Contents

zMUD allows you to connect to and play MUDs on the Internet, and provides you many useful tools, such as aliases, actions, macros, keys, buttons, scripts, etc, to make your MUD life easier and more profitable.   See the disclaimer for support information.

zMUD was designed based upon ideas from various versions of TINTIN, the popular UNIX MUD client.   I have tried to provide compatibility with TINTIN so that users of that client will feel at home.   However, since I have never actually used TINTIN myself, I cannot promise full compatability.   Concepts, functions, and syntax are similar, but may differ.   Of course, the graphical environment of Windows allows functions that were not possible in the text-based UNIX world, giving MUD players even more power.

zMUD has been optimized for use on DIKU and LP combat MUDs.   Since I dont play the social MUDs, zMUD may or may not be useful in those cases.   However, in combat MUDs, zMUD excels, providing many mechanisms for outlaws, robots, etc.

Features

zMUD has unique features for both beginning MUD players, as well as for power wizards builders and coders accustomb to clients like TINTIN.   Here is a list of major features:

Macro keys	assign text or commands to *any* key combination on the keyboard
Aliases	assign text or commands to shortcut names to save typing
Triggers	execute commands based upon patterns received from MUD. Sophisticated pattern matching functions are provided.
Variables	both text and numeric variables saved with your session
Functions	Built-in as well as user defined functions. zMUD contains a poweful programming language.
Buttons	a unique GUI interface allows commands to be executed by clicking buttons, or allows you to easily toggle features such as Triggers.
Paths	let you record directions to MUD locations, and even play them back in reverse. Called **speedwalking** in some clients. Path commands can be customized.
Multiple Chars	Using a Multiple Document Interface (MDI), zMUD allows you to play multiple characters at the same time in different windows. Commands can easily be sent between windows, or to all windows.
Spam protection	prevents you from sending the same string to the MUD too many times and being flagged a spammer
ANSI	full ANSI color support. Colors are user-configurable.
Multimedia	Allows you to trigger sounds, MIDI, movies, etc.
Scripts	store commands in a text file and read them in as a script
Character Database	keeps a database of all of your MUD characters, and detailed notes for each one.
Tab completion	allows you to enter long strings of text by typing the first few characters and then pressing <TAB> to fill in the rest.
History	command history of last commands with customizeable storage limit
Logging	log your session to a file for review at a later time
Timer	a built-in timer allows you to take control of ticks
Status line	customizeable line shows the status of variables and triggers
Customizeable	colors, fonts, sounds, special characters can all be modified and saved.
Settings files	saves all settings (aliases, macros, etc). You can have a single settings file for several characters.
Online Help	extensive online help system provides both reference and examples. Context-sensitive command help is also a keystroke away.
Connection Wizard	contains a master list of MUDs to make connections easier
Compatability	90% compatability with TINTIN and TINTIN+ text-based clients. Includes a TINTIN++ script importing and conversion facility.
WinSock	uses WinSock networking to provide interoperability with all systems, as well as SLIP/PPP with Trumpet WinSock software
GUI	provides both a GUI interface as well as a traditional command line interface. Output can be scrolled and word-wrapped, and window can be frozen to prevent unwanted scrolling. Scrollback buffer can be as large as 4MB in 32-bit version (64k in 16-bit version)
Windows 95	developed and tested as both a 16-bit and 32-bit Windows '95 application. 16-bit version also works on Windows 3.x. Both versions work with Windows NT.

Disclaimer

zMUD is FREE.   It is not Shareware, it is FREE.   You are allowed to redistribute zMUD to anyone as long as the entire package as provided remains intact.   However, you are not allowed to sell zMUD for any fee as part of any package or book without advance approval of the author of zMUD.

Hopefully this is a case where you get much more than you pay for.   I have been using zMUD extensively for several months.   However, the only reason I have released it to the public is with the understanding that there is no support.   I do not guarentee any bug fixes or new versions.   In reality I will probably continue to improve it for my own use, but I reserve the right to terminate development of this project at any time.

zMUD has only been tested on a 486/66 and a Pentium/133 running Windows '95.   If you have troubles running it on a fairly standard and common system, let me know.   I will not support Windows NT, and I will not support any proprietary network stacks.   Since this works with the Microsoft Winsock DLL, it should be compatible with any other vendor that properly implements the WinSock standard.   It has been tested with various screen resolutions and color depths.   I will not support this program on a machine slower than a 486/66.   Since I no longer have a system running Windows 3.x, I have not been able to test it, although it should work.   Ultimately, zMUD will become a 32-bit only application so Windows 3.x users should plan on migration to Windows 95 or Windows NT.

I will maintain a zMUD home page on the World-Wide-Web as a mechanism for distributing zMUD and information about it.   Via this home page you can send me email about zMUD, but I may not be able to respond in a timely manner.   I will not provide general MUD help or advice, nor will I answer questions that are already answered in this help file.   I will keep a list of reported bugs and will pay the most attention to suggestions for new features.

Currently, the zMUD home page is maintained at: http://www.rt66.com/zugg
Email can be sent to: zugg@rt66.com

If you wish to send a donation to the author of zMUD, send a check or money order to:

Mike Potter
681 43rd St.
Los Alamos, NM.   87544

Your name will then be added to the credit listing in the About dialog.

Getting Started

When you start zMUD, you are presented with the welcome dialog.   Behind this dialog you can see the texture-mapped background of the main window, with a menu bar along the top, and the command input window along the bottom of the screen.   As you connect to MUDs, individual MUD windows will be displayed on top of the main windows background.

The steps in getting started with zMUD are:

Quick Start

The easiest way to get started quickly with zMUD is using the Connection Wizard.   Either press the Connection Wizard button in the welcome dialog, or select Connection Wizard from the File menu.

The Connection Wizard will display a list of most known MUDs on the Internet, sorted by their name.   You can press keys to scroll quickly to the MUD name that begins with the key you press.   As you select MUDs from the list, or scroll through the list, the details of the MUD will be shown in the fields on the right.   You can also edit the information on the right if you desire.   When you find the MUD you want to connect to, simply press the Connect button.   Press cancel to exit the wizard.

zMUD will attempt to connect to the MUD that you have selected.   Once connected, the MUD will normally ask for your username, followed by your password.   Once you have entered this information, zMUD will ask if you want to create an auto-login trigger for this MUD.   If you select Yes, auto-login triggers will be created, and your character name and password will be saved to the character database.   Be sure and save your settings file before you exit so that this login trigger will get saved.

Thats all there is to it!   You are now connected and playing a MUD.   You can continue through this help section for more information to help you get started with zMUD.

Note: the file used for the master MUD list was generated by Scott Geiger.   The latest version can be downloaded at ftp.interplay.com.   Simply name this file mudlist.txt and place it into your zMUD directory, and the connection wizard will read the new version.

Troubleshooting: If you have trouble connecting to a MUD and get the error message cant lookup address, then you probably have an incorrect nameserver setup.   Setting up a proper nameserver for your Internet connection is beyond the scope of this document, however, there is a work-around for this.   When using the Connection Wizard, both the host *name* and the host *address* are displayed.   If you are having trouble connecting to the name, simply click the USE button to use the address rather than the hostname.   This will prevent zMUD from trying to look up the address from a nameserver.   This will typically allow you to connect to the MUD.   If the MUD administrators ever change the address of their machine, your connection will stop working until you enter the new address.

Creating a Character

To create a new character, click the Character button from the welcome dialog, or select the Another Character option from the File menu.   The Character Database dialog will be displayed.   zMUD keeps track of all of your MUD characters in this database.   To start with, the database will be empty.   To create a new character, click the New button.

The New Character dialog has several fields for you to enter data.   The cursor is initially placed in the ID field.   Enter a unique short name for this character -- this field is used for the title of the window, and is used in the #SESSION command.   Next, enter the title or name of the MUD you will be connecting to into the Title field.   This is not necessarily the Internet name or address, but is a more description name. Press <TAB> to move to the Host field.   Enter the Internet host name or host address of the machine you wish to connect to. Press <TAB> again and enter the port number that the MUD is running on. Now, click the Connect button to connect to the MUD.

If you leave the character Name and Password fields blank, zMUD will attempt to auto-detect these values and set up an auto-login trigger for you.   If you dont want zMUD to create an auto-login trigger, go ahead and enter values into the Character and Password fields.

Note: your character name and password are not required to use zMUD.   They are used by the #CH (%char) and #PW (%pw) commands (functions).   Dont worry, your password stored in the Character Database is scrambled.   As usual, use a unique password for each MUD, and never use a password that is the same as any other computer you might have an account on.

Settings Files

Associated with each MUD character is a file containing all of your preferences, colors, triggers, aliases, macro keys, buttons, etc.   This file is called your Settings File.   zMUD will automatically choose a setting file name based upon the title of the MUD.   However, you can change this by clicking the Settings tab in the Character Database.

Here, you will see two file names: the primary settings file, and the inherited settings file. zMUD actually loads three settings files for each MUD character.   First, the DEFAULT.MUD settings file is loaded.   This file is used to set overall program defaults, such as default colors, fonts, etc.   Next, the Inherited Settings file is loaded.   Typically you will have an Inherited file related to the type of MUD (LP, DIKU, MUSH, etc) that you are playing.   Finally, the Primary Settings file is loaded, which contains the triggers, macros, etc specific to this character.   In several preferences dialogs you will see options for using Inherited Settings, or using the Primary Settings.   Note that you can normally only edit the contents of your Primary file.   To edit an Inherited File or to edit the Default file, open an Empty window (using the Empty button in the Character dialog), and use the Settings/Load menu to load the file you wish to edit.   Then make your changes and save this file.

Other Character Fields

The drop down box to the right of the MUD Title allows you to specify the type of MUD you will be connecting to.   Currently, this is not used for anything other than for your own information.   The Comment field is a place for a short comment (like the class, race, or level of your character).   By clicking the Notes tab, you can enter free-form text.   This allows you to keep miscellaneous notes about the MUD and character you are playing.

Character Database

In the Character Database, accessed by selecting Character from the main splash screen, or by selecting Another Character from the File menu, you create and manage all of your MUD characters.   To create a new character, click the New button.   To edit an existing character, select the character by clicking on one of the lines, and changing the information shown to the right.   To delete a character, select it by clicking on it, then click the Delete button.   You can also make a copy of a character by selecting it and clicking the Copy button.

To connect to the MUD associated with a character, select the character by clicking on it, then click the Connect button.   A TELNET connection will be made between your computer and the computer listed in the Host field for your character.   Several attemps to connect are made.   If your computer is unable to connect to the host a Cannot Connect dialog will be shown.   It is possible that the Host MUD is currently unavailable.   You should also check your network connection to make sure it is working.

If you want to work on your settings while disconnected from the network, select the desired character and click the Offline button.   This will load the settings and open the MUD window, but no text we be sent to or from the network.   If you click the Empty button, a blank MUD window will be opened.   This is useful for loading and editing individual settings files in order to edit them.

Note: You must connect your computer to the Internet before connecting to the MUD using zMUD.   Ensure that your ethernet connection is working, or ensure that you have established a working SLIP/PPP connection before using zMUD.   A good rule of thumb is: if you can directly access the World-Wide-Web through tools like Mosaic or Netscape, then you can easily use zMUD.   You cannot use zMUD through a terminal link to another Internet host -- you must have a direct connection to the Internet.   Also, zMUD does not work through firewalls which might be set up to allow Netscape (WWW) connections, but wont allow MUD connections.

Basic Usage

Once you are connected to the MUD, you will typically be prompted for your character name and password. zMUD tries to autodetect this, and will pop-up a dialog with your character name and password and ask if you want zMUD to create an auto-login trigger for you. If you click OK, then the next time you log-in, zMUD will automatically enter your character name and password. If you click Cancel, this trigger will not be created for you. Of course, the next time you log in, zMUD will ask about creating the trigger again. If you want to stop zMUD from auto-detecting your login process, simply go into the character database and fill in a value for your character name.

Note that the text you type appears in the bottom Command Line entry field. When you press <Enter>, the text in this field is sent to the MUD. It is also echoed to your text window if you have the Echo flag enabled (default is enabled) using the current command color (changeable in the Color preferences).

When you enter your password, it will also be echoed to the screen. To prevent this, use the #PW command. Type #PW and press <Enter>. The password for this character that you entered in the Character Database will be sent to the MUD, but it will not be echoed to the text window. Also note that # is the default command character and you can change this in the Preferences dialog as described later. Instead of using the #PW command.

The Command Line

Once you have entered your character name and password, answer any other questions displayed by the MUD. All of the text that you type will be shown in the command line. You can use the <Backspace> key to edit this line. You can also move the insertion point withint the command line by clicking the mouse at the point you want to start typing. If you have a separate set of arrow keys on your keyboard, you can use the right and left arrow to move within the command line. Note that the keys on the numeric keypad have macros assigned to them by default so they cannot be used as arrow keys. When you press <Enter>, the text in the command entry field is sent to the MUD. If the Echo flag is enabled, it will be shown in the large text window in the current command color

You can enter multiple commands on the same line using the Seperator character, which defaults to semi-colon (;). Thus, eat bread;drink water will send the two commands eat bread and drink water to the MUD in quick succession. Also, when commands are sent to the MUD from the command line, a newline (CR/LF) is always added to the end automatically. To prevent this newline from being sent, you can put a tilde (~) character at the end of the line. The text (without the tilde) will be send to the MUD without an automatic newline.

The Output Window

You can scroll the main MUD window by clicking the scrollbars to the right and below the main window. You can also use the PgUp and PgDn keys on the keyboard (but again, not the number pad). When you scroll the window, the border changes to red to indicate that the screen is automatically frozen. This is done so that when connected to a MUD you can review the scrollback buffer and read text without it continuing to scroll. Once the screen is locked, the up and down arrows can be used to move line by line, or the PgUp and PdDn keys can be used to scroll a page at a time. You can still type commands in the command line and send them to the MUD while frozen. Also, text is still being received from the MUD and your triggers are still being processed. To unfreeze the window and automatically return to the bottom of the scroll buffer, click the Pause button in the lower right corner of the window, or press the Ctrl-Z key, or press the ScrollLock key, or select the Freeze command from the Window menu, or type #FREEZE on the command line.

