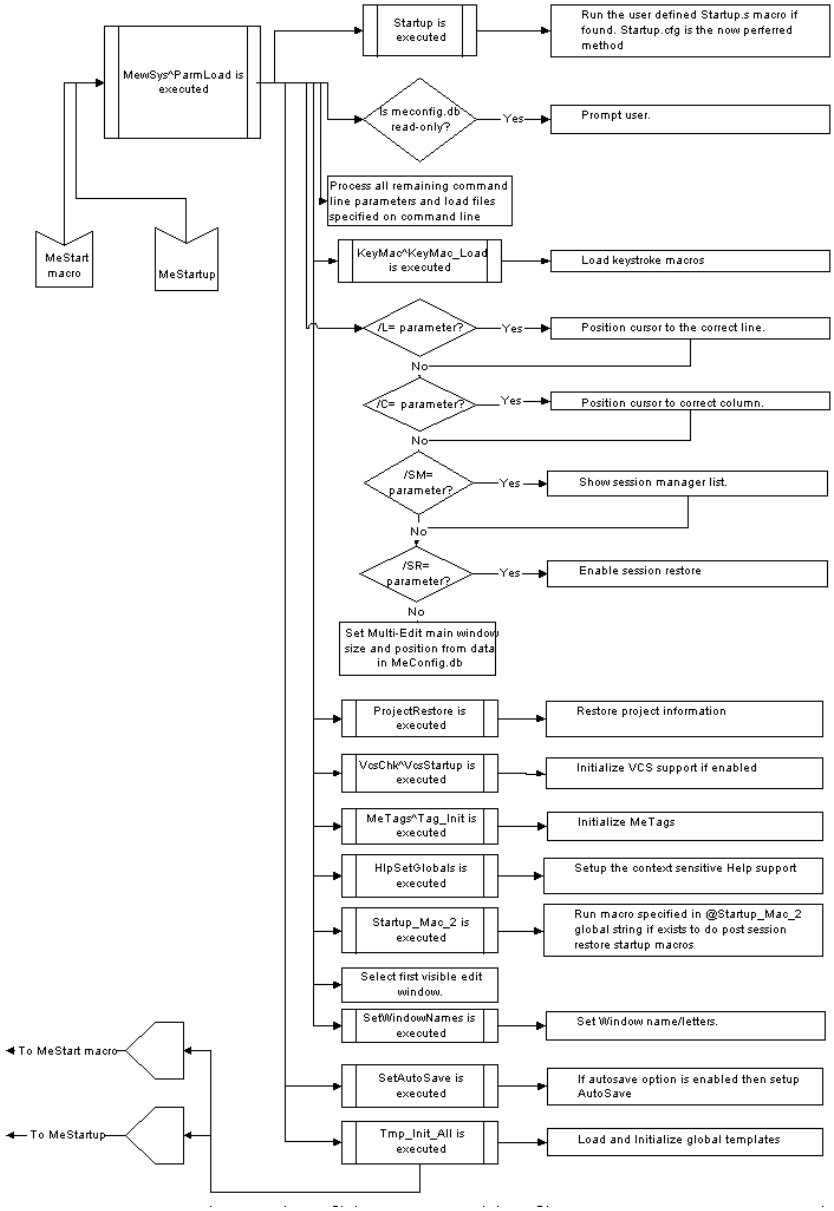
Multi-Edit Startup



Mestart Macro



Parmload Macro



Command Line Switches

What are Command Line Switches?

Multi Edit offers the ability to configure almost all of its options via the Customize dialog box. 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.

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

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 feature to replace the Startup.s file was added to Multi-Edit 9.0. 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 setup 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.

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 Defaults sub-directory.

Adding Features to Multi-Edit

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:

.CFG	contain scripts to configure Multi-Edit, a good way to replace STARTUP.S.
.DAT	contain scripts to install add-on packages. Used by the Install macro.
.DEL	contains scripts to uninstall add-on packages. Used by the Install macro.
.UPD	contain scripts to update db files.

