

Genie3

Documentation

Revision 1

Copyright © 2009 [genieclient.com](#). All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner.

Table of Contents

1. Beginners Guide	7
2. Installation Troubleshooting	8
3. Profiles	9
3.1. What are profiles?	9
3.2. How do I create a profile?	9
3.3. How do I use a profiles?	9
4. Copy / Paste	10
5. Macros	11
5.1. What is a macro?	11
5.2. How do I add a macro?	11
5.1. Using the Command Line	11
5.1. What Can I Bind?	12
5.2. Intermediate Macro Functionality	12
5.2.1. Multiple Commands	12
5.2.2. Macros that do not auto trigger	13
5.3. Advanced Macro Functionality	14
5.3.1. If Functions in Macros	14
5.3.2. Setting Variables in Macros	14
5.3.3. Nested If Functions	15
5.3.4. AND / OR Functionality in an IF Macro	15
6. Ignores	17
6.1. What is an ignore?	17
6.2. How do I add / edit / delete an Ignore	17
6.2.1. Ignore Case	17
6.3. Gagging with Regular Expressions	17
6.4. Things to be Wary Of	18
6.5. Parse Order	18
7. Aliases	19
7.1. What is an Alias?	19
7.2. How do I add / edit / delete an Alias	19
7.3. Using Variables in an Alias	20
8. Substitutes	22
8.1. What is a Substitute?	22
8.2. How do I Add / Edit / Delete a Substitute?	22
8.3. Using Variables / RegExp in a Substitute	22
9. Triggers	24
9.1. What is a Trigger?	24
9.2. How do I Add / Edit / Delete a Trigger?	24
9.3. Setting a variable with a Trigger	25
9.4. Eval Triggers	25
9.5. Advanced Triggers	25
10. Highlights	27
10.1. What Is A Highlight?	27
10.2. Types of Highlights	27
10.3. How do I Add / Edit / Delete a Highlight?	27

10.3.1. Highlight Types.....	28
10.3.1.1. String.....	28
10.3.1.2. Line.....	28
10.3.1.3. Begins With.....	28
10.3.1.4. RegExp.....	28
10.3.2. Case Sensitive.....	29
10.3.3. Colors.....	29
11. Plug Ins.....	30
11.1. What is a Plugin?.....	30
11.2. How Do I Install A Plugin?.....	30
11.3. Configuring Plug Ins.....	30
11.4. Writing a Plug In.....	30
11.5. Useful Plugins.....	31
12. Status Bar.....	32
12.1. What is the Status Bar?.....	32
12.2. How do I add something to the statusbar?.....	32
12.3. EVAL Triggers.....	33
12.4. Advanced Statusbars.....	33
13. Automapper.....	35
13.1. What is the Automapper?.....	35
13.2. How to use the Mapper.....	35
13.2.1. Levels of a Map.....	36
13.2.2. Zoom.....	36
13.2.3. #goto.....	36
13.2.4. #goto <room id number>.....	36
13.2.5. How to Record a Map.....	36
13.2.6. How to Edit a Map.....	37
13.2.6.1. Moving Nodes and Labels.....	37
13.2.6.2. Labels.....	37
13.2.6.3. Deleting Nodes.....	37
13.2.6.4. The Edit Panel.....	38
14. Layout.....	39
14.1. What are layouts?.....	39
14.2. How to Add a Window.....	39
14.3. How to Remove a Window.....	39
14.4. How to Reposition a window.....	39
14.5. Changing Font.....	39
14.5.1. Fixed Width Fonts.....	40
14.6. Time Stamp.....	40
14.7. Layouts.....	40
14.7.1. Named Layouts.....	40
14.7.2. Bars and Panels.....	40
14.7.2.1. Icon Bar.....	41
14.7.2.2. Script Bar.....	41
14.7.2.3. Health Bar.....	41
14.7.2.4. Magic Panels.....	41
14.7.2.5. Status Bar.....	41
15. File Menu.....	42