You can search for text in the output window using the Find command in the Edit menu, or by pressing Ctrl-F.   You can search backwards (default) and forwards through the buffer. Since this buffer can grow quite large, especially in the 32-bit version of zMUD, the Find command is very handy for rapidly locating past text.   The screen is automatically frozen when the text is found.

You can also copy text from the output window and paste it into other programs.   When you highlight text with the mouse (click the left button at the starting location, drag the mouse with the left button still held, then release the mouse button at the end location) the screen is automatically frozen to prevent scrolling from disturbing your selection.   When you release the mouse button, the text is automatically copied to the clipboard and the text is unhighlighted.   You can select large portions of text (more that a screenful) by left clicking the start location, the left clicking while holding the Shift key at the end location.   If you double-left-click, the word under the mouse is highlighted then copied to the clipboard.

Contents of the screen are stored to the clipboard in both plain ASCII text format and in color ANSI format.   When you paste to an external program like Notepad, the plain text format is used.   If you paste to the Command Editor the ANSI color format is used to preserve MUD colors.   If you paste to the Command Line, plain text is used unless you have defined a color translation syntax in the color preferences in which case the ANSI color is converted to color commands for your MUD.

Introduction to Macros

One of the first ways to make zMUD do more than just a dumb TELNET client is to assign commands to keys on your keyboard.

To assign a command to a key, press Control-K, or select Define Key from the Action menu. You will be prompted to press the key combination that you wish to assign a command to. Almost any key on the keyboard can be assigned a command, in combination with the Shift, ALT, and Ctrl keys. You cannot override any of the zMUD command keys (like Control-K). If you assign a command to a Windows key (like F10, or Alt-A), your command will work and the Windows function will be disabled - be careful. Also, note that zMUD automatically activates the NUMLOCK mode to allow you to assign macros to the keypad. If NUMLOCK is off, the keypad functions as the cursor keys. Since the cursor keys are used to edit the command buffer and recall from the command history, you should refrain from assigning macros to the arrow keys (although you can if you want).

Once you press the key you want to assign a command to (for example, the NUM8 key, the 8 key on the keypad) you will be prompted for the command. Enter the text you wish to be sent to the MUD when you press this key. For example, if you were assigning a command to the NUM8 key, you could enter the text north. Now, when you press the NUM8 key, the text north is sent to the MUD. Notice that when you press a macro key the text in the command entry field is selected, but is otherwise undisturbed. This is a very useful feature. For example, if there is a monster to the north that you want to hit quickly before it can hit you, you can enter kill monster in the command entry field, press the NUM8 key (to move north) quickly followed by pressing <Enter> to send the kill command to the MUD.

The setting files DIKU.MUD and LPMUD.MUD contain some sample macro key assignments that I have found useful for these two types of combat MUDs.

Macro Chaining

If you want the text assigned to the macro key to be put into the Command Line rather than sent directly to the MUD, put a tilde (~) at the end of the text assigned to the key. Then, when you press the macro key, the text will be added to the command line and your cursor will be placed right after the text so that you can complete the command and press return.

You can also chain macros using the above feature. When you press the macro key that has the tilde at the end, the text is placed into the command line. If you then press a normal macro key, the text is added to the command line and the finished command is sent to the MUD.

Heres an example: assign the text open door ~ to the F8 key. I assume you have the normal directions like north, south, east, etc assigned to the numeric keypad. When you press F8, the text open door is put into the command line. If you now press the 2 key on the keypad (south), the word south is added and the command open door south is sent to the MUD. All in just two keystrokes!

Introduction to Aliases

Aliases are another way to simplify your life of MUD playing. Basically, aliases allow you to assign any command to a shortcut abbreviation.

The easiest way to create an alias is to type the command or words you want to make a shortcut for, then press Control-A, or select Make Alias from the Action menu. You will be prompted for the shortcut abbreviation you wish to assign the command to. For example, enter the text fill waterskin statue and press Control-A. Then enter fs and click OK. Now, whenever you enter fs in the command buffer, the string fill waterskin statue will be sent to the MUD.

Note that aliases are only translated if they are the first word in a command. In the example described above, if you entered the text say fs on the command line, the string say fs is sent to the MUD and fs is not translated.

Alias Preference dialog

You can edit all of your aliases using the Settings/Edit/Aliases menu command. This brings up the alias dialog. All of your aliases (like fs from the above example) are displayed in the list on the left. As you click on these aliases you can edit them on the right. To define a new alias, click the New button and enter the name and commands for the alias. You can copy an existing alias using the copy button, and delete an alias using the button with the picture of a trash can on it.

This is an example of a standard zMUD Preferences dialog. It has a list on the left with edit controls on the right. The OK, Cancel and Help buttons are along the button. Along the top of the dialog is a menu strip that lets you access the other preferences such as macros, triggers, etc. To the left of the menu strip is a button with a stick pin on it. When you click this button, the window will stick to the top of the screen and not get covered up by other windows. This button toggles so if you click it again the dialog returns to a normal window that can be obscured by other windows. Note that the status of the stick button and the position of the dialog is saved in your ZMUD.INI file.

The ALIAS command

Another way to define an alias is using the #ALIAS command. Commands are typed entiredly in the command input line at the bottom of the screen, but perform a function for zMUD and normally don't send any text to the MUD. Commands are provided for users of text-based MUD clients like TINTIN, and are similar in syntax. To create an alias with the ALIAS command, type #ALIAS shortcut 'command text'. The command text will then be assigned to the shortcut abbreviation that you supply. You can also list all aliases by just entering #ALIAS, or you can list the definition of a single alias using #ALIAS shortcut.

Aliases can also contain *Parameters*. Parameters are the text following the shortcut. For example, if you enter fs foo bar, fs is the alias shortcut, foo is the first parameter, bar is the second parameter. Parameters are assigned to specific numeric variables %1 through %9. In the previous case, %1 would contain foo, and %2 would contain bar. You can use these parameters in the alias itself.

For example, define the alias #ALIAS k 'kill %1'. Now when you enter k rabbit, the command kill rabbit is sent to the MUD. Now, this isn't a very useful example, because if you add text after an aliases (like rabbit in the above example) and the alias doesn't use it as a parameter, the extra text will just be appended to the result of the alias translation.

Thus #ALIAS k kill followed by k rabbit will do the same thing.   However, with parameters you can get more sophisticated.   For example, the alias #ALIAS kk 'kill %1;kick %1' followed by kk rabbit will send the commands kill rabbit and kick rabbit to the MUD.

One last tidbit...if you assign a command to the alias atconnect, it will be executed whenever you connect to the current MUD.   Other special aliases include: atexit which is executed when you exit zMUD, and atdisconnect which is executed when you disconnect from the current MUD.

Introduction to Variables

Variables are very similar to aliases. The important difference between aliases and variables is that aliases are only expanded when at the beginning of a command, while variables are expanded anywhere. To expand the variable, you preceed its name with the @ character. Note that this is different then TINTIN where variables start with a $. You can change the variable character in the Preferences dialog if you wish.

To define a variable, you still use the #VARIABLE command. For example, #VAR container waterskin stores the string waterskin into the variable container. To return the contents of the variable, preceed its name with the @ character. For example, fill @container would expand to fill waterskin.

Another assignment syntax is also provided. As with some programming languages, you can use the syntax variable = value to assign a value to a variable. All variables are stored internally as 80 character strings, just like aliases. Note that using this syntax, you can put expressions on the right side. For example, a=2 then b=a+1 assigns the value of 3 to the variable b.

So, the illustrate the use of variables, with the variable @container defined as shown above, you could now create an alias #ALIAS fs 'fill @container statue'. Now when you enter fs on the command line, the current value of the container variable (waterskin from the above example) is expanded and the command fill waterskin statue is sent to the MUD.

Understanding Quotes

For this example, make sure the Expand Variable option in the Preferences dialog is turned ON.

In TINTIN, the commands are surrounded by curly braces {} instead of single quotes. You can do this in zMUD also, but you must understand variable expansion rules. Normally, when you enter a command into the command line, all variables are immediately expanded before the command is parsed. Putting text within single quotes prevents this immediate variable expansion. Curly braces allow you to group text much like quotes, but variables within the braces are expanded immediately. Lets look at two examples:

```
container=waterskin
#ALIAS fs fill @container statue
#ALIAS fs2 {fill @container statue}
```

The first line assigns a value to the container variable. The second line is the standard way to define an alias in zmud. The third line is the standard TINTIN syntax. Go into the Alias dialog and look at what got assigned to these aliases:

fs	fill @container statue
fs2	fill waterskin statue

Note that in the second example, the value of @container was expanded *before* the text was assigned to the alias. Just like in TINTIN, the way around this is to delay the expansion of @container by adding a second @ character to the variable.

```
#ALIAS fs3 {fill @@container statue}
```

results in

fs3	fill @container statue

Notice that the extra @ was stripped when the variable expansion phase occured, and the result that was assigned to the alias is just what we want.

You can turn off this behavior by turning off the Expand Variables option in the Preferences dialog. With this option off, variables in the command line are not immediately expanded before the command is executed.

System Variables

There are also several predefined system variables that are maintained by the system. These variables all begin with the parameter (%) character. Changing temporary variables does not change the modified status of your settings file, so you aren't prompted to save your settings when you exit. These system variables are used just like regular variables, except with the % character instead of the @ character. To perform delayed expansion you use %% instead of @@.

Predefined Variables

The system maintained several predefined variables for you to use. Each of these variables begins with an underscore character to signify that it is a temporary variable.

%action	the action executed from the last trigger
%char	the name of your MUD character
%ctime	the number of seconds you have been connected to the MUD
%host	the host name of the current MUD
%lastcom	the last command executed
%lastcom2	the command before the last command executed
%lastcom3	the command before %lastcom2
%lastinput	the last line of commands executed
%line	the last line received from the MUD
%line2	the line before the last line received
%line3	the line before the line before the last line received
%param1	the first parameter from the last trigger match
%param2..%param10	the parameters from the last trigger match
%port	the current port connected to
%random	a random number from 0 to 99
%repeatnum	the current index during repeating commands, or loop command
%i	same as %repeatnum
%selected	returns the text currently selected in the output or command buffer
%title	the title of the current MUD
%trigger	the line that caused the last trigger
%window	the name of the current window

Introduction to Paths/Speedwalking

Paths are a very powerful feature that allow you to save the directions to a location and then replay these directions at high speed at a later time. This is also called *speed walking* in other MUD clients. Not only is this useful for getting to common areas within the MUD, but the directions are replayed fast enough to get you past some aggressive monsters (agros). This is not fool-proof as some MUDs have high aggressive monsters that will hit you anyway. Also, when leading a party, sometimes the monster will still hit one of your other party members. However, it works most of the time and you will find yourself using Paths quite a bit.

To record a path, enter the #MARK command, or select Speedwalking from the Action menu and press the Start Recording button. Then use your normal movement commands to walk to the final destination. Then, use the #PATH command to save the directions by entering #PATH shortcut. Or, you can select Speedwalking from the Action menu, press the Stop Recording button and enter the shortcut name for this path.

Paths are saved as aliases preceeded by a special character called the Movement character, which defaults to a period (.). To replay the path directions, go to the start of the path that you saved eariler with the #MARK command, and enter .shortcut. The directions saved in the path shortcut will then be replayed at high speed.

You can also send a set of directions at high speed using the Movement character by entering directions. For example .neesuwd will send north, east, east, south, up, west, down to the MUD at high speed. You can preceed any direction with a number to repeat it that many times. The above path could be abbreviated .n2esuwd.

While recording a path (after entering the #MARK command), you can inspect the current path being recorded at anytime by entering #PATH (with no parameter). The current path relative to the marked starting location will be shown in the direction syntax described above. If you make a mistake and go the wrong direction, you can use the #BACKUP command to erase the last direction from the currently recorded path. It will also attempt to move you to your previous location (e.g. if you went north by mistake, then did #BACKUP, you will be moved south).

A useful function that zMUD has added over other clients that have path features is the ability to reverse a path. For example, let's say you have the path .2s2wn assigned to a shortcut called magic (gets you from temple to magic shop). When you enter .magic while in the temple, you are taken to the magic shop. Now you buy whatever you need, and then you want to return to the temple. You can use the #REVERSE command to reverse the path by entering #REVERSE magic. You can also use the shortcut syntax of two dots: ..magic. The path .s2e2n will be sent to the MUD. If you enter #REVERSE with no parameters, than the currently recorded path (since the last #MARK) will be reversed. Kind of like leaving a trail of breadcrumbs. Note however, that in many cases, going east to a room doesn't necessarily mean that entering west will go back. The #REVERSE command only works in "euclidean" areas of your MUD.

Note that speedwalking relies upon definitions of commands like north, south, down, etc. You can add your own commands and short-cuts (such as o for open door) using the Settings/Edit/Directions menu command.

Introduction to Triggers

Triggers can be tricky, but are also the most powerful feature of zMUD. Triggers (called actions on other MUD clients) allow you to execute a command whenever a particular string of text is received from the MUD. While this sounds simple, it has powerful implications.

To define a trigger, you use the #TRIGGER (or #ACTION) command. The syntax is #TRIGGER 'pattern' 'command'. Whenever the pattern text is received from the MUD, the command is executed. You can also define and edit triggers using the Trigger Preferences from the Settings/Edit/Trigger menu.