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

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

/*

***/**

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 *expression*. An *expression* that evaluates to True will cause all the following lines 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.

== ***number***	evaluates True when Result is equal to number
< ***number***	evaluates True when Result is less than number
> ***number***	evaluates True when Result is greater than number
<= ***number***	evaluates True when Result is less than or equal to number
>= ***number***	evaluates True when Result is greater than or equal to number
!= ***number***	evaluates True when Result is not equal to number

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** is found are ignored.

Expression is defined as *statment1 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:

==	evaluates True when statement1 is equal to statement2
<	evaluates True when statement1 is less than statement2
>	evaluates True when statement1 is greater than statement2
<=	evaluates True when statement1 is less than or equal to statement2
>=	evaluates True when statement1 is greater than or equal to statement2
!=	evaluates True when statement1 is not equal to statement2

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.

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 c**ommand.

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.

The result code is set to _noError if at least one matching record was found, otherwise _errorNoDbRecord is returned.

Example:

```
;--- 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 setup 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 setup 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 File blank to use the user's default CmdMap file.

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, or to use 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 setup 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 File blank to use the user's default CmdMap file.

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.

*This overrides a **Set Btn** command.*

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 setup 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. The following table shows the meaning of each state.

Off	Commands will not write progress and error messages to the log file
On	Commands will write progress and error messages to the log file.
Reset	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
Msg *text*	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.

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.

Off	The message box will not be used to display progress and error messages.
On	Progress and error messages will be shown in a modeless message box dialog.
Reset	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.

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.

Off	The user will not be prompted before a data command is executed
On	The user will be prompted to verify their desire to proceed with the current data command.
Reset	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.

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 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).

Add (Default)	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.
Delete	The Delete state will cause a data command to run the appropriate DbDelXxx macro in DbTools to delete the specified data.

---

*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 use 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.

After (Default)	Will cause the data to be inserted after the specified data record or at the end of the page when no data is found.
Before	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.

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).

False (Default)	Will not allow a duplicate record to be added to the file when a data command is executed.
True	Will allow duplicate records to be added to the file when a data command is executed

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.

Off (Default)	Will cause the command line to be used exactly as it appears.
On	Will cause all metacommands in the command line to be translated before the data command is executed

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 to insert the new section before or after. It also sets the CmdMap section the following **Cmd** command place it's 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.

NAME=*name*	Used to locate the CmdMap record by name, where name is a text string specifying the command name, i.e. Delete Window.
KEY=*keystr*	Used to locate the CmdMap record by assigned key value, where keystr is a text string of the following form. /KL=F3/K1=114/K2=1 Both primary and secondary keys will be searched.
WCMD=*number*	Used to locate the CmdMap record by WCmd number, where number is a WCmd number

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:

/WCMD=*number*	Specifies the WCmd number used to locate the menu by WCmd, where number is a WCmd number.

or

/IN=level	Specifies the indent level of the menu to locate, where level is the level number. /#= entries only need to be specified up to # equal to level
/1=menu_str	Specifies the text of a level 1 menu entry, where menu_str is the text string, i.e. &File.
/2=menu_str	Specifies the level 2 menu entry, where menu_str is the text of a menu entry, i.e. &Open.
/3=menu_str	Specifies the text of the level 3 menu entry.
/4=menu_str	Specifies the text of the level 4 menu entry.

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, , replaces, 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 command 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

Instead of having each add-on package developer design and code their own installation and management macro, ACI 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.

P=	The package name (Required)
V=	The version of the package (Required)
F=	A script file to copy files etc “MyAddOn.dat” (Required)
CF=	A script file to add package commands “MyAddOn.upd”
D=	The default install directory
R=	A readme file
U=	A script file to delete package “MyAddOn.del”
I=	A macro command to initialize the package upon startup and session switches
POST=	A macro command that is run after a session is restored.
EM=	A macro command that is run when the Enable button in the Manage Add-On Package dialog is selected.
DM=	A macro command that is run when the Disable button in the Manage Add-On Package dialog is selected.
DLL=	A .DLL file that could contain bitmaps etc
NDC=	No DLL Copy flag, 0-Copy DLL to Multi-Edit dir, 1-Do not copy

.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 Addon 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

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.*

Name	Definition	Comment
execute_string	command+	An execute request consists of one or more commands
command	[command_name args*]	Each command is enclosed in square brackets. A command may have an optional argument list.
command_name	grchar+	A valid command name consists of one or more alphanumeric characters.
Grchar	[a-zA-Z0-9!#$%^&()-_{}~]	The set of legal MS-DOS® file system characters.
args	(arglist)	Command arguments are contained within parentheses.
arglist	Arg or any arg, arglist	An argument list consists of zero or more arguments separated by commas.
arg	"string" or string	An argument consists of a string or a string enclosed in double quotation marks. The argument may be empty.
string	grchar*	A series of valid string characters.

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. The examples below show the command string and resulting text.

Examples of Embedding Special Characters in Argument Strings

Argument	Resultant string
"""Woof"", said the big dog.”	"Woof", said the big dog.
"\"Woof\", said the big dog."	"Woof", said the big dog.

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")]

Command	Topic	Description
RunMacro	System	Takes on parameter, which is the name of the macro and any string parameters

Example: [RM('Macro_File^Macro_Name /Parm1=foo/Parm2=other foo ')]

Command	Topic	Description
RM	System	Takes on parameter, which is the name of the macro and any string parameters

Example: [PrepareExit]

Command	Topic	Description
PrepareExit	System	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.

Example: [Quit]

Command	Topic	Description
Quit	System	Shuts down Multi-Edit.

Example: [LineCol(4,6)]

Command	Topic	Description
LineCol	CurrentWindow	Sets current line and column. Accepts two parameters, where first is the line and second is the column.

Topics and Items Supported

Topic	Item	Description
CurrentWindow	Name	Complete filename of the current window.
CurrentWindow	LineCol	Line and Column position of the current window.

COM

Multi-Edit 9.0 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 9.0 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 page 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.