15.1. Connect	42
15.2. Auto Log	42
15.3. Open Log In Editor	42
15.4. Auto Reconnect	42
15.5. Ignores/Gags Enabled	42
15.6. Triggers Enabled	43
15.7. Plugins Enabled	43
15.8. Automapper Enabled	43
15.9. Mute Sounds	43
15.10. Show XML Raw Data	43
16. Variables	44
16.1. What Is A Variable?	44
16.2. How to Set / Edit a Variable	44
16.2.1. Macro	45
16.3. Built in Variables	45
17. Command Line	48
17.1. Command Line Reference	48
17.1.1. #beep	48
17.1.2. #clear	48
17.1.3. #colors	49
17.1.4. #comment	49
17.1.5. #connect	49
17.1.6. #echo	50
17.1.7. #edit	51
17.1.8. #eval	51
17.1.9. #evalmath	52
17.1.10. #event	52
17.1.11. #flash	53
17.1.12. #help	53
17.1.13. #if	53
17.1.14. #keys	54
17.1.15. #log	54
17.1.16. #math	54
17.1.17. #parse	55
17.1.18. #play	55
17.1.19. #playsystem	56
17.1.20. #put	56
17.1.21. #queue	57
17.1.22. #script	57
17.1.23. #send	58
17.1.24. #statusbar	58
17.2. Configuration Commands	59
17.2.1. #alias	59
17.2.2. #class	59
17.2.3. #config	60
17.2.4. #gag	60
17.2.5. #highlight	61
17.2.6. #layout	61

17.2.7. #load	61
17.2.8. #macro	62
17.2.9. #name	62
17.2.10. #preset	63
17.2.11. #profile	63
17.2.12. #save	63
17.2.13. #subs, #substitute	64
17.2.14. #trigger	64
17.2.15. #un*	65
17.2.16. #var, #var, #variable	65
17.2.17. #window	66
17.3. Automapper Commands	66
17.3.1. #goto	66
17.3.2. #mapper	66
17.4. Evaluate	67
17.4.1. Keywords	67
17.4.2. Functions	67
17.5. Evaluate Math	68
17.5.1. Keywords	68
17.5.2. Functions	68
18. RegExp	69
18.1. What is a Regular Expression?	69
18.2. Beginners Guide to the Essential Regular Expressions	69
18.3. How do I use a Regular Expression?	70
18.4. Other Regular Expressions	71
18.5. Advanced Regular Expression Examples	73
18.6. Regular Expression Language Elements	74
18.7. Testing your Regular Expression	74
19. Scripts	75
19.1. What is a Script?	75
19.2. How do I write a Script?	75
19.3. Creating / Editing A Script	75
19.4. Differences with the Genie Scripting Engine	76
19.5. Multiple Scripts?	76
19.6. How to Download and Run Scripts	76
19.7. Script Variables	77
19.8. Global Variables	78
19.9. Essential Commands	79
19.10. Beginners Script	79
19.11. Intermediate Scripting	81
19.12. Script Command Reference	82
19.12.1. action	82
19.12.2. action classes	83
19.12.3. counter	83
19.12.4. debug	83
19.12.5. delay	84
19.12.6. echo	84
19.12.7. eval	84

19.12.8. evalmath	84
19.12.9. exit	84
19.12.10. gosub	84
19.12.11. goto	85
19.12.12. if	85
19.12.13. if_X	85
19.12.14. else	85
19.12.15. include	85
19.12.16. match	85
19.12.17. matchre	86
19.12.18. matchwait	86
19.12.19. math	86
19.12.20. move	86
19.12.21. nextroom	86
19.12.22. pause	87
19.12.23. put	87
19.12.24. random	87
19.12.25. return	87
19.12.26. save	87
19.12.27. send	87
19.12.28. shift	88
19.12.29. timer	88
19.12.30. var	88
19.12.31. unvar	88
19.12.32. wait	88
19.12.33. waitfor	89
19.12.34. waitforre	89
19.12.35. labels	89
19.12.36. script blocks	89

1. Beginners Guide

If you are new to genie, it is worth having a look at the sections on [Regular Expectations](#) (particularly the [Beginners Guide to the Essential Regular Expressions](#)) and [Variables](#). A good understanding of these sections will enable you to make the most of Genie's functionality. The best way to master these concepts is to incorporate some basic examples into your version of Genie. Gradually as you try to do more and more with them your knowledge will increase. The forums are an excellent source of information and while some examples may not be directly applicable to your character or needs, you can often learn a lot from examining other peoples examples to understand why they work.