Let's start with a simple example. When you are working in a group, it is important to see anything that someone in the group has to say. When someone in the group talks, the MUD usually says something like Zugg tells the group 'heal me'. To ensure you don't miss this important information, let's change the color of the line to red using the #COLOR red command. Thus, the trigger would be defined as #TRIGGER 'tells the group' '#COLOR red'.

That was easy, and triggers like this can really enhance your MUD playing. Here's another useful example: #TRIGGER 'You are thirsty' 'dr'. With an alias like #ALIAS dr 'drink @container' this trigger will keep your stomach happy and full by automatically drinking whenever you are thirsty from whatever container you have.

Extracting text from the MUD

Patterns can contain more complicated expressions and wildcard characters, and parts of the matched pattern can be stored in special parameters for use in the command string. Parameters were introduced when Aliases were discussed. The way you store part of the pattern into a parameter is by surrounding the part of the pattern with parenthesis. One of the wild-card strings for a patter is %w which matches any word. So, for example #TRIGGER '(%w) tells you' 'tell %1 I am busy' will match any string from the MUD that has a word followed by the string tells you. Thus, when you receive the string Zugg tells you 'Hi', you will automatically send the command tell Zugg I am busy to the MUD.

Here's another really useful one: #TRIGGER 'You get (%d) coins' 'split %1'. Since %d matches any set of numeric digits, whenever you pick up some gold coins, you will automatically split them to your group!

Trigger Classes

Now, you wouldn't want the above trigger to be active all of the time. Splitting coins when you are not in a group is not recommended. To assign a name (called a Class name) to a trigger, provide the name as the optional third parameter to the #TRIGGER command. For example: #TRIGGER 'You get (%d) coins' 'split %1' autosplit. Then you can turn the trigger on with #T+ autosplit, or turn it off with #T- autosplit. Assign these two commands to macro keys or buttons and you have full control over when you split and when you don't.

Introduction to Buttons

Combining the concepts of aliases, triggers, and variables are Buttons. Buttons not only make it easier for novice users to use these features, but provides significant functionality to power users that is not found in text-based clients such as TINTIN. Buttons can be clicked to execute a command, their caption can display the value of a variable, and they can act as a toggle to turn a feature (like triggers) on and off easily.

To define a button, right click on the button you wish to edit, or select Make Button from the Action menu. There are currently two types of buttons: push buttons and toggle buttons. A push button is clicked to execute a command and releases as soon as you release the mouse. A toggle button changes between being off (up) or on (down). The type of button is controlled by the *Variable* field. If you assign the name of a variable to this field, the button will be a toggle button, and the variable will contain the state of the button (0 for up/off, 1 for down/on).

In the *Off Caption*, enter the text you wish to display on the face of the button when the button is in its normal, off position. The next field is for the caption to be displayed when the button is pressed in, or on (this field is grayed out if the button is not a toggle button). If you leave the *On Caption* blank, it will use the same value as the *Off Caption*. You should limit captions to about 10 characters so that they fit on the button face. The value of the caption fields are evaluated, so you can put an expression containing variables and the result will be displayed on the button.

We'll skip the *Value* field for a moment, and move to the *On Command* field. Enter the command you wish to execute when the button is pressed (from Off state to On state). In the *Off Command* field, enter the command you wish to execute when then button is released (from On state to Off state).

The *Value* field is used to externally control the state of the button. You can do this with just the variable (e.g. if you assign 1 to the variable in the *Variable* field, the button will activate itself, if you assign 0 to this variable the button will deactivate itself). However, the *Value* field allows greater control as to whether the button is pressed or not. If the expression in the *Value* field is true, the button is in the On state (pressed). If the expression in the *Value* field is false, the button is in the Off state (released).

So, how about an example? In the Introduction to Triggers we created an autosplit trigger. We can make this trigger much more user friendly by using it with a button. Select Make Button from the Action menu. In the *Off Caption* field, enter the text AutoSplit. In the *On Command* field enter emote is auto-splitting;#t+ autosplit. In *the Off Command* field enter emote stops auto-splitting;#t- autosplit. In the *Variable* field, enter autosplit. Click OK to save the button definition. Now you have a button labelled AutoSplit, and it is currently off. Click the button. The command emote is auto-splitting is sent to the MUD (telling your group members what you are doing), and the autosplit trigger is enabled. The semi-colon (or Seperator character) allows you to specify more than one command on the same line. Note that the button now appears in the On state (is pressed in). This gives you the visual clue that your AutoSplit function is enabled so that you don't forget. Click the button again. The text emote stops autosplitting is sent to the MUD, and the autosplit trigger is disabled. The button now appears in the Off position. Now you don't have to waste two keys on the keyboard to turn your autosplit trigger on and off, and in addition you have a nice visual clue as to whether your trigger is active or not. Try to beat this in a text-based MUD client!

Introduction to Multiplaying

If you connect to more than one character or more than one MUD (using the Another Character menu command or the Connection Wizard), zMUD will put each character into a different window. This allows you to control multiple characters at once (called *multiplaying* and is banned on many MUDs).

The commands that you type in the command box are sent to the character window that currently has the focus. This is usually the window on top. You can change the currently focused window in several ways: select the window from the list given in the Window menu, or click on the window box just to the left of the command input window to popup a list of current character windows. You can also cycle through the open windows using the Ctrl-N or Ctrl-Tab keys.

Each window has a name associated with it. The default name for a window is the ID of your MUD character. You can change this name using the #NAME command.

To send commands to a different window, preceed your command with name: where name is the name of the window you wish to get focus, and : is the focus character (which can be changed in the Preferences). The indicated window will be brought to the top and focused, and the command will be sent to that character. To send a command to a different window without changing the focus, preceed your command with :name: where name is the name of the window you want to command sent to. In this case, your current window will be unchanged. If you do not specify any window name, and just enter :hi then hi will be sent to all windows.

Note that when refering to windows by name, you dont have to spell out the entire name, just use enough characters to uniquely determine the window. For example, if you have a window named zugg, then typing z:hi is enough to send hi to the zugg window. You can also refer to windows by there number (shown next to the name in the Window menu). For example 1:hi sends hi to the first character window. Variables are also allowed before the colon. If the variable @tank contains the value zugg, then @tank:hi will send hi to the zugg window.

Note that you can also mix these focus commands within your command line. For example, entering zugg:eat;aurora:drink will send the eat command to zuggs window, and the drink command to auroras window. Auroras window will have the focus at the end of this. Lets look at some more examples to make this clearer. In all of these examples, assume that we have two windows, Zugg and Aurora, and that Auroras window currently has focus.

eat;zugg:drink	tell aurora to eat and zugg to drink. Gives zugg the focus.
zugg:eat;drink	tell zugg to eat and drink and give zugg the focus
:zugg:eat;drink	tell zugg to eat (dont change focus) then tell aurora to drink
zugg:eat;::drink	tell zugg to eat and give him the focus. Then, change the focus back to aurora and tell her to drink.

Finally, you can use tab completion to change window focus without sending any commands to the window. Typing zugg: and pressing <TAB> will give the zugg window focus and bring it to the top. The command buffer will then be cleared, awaiting commands to be sent to this window.

Advanced Topics

zMUD is a very complex program.   The combination of triggers, variables, and aliases combine to form a powerful event-driven programming language.   To enhance this language are functions, introduced in this section.

Also, zMUD isnt just for players.   It includes many features designed for world builders and MUD coders.   One of these is the ANSI editor window which allows you to send more than one line to the MUD, and imbed ANSI color sequences.

The following topics are covered in more detail:

Advanced Editing

In addition to the single line of input that you can send from the command input window, zMUD contains a full-window ANSI editor for composing long messages. To open this editor window, select Command Buffer from the Window menu, or press Ctrl-Enter.

The command editor works like most editors. You can type text, use the arrow keys (or mouse) to move around, cut, copy, and paste text within the editor, or copy text from the MUD output window and paste it into the editor window. Lines can be any length, and you can use the scroll bars to display other parts of the window. The size of the editor is the same as the output window. In the 16-bit version, this limit is 64k bytes. In the 32-bit version, the limit is 4MB, enough to edit large log files.

The menu in the editor window allows you to load text from a file, or save the current contents to a file. The **Import** command inserts a file at the current cursor position, rather than replacing the buffer contents with the file. The **Send** command sends the contents of the editor to the MUD, and can also be activated by pressing Ctrl-Enter. Using the Prefix string to the right of the speed buttons, or by selecting Strings in the Options menu, you can change the string of text that preceeds each line as it is send to the MUD. You can also change how blank lines are sent to the MUD. Normally, each line is parsed for commands before being sent to the MUD. You can send the text verbatim by pressing the Parse speedbutton to toggle command parsing.

The **Capture** command allows you to take the last N lines from the MUD output window and insert them into the editor. The #CAPTURE command can be used in triggers to send MUD output to the editor window. The Capture Toggle speed button can be used to turn off capturing while you are editing.

In the Text menu, you can change the bold attribute, or the color of your text. Standard ANSI color sequences are used, and note that most MUDs filter ANSI sequences that you send (although this might change when more people using zMUD demand this colorful feature). If you have defined a color syntax for your MUD, zMUD can convert the ANSI color in this window to the color commands used by your MUD if the Color Syntax is enabled.

The buttons along the speedbar replicate the Open, Save, Capture, Send, Bold, Color, Capture Toggle, and Parse toggle menu commands. To close the window, select Close from the File menu, or press <ESC>.

Note that the text is not only sent to the MUD, but it is also parsed and executed just as if you entered it line by line into the single line buffer. Thus, you can load a script from a text file, and execute it using the Send command, or by pressing Ctrl-Enter. If you turn off the Parse option (using the speed button or menu command), then text will be sent to the MUD verbatim - it will not be parsed for commands. You can also edit the string that is sent as a prefix to each line, and control what is sent in place of blank lines.

Advanced Programming

In the section on variables and triggers you learned the basics about programming in zMUD. However, zMUD is a powerful event-driven programming language, and you have barely touched the surface of what is possible. In this section, a more detailed explanation of command parsing will be presented, and *functions* will be introduced. If you fully understand how zMUD parses its commands, you will find that zMUD is capable of doing most anything you desire.

Command Syntax and Parsing

Everytime you enter a command and press Enter, the command text is parsed. This parsing several steps:

    break the input using the seperator character (;) into individual commands
    determine the focus of the command
    replace variables and functions in the command
    execute the command

Now you can understand when you need a single @ to expand a variable, and when you need two (@@). Since all variables and functions are replaced in the command before the command is executed, the value of variables at the time you enter the command are inserted if you specify a single @ character. If you use multiple @ characters, the value of the variable is not replaced, one of the @ characters is simply be removed. Thus, in unusual situations, you might actually use more that 2 @ characters to delay variable expansion even further.

Variables are not expanded if they occur within quotations (single or double). Curly braces {} act like quotations, but variables within braces are expanded normally. Note that after a command is split into its individual words, the enclosing quotes or braces are removed. Thus, the command #VAR b {testing} assigns the value testing to the variable @b -- the braces are removed. If the variable @a has the value hello, then #VAR b @a there assigns @a there to the variable @b, while #VAR b {@a there} assigns hello there to the variable @b. Note that if the Expand Variable option in the Preferences is turned off, variables are never expanded within the command line.

Note that the same thing happens with pre-defined variables and functions that start with the % character. To delay expansion, simply use multiple % characters.

One final word about parsing and variable expansion. Normally, each variable must be seperated by spaces to allow proper parsing. Thus, if the variable @a has the value test, you must normally say @a ing to get @a to expand. If you say @aing zMUD looks for a variable called aing. You can use the curly braces {} to surround the name of the variable to solve this problem. Thus, @{a}ing expands to testing.

You can use the above syntax to perform indirect variable addressing. Lets say that the variable @b has the value a. Refering to @{@b} expands the value of @b, resulting in @{a} which expands to test.

If you ever want to use one of the special zMUD characters like @ or %, you can use the tilde character (~) to quote the special character. Thus, zugg~@rt66.com would tell zMUD not to interpret @ as a variable character. To actually use a tilde character, use two of them.

Functions

In addition to variables, zMUD allows you to define functions. Think of functions as variables with parameters. Parameters are used much like they are with aliases, except that since they are expanded within a command string, the syntax for calling a function is a bit different.

You declare functions just like variables, using the #VARIABLE command.   However, in the definition of the function, you can use %1, %2, etc to refer to parameters to be supplied by the caller.   For example, #VARIABLE kk kill %1;stun %1 defines a function called @kk that takes one parameter.   You expand and execute this function by using the @ character in front of the function name, then put the required parameters in parenthesis (much like in a programming language).   Thus @kk(zombie) will expand to kill zombie;stun zombie.   Recall that this is similar to the kk alias that we defined in the Introduction to Aliases section, but unlike aliases, functions are expanded during the parsing phase anywhere in the command, rather than performed at the execution phase.

To add more power to zMUD, several predefined functions are supplied.   They provide the tools necessary for some very powerful trigger processing.   By using the predefined functions in combinations with your own functions, the sky is the limit!