```
' ====================================================================
' 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
    ' ----------------------------------------------------------------
    ' 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!" )

'    WScript.Echo objMEW32.GlobalInt( "!ComTest" )

'    WScript.Echo objMEW32.GlobalStr( "@Default_Ext_List" )
'    WScript.Echo

'    objMEW32.RunMac( "Remote^RemGetActiveFile /GN=!RemActiveFile" )
'    WScript.Echo "ActiveFile: " + objMEW32.GlobalStr( "!RemActiveFile" )

'    objMEW32.GlobalInt( "!ComTest" ) = 6
'    WScript.Echo objMEW32.GlobalInt( "!ComTest" )

'    objMEW32.GlobalStr( "!ComTestStr" ) = "ComTest"
'    WScript.Echo objMEW32.GlobalStr( "!ComTestStr" )

    ' 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

Macromedia Integration

Macromedia

What the **Macromedia IDE integration package** does is allow you to switch back and forth between Multi-Edit and a Macromedia IDE, and have both environments reflect any editing changes. It also makes available a Macromedia menu in Multi-Edit, which allows you to launch Macromedia IDE commands (like Compile) directly from within Multi-Edit.

The rest of this document contains information specific to each of the supported Macromedia products.

ColdFusion Studio 5 and Homesite 5

When switching back and forth between Multi-Edit and CF Studio / Homesite, the actual files are saved to disk on every switch. To keep CF Studio/Homesite from showing a reload prompt on switching back to it when a file was changed in Multi-Edit, the CF Studio/Homesite "**Options | Settings | File Settings**" option "If another process modifies the document - Always reload" must be enabled.

Synchronization of files will only take place when CF Studio / Homesite is in Edit mode.