You might also want to look at the [Scripting Section](#), particularly the [Beginners Script](#) sub-section if you plan to make use of scripts.

2. Installation Troubleshooting

The most common problem encountered when installing genie is with your .NET Framework. Make sure you have .NET installed. If Genie still does not work, try a full uninstall and then a clean install of the latest Microsoft .NET Framework version before trying again.

If you still can not get it to work, please contact:

support@genieclient.com

3. Profiles

3.1. What are profiles?

Profiles are designed to store the configuration specific to your character on your computer for future use. This allows you to click on 'Connect Using Profile' instead of retyping your account details in the Connect window every time you wish to connect to the game.

3.2. How do I create a profile?

To create a profile you need to first connect to the game, then click on 'Save Profile' in the Profile Menu. The profile is now saved.

You may also choose to save the account password. The Password is saved, encrypted, on your local disc to increase security.

Warning: If you use this option anyone with access to your computer files may connect to your character!

3.3. How do I use a profiles?

To connect to the game using your profile simply click on the 'Connect Using Profile' option in the File Menu. You will not have to fill in your details as they are now saved in your Profile.

You can also connect using your profile via the Command Line by typing

```
#connect <profile name>
```

This command can be assigned as a Macro for convenience.

4. Copy / Paste

Genie is designed to make copying text a quick and easy process. Instead of highlighting the text and right clicking to copy or using the Ctrl+C shortcut as you normally would, Genie automatically sends any highlighted text to the Clipboard. This means that all you need to do to select text from a window, then click Ctrl+V (paste) in your desired location.

5. Macros

5.1. What is a macro?

Simply put Macros bind actions within the game to a specific key on your keyboard. This enables you to program commands that you perform repeatedly into Genie so that you can execute them at the press of a button rather than repeatedly typing them out.

The most basic macros involve binding a single command. For example you could set Genie to use the ‘Slice’ attack whenever you press the number 5 on your keyboard. More complex Macros are capable of handling multiple commands and even logic functions.

5.2. How do I add a macro?

The macro entry window can be found by going to the [Edit](#) menu and selecting [Configuration](#). In the Configuration window choose the ‘Macros’ tab, this will display a list of all your current macros.

To add a new macro click the ‘Add’ button below the list of tabs in the Configuration window. There will be two blank text boxes below the list of macros. On the left is the ‘Macro (Press Keys)’ field. Select this box with the mouse and simply press the key you wish to bind your macro to. For example number 5 on the keyboard.

Now select the ‘Action’ field. This is the command that will be sent to the game whenever you press the specified key. This command will be executed automatically, no additional text is required. So to complete our example enter ‘slice’ in the box and click apply.

Your new macro will now appear in the list above. Press the appropriate key to ensure it works.

5.1. Using the Command Line

It is also possible to save a macro using the [command line](#). The syntax for this is:

```
#macro {key to be bound}{action to be bound}
```

So if we wanted to bind “slice” to 5 on the number pad, you would enter

```
#macro {NumPad5}{slice}
```

Make sure you enter [#macro save](#) into your command line and hit return if you save your macros in this way. Genie will notify you if your macros were successfully saved. For more see the [#macro](#) section of this help file.

5.1. What Can I Bind?

Macros are a way of reducing the strain of repeated typing so the best commands to bind are ones that you will use frequently. Common examples include movement (many of these are already built into Genie, though you may wish to change the keys to suit your preferences) weapon attacks, spells and buffs.

Be aware of 'Num Lock' when you are binding Number Pad buttons. Most buttons on the Number pad have two states. One when Num Lock is on and another when it is off and each can be bound to a separate macro.

5.2. Intermediate Macro Functionality

5.2.1. Multiple Commands

Genie can assign multiple commands to a single macro. To do this you [add a macro](#) as normal, separating the commands with a semi-colon. For example you might want to bind a spell preparation and targeting to a single key:

```
prep bolt 15;target
```

The number of commands entered in this way is restricted by your type ahead lines. The amount you have depends on your account type. Standard users have a single type ahead line, while premium users have two. A third can be acquired through LTB points.