Predefined Functions

The following functions are defined within zMUD:

%abs(i)	return the absolute value of I
%alias(s)	expand the value of alias s
%begins(s1,s2)	true if s1 starts with s2
%case(i,s1,s2,s3...)	if I=1, return s1, if I=2, return s2, etc. Up to 8 strings can be given
%char(i)	return the ASCII character associated with the number I. This function is also used to translate the system characters (;:@%!.)to their current values if they have been changed.
%color(fore,back)	converts a descriptive color into an attribute value
%concat(s1,s2,s3..)	return all strings concatenated together (up to nine parameters)
%copy(s,i,n)	return a portion of string s, starting at character position I, and returning n characters
%dde(serv,topic,item)	fetch data from a dde server
%ddemacro(serv,topic,s)	tell the specified dde server to execute the macro in s
%ddepoke(serv,topic,item,value)	poke the data in value to a dde server
%delete(p,i,n)	return the string s with n characters starting at position I deleted.
%ends(s1,s2)	true if s1 ends with s2
%eval(p)	evaluate parameter p as an expression and return the result
%format(f,a,b,c,d...)	use a format string to format the values of a,b,c, etc. Format strings consist of a string of characters, with special format specifiers of the type %w.dx where w is the width of the field, d is the number of decimal places, and x is the format type. Format types for x include: s for string, n for number (commas every 3 digits), f for floating point, m for money (currency).
%getglobal(name)	return the value of global variable name (stored in the INI file)
%greg(i,s)	search the ith file and return lines that match the pattern in s
%if(expression,true-value,false-value)	if expression is true, return the true-value otherwise return the false-value
%insert(p,s,i)	return the string s with pattern p inserted at position I.
%isnumber(s)	true if s represents a valid number
%left(s,n)	return the leftmost n characters of the string s
%leftback(s,n)	return the leftmost part of s, n characters from the end
%len(s)	return the length of the string s
%lower(s)	return the string s in lowercase
%max(a,b,c,d...)	return the maximum value of up to 9 parameters
%min(a,b,c,d...)	return the minimum value of up to 9 parameters
%numwords(s,d)	return the number of words in string s, delimeted by string d (if d is missing, a space is used as the word delimeter)
%pick(s1,s2,s3,...)	display a picklist and let the user choose one or more strings from the list. If more than one are chosen, they are returned seperated by the current command seperator character (;).
%pos(p,s)	return the position of pattern p in string s. Return 0/false if not found
%proper(s)	convert s to proper case (lowercase except for first letter)
%prompt(v,p)	prompt the user for a value for the variable v. If p is present, use password mode so that the value entered by the user is not echoed.
%random(i,j)	return a random integer >= I and < J. If J is omitted, then I specifies the maximum value, and 0 is used as the minimum value.
%read(i,rec)	read the specified record from the ith file. If rec is omitted, zero is assumed. For text files, rec is the line number to be read (0 reads the next sequential line).

%remove(p,s)	Remove substring p from string s
%repeat(s,n)	return s repeated n times
%replace(s,p,r)	return s with all occurences of p replaced with r
%right(s,n)	return the rightmost part of s following n characters
%rightback(s,n)	return the rightmost n characters of the string s.
%setglobal(name,value)	set the value of the global variable name (stored in the INI file)
%time(format)	return the current date/time. If format is nil (or missing), a long format is used. Otherwise, use characters such as dd, mm, mmm, yy, hh, mm, ss, etc in the format string to return that part of the current date/time
%trim(s)	trim spaces from beginning and end of s
%trimleft(s)	trim leading spaces from s
%trimright(s)	trim trailing spaces from s
%upper(s)	return the string s in uppercase
%word(s,i,d)	return the ith word of string s, delimeted by string d (if d is missing, a space is used as the word delimeter
%write(i,s,rec)	write string s to the ith file at record rec. For text files, rec is ignored and s is appended to the file. If rec is 0, s is written to the end of the file.

Menu Reference

Click on the menu command that you want help on. The values in parenthesis are the names of the commands as used by the #MENU command.

File (file)	
Connection Wizard	(wizard) display master list of MUDs
Another char	(char) connect to a new character or host
Reconnect	(reconnect) re-establish connection to current character
Disconnect	(disconnect) disconnect from the current session
New Log	(new) open a new log file
Append Log	(append) append to an existing log file
Log	(log) toggle logging state
Print Setup	(setup) setup the current printer
Print	(print) print the current screen (screen), buffer (buffer), or selection (selection). You can also toggle color or B&W (color).
Exit	(exit) disconnect from MUD and leave the program
Edit (edit)	
Cut	(cut) cut text from the input command line
Copy	(copy) copy selected text from the input command line
Paste	(paste) paste text into the command line
Select All	(all) select the entire contents of the command line
Clear	(clear) clear contents of command line
Settings (settings)	
New	(new) clear all settings
Load	(load) load settings from a file
Save	(save) save settings to a file
Save As	(saveas) save settings to a different file
Import	allows you to import ASCII (ascii) script files, or TINTIN++ (tintin) script files
Export	(export) Allows you to export macros, aliases, triggers, etc in ASCII format for later importing.
Edit	(edit) edit current settings for general preferences (general), special characters (chars), aliases (aliases), triggers (triggers), macros (macros), buttons (buttons), tab completion (tab), directions (directions), slow walking (slow), sounds (sounds), fonts (fonts), and colors (colors).
Actions (actions)	
Make Alias	(alias) create an alias
Make Trigger	(trigger) create a trigger action
Define Keys	(macro) assign a command to a key
Make Button	(button) create a new button
Add Tab Word	(tab) add word to tab completion list
Speedwalking	(speed) manipulate paths and control speed walking
Timer	(tick) manipulate the tick timer
Synch Timer	(sync) resynch the tick timer
Window (window)	
Tile	(tile) Tile all windows so you can see them all
Cascade	(cascade) Place all windows on top of each other
Arrange	(arrange) arrange the minimized windows at the bottom of the screen
Freeze	(freeze) freeze output window to prevent scrolling
Refresh	(refresh) redraw the current window

Parse	(parse) toggle parsing of command line
Command Buffer	(buffer) open the full-window command editor
History	(history) display the command history

Help (help)

Contents	(contents) show table of contents
Search	(search) search the online help database
Reference	(reference) display command reference
Getting Started	(started) display the Getting Started help section
Command Wizard	(wizard) get help and examples of zmud commands
Tip of the day	(tip) display a useful usage tip
Startup Screen	(startup) display the initial welcome dialog
About	(about) version, credits, and copyright info
Feedback	(feedback) send feedback to the author of zMUD

Reconnect

Allows you to quickly reconnect to your current MUD session.   The network link to the MUD is dropped, and then zMUD immediately tries to reconnect.   If the MUD cannot be connected, a retry dialog is displayed showing a 10 second countdown.   At the end of this 10 seconds, a reconnection is attempted again.   You can cancel the reconnection attempt by clicking the Cancel button.   You can force it to try again without waiting for 10 seconds by clicking the Retry button.

This is a great function to use after a MUD reboot to ensure that you get connected as quickly as possible.

zMUD will not allow you to change this 10 second delay.   If all players using zMUD were able to attempt reconnects continuously, it would bring the poor MUD server that is trying to reboot to its knees and upset a lot of system administrators.

Make Alias

You are prompted to name an alias created from the following text:

any highlighted text in output window
any highlighted text in command line
entire contents of command line

Note that any expansion of variables is delayed so that two variable characters are not needed.

Make Trigger

You are prompted for the commands to be assigned to the pattern.   The pattern is taken from any highlighted text in output window

Note that any expansion of variables is delayed so that two variable characters are not needed.

Add Tab Word

Adds a word to the tab completion list.   The word is taken from

> any highlighted text in the output window
> any highlighted text in the command line
> last word in the command line

Save Path

You are prompted for a name for the currently recorded path.   Note that the Movement character (.) at the beginning of the path name is optional and will be added for you.

Run Path

A drop-down list of saved paths is displayed.   Click the down-arrow icon to drop-down the list, and click on the path you wish to run.   Then click OK to run the path.   You can also double-click the path name to run it.   Click Cancel to abort.

Reverse Path

A drop-down list of saved paths is displayed.   Click the down-arrow icon to drop-down the list, and click on the path you wish to run.   Then click OK to run the path.   You can also double-click the path name to run it.   Click Cancel to abort.

Note that the path will be run in reverse to allow you to retrace your steps.

Command Help

You can get help on the command currently in the command line by selecting this function. You can also press Control-H or F1 (as long as you have not assigned any macros to those keys).

Speedwalking Dialog

This dialog allows you to manipulate all speedwalking and path commands.   On the left is a list of all defined paths.   Click on a path to move its definition into the Path Ahead field. The directions in the Path Ahead field are those that will be sent by the walking commands. The directions in the Path Behind field are those that have already been sent or recorded.

The Speed Walk button will send the directions in the Path Ahead field to the MUD quickly. Once sent, there is no way to abort them.   The Slow Walk button sends the first direction in the Path Ahead field, then waits for confirmation from the MUD that the move was successful.   These confirmation triggers can be set by clicking on the Settings tab.   There is also a timer that can be set to abort the Slow Walk if it times out.   Once confirmed, the next direction is sent, then another wait for confirmation occurs.   This combination of sending a direction, then waiting for confirmation is continued till no more directions are in the Path Ahead buffer.

While Slow Walking is active, a Stop button will appear that lets you abort the Slow Walk process.   The Step button moves one step in the next direction held in the Path Ahead buffer.   The Backup button moves backwards one step, effectively undoing the last direction stored in the Path Behind buffer.   The Turn Around button reverses the Path Ahead and Path Behind buffers.

Finally, the Start Recording button is used to start recording directions into the Path Behind buffer.   The Stop Recording button terminates a recording and prompts you to save the directions in the Path Behind buffer to a named path.

Timer Dialog

This dialog allows you to easily manipulate the tick timer commands in zMUD.   The current value of the timer (counting down to zero) is shown on the right.   The timer can be started and stopped using the Start and Stop buttons.   The Synch button is used when a tick on the MUD actually occurs.   The current tick interval will be modified to reflect the actual tick on the MUD.   Note that most tick intervals vary depending upon the lag on the MUD, so dont think this timer will save your life or anything.   The Reset button resets the timer back to the tick interval.

The Tick interval can be set from this dialog to a given number of seconds.   The Timeout Margin indicates the point at which the Timeout Command is executed.   For example, if the Margin is 5, the Command will be executed when the timer reaches 5 seconds.

Command Example

You can get an example of how to use the command currently in the command line by selecting this function.   You can also press ALT-F1 (as long as you have not assigned any macros to this key).

Command Reference

Each command should be preceeded by the Command character. The default command character is # but can be changed in the Preferences dialog. You can abbreviate each command using the letters shown in boldface.

[number]	repeat following text [number] times
ABORT	abort further parsing of the current command line
ACTION	create or display a trigger action
ADD	add a value to a variable
ALARM	create an alarm trigger
ALIAS	create or display an alias
ALL	send a command to all windows
BACKUP	remove last direction from current path
BEEP	beep the speaker (or play a wave file)
BUTTON	trigger a button
C+	start capturing to a window
C-	stop capturing to a window
CAPTURE	capture lines and send them to the editor or a window
CASE	select a command from a list
CHARACTER	returns the name of your character
COLOR	change color of the last line of text
CONNECT	reconnect to the current session
CR	send a new line
CW	color the matched word on the last line
DDE	send a DDE macro to a server.
DISCONNECT	disconnect from the current session
ERASE	erase a file from the disk
EXEC	execute a command
FILE	open a file for reading and writing
FREEZE	freeze the screen from scrolling
GAG	remove last line from screen
HELP	get help on commands
H+	retrieve the next command in the history buffer
H-	retrieve the previous command in the history buffer
HISTORY	display the history of previous commands
HIGHLIGHT	highlight the last line of text
HOST	return the name of the current host
IF	perform a conditional test
IGNORE	toggle the processing of trigger actions
INPUT	put text into the command buffer
KEY	define a macro key
KILLALL	delete all aliases, macros, trigger actions, tab-completion words
LOAD	load a settings file
LOOP	execute command several times in a loop
LOG	start a log file or toggle logging
MAP	add a direction to the current path
MATH	perform complex math and expression parsing
MARK	mark the beginning of a path
MEDIA	send commands to your multimedia device
MEMORY	display the remaining memory
MENU	execute a menu command
NAME	change the name of the current window
NOOP	nothing
OK	confirm a Slow Walk step

PATH	save or display the current path
PICK	select commands from a list
PLAY	play a wave, midi, avi, cd player, or other multimedia
PROMPT	prompt for the value of an alias/variable
PW	return your current password
READ	read and execute a script from a file, or read a record from a file
RECORD	record an alias
RESET	reset the file back to the beginning.
RETRACE	retrace a path
SAY	echo text to the screen
SAVE	save the current settings file
SEND	send a text file to the MUD prefixed by a command
SESSION	open a new session
SHOW	echo text to the screen
SLOW	execute a path in Slow Walk mode
STATUS	set the definition of the status bar
STEP	resume an aborted Slow Walk and step ahead
STOP	abort a Slow Walk
T+	turn on a class of triggers
T-	turn off a class of triggers
T?	display time remaining in timer
TAB	add word to tab completion list
TIMER	toggle the timer
TRIGGER	create or display a trigger action
TS	set the time or origin of the timer
TYPE	display all or part of a text file to the screen
UNALIAS	remove an alias
UNGAG	prevent the line from being gagged
UNKEY	remove a macro key
UNTRIGGER	remove a trigger
UNVAR	remove a variable
VARIABLE	assign a value to a variable
VERSION	display the current version of zMUD
WAIT	delay further processing until next line is received
WINDOW	open a new window
WIZLIST	display the credits for zMUD
WRITE	write a record to a file

Preferences

The Preferences dialog is accessed via the Edit command in the Settings menu and allows you to change all of the parameters stored in a zMUD settings file.   This settings file is associated with your MUD character in the Character database.

Select the tab of the dialog that you want help on:

General	change miscellaneous parameters
Special Characters	allows you to change the special characters used by zMUD
Memory	allows you to monitor and change scrollback and editor memory usage.

Settings

This dialog allows you to change some of the system settings.