All of the integration settings are being stored in the Registry under:

HKEY_CURRENT_USER\Software\American Cybernetics\Multi-Edit\9.0\Addon

---

The specific Macromedia settings are stored under the "Macromedia" subkey of the above. The ColdFusion Studio specific settings are stored under the subkey "Macromedia\Studio5" and the Homesite specific settings are stored under the subkey "Macromedia\Homesite5" of the above key.

The install process creates the needed registry entries and then will build a Multi-Edit menu entry for the Macromedia tools. The "Enable" option in the Multi-Edit "**Addon | Macromedia | Configure**" dialog enables/disables the loading of our AddIn DLL when CF Studio or Homesite starts.

Important requirements:
You **MUST** have "Network file locking" **OFF**!!! This is because both Multi-Edit and CF Studio/Homesite must have full read/write access to the source files.

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 and 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 not 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 “**Backups | Temp files | Autosave**” dialog. You must also at least have “Determine changed file by” set to “Date/Time”. See **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 **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 "**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 actually 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 evertime you switch back to the IDE. If you do not have this option 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\American Cybernetics\Multi-Edit\9.0\Addon\Borland*

All of the settings are stored in the Registry under the "Software\American Cybernetics\Multi-Edit\9.0\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. The following registry entries are added/modified:

HKEY_CURRENT_USER\Software\Borland\Delphi\6.0\Experts
HKEY_CURRENT_USER\Software\Borland\Delphi\6.0\Transfer
HKEY_CURRENT_USER\Software\Borland\Delphi\5.0\Experts
HKEY_CURRENT_USER\Software\Borland\Delphi\5.0\Transfer
HKEY_CURRENT_USER\Software\Borland\Delphi\4.0\Experts
HKEY_CURRENT_USER\Software\Borland\Delphi\4.0\Transfer
HKEY_CURRENT_USER\Software\Borland\Delphi\3.0\Experts
HKEY_CURRENT_USER\Software\Borland\Delphi\3.0\Transfer
HKEY_CURRENT_USER\Software\Borland\Delphi\2.0\Experts
HKEY_CURRENT_USER\Software\Borland\Delphi\2.0\Transfer
HKEY_CURRENT_USER\Software\Borland\C++Builder\1.0\Experts
HKEY_CURRENT_USER\Software\Borland\C++Builder\1.0\Transfer
HKEY_CURRENT_USER\Software\Borland\C++Builder\3.0\Experts
HKEY_CURRENT_USER\Software\Borland\C++Builder\3.0\Transfer
HKEY_CURRENT_USER\Software\Borland\C++Builder\4.0\Experts
HKEY_CURRENT_USER\Software\Borland\C++Builder\4.0\Transfer
HKEY_CURRENT_USER\Software\Borland\C++Builder\5.0\Experts
HKEY_CURRENT_USER\Software\Borland\C++Builder\5.0\Transfer

Limitations:

Synching will not be active when debugging/running your app 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 Addon menu for switching to one of the IDEs

All of the integration settings are being stored in the Registry under:

HKEY_CURRENT_USER\Software\American Cybernetics\Multi-Edit\9.0\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 the DevStudio registry entries to install the MeDsSync.dll AddIn, register the server in the MeDsSync.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 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\American Cybernetics\Multi-Edit\9.0\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, 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 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.

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.

Watcom Integration

C/C++

The **Watcom C/C++ IDE integration package** allows Multi-Edit to be used instead of the editor provided with the Watcom C/C++ compiler package. This package allows Multi-Edit to provide the same features that are available while using the Watcom editor. Support includes the Windows IDE for version 10.0 and the Windows 9.x / NT IDEs for version 10.5, 10.6 and early version of 11.0.

Features

Integration with the following IDEs:
- Windows hosted IDE.
- WinNT hosted IDE. (Also works under Win95)

Auto starts Multi-Edit and loads files selected in the IDE.

Closes Multi-Edit on exiting the IDE when Multi-Edit was started by the IDE.

Loads a file and positions the cursor on the error when an error message is selected in the IDE.