You can get around this restriction by using the [send command](#). This will place the action in a queue that Genie will attempt to resolve as soon as your character is unaffected by runtime. To use this functionality add a “-” sign before your command. So using the example above we would have:

```
-prep bolt 15;-target
```

Using the send command is useful for commands that you wish to execute while other commands are being entered into the game. So for example, if your combat script is inputting weapon attack commands but you wish to manually use Targeted Magic, using send in a macro would stop Runtime incurred by your script's commands from conflicting with your macro. You will still have to wait for the runtime to finish, but it eliminates the need to perfectly judge your key presses if you are in runtime.

Since “-” is a shortcut to the #send command, you may also specify a number value to wait. In the example below it would execute the first command, then wait 8 seconds before the second:

```
hunt;-8 hunt 1
```

5.2.2. Macros that do not auto trigger

If you wish to bind a macro that does not automatically input its command to the game you must include the @ symbol. The @ tells the cursor where to appear and Genie will display the text contained within the 'Action' field in your command line rather than inputting it directly into the game. For example:

```
say /quietly @
```

This will enter say /quietly in your command line with the cursor positioned after the word quietly. This functionality eliminates the need to repeatedly type out commands that are used in a variety of different situations. Another example would be

```
target @ with my crossbow
```

The cursor would appear between the words 'target' and 'with' allowing you to simply input your preferred target.

You might also want to use [Aliases](#) in situations like this.

5.3. Advanced Macro Functionality

5.3.1. If Functions in Macros

Genie is capable of using IF functions within its macros. This means that the macro can check if a statement is true or not when it is pressed. If the condition is true, it will send one command to the game. If the condition is false, it will send a different command.

To do this you have to use [global variables](#) within your macros. Any global variable can be called by a macro. To see how to create variables see the section on creating [global variables](#).

A basic example might use a macro to draw and sheath a weapon. It will utilize the \$righthand variable which describes the item in your character's right hand.

The syntax for an IF function in a macro is:

```
#if {condition}{outcome 1}{outcome 2}
```

See the [#if command](#) section of this help file for more information.

So in our example we want to draw our sword but only when we do not already have it in our hand. Using the \$righthand variable to check this statement our macro would look like this:

```
#if {"$righthand" = "bastard sword"}{#put sheathe sword}{#put draw sword}
```

It is a good idea to surround any string in "quotation marks" to ensure it is accurately identified by genie.

It is possible to assign multiple commands to each section of an IF function macro. Using the above example we can add in our shield:

```
#if {"$righthand" = "bastard sword"}{#put sheathe sword;#put wear shield}{#put draw sword;#put remove shield}
```

5.3.2. Setting Variables in Macros

It is possible to set a [global variable](#) using a macro. This can be combined with actions so that your macro captures information that you can use later on in [scripts](#). An example would be to set a variable called "\$stance" that records which combat stance you are in. The syntax for this is

```
#var <variable name> <variable result>
```

So in our example we would bind the following

```
stance shield;#var stance Shield
stance dodge;#var stance Dodge
stance parry;#var stance Parry
```

For more on how to use [variables](#) once they have been captured see the appropriate section of this help file.

Also, remember that any variable can be displayed on the [statusbar](#).

5.3.3. Nested If Functions

Genie supports nested IF functions. This means that you can have more than two outcomes for your macro. This example will get your hand mallet if you do not have it. If you have it in your right hand it will input the command “throw right”. If you have it in your left hand it will input the command “throw left”.

```
#if {"$lefthand"="hand mallet"}{#put throw left}{#if {"$righthand"=
"hand mallet"} {#put throw} {#put get hand mallet}}
```

There is no limit to the number of nested IF functions you can have in a single macro, but the longer a macro becomes the harder it is to read and to edit.

5.3.4. AND / OR Functionality in an IF Macro

Genie can use the OR function to compare variables within a macro. For example:

```
#if {"$righthandnoun" = "blade" OR "$righthandnoun" = "blades"}{#put
throw}{#put get blade}
```

Because the stackable throwing blades change name when they are split they can be either a “throwing blade” or “throwing blades”. This macro will input the throw command if your \$righthandnoun variable is either “blade” OR “blades”. If it is neither of these options the macro will input the command “get blades”.