ANSI Color	toggle whether the window interprets ANSI color commands
Word Wrap	toggle word wrapping at the window border. Lines without spaces are not wrapped. You can specify the specific column to wrap at, or select Auto Wrap to automatically track the width of the window.
Auto Clear Input	Normally, when you send a command to the MUD, the text is highlighted so that you can type over it, or press Return to send it again. If this flag is on, the command buffer is always cleared after you send a command.
Show Triggers	turns on verbose debugging information to show the details of trigger processing
Echo Commands	if on, commands sent from the input window are also echoed to the output window using the current command color
Connection Timer	enables or disables the tracking of connect time and display of the connection timer.
Help Baloons	toggles the help messages that popup when the cursor is moved over various buttons.
Auto NumLock	determines whether NumLock is automatically enabled when zMUD is started. Should be disabled on laptops without a keypad.
Expand Variables	if on (default), variables in the command line are expanded unless they are within quotes.
Use = syntax	If off, the var = value syntax is disabled allowing you to use the = character for your MUD.
Spam Count	number of times you can safely repeat the same command without being flagged a spammer
Spam Length	ignore commands with this length or less in the spam count
Spam Command	command to insert after you have repeated another command Spam Count times
Tick Interval	current timer interval
Scroll Amount	determines how often zMUD updates the screen. If set to zero, zMUD only updates the screen when no text is being received from the MUD. This is the fastest setting, but you may miss text scrolled of the screen. If set non-zero, it indicates after how many lines received from the MUD to update the screen. The default is 5.
Command Char	character used to begin a command. Default is #
Separator Char	character used to separate multiple commands on a single line. Default is ;
Variable Char	character used to preceed a variable. Default is @
Parameter Char	character used to preceed a parameter reference. Default is %
Movement Char	character used to begin a path variable or command. Default is .
History Char	character used to recall a command from the history buffer. Default is !
Focus Char	character used to specify the window that will receive the command. Default is :

Click OK to save your changes. Click Cancel to abort. Click Help to display this help screen.

Macro Keys

This dialog allows you to view and edit all of your macro keys.   In the main list is each macro, with the name of the key shown to the left, and the command assigned shown to the right.   Click on a key definition, and the key value is copied into the *Key* field, and the command is copied into the *Macro* field.   You can then edit the command by editing the *Macro* field, or change the key by clicking the *Key* button and pressing the new key combination.

To create a new macro, click the New button.   To delete the currently selected macro, click the Delete key.   You can copy the selected macro using the Copy button.   Close the dialog by clicking the OK button.   Display this help screen by clicking the Help button.

Along the top is the standard Preferences banner containing a menu bar to select other Preferences dialogs, and a stick-pin button that can be used to keep this dialog at the top of the screen when pressed in.   The position of this dialog and the status of the stick button are stored in your ZMUD.INI file.

Aliases

This dialog allows you to view and edit all of your aliases.   In the main list is each alias, with the name of the alias shown to the left, and the command assigned shown to the right.   Click on an alias, and the shortcut name is copied into the *Alias* field, and the command is copied into the *Command* field.   You can then edit the command by editing the *Command* field, or change name of the alias by editing the *Alias* field.

To create a new alias, click the New button.   To delete the currently selected alias, click the Delete key.   To make a copy of the selected alias, click the Copy button.   Close the dialog by clicking the OK button.   Display this help screen by clicking the Help button.

Along the top is the standard Preferences banner containing a menu bar to select other Preferences dialogs, and a stick-pin button that can be used to keep this dialog at the top of the screen when pressed in.   The position of this dialog and the status of the stick button are stored in your ZMUD.INI file.

Triggers

This dialog allows you to view and edit all of your triggers. In the main list is each trigger, with the pattern shown to the left, and the command assigned shown to the right. Click on a trigger, and the class name is copied to the *Name* field, the pattern is copied to the *Pattern* field, and the command is copied to the *Command* field. You can then edit any of these fields as desired.

To create a new trigger, click the New button. To delete the currently selected trigger, click the Delete key. To make a copy of the selected trigger, click the Copy button. Close the dialog by clicking the OK button. Display this help screen by clicking the Help button.

The Enable Class button is used to enable all triggers with the class name listed in the *Class* field. The Disable Class button is used to disable all triggers with this class name. The Enable/Disable button enables or disables just the current trigger. This is also reflected with a check mark next to the trigger in the list on the left, and can also be toggled by double-clicking on the trigger in the list on the left.

The Enabled at Init box determines whether this trigger is automatically enabled when the settings file is loaded.

The Trigger on CR box determines whether the pattern for this trigger is checked after each line is received from the MUD, or whether it is only checked after a block of text is received. You will normally leave this box checked. However, when triggering on text that is not followed by a newline (such as username or password prompts), you should deselect this option.

At the bottom of the screen is a test area. You can enter a string of text that might be sent by the MUD and click the Test button to see if the current trigger will fire. The values of any parameters extracted from the text will also be shown. Note that zMUD will automatically create a test pattern for you based upon what you enter in the Pattern field. To prevent this, click the Lock button to the left of the test pattern.

Along the top is the standard Preferences banner containing a menu bar to select other Preferences dialogs, and a stick-pin button that can be used to keep this dialog at the top of the screen when pressed in. The position of this dialog and the status of the stick button are stored in your ZMUD.INI file.

Directions

This dialog allows you to view and edit all of your direction definitions.   These definitions are used in conjunction with the path features.   In the main list is each direction, with the character defining the direction shown to the left, and the typed commands for the direction shown to the right.   Click on a direction, and the defining character is copied into the *Char* field, the typed command is copied into the *Input* field, and the character used to reverse the direction is copied into the *Reverse Char* field.   You can then edit these fields as desired.

If you want more than one command to trigger the given direction, separate the multiple commands with a vertical bar (|).   For example, if you want the commands n and north and nor to trigger the n direction, put n|north|nor in the *Input* field.

Tip: Sometimes you need to open a door while executing a path.   To do this, define a new direction with the character o in both the *Char* and *Reverse Char* fields, and then put open door in the *Input* Field.   Now when you are recording a path and you enter open door, the character o will be added to the path.

Tip 2:   Some MUDs use the directions ne, nw, se, sw.   The paths can only contain a single character for each directions.   Thus, make some up.   For example, create a direction with a *Char* of q and a *Reverse Char* of r and put ne in the *Input* field.   Then create another direction with a *Char* of r and a *Reverse Char* of q with sw in the *Input* field.

To create a new direction, click the New button.   To delete the currently selected direction, click the Delete key.   To make a copy of the current direction, click the Copy button.   Close the dialog by clicking the OK button.   Display this help screen by clicking the Help button.

Tab Completion

The Tab Completion dialog shows all of the words that can be completed automatically by enter the first few characters in the command line and pressing <TAB>.   Aliases and variables are automatically expanded.   To expand any other text you must add it to the tab completion list shown here.   The box on the left is fully editable.   To add a new word, click in the box and begin typing the word, ensuring that it is on a line by itself.   To delete a word, use the <Backspace> key or <Del> key to remove the word.   This box acts like a regular text editor.

To save your changes to the tab list, click the OK button.   To cancel your changes and close the dialog, click the Cancel button.   You can load any text file into this list using the Load button.   You can also save this list to a text file using the SaveAs button.   The Help button displays this help screen.

Sounds

This dialog allows you to enable or disable sound in zMUD.   Also, you can select the sound played by the #BEEP command, and played when you start zMUD.   In the input boxes, you can specify the name of a WAV file, or can enter the numeric value of the window event you wish to trigger.   Window event 0 is the standard beep sound.   Other sounds occur at 16, 32, 48, 64, etc.
These values are stored in the ZMUD.INI file and effect all windows.

In this dialog you can also enable or disable sounds support.   Also, if you are having trouble with the sound support in zMUD, you can put the line Sound = 0 in the [Settings] section of your ZMUD.INI file.

You also specify the sounds used for connection and disconnection from the MUD in this dialog.

Fonts

This dialog allows you to change the command input font, or the output font of the current window.   The default font is Courier 10pt.   Note that zMUD does not perform wordwrapping properly if you select a font that is not fixed-pitch, or in which the bold font has a different size than the regular font.   Courier has this problem, for example, so bold text doesnt word wrap properly.   True-type fonts work better, but Courier was chosen over Courier New because more lines can be fit on the screen, and there is better compatibility with Windows 3.1.

To change the font, just click on either the input or output box.   A standard Windows font change dialog box will be shown.   Select the desired font and size and click OK to save your change.   Changes to fonts take effect immediately.   Note that style changes (bold/italic) to the Input font are currently ignored.

Colors

The system tab displays the colors used for zMUD for various text.   To edit one of these text colors, click the button to the right of the sample text and select the foreground and background colors from the dialog.

The Foreground tab allows you to specify the foreground color displayed for the 16 ANSI color values. The actually color displayed by each color index is shown in the boxes on the screen.   Click on one of these boxes to change the actual color.   This allows you to take full advantage of all the colors of your video card.   Buttons are provided to set all of these colors to either normal (dim) or bright colors.   The upper 8 colors are only used if you select the Use Highlight Color option on the right, otherwise the Bold font is used rather than the upper colors.

The Background tab is similar to the Foreground in that it lets you specify the mapping for the 8 background ANSI colors.

The Syntax tab allows you to enter a color syntax command language used by your MUD. Some MUDs have special ways that you can change the color of your tell and gossip commands.   Next to each color, enter the letter or number used by your MUD to represent that color.   Then, in the command sections on the right, enter the syntax used by your MUD and use F to represent the code of the foreground color, and B for the code of the background color.   Also enter the syntax used by your MUD to return to the default color. This syntax, if enabled, is used to translate copied color text from the output window into the command line, and for text sent from the Command editor.

In the Window tab you can control the look of the background window.   The background window can be a solid color, or a bitmap texture.   If you disable the Use Bitmap option, the color shown will be the solid background color (you change it by clicking on the color box). If you enable textures, then the bitmap specified will be used for the background.   If you leave the name of the BMP filename blank, the internal marble texture will be used.   A sample of the texture is shown below the name of the BMP file.   Changes to background color or texture do not take effect until you restart zMUD.

Slow Walking

This dialog allows you to control the settings use to perform Slow Walking. On the left is a list of patterns received from the MUD that should confirm a successful Slow Walk step. These are turned into trigger automatically for you.

For example, a common setting is ^Exits which is usually displayed by the MUD at the bottom of a room description.

On the right you can control the Slow Walk timer. Its value is set in milliseconds (e.g. 5000 is 5 seconds). If enabled, a timeout will terminate a Slow Walk. If enabled, the step is automatically confirmed.

ACTION

Syntax: #AC pattern command *classname*

Related: #TRIGGER #T+ #T- #IGNORE
Example

This is one of the most powerful features of zMUD. It allows you to define a command to be executed whenever the matching text is received from the MUD.

The first parameter is the text to be matched. If the text contains a space, you need to enclose it in quotes. This pattern can contain special pattern matching symbols and wildcards. The second parameter is the command to be executed when the pattern is received from the MUD. Since this command usually consists of more than one word, you must enclose it in quotes. The third parameter is optional, and is the name of the trigger action class that this action is part of. Triggers can be enabled and disabled when part of a class.

For advanced trigger options, you must go to the Preferences dialog. In this dialog, you can determine whether the action is triggered at the end of each line received from the MUD, or if it is just triggered at the end of receiving a block of data from the MUD. Responding to MUD prompts such as Username and Password require a trigger that activates after a block of text is received since these prompts are not normally followed by a newline.

In the Preferences dialog you can also determine whether the trigger action is enabled when you first load the data file. You can also individually enable or disable triggers individually as well as by class name.

Pattern Matching

Patterns can contain several special character for wild-card matching.

*	match any number of characters or white space
?	match a single character
%d	match any number of digits (0-9)
%w	match any number of alpha characters (a-z) (a word)
%a	match any number of alphanumeric characters (a-z,0-9)
%s	match any amount of white space (spaces, tabs)
%x	match any amount of non-white space
[range]	match any amount of characters listed in range
^	force pattern to match starting at the beginning of the line
$	force pattern to match ending at the end of the line
(pattern)	save the matched pattern in a parameter %1 though %9
~	quote the next character to prevent it to be interpreted as a wild card.
{val1|val2|val3|...}	match any of the specified strings
{^string}	do not match the specified string

In specifying a range, you can list specific characters such as [abc] or you can use a range [a-c]. To use a wild card character in the string itself, preceed the special character with the ~ quote character. For example, the pattern ~[test~] will match the string [test] rather than being interpreted as a range wild-card pattern. Note that the quote character can be changed in the Preferences section.

You can also include variables in your pattern, and the name of the variable will be replaced with its value before the pattern match is performed.

ABORT

Syntax: #AB

Example

Aborts the processing of the current command line.

ABORT example

```
get all corpse;#ABORT;split
```

OK, this is somewhat contrived, but the command get all corpse is sent, then the #ABORT stops further processing so that the split command is ignored..

ACTION example

A simple trigger action

#AC 'chats' '#COLOR red'

whenever a line containing the word chat is received, the color of the line is changed to red.

Triggers for automatic logins

#AC '^Username:' '#CH'
#AC '^Password:' '#PW'

In the Preferences dialog, turn off the Trigger on CR option so that these macros dont wait for a newline character.   Note that the ^ character at the beginning of each pattern forces them to match the beginning of a line.

Parameters in triggers

#AC '^You get (%d) coins' 'split %1' autosplit

Whenever you see a line like You get [number] coins the number of coins is stored in the %1 parameter.   The command then uses this value to split the coins amoung the party members.   A class name of autoplit is used so that you can enable and disable the trigger using the T+ and T- commands.

ALIAS

Syntax: #AL *aliasname string*

Related: #VARIABLE

Example

Assign the command string to the shortcut aliasname. Variables in string are expanded when the ALIAS command is executed. To delay expansion of variables, use two variable characters.

If ALIAS is used with no parameters, all aliases are listed to the output window. If ALIAS is given a single parameter, the definition of aliasname will be displayed.

Aliases can also be expanded via tab completion. If the aliasname is entered into the command line and <TAB> is pressed, the aliasname will be replaced with the string assigned to that alias.