Menu entries are added in Multi-Edit to activate the Watcom IDE.

A configure dialog is added in Multi-Edit to change the integration settings.

The ability to uninstall the Watcom C/C++ IDE integration.

Details
The Watcom IDE does not contain an internal editor and must use an external editor for editing text. There are two ways an editor can be integrated into the IDE. The simplest is by specifying the editor command line the IDE will run when editing a file. This allows using almost any editor available but there are a few limitations by using this method. The second method uses a special DLL file that uses the editor API and DDE to communicate with the editor. Since we wanted to provide the best integration possible we chose the second method.

This package provides two DLL files used to hook into the IDEs, WEDMEW.DLL for the Windows hosted IDE and WEDMEWN.DLL for the NT hosted IDE. These files by default are located in the <ME_PATH>WATCOM directory and are hooked into the IDE differently depending upon the version of Watcom C/C++ compiler package being used. For version 10.0, the IDEX.CFG file located in the Watcom \BINW directory is modified to specify that the WEDMEW.DLL file is to be used for all editor communications. While for version 10.5-11.0 the IDEINIT.CFG file in the main Windows directory is modified to do the same but specifies the WEDMEWN.DLL file when the NT hosted IDE is selected. This is automatically done when the integration package is installed and is also changed when the Watcom->Configure dialog is used to change options. The old configuration options are saved so that if/when this package is uninstalled the IDE will be configured back to what it was prior to installing this package.

A file named MEADDON.INI in the main Windows directory is created and used by Multi-Edit to configure most of our integration packages and is used by this package. This is a standard Windows INI file and uses the following information for the Watcom C/C++ integration.

[MULTI-EDIT]	All general Multi-Edit options
MEWPath=	The full pathname of the Multi-Edit executable
[Watcom]	All Watcom specific options
MEWOptions=	The Multi-Edit command line options stated by IDE
Version=	The version of the Watcom package
Host=	The IDE host name being used
Shutdown=	Option to cause Multi-Edit to auto shut down
CfgFile=	The full path for IDEX.CFG file (10.0 only)

By holding down the Shift key while pressing the arrow keys

[Watcom_Ide]	Saved IDE information
editor=	Old editor name (10.5-11.0 only)
dll_editor=	Old dll flag (10.5-11.0 only)
editor_parms=	Old editor param string (10.5-11.0 only)
editor_line=	Old editor line from IDEX.CFG (10.0 only)

The above options are set when installing and configuring the Watcom C/C++ IDE integration package and shouldn't be changed manually.

The only entry that you might need to change manually is MEWOptions. By default this is set to /NR - no session restore, but you might also want to add the /NOSPLASH option to have the Multi-Edit splash screen not show when Multi-Edit is started.

Watcom Configure
For more information on the Watcom Integration see Watcom C++ Integration

C/C++ Version
The version of the Watcom compiler being used. Supports 10.0, 10.5, 10.6 and 11.0.

IDE host

Windows: The 16-bit Windows hosted IDE
NT: The 32 bit Windows hosted IDE

Auto Close Multi-Edit on IDE exit
If checked shuts down Multi-Edit when the IDE is shutdown.

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 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.

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.

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/.

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 a 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.

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.

COMSPEC

The COMSPEC environment variable contains the path where command.com can be found. This variable is automatically set by DOS during the system's boot process and you normally do not need to be concerned with it.

CUA

Common User Access, a set standard 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.

Fortran-77

FORTRAN 77 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 enables programmers to easily create applications with multiple windows.

Modeless Dialog

Modeless dialog boxes are dialogs that stay on the screen and are available for use at any time but permit other user activities.

Modula-2

Modula-2 addresses 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, 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.

VBasic

A programming language and environment developed by Microsoft. Based on the BASIC language, Visual Basic was one of the first products to provide a graphical programming environment and a paint metaphor for developing user interfaces.

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

D

E

F

G

H

N

O

P

Q

R

S

T

U

V

W

Y

Z