In a similar way Genie can use the AND function to match multiple variables to a condition. For example, assume I have a [global variable](#) called “\$chamber” that tells me if my Repeating Crossbow has a bolt in the chamber or not. This variable can either be “Full” or “Empty”:

```
#if {"$righthand" = "repeating crossbow" AND "$chamber" = "Empty"}{#put
push my crossbow}{#put fire}
```

This macro will push the repeating crossbow to fill the chamber only if you have the repeating crossbow in your hand and the “\$chamber” variable indicates that the chamber is empty. Otherwise it will input the “fire” command.

Don't forget that you can use [triggers](#) to set [global variables](#) for use in your macros.

AND and OR functionality can be combined with [nested IF functions](#) to create extremely advanced macros. Bear in mind that the longer a macro becomes the harder it is to edit.

6. Ignores

6.1. What is an ignore?

Ignores or Gags are a way of preventing specific text from appearing in the game window. This is useful for text that occurs repeatedly but adds little to your game experience except for scroll. For example many people choose to gag juggling messaging or the text that appears when you collect your thrown weapon or ammunition in combat.

6.2. How do I add / edit / delete an Ignore

The Ignore entry window can be found by going to the [Edit](#) menu and selecting [Configuration](#). In the Configuration window choose the 'Ignores' tab, this will display a list of all your current gags.

To add a new gag click the 'Add' button below the list of tabs in the Configuration window. There will be a blank text boxes below the list of gags labelled Gag (Regular Expression). Type (or even better, copy and paste) the text you wish to ignore into the field and click apply. Your new gag will now appear in the list above.

To Edit or Remove an existing gag, simply access the list of gags by going to the Edit menu and selecting [Configuration](#). In the Configuration window choose the 'Ignores' tab and select the gag you wish to edit or delete.

To edit the gag, simply change the text that appears in the Gag (Regular Expression) field, then click Apply.

To delete the gag, simply click the 'Remove' button below the list of tabs in the Configuration window.

You can also add an ignore using the [#ignore](#) command

6.2.1. Ignore Case

If you check this box Genie will ignore the case of your gag. So for example, if you choose to gag the word Arrow with Ignore case selected Genie would gag both Arrow and arrow. This feature prevents you from having to add separate gags for the same term or sentence if it occurs with multiple case options.

6.3. Gagging with Regular Expressions

Genie uses [Regular Expressions](#) within its gags. This means that you can account for variation in text when trying to gag a phrase that recurs frequently. For example:

```
^The shadowling mews piteously and covers its face with its paws\.$
```

However, you also need to be aware of the special characters used by Regular Expressions when setting up gags. See the [Regular Expression](#) section for more on these characters, but it is worth remembering that the `\` character designates a character as simply a character, rather than a regular expression. IE/ you will need to use `\.` to gag a period, not simply `.` which is a Regular Expression.

6.4. Things to be Wary Of

You should take care when adding gags to Genie as the program cannot discriminate based on context. In other words, if the text you have gagged comes up in a different situation you will not see it. Be particularly wary of gagging single words or short phrases that could occur in a multitude of situations.

Gagging error messages can seem like a good idea, but if you are trying to figure out why an action is not working and the error message has been gagged it can be frustrating.

It is best to be as specific as possible with a gag. Copy and paste the whole line across to avoid conflicts and typos and try to limit your gags to text that impacts your game experience.

You can always enter `#gag clear` into the command line to remove all gags. This is useful if you suspect a gag is causing desired text to be suppressed.

6.5. Parse Order

Also take note of the order Genie parse text:

1. Highlight "Begins with"
2. Highlight "Line contains"
3. Triggers
4. Gags/Ignores
5. Substitutes
6. Highlight "RegEx"
7. Highlight "String"
8. Names

It's important to remember this order since for example a RegEx highlight will trigger off your substituted text.

7. Aliases

7.1. What is an Alias?

An Alias is a way of binding a certain word or phrase to an action. When the word is entered into your command line and sent to the game Genie will instead enter the Action you had assigned to that Alias.

For example if your alias was the word 'hello' and the action assigned to it was 'say /quietly Hello there!', simply entering the word hello would trigger your speech action. Hello has become an alias for 'say /quietly Hello there!'