Text following the aliasname in the command line is stored in parameters. These parameters %1 through %9 can be used in the string definition of the alias. Special parameters %%1 through %%9 are also defined which repesent the parameter plus all text following it. Thus, %%1 contains all text following the alias. %%2 contains everything past the first parameter, and so on. Thus, in the example alias foo bar, alias is the aliasname, foo is assign to %1, bar is assigned to %2, foo bar is assigned to %%1, and bar is assigned to %%2. Any text following the aliasname that is not used as a parameter is appended to the results of the alias expansion.

ALIAS example

Simple alias

```
#AL fs 'fill waterskin statue'
```

When fs is entered, the string fill waterskin statue is sent to the MUD.

Using delayed expansion

```
#AL fs 'fill @container statue'
```

When fs is entered, the value of @container is expanded, and the result is send to the MUD. If @container has the value of jug, then the string fill jug statue is sent to the MUD.

Using parameters

```
#AL kk 'kill %1;kick %1'
```

Used with a parameter.   If kk rabbit is entered, the commands kill rabbit and kick rabbit are sent to the MUD.

ADD

Syntax: #AD variable amount

Example

This command allows you to perform simple arithmetic to variables.   The value given by the amount parameter is added to the current value of the variable.   If amount is not numeric, an error occurs.   amount can also be a reference to another variable, adding its current value to the value of the listed variable.   To subtract a value, use a negative amount.

This is the only math function currently implemented in zMUD.   A full MATH command, ala TINTIN, is planned for a future version.

ADD example

```
#AD moves 1
```
Add one to the @moves variable

```
#ACTION 'You get (%d) coins' '#AD gold %1'
```
When you pick up some coins, add their value to the @gold variable.

ALARM

Syntax: #ALA timepattern command

Related: #TRIGGER

Example

Allows you to set up a trigger based upon the time, rather than what is received from the MUD. The timepattern can contain a specific time, or can include wildcards as shown below. If preceeded with a minus (-), the connection time is used rather than the current time.

Typically, the timepattern has the format hours:minutes:seconds, although the hours and minutes are optional. If missing, the hours or minute parameter is assumed to be an asterisk wildcard. In place of a specific numeric value, you can use an asterisk to match any value, or you can list several values seperated by the OR operator (|). You can also use the special wildcard *value which will match when the time MOD the value is zero. E.g. *10 matches 10, 20, 30, etc. Finally, you can put parenthesis around the wildcards to save the values matched to the %1..%9 parameters.

ALARM example

#ALARM -30:00 {save}
The hour isnt specified, so it defaults to *.   Thus, this trigger saves your game every 30 minutes of connect time.

#ALARM 3:00:00 {gossip Why arent you sleeping?}
This triggers at 3am local time.

#ALARM -59:(55|56|57|58|59) {say %eval(60-%1)}
Ok, heres a complicated one.   The pattern starts with a minus sign, so its going to look at the connection time.   Then the hour parameter is missing, so any hour will match.   The minutes is specified at 59.   The seconds match 55 or 56 or 57 or 58 or 59, and the actual value matched is saved to %1 because of the parenthesis.   The command then takes %1 (the matched seconds), subtracts it from 60 and says the result.   The final result of this trigger is on the last 5 seconds of every hour, you say 5 4 3 2 1.

ALL

Syntax: #ALL command

Example

Sends the specified command to all character windows.

ALL example

```
#ALL quit
```

sends the quit command to all active character windows.

BACKUP

Syntax: #BA

Related: #PATH #RETRACE
Example

Remove the last direction from the currently recorded path.

BACKUP example

If the current path is .nsew then #BA will set the path to .nse.   If the current path is .n4s then #BA will set the path to .n3s.

BEEP

Syntax: #BEEP [value]

Related: #PLAY

Example

Plays the current beep sound. If you specify a numeric value, then the cooresponding windows event is played instead. The default beep sound has a value of zero.

Note that zMUD does not pause while the sound is playing. Thus, playing two beeps in a row will only sound one beep, since the first one is still playing when you start the second. To play two beeps, use the WAIT command to insert a delay.

BEEP example

```
#BEEP 16
```
play the sound for windows event 16

```
#BEEP;#WAIT 500;#BEEP
```
sounds two beeps with a 1/2 second delay.   Note that zMUD continues to process other events during the delay, so this wont slow you down.

BUTTON

Syntax: #BU number

Example

Triggers the numbered button (from 1 to 16). This is typically assigned to a macro key. The number parameter can be a variable reference, but must evaluate to a numeric value.

BUTTON example

#BU 1
triggers the first button, just as if you had clicked it.

C+

Syntax: #C+ [name]

Related: #C-
Example

Starts capturing lines and sending them to the specified window.   If name is omitted, lines are sent to the command editor (assuming capturing is enabled within the editor). Otherwise, the lines are sent to the named window.   The window is created if it doesnt exist.

C+ example

```
#C+ temp
```
starts copying all lines received from the MUD to the window named temp.

C-

Syntax: #C-

Related: #C+

Example

Stops capturing text from the MUD.   Opposite of C+.

C- example

#C-
Wow, imagine that!

CAPTURE

Syntax: #CAP [number] [name]

Related: Editor window

Example

Captures the last number lines of text, and copies them into another window. If number is missing, the last line is copied. If number is -1, all lines are copies. If name is specified, the lines are sent to the named window (the window is created if it doesnt exist). If name is omitted, the lines are sent to the command editor window.

CAPTURE example

```
#CAP
```
capture the last line from the MUD and copy it into the editor window.

```
#TRIGGER {tells you} {#CAP tell;#GAG}
```
When a line containing the string tells you is received from the MUD, the capture command copies the line to the tell window, and the gags it from the current window.

CASE

Syntax: #CA index command1 *commandn*

Example

Allows you to select a command from a list to be executed. The index parameter determines the command to execute from the list given by command1..commandn. If index is greater then the number of commands, it wraps around. For example, if there are four commands and you ask for the fifth, the first is returned. This allows you to use the predefined variable %random to select a random command.

If the index is negative, results are undefined.

CASE example

#CASE 2 'first command' 'second command' 'third command'
sends the string second command to the MUD

#CASE @joincmd 'join' 'rescue'
if the variable @joincmd is 1 (or 3,5,7...) the string join is returned, otherwise the string rescue is returned.

#CASE %random 'Hello' 'Hi there' 'Hiya' 'Hi'
returns a random string from the given list to the MUD.

CHARACTER

Syntax: #CH

Related: #HOST #PW
Example

Returns the name of the current character from the Character Database

CHARACTER example

If the current character name is Zugg, then #CH sends Zugg to the MUD.

CLOSE

Syntax: #CL filenum

Related: #FILE

Example

Close the file given by filenum. It must have already been opened using the FILE command.

CLOSE example

```
#CLOSE 1
```

Closes file number 1

COLOR

Syntax: #CO attribute *pattern*

Related: #HIGHLIGHT

Example

If the pattern parameter is left out, this command changes the color of the last line received from the MUD. The color attribute can be a numeric attribute (compatible with the attribute values used by the text modes of DOS) or can be a combination of string values listed below, seperated by commas.

If the pattern is included, a trigger is created to color any line matching the given pattern with the specifed color.

Color values:

black	0
blue	1
green	2
cyan	3
red	4
magenta	5
brown	6
gray	7
yellow	14
white	15
bold	128

to make a color brighter, add 8 to the base value. For example, 9 is bright blue. To change the background color, rather than the foreground, multiply the base value by 16. For example, to get a red background, use 4*16 or 64. To make the foreground font bold, add 128 to the value.

Thus, a bold white on a blue background would be 128 + 1*16 + 15 = 159.

COLOR example

#CO red
changes the color of the last line received to red.

#CO bold,red
changes the last line to bold font and colors it red

#CO 159
set color of last line received to bold white on blue background.

#CO red 'tells the group'
same as #ACTION 'tells the group' '#COLOR red'.   Whenever a string is received from the MUD containing the pattern tells the group, the phrase is colored red.   Only the phrase is used because #COLOR secretly uses the #CW command.   If you really want the entire line colored, you must create the trigger manually using:

#TRIGGER tells the group #COLOR red

CONNECT

Syntax: #CON

Related: #DISCONNECT
Example

Disconnects, then reconnects to the current MUD session. Save as File/Reconnect menu command.

CONNECT example

```
#CON
```

drops the session, then reconnects.   If you are using auto-login triggers, you should be back to where you were.

CR

Syntax: #CR

Example

Send a blank line to the MUD.

CR example

```
#CR
```

send a blank line to the MUD

CW

Syntax: #CW color

Related: #COLOR

Example

If used after a successful trigger, this command will color the phrase matched by the trigger with the specified color.

CW example

```
#TRIGGER {Zugg} {#CW red}
```

Whenever a line is received containing the word Zugg, these words are colored red.   Note that this is similar to the #COLOR command which would color the entire line red instead.

DDE

Syntax: #DDE server topic macro

Example

This command allows you to communicate with an external program via Microsoft Windows Dynamic Data Exchange (DDE). You must consult the documentation of your external program to determine the syntax of its macros, and its defined topic names. The server name is usually the name of the program itself (without the .EXE). There are also built-in functions for DDE:

    %dde(server,topic,item)

Communicates with the specified server and topic and returns the requested item as the value of the function call.

    %ddepoke(server,topic,item,value)

sends value as the new item to the specified DDE server and topic.

    %ddemacro(server,topic,macro)

does the same as this #DDE command.

Note the zMUD also acts as its own DDE server. The server name is zmud, the topic name is also zmud, and the only defined item name is data. Using a DDEPoke you can set data to any command string and cause all variables and function calls to be expanded. You can then yuo a DDE call to retrieve the resulting value of data. Using DDEMacro you can send a command string to zMUD and cause it to be executed as if it were typed.

DDE example

Since DDE is somewhat obscure, I have included a bunch of useful examples:

```
#DDE NETSCAPE WWW_OpenURL http://www.rt66.com/~~zugg/zmud.html
```
If you have NetScape, this command will send the specified URL and cause it to be opened. Note that because the tilde character is used in zMUD to quote special characters, you must use two of them to send one to Netscape.

```
#DDE ZMUD ZMUD remove all;drop all
```
Causes the indicated commands to be sent to zMUD and executed!   Watch out!

```
%dde(Excel,TEST.XLS,R1C1)
```
Tells Microsoft Excel to load the spreadsheet TEST.XLS and returns the value of cell R1C1 (row 1, column 1)

```
%ddepoke(Excel,TEST.XLS,R1C1,@tank)
```
Sends the value of the variable tank to Excel which overwrites R1C1 in spreadsheet TEST.XLS

DISCONNECT

Syntax: #DI

Related: #CONNECT
Example

Disconnects your current session. Careful, because it doesnt ask if youre sure!

DISCONNECT example

```
#TRIGGER {You are BLEEDING} {#DI}
```

Actually not a good idea, but this will disconnect you when you start bleeding, leaving you linkdead.   Note that on most MUDs the monster will keep hitting your linkdead character.

ECHO

Syntax: #EC string

Related: #SAY

Example

Echo the string to the top window. Like the SAY command, except SAY echoes to the window it was issued from. The difference is when performing trigger actions. Using SAY, the trigger will echo the string to the window that issued the trigger. Using ECHO it will echo to the window the user is currently viewing.

ECHO example

```
#TRIGGER {The glow fades} #ECHO Sanc out in window %window
```

When your sanctuary spell runs out in any window containing this trigger, the window that currently has focused is given the message that Sanctuary has run out.

ERASE

Syntax: #ERA filenum

Related: #FILE

Example

Erases the file opened as file number filenum.

ERASE example

```
#FILE 1 old.log
#ERA 1
```

erases file named old.log.   Note that the FILE command prevents you from accessing files outside the ZMUD directory, or from accessing EXE, HLP, or MUD files.

EXEC

Syntax: #EXEC command

Example

Executes the specified command.   Since the command can contain variables which are expanded, this command can be very powerful.
Note: Use this command at your own risk!
If you use this in a trigger, you open yourself up to abuse from other players that will try to set off yor triggers themselves.

EXEC example

```
#TRIGGER ^Zugg tells you (*) #EXEC %1
```

Wow, what a trigger!   Whenever Zugg tells you to do something, you will do it.   Note the importance of using the ^ character to anchor the pattern to the beginning of the line. Without this precaution, somebody could say Aurora tells you Zugg tells you to drop all which would cause you to inadvertantly drop everything you are carrying!

FILE

Syntax: #FI number name

Related: #READ #WRITE

Example

Open a file for reading and writing. zMUD provides 10 files. Files numbered 1-5 are text files that can be read sequentially, or appended to. Files numbered 6-10 are string record files that can be read and written randomly. If the numbered file is already opened, the previous file is closed. The filename given in name is restricted to the current directory containing ZMUD.EXE and cannot refer to a EXE, HLP, or MUD file. This protects you from accidentally modifying important files on your disk.

FILE example

```
#FILE 1 test.txt
```

assigns text.txt to file 1.

FORALL

Syntax: #FO list command

Example

The specified list contains items separated by | characters. This command loops through the list, assigning each element in turn to the %i variable, and executes the command.

FORALL example

```
list=sword|ring|shield
#FORALL @list repair %i
```

Loops through the equipment in the list and repairs each one in turn.

FREEZE

Syntax: #FR *value*

Example

This command causes the output screen to freeze.   Text from the MUD continues to be received, and triggers continue to execute, however, the screen does not scroll.   If value is 0, the screen is unfrozen, otherwise the screen is frozen.   If value is omitted, the current freeze state is toggled.   This command is the same as pressing Control-Z, selecting Freeze from the Window menu, or clicking in the lower right corner of the output window next to the scroll bars.

While the screen is frozen, the output window is surrounded by a red border, rather than a grey border.

FREEZE example

#FR 1
freezes the window

#FR
toggles the current freeze window state

GAG

Syntax: #GA *pattern*

Related: #UNGAG

Example

If the pattern is omitted, this command deleted the last line received from the MUD.   If pattern is included, any line from the MUD matching the pattern is deleted from the input screen.   This allows you to remove text that you don't want to see.   The last syntax is equivalent to #ACTION pattern '#GAG'.