An alias is activated by entering the text used to trigger it and pressing return. It will not trigger when you enter the text, so you are free to bind commonly used words as Aliases without fear that it will cause problems when you are entering speech.

This is useful in a variety of situations. Firstly it allows you to condense long strings of text that you use repeatedly but do not wish to bind to a macro, allowing you to avoid the strain of constant retyping. Secondly, more advanced Aliases make use of [variables](#) to adapt the Action to a variety of situations. This allows for great flexibility in the level of control you have over commands sent to the game while still making these commands easy and quick to enter.

7.2. How do I add / edit / delete an Alias

The Alias entry window can be found by going to the [Edit](#) menu and selecting [Configuration](#). In the Configuration window choose the 'Alias' tab, this will display a list of all your current aliases.

To add a new Alias click the 'Add' button below the list of tabs in the Configuration window. There will be two blank text boxes below the list of Aliases labelled Alias and Action. 'Alias' is the text that you will enter to trigger the alias. Select the Alias field and enter the appropriate text. The Action field is the action that genie will take when you enter the alias. Enter the action you wish to use into the field and click apply. Your new alias will now appear in the list above.

To Edit or Remove an existing alias, simply access the list of aliases by going to the [Edit](#) menu and selecting [Configuration](#). In the Configuration window choose the 'Aliases' tab and select the alias you wish to edit or delete.

To edit the alias, simply change the text that appears in the Alias and Action fields, then click Apply.

To delete the alias, simply click the 'Remove' button below the list of tabs in the Configuration window.

You can also add an Alias by using the [#alias](#) command.

7.3. Using Variables in an Alias

Aliases become more useful and more powerful when you use them in conjunction with [local variables](#). These variables are entered after the text used to trigger the alias. The syntax for the variable is \$1 and this must be entered in the action field when setting up your alias.

For example let us consider an alias that shoots your crossbow at a target for PvP. Your target will change constantly as you never know who your opponent may be. In this case we would set up the following alias:

Alias

```
fire
```

Action

```
fire my crossbow at $1
```

This alias would be triggered by entering fire in the command line followed by the \$1 variable, in this case the name of your target. So if you enter 'fire Bob' into your command line, Genie will input

```
fire my crossbow at Bob
```

You can make this even more flexible by using multiple variables within a single Alias. The syntax is the same, each variable is simply assigned a different number. The first variable is \$1, the second \$2, the third \$3 etc.. This functionality can be combined with the ; command to have a single alias send multiple commands and can also be combined with the [#send command](#) in case your character is currently in roundtime. So for example, we might expand upon our previous example to create an Aiming and Firing Alias.

Alias

```
fire
```

Action

```
#send aim $0;#send fire $1
```

This alias is triggered by entering fire in the command line followed by the variables \$1 and \$2. So in this example 'fire Gandalf head" will cause Genie enter the following

```
aim Gandalf  
fire head
```

Finally it is possible to use the \$0 variable within an Alias. This works in a similar way to \$1 but it allows for the inclusion of multiple argument words within the same variable entry. For example let us consider the following Alias

Alias

```
sn
```

Action

```
snipe $0
```

In this case Genie will pick up any commands you enter after the word 'snipe' and use them all as variables. So for example, if you enter “snipe Bob head”, Genie will input

```
snipe Bob head
```

This is useful for commands that may use multiple words, or not, depending on the circumstance.

In this way Aliases can be set up to simplify actions which, although repeated frequently, are used in such varying contexts that binding them to a macro becomes problematic.

8. Substitutes

8.1. What is a Substitute?

A substitute allows you to replace text output by the game with text of your own choosing. This can range from a simplification to reduce the amount of text on screen to the addition of useful information after an item. For example, you could replace the log on and log off text that accompanies a players arrival or departure with simply their name and a + sign if they are logging on or a – sign if they are logging off. Alternatively you could replace the word 'yelith root' with the phrase 'yelith root (limbs internal)' to remind yourself what body part this particular herb cures.

8.2. How do I Add / Edit / Delete a Substitute?

The Substitute entry window can be found by going to the Edit menu and selecting [Configuration](#). In the Configuration window choose the 'Substitutes' tab, this will display a list of all your current substitutes.