GAG example

#GA
removes the last line received from your screen.

#GA Zugg
removes any lines received from the MUD containing the string Zugg

#GA gossips
removes any lines received from the MUD containing the string gossip.

H+

Syntax: #H+

Related: #H-

Example

Retrieves the next command from the command history. This only works if you have previously used #H- to retrieve the previous command. This command is normally assigned to the down-arrow key.

H+ example

```
test1
test2
#H-
#H+
```
the #H- retrieves the test1 command, while the #H+ retrieves the test2 command.

H-

Syntax: #H-

Related: #H+

Example

Returns the previous command from the command history buffer. This command is usually assigned to the up-arrow key.

H- example

```
test1
test2
#H-
```

Retrieves the command test1 from the history buffer.

HELP

Syntax: #HE [command]

Example

Without any parameters, this command displays the Help table of contents. If you specify a command, help on that command is displayed.

HELP example

```
#HELP alias
```
displays help about the alias command

HISTORY

Syntax: #HIS

Example

Displays the last 20 commands in the output window. The number preceeding each command is the command line number. You can execute one of the previous commands by preceeded the command line number with the history character, which defaults to !. !! executes the most recent command. You can also execute a previous command using ! pattern where pattern matches the text at the beginning of a previous command.

You can also pop up an interactive history dialog by right clicking in the output window, or by left clicking on the arrow button at the left of the command input line. The dialog shows the last 20 commands. If you single-click on a line, the command is copied into the edit line at the top of the dialog. You can then edit the command and press <Enter> to execute it. If you double-click on a command, it is sent directly to the MUD without allowing you to edit it, and the history dialog will close. To close the dialog without executing a command, right click on it.

Tab completion can also be used with history. If you use the history character (!) to specify a command, either numerically, or by using a pattern, then press <TAB>, the matching command will be copied into the command input buffer for editing.

HISTORY example

#HI
displays the last 20 commands

!!
executes the last command again

!3
executes the third command in the history buffer

!k
executes the last command beginning with the string k

!k<TAB>
retrieves the last command beginning with the string k for editing in the command line.

HIGHLIGHT

Syntax: #HI *pattern*

Related: #COLOR

Example

If pattern is omitted, this command makes the last line received from the MUD bold. If pattern is included, any line matching the pattern received from the MUD is made bold. This last syntax is equivelant to the command #ACTION pattern '#HIGHLIGHT'.

HIGHLIGHT example

#HI
make the last line received from the MUD bold.

#HI Zugg
highlight any line received from the MUD containing the string Zugg.

HOST

Syntax: #HO

Related: #CHAR #PW

Example

Send the name of the current MUD host.

HOST example

If the current host name is foo.bar.edu then #HO sends the string foo.bar.edu to the MUD.

IF

Syntax: #IF expression true-command *false-command*

Example

Allows conditional execution. If the expression is true, then the true-command is executed. If the expression is false, then the false-command (which is optional) is executed. Expressions can contain variables and operators.

Expressions

zMUD implements full expressions. Expressions can contain variables, and most common operators. Parenthesis can be used to override default operator presedence. When evaluating an operation, if all parameters of the operation are numeric, then a numeric operation is used, otherwise a string operation is used. The following operators are recognized (v1 and v2 represent variables, or other expressions):

v1 + v2	add value1 to value2. If values are not numeric, the text values are concatenated.
v1 - v2	subtract value2 from value1
v1 * v2	multiply value1 by value2
v1 / v2	divide v1 by v2. Any fraction is discarded.
v1 | v2	divide v1 by v2 and return the modulo
v1 & v2	returns the logical AND of value1 and value2
v1 and v2	same as above
v1 | v2	returns the logical OR of value1 and value2
v1 or v2	same as above
v1 xor v2	returns the logical XOR of value1 and value2
v1 = v2	true if value1 is the same as value2
v1 > v2	true if value1 is greater than value2
v1 < v2	true if value1 is less than value2
v1 >= v2	true if value1 is greater than or equal to value2
v1 <= v2	true if value1 is less than or equal to value2
v1 <> v2	true if value1 is not equal to value2
v1 != v2	true if value1 is not equal to value2
v1 =~ v2	true if the pattern in value1 is contained in value2
v1 ~= v2	same as =~
-v1	return the negative of value1
!v1	return the logical NOT of value1

If the pattern matching=~ operator is used, any saved pattern parameters are available in the true-command or false-command if expression is part of an IF command.

IF example

#IF @autosplit 'split @gold'
If the @autosplit variable is non-zero, then the value of @gold is expanded, the string split is sent to the MUD followed by the value of @gold.

#IF (@gold < 100000) 'emote is poor' 'emote is RICH!'
If the value of the @gold variable is less than 100000, then the string emote is poor is sent to the MUD, otherwise the string emote is RICH! is sent to the MUD.

#IF (@line =~ "You receive (%d) coins") 'split %1'
If the value of the variable @line matches the pattern You receive %d coins, then the number of coins matched is stored in the %1 parameter, and the string split is sent to the MUD, followed by the parameter.   Note the nested quotation marks needed to properly parse this command.

IGNORE

Syntax: #IG

Related: #T+ #T-
Example

Toggle the execution of all trigger actions.

IGNORE example

#IG
start ignoring all triggers.

#IG
turn execution of triggers back on again.

INPUT

Syntax: #IN string

Example

Copy the string into the current command buffer, replacing the current contents.

INPUT example

#IN get @item
Expand the value of the @item variable and place the command get followed by this value into the current command buffer.

KEY

Syntax: #KE key command

Example

Assign a command to a key. key should be the full name of the key, for example, F1, or CTRL-A, or ALT-F2.

As an alternative syntax, you can use <key>=command as an assignment statement.

KEY example

```
#KEY F1 eat bread
```
assign the eat bread command to the F1 key

```
<ALT-D>=drink water
```
assign the command drink water to the ALT-D key

KILLALL

Syntax: #KILLALL

Example

Erase all macro keys, aliases, variables, triggers, tab completion words

KILLALL example

#KILLALL
deletes everything (well almost).   This command cannot be abbreviated.

LOAD

Syntax: #LOA filename

Related: #SAVE

Example

Load the specified settings file into the current window. Note the the current settings file is not saved! Any variables in filename are expanded.

LOAD example

```
#LOAD dc
```
Load the dc.mud settings file (.MUD is the default extension)

```
<F1>=#LOAD combat;<F2>=#LOAD social
```
loads the combat.mud file when you press F1, and loads the social.mud file when you press F2

LOOP

Syntax: #LOO range command

Example

Execute the command a number of times given by the range. The range consists of a minimum numeric value, followed by a maximum value, seperated by a comma. If only the minimum value is given, then a loop of 1,value is assumed (executing the command the number of times stated by the value). The current value of the loop variable is stored into the %i variable for use in the command.

LOOP example

```
#LOO 3 'north'
```
sends the command north to the MUD three times

```
#LOO 1,4 'get coins %i.corpse'
```
sends the commands get coins 1.corpse, get coins 2.corpse, get coins 3.corpse, get coins 4.corpse to the MUD

```
#LOO @num 'eat bread'
```
sends the eat bread command to the MUD the number of times contained in the @num variable.

LOG

Syntax: #LO *filename*

Example

Given a filename parameter, this command creates a logfile with the specified file name. If the file already exists, it is opened for appending. If the file does not exist, it is created. If the filename is omitted, then the logging flag is toggled.

LOG example

```
#LO test.txt
```
start logging all input from the MUD to the file test.txt

```
#LO
```
toggle the logging flag.

MAP

Syntax: #MAP direction

Related: #PATH

Example

Add the specified direction to the current path being recorded.

MAP example

#MAP north
if the current path is .s then the path is updated to be .sn.   If the current path is .2n then the path is updated to be .3n.

MATH

Syntax: #MAT variable expression

Related: #ADD

Example

Assigns the value of the expression to the given variable.   Expressions can contain numeric, logical, and text functions.   Any variables in the expression are also expanded.

MATH example

```
#MATH test (1+3)*4
```
assigns the value of '16' to the variable @test.

```
#MATH test2 @test-4
```
if @test has the value of 16, the value of 12 is assigned to @test2

```
#ALIAS add '#MATH value %1+%2'
add 3 4
```
the value of 7 is assigned to the variable @value

MARK

Syntax: #MA

Related: #PATH

Example

Mark the beginning of a path. Clears the currently recorded path.

MARK example

#MA
clears the current path and marks the beginning of a new path.   Turns on path recording.

MEDIA

Syntax: #ME function

Related: #PLAY

Example

Sends a command to your current multimedia device. Usually, you use this command after starting to play something with the play command. Any variables in function are expanded. Possible values of function are:

back	step the media backwards
close	close the current file
eject	eject the media
next	go to next track
pause	pause the playing of the media
play	start playing the media
prev	go to the previous track
resume	resume playing after a pause
rewind	send media back to beginning
step	step the media forwards
stop	stop playing the media

MEDIA example

```
#MEDIA next
```
if you are playing a CD, this moves to the next track

MEMORY

Syntax: #MEM

Example

Displays the amount of Windows memory you have left.   Mostly for debugging purposes.

MEMORY example

```
#MEM
16575234 bytes available
```

MENU

Syntax: #MEN command

Example

Executes the specified menu commands. The title of each menu command is given in the menu reference section. Each menu must be listed seperated by | characters. Thus, the Exit command is specified as File|Exit. There is no way for zMUD to fill in any dialog value, this command just executes the menu command as if the user clicked on the menus.

MENU example

#MENU File|Exit
Exits zMUD! (You wouldnt really do this, would you?)

#MENU Settings|Edit|Triggers
Pops up the trigger dialog box.

MESSAGE

Syntax: #MES string

Example

Displays a small window containing the specified string as a message. The window is small and surrounded by a red border. There is a button in the window to close it. If the window is not closed within 10 seconds, it closes automatically.

MESSAGE example

```
#MESS Sanctuary is out!
```
Displays the specified message in an alert window.

NAME

Syntax: #NA string

Example

Change the name of the current window. The default name of a window is the name of the MUD character. With this command you can change the name to anything you want. This name is saved with the character.

NAME example

#NAME tank
gives the current window the name of tank.   Now when you say tank:command the command is sent to this window.

NOOP

Syntax: #NO

Example

Does nothing! (Wow, what a powerful command)

NOOP example

You really need an example of this one???

OK

Syntax: #OK

Related: #SLOW #STOP

Example

Confirms the currently pending slow walk direction. This command is typically triggered by a macro to indicate that the previous movement was successful.

OK example

```
#TRIGGER {^Exits} {#OK}
#TRIGGER {^It is pitch black} {#OK}
```

When either of these two strings are received from the MUD, the pending Slow Walk direction is confirmed, allowing slow walking to continue.

PATH

Syntax: #PA *pathname*

Related: #MARK #RETRACE #MAP
Example

If pathname is omitted, the current path being recorded is displayed. If pathname is given, the currently recorded path is saved to the variable specified by pathname. The direction character (.) is prepended to the variable name automatically.

PATH example

#PA
displays the current path being recorded

#PA magic
saves the current path to the variable .magic.

PICK

Syntax: #PI val1 val2 val3 ...

Example

Display a pickbox with val1, val2, val3, ... listed on each line (up to nine values can be specified). The user can select one or more of these lines and return them. The values are returned seperated by the current command seperator (;).

PICK example

#PI get all corpse get all.coins corpse sac corpse
displays a picklist with three values.   If the user selects the first and third option, the string get all corpse;sac corpse is returned and executed.

PLAY

Syntax: #PL filename

Related: #MEDIA
Example

Plays the specified multimedia file.   The file type is determined from the file extension of the filename.   Tested formats are WAV, MID, AVI.   If filename refers to a drive (as in D:), the drive is assumed to point to a CDROM with a musical CD in it.

Note that zMUD does not pause while the media is playing.   Also, only one media can be playing at any given time.

PLAY example

#PLAY start.wav
play the startup wav audio file

#PLAY D:
start playing the CD on drive D:

sound=ouch.wav
#TRIGGER hits you #PLAY @sound
plays the file ouch.wav whenever you are hit.   There are real possibilities here!

PROMPT

Syntax: #PR aliasname

Example

Pops up a dialog box to prompt you for the value of the specified alias/variable.

PROMPT example

#PR tank
Displays a dialog box asking you to enter the value for the @tank variable.   The current value of the variable is the default.

PW

Syntax: #PW

Related: #CHAR

Example

Sends the password of the current character to the MUD. This password is not echoed to the output window.

PW example

```
#PW
```
sends your password to the MUD.

READ

Syntax: #REA filename

#REA n rec

Example

Open the file given by filename and read it line by line, executing each line.   This allows you to store commands in a script file and then execute this script.   Typically the KILLALL command will be used to clear memory before reading the file.

The second form of this command reads data from the nth file (opened with the FILE command).   If n is 1-5, then the file is a text file and rec is the line number to read.   If rec is zero or omitted, the next sequential line is read.   If n is 6-10, then the file is a structured file, and the record indicated by rec is read.   If rec is zero or omitted, the next record is read.

READ example

```
#REA mud.txt
```
Read the file mud.txt line by line and execute each line as if you had typed it manually.

```
#FILE 1 mudlist.txt
#READ 1 10
```
read the 10th line from the file mudlist.txt

RECORD

Syntax: #REC [aliasname]

Related: #ALIAS

Example

Toggles the recording of an alias. When you first enter #RECORD, zMUD starts recording all the commands you send to the MUD. You can monitor this recording by entering #RECORD again during the recording. When you are done and want to save the recording, enter the #RECORD command followed by the name of the alias you wish to create. If you use an alias value of off (#RECORD off), the recording is stopped and not saved.