To add a new Substitute click the 'Add' button below the list of tabs in the Configuration window. There will be two blank text boxes below the list of Substitutes labelled Substitute (Regular Expression) and With. 'Substitute' is the text that you wish to replace. Select the Substitute field and enter the appropriate text. The With field is the text that Genie will display in the place of your substituted text. Enter the replacement text you wish to see into this field and click apply. Your new substitute will now appear in the list above.

For example if you enter 'War Stomper' into the Substitute field (a Barbarian title) and 'Barbarian' into the 'With' field, Genie will replace the War Stomper title with the word Barbarian.

To Edit or Remove an existing substitute, simply access the list of substitutes by going to the Edit menu and selecting [Configuration](#). In the Configuration window choose the 'Substitutes' tab and select the substitute you wish to edit or delete.

To edit the substitute, simply change the text that appears in the Substitute and With fields, then click Apply.

To delete the substitute, simply click the 'Remove' button below the list of tabs in the Configuration window.

You can also add a substitute by using the `#sub/#substitute` command.

8.3. Using Variables / RegExp in a Substitute

Substitutes use Regular Expressions to replace text and are at their most useful when combining

both [Variables](#) and [Regular Expressions](#) within a single substitute. For example, you might wish to simplify the text that precedes a message over Gweth, changing 'Your mind hears Bob thinking, Greetings Theren!' to 'Bob: Greetings Theren!' To do this you

Substitute

```
^Your mind hears (\S+) thinking, (.*)$
```

With

```
$1: $2
```

Or you might wish to reduce juggling text by substituting as follows:

Substitute

```
^(\\S+) begins to juggle (.+),
```

With

```
$1 juggles $2.
```

However, because they always use Regular Expression, you need to be aware of the special characters used in a RegExp when setting up substitutes. For full details and a list of all of these characters, with examples, see the [Regular Expression](#) section of this help file. However, remember that in a Regular Expression the `\` character causes any following character to be regarded as a normal character, and not a regular expression character. So for example if you wanted to use a period in your substitute you would need to use `\.` and not simply the period.

9. Triggers

9.1. What is a Trigger?

A trigger is a way of performing a certain action every time the game outputs a specific piece of text. For example, a trigger could automatically listen when someone tries to teach you a lesson or set a variable that captures how many TDP's you have whenever the game displays the information. Triggers are especially useful for setting [variables](#) and [statusbar](#) information, and work best with [Regular Expressions](#).

9.2. How do I Add / Edit / Delete a Trigger?

The Trigger entry window can be found by going to the Edit menu and selecting [Configuration](#). In the Configuration window choose the 'Triggers' tab, this will display a list of all your current triggers.

To add a new Trigger click the 'Add' button below the list of tabs in the Configuration window. There will be two blank text boxes below the list of Triggers labelled Trigger (Regular Expression) and Action. 'Trigger' is the text that you will be using to activate your trigger. Select the Trigger field and enter the appropriate text. The Action field is the action that Genie will perform once the specified text appears. Enter the appropriate action into this field and click apply. Your new trigger will now appear in the list above.

For example if you enter

```
^To learn from (him|her), you must LISTEN TO (\w+)
```

into the Trigger field and

```
#send listen to $2
```

into the 'Action' field, Genie will automatically listen when someone tries to teach you. Obviously you may need to be careful with certain classes if you are a Paladin (stealing) or a Barbarian (magic).

To Edit or Remove an existing trigger, simply access the list of triggers by going to the Edit menu and selecting [Configuration](#). In the Configuration window choose the 'Triggers' tab and select the trigger you wish to edit or delete.

To edit the trigger, simply change the text that appears in the Trigger and Action fields, then click Apply.

To delete the trigger, simply click the 'Remove' button below the list of tabs in the Configuration

window.

You can also add a trigger using the `#trigger` command.

9.3. Setting a variable with a Trigger

Triggers are a useful way to set variables. For example, you may wish to record your available tdp's either for use in a `script` or to display on your `statusbar`. This type of trigger uses the `var command` as the action that is triggered. In this case you would set up your trigger as follows:

Trigger

```
^You have (\d+) TDPs.
```

Action

```
#var tdp $1
```

9.4. Eval Triggers

Eval triggers allow you to trigger an action when a specified variable changes. This is particularly useful for the statusbar, where you can set the trigger to update your statusbar if the variable in question changes.