RECORD example

```
#REC
starts recording
n
w
open door
#REC
displays: Current alias: n;w;open door
#REC temple
saves the commands n;w;open door to the alias named temple and turns off recording.
```

RESET

Syntax: #RES *n*

Related: #FILE

Example

Reset the nth file back to the beginning. The file must be opened using the FILE command.

RESET example

```
#RES 1
```

reset file 1 to the beginning.

RETRACE

Syntax: #RE *pathname*

Related: #PATH

Example

Executes the specified path backwards, allow you to retrace your steps as long as you are in a euclidean section of the MUD. If the pathname is omitted, the path currently being recorded is reversed.

RETRACE example

#RE magic
If the .magic path contains .2s2wn then this will execute the path .s2e2n.

SAVE

Syntax: #SAV [filename]

Related: #LOAD

Example

Saves the current settings file. If you specify a filename, the settings are saved to that file instead. Note that the name of the settings file associated with your current character is not changed by this command.

SAVE example

```
#SAVE new
```
saves the current settings to the file new.mud

SAY

Syntax: #SA text

Example

Same as the SH command. Echoes the specified text to the screen without sending it to the MUD.

SAY example

#SA You have @gold coins
Prints You have nnnn coins to the screen where nnnn is the current value of the @gold variable.

#ACTION 'aura fades' '#SA SANC IS OUT!!!!;#COLOR red'
If the string aura fades is received from the MUD, the string SANC IS OUT!!!! is displayed to your screen and colored red.

SEND

Syntax: #SE filename prefix

Example

Sends the contents of filename to the MUD. Each line of the file is prefixed by the prefix string before being sent.

SEND example

```
#SEND notes.txt tell zugg
```

Sends the file notes.txt and prefixes each line of the file with tell zugg

SESSION

Syntax: #SES [character-name|hostname port]

Example

Opens a new MUD session to the specified character or host. If you enter a single parameter, it should be the ID of a character in the character database that you want to run. If you specify two parameters, the first is the host name or IP address of a MUD, and the second parameter is the port number.

SESSION example

```
#SES Zugg
```
Opens a new session to the entry in the character database with an ID of Zugg

```
#SES jitter.rahul.net 6666
```
opens a new session to the MUD on host jitter.rahul.net port 6666.

SHOW

Syntax: #SH text

Example

Echoes the specified text to the screen without sending it to the MUD. Similar to SAY except that the text is processed just as if it were received from the MUD (usually for testing triggers). Note that SHOW does not automatically send a newline character.

SHOW example

#SH You have @gold coins%char(10)
Prints You have nnnn coins to the screen where nnnn is the current value of the @gold variable.   the %char(10) at the end forces a newline to be sent.   Any trigger set up for this type of pattern will get triggered.

SLOW

Syntax: #SL path

Related: #STEP #STOP #OK

Example

Executes the specified path in Slow Walking mode. In this mode, a single direction is sent to the MUD, then zMUD waits for confirmation before sending the next direction. Directions are confirmed with the #OK command, and aborted with #STOP. If a Slow Walk was aborted, it can be resumed with the #STEP command

SLOW example

```
#SL .n2es
```

Sends the north command to the MUD.   Then waits for confirmation.   If confirmed, then east is sent, and so on.

STATUS

Syntax: #ST *text*

Example

Sets the definition used for the status line displayed beneath the output window.   This line can contain variables which are expanded before the status line is displayed.   The status line is updated whenever a variable is changed.

STATUS example

```
#ST Gold: @gold   Tank: @tank
```

If the value of @gold is 1234 and the value of @tank is Zugg, then the text

```
Gold: 1234   Tank: Zugg
```

is displayed on the status line.

STEP

Syntax: #STE

Related: #SLOW #STOP #OK

Example

Resumes a previous aborted Slow Walk by executing the next step in the path.

STEP example

#SLOW .n2es
North is sent, but never confirmed, so Slow Walking stops
#STEP
sends north once again to restart the Slow Walk.

STOP

Syntax: #STO

Related: #SLOW #STEP #OK

Example

Aborts the current Slow Walk. Typically used in triggers.

STOP example

```
#TRIGGER {A gang member is here} {#STOP;kill gang}
```
If the pattern A gang member is here is received from the MUD, any Slow Walk is aborted and the gang member is killed.   You can then resume your walking with #STEP after the gang member is dead.

STW

Syntax: #STW string

Related: #STATUS

Example

Specifies the definition of the status window. The status window is like the status line except it can contain more than one line, and can contain %ansi color sequences. The status window can be positioned and sized anywhere on the screen (the position and sized is remembered). You can use the %CR function to insert a newline, and the %ANSI function to change the color of text. Right clicking on the status window also lets you set its definition string.

STW example

```
#STW   Hp: @hp %cr Exp: @exp %cr %ansi(red)Tank: @tank
```

Defines a three line status window.   The first line shows that current hitpoints in the @hp variable, the next line shows the experience in the @exp variable, and the last line shows the name of the current tank in red.

SUBSTITUTE

Syntax: #SU string

Example

This command is used in conjunction with triggers to change the text matched by the last trigger pattern to something else. It is useful for removing clutter on the screen.

SUBSTITUTE example

```
#TRIGGER {(*) tells you,} {#SUB {%1:}}
```

Replaces all lines received from the MUD of the form xxx tells you, with the text xxx: .   This saves room on the screen and is expecially useful in a subwindow that captures text matching a pattern.

T+

Syntax: #T+ classname

Example

Enable execution of all triggers with the specified classname.

T+ example

#T+ autosplit
turn on the triggers in the autosplit class.

T-

Syntax: #T- classname

Example

Disabled execution of all triggers with the specified classname.

T- example

#T- autosplit
turn off all triggers in the autosplit class.

T?

Syntax: #T?

Related: #TIMER #TS
Example

Display the amount of time remaining in the timer.

T? example

```
#T?
```

If 10 seconds are left in the timer, the string 10 secs is displayed.

TAB

Syntax: #TA word

Example

Add the specified word to the tab completion list. If you type the first part of this word and press <TAB> the rest of the word will be filled in for you.

TAB example

#TA zugg
Add zugg to the tab completion word list.   If you now enter z then press <TAB> the ugg will be filled in for you.

TIMER

Syntax: #TI

Related: #T? #TS

Example

Toggles the timer. If the timer is off, it is turned on. If it is on, then it is turned off. Note that the amount of time left in the timer is not effected by this command.

TIMER example

#TI
turns on or off the timer.

TS

Syntax: #TS *value*

Related: #TIMER #T?

Example

Sets the value of the timer and starts the countdown. At 5 seconds before the counter hits zero the string TICK IN 5 SECONDS. is displayed on the screen. If the value is omitted, then the origin of the timer is reset.

This timer is typically used to time ticks in the MUD. To get started, enter #TS value where value is the approximate time between ticks. When a tick actually arrives, enter #TS without a parameter to fine tune the timer interval. You can then set up an action that triggers on the TICK IN 5 SECONDS. string to perform an action such as resting.

TS example

#TS 60
set the tick timer to 60 seconds and begin the countdown

#TS
refine the timer interval.   Use this when the tick actually arrives.

TRIGGER

Syntax: #TR pattern command *classname*

Example

This is one of the most powerful features of zMUD. It allows you to define a command to be executed whenever the matching text is received from the MUD.

The first parameter is the text to be matched. If the text contains a space, you need to enclose it in quotes. This pattern can contain special pattern matching symbols and wildcards. The second parameter is the command to be executed when the pattern is received from the MUD. Since this command usually consists of more than one word, you must enclose it in quotes. The third parameter is optional, and is the name of the trigger action class that this action is part of. Triggers can be enabled and disabled when part of a class.

For advanced trigger options, you must go to the Preferences dialog. In this dialog, you can determine whether the action is triggered at the end of each line received from the MUD, or if it is just triggered at the end of receiving a block of data from the MUD. Responding to MUD prompts such as Username and Password require a trigger that activates after a block of text is received since these prompts are not normally followed by a newline.

In the Preferences dialog you can also determine whether the trigger action is enabled when you first load the data file. You can also individually enable or disable triggers individually as well as by class name.

TRIGGER example

A simple trigger action

```
#TR 'chats' '#COLOR red'
```

whenever a line containing the word chat is received, the color of the line is changed to red.

Triggers for automatic logins

```
#TR '^Username:' '#CH'
#TR '^Password:' '#PW'
```

In the Preferences dialog, turn off the Trigger on CR option so that these macros dont wait for a newline character.   Note that the ^ character at the beginning of each pattern forces them to match the beginning of a line.

Parameters in triggers

```
#TR '^You get (%d) coins' 'split %1' autosplit
```

Whenever you see a line like You get [number] coins the number of coins is stored in the %1 parameter.   The command then uses this value to split the coins amoung the party members.   A class name of autoplit is used so that you can enable and disable the trigger using the T+ and T- commands.

TYPE

Syntax: #TY filenum [pattern]

Related: #FILE
Example

If pattern is omitted, the entire contents of the numbered file are displayed on the screen (starting from the current position of the file).   If pattern is given, only lines matching the pattern are displayed.   Pattern can contain full pattern-matching commands.

TYPE example

```
#FILE 1 mudlist.txt
#TYPE 1
```
displays the entire contents of the file mudlist.txt to the screen

```
#TYPE 1 castle
```
only displays lines from mudlist.txt containing the word castle

TZ

Syntax: #TZ

Related: #TS

Example

Resets the tick timer to zero but does not change its duration.

TZ example

```
#TZ
```
Resets the tick timer to zero

```
#TRIGGER {The sun rises} {#TZ}
```
Resets the tick timer when the text The sun rises is received from the MUD.

UNALIAS

Syntax: #UNA alias

Related: #ALIAS

Example

Deleted the specified alias from memory.   Careful, there is no way to get it back after you do this.

UNALIAS example

```
#UNA kk
```

removes the alias called kk

UNGAG

Syntax: #UNG

Related: #GAG

Example

Prevents the current line from being gagged. Typically used in a trigger to undo the GAG action of a previous trigger.

UNGAG example

```
#TRIGGER {Zugg} {#GAG}
#TRIGGER {tells you} {#UNGAG}
```

Normally, the first trigger would gag all lines that contain the string Zugg. However, the second trigger looks for the string tells you and ungags it. Thus, the string Zugg tells you will still be displayed.

UNKEY

Syntax: #UNK key

Related: #KEY

Example

Deleted the specified key macro from memory.   Careful, there is no way to get it back after you do this.

UNKEY example

```
#UNK <F1>
```

removes the macro assigned to the F1 key

UNTRIGGER

Syntax: #UNT pattern

Related: #TRIGGER
Example

Deleted the trigger assigned to the specified pattern from memory.   Careful, there is no way to get it back after you do this.

UNTRIGGER example

```
#UNT {tells you}
```

removes the trigger associated with the pattern tells you

UNVAR

Syntax: #UNV variable

Related: #VAR

Example

Deleted the specified variable from memory.   Careful, there is no way to get it back after you do this.

UNVAR example

#UNV tank
removes the variable called tank

VARIABLE

Syntax: #VA variable value

Example

Similar to the ALIAS command. Assigns the specified value to a variable. You do not need to specify the @ variable character. This allows you to define variables independent of the user's variable character setting.

An alternative syntax is variable = value or variable := value.

VARIABLE example

#VA coins 1000
assign 1000 to the @coins variable

coins = 1000
same as above.

VERSION

Syntax: #VE

Example

Displays the current version and date of zMUD

VERSION example

You dont really need an example of this, do you?

WAIT

Syntax: #WA *time*

Example

Delays processing of further commands on the line until text is received from the MUD. Sometimes command from zMUD can be executed too fast for the MUD. This command is used to slow them down.

If the time parameter is specified, the processing of further commands on the line is delayed the amount of time specified. Time is in units of milliseconds - e.g. 1000 is 1 second. Note that the commands remaining on the line are queued up for this time. You can still enter other commands on the command line while waiting. Only one WAIT can be active at a time. Starting a second WAIT cancels the first and discards the commands in the queue.

WAIT example

west;#WA;kill citizen
sends the west command to the MUD, then waits for some output from the MUD, then sends the kill citizen command.   Without the #WA command, on some MUDs the kill citizen would be sent before you actually moved west.

#WA 2000;kill citizen
wait for 2 seconds, then send the kill citizen command to the MUD

WINDOW

Syntax: #WIN name [filename]

Example

Bring the specified window to the top.   If the window doesnt exist, it is created.   Note that when the window is created, the file name.ZCR or name.MUD is loaded (ZSC is a text script file, MUD is a binary settings file).   If filename is specified, then that settings file is loaded instead.

WINDOW example

#WIN tell
create a new window named tell.   It tried to load tell.zsc or tell.mud as the settings for this window.

WIZLIST

Syntax: #WI

Example

Display the credits for zMUD.

WIZLIST example

Try it and see!

WRAP

Syntax: #WR [column]

Example

Toggles the auto word wrap mode.   If column is specified, the text is wrapped at the given column.

WRAP example

#WRAP
Turns on word wrapping

WRITE

Syntax: #WR n value [rec]

Related: #READ

Example

Write a value to the nth file. If n is 1-5 then the file is a text file, and the value is appended to the end - rec is ignored. If n is 6-10, then the file is structured, and value is written to the record given by rec. If rec is zero or omitted, value is appended to the file.

WRITE example

```
#WR 1 logged onto Dark Castle
```
append the string to the end of text file number 1.

```
#WR 6 this is record 3 3
```
writes the string as the third record in file 6.

Repeating Commands

Syntax: #number command

Example

The specified command is sent to the MUD the number of times given by the number parameter. This number must be a constant. To use variables, see the LOOP command. The current value of the repeat counter is saved in the predefined variable %repeatnum for your use in the command.

Repeating example

```
#10 kill mound
```
send the command kill mound to the MUD 10 times.

Zugg's Multi-User Dungeon client

Multi-User Dungeon