To designate a trigger as an EVAL trigger, simply check the Eval box to the right of the Trigger field.

In the Trigger field enter the variable you are monitoring. This has to be a global variable, so for example `$monstercount`. Enter the action as usual. For example:

Trigger

```
$monstercount
```

Action

```
#statusbar Monstercount: $monstercount
```

Or a more complex example, using an Eval Expression:

Trigger

```
$monstercount > 2
```

Action

```
#echo Watch out! $monstercount monsters in the room!
```

9.5. Advanced Triggers

Some examples of advanced trigger actions include:

```
#if {contains("$roomplayers", "Gandalf")} > 0} {#put say Ahoy hoy
```

```
Gandalf!}  
#if {$hidden = 1} {#put snipe $1} {#put fire $1}
```

10. Highlights

10.1. What Is A Highlight?

A Highlight is a way of coloring specific text to make it easier to see. A very basic example that most people highlight is the word 'you'. This makes it easier to pick up on lines that relate to your character, especially in high scroll rooms.

10.2. Types of Highlights

There are three sub tabs within the Highlights tab. Strings, Names and Presets. Presets are the default highlights that must be set by Genie, and though you can change the color you cannot delete the entries themselves. These include room name, creatures, thoughts and whispers.

The Names tab is used for highlighting names. Friends, enemies, GMPCs or noteworthy people of any kind can be added here. Name highlights require an exact match and also require that the word not be within another word. For example if Kai was highlighted Genie would not highlight kai or any part of Kaiser. In addition with a name highlight you can right click on a window and ignore anything that does not match a name in your name list.

Finally strings includes everything not covered by the two other categories, and includes the majority of your highlight entries.

10.3. How do I Add / Edit / Delete a Highlight?

The Highlight entry window can be found by going to the Edit menu and selecting [Configuration](#). In the Configuration window choose the 'Highlights' tab, this will display a list of all your current highlights.

To add a new Highlight click the 'Add' button below the list of tabs in the Configuration window. There will be two blank text boxes below the list of Highlights labelled Highlight and Color. 'Highlight' is the text that you will be coloring. Select the Highlight field and enter the appropriate text. The Color field is the color that Genie will use to highlight your specified text. For full details of how to alter the color see the [color](#) section. Enter the appropriate color into this field and click apply. Your new highlight will now appear in the list above.

To Edit or Remove an existing highlight, simply access the list of highlights by going to the Edit menu and selecting [Configuration](#). In the Configuration window choose the 'Highlights' tab and select the highlight you wish to edit or delete.

To edit the highlight, simply change the text that appears in the Highlight and Color fields, then click Apply.

To delete the highlight, simply click the 'Remove' button below the list of tabs in the Configuration window.

You can also add a highlight using the `#highlight` command.

10.3.1. Highlight Types

Genie has several options for highlighting. These can be found below the Highlight field as a row of four radio buttons:

10.3.1.1. String

This is the most basic example of a highlight. A string highlight will highlight every occurrence of the specified word even if it occurs within or as part of another word. For example, you might choose to highlight 'back'. If you use a string highlight you will highlight 'back' but also the first four letters of backpack and the last four letters of throwback.

Strings should be used for words that you wish to highlight in all circumstances and the more specific you are the less chance you have of highlighting text in error. For example, you should highlight 'at melee range' rather than just 'melee'.

10.3.1.2. Line

This will highlight the entire line when it matches the word or phrase you entered. So for example you might highlight 'closes to melee range on you'. Whenever this text appears Genie will highlight the whole line red. Line highlights are useful for drawing attention to lines rather than individual words. Another example would be to highlight:

```
you. You dodge.
```

10.3.1.3. Begins With

The Begins With highlight will only highlight if the specified text is placed at the beginning of a line. For example highlighting '[You're' will highlight the balance line in combat but would not highlight the following text

```
Bob says, [You're nuts Bill!
```

10.3.1.4. RegExp

RegExp highlights allow you to use regular expressions within a highlight. This allows you to

account for variation where you might not know exactly what will accompany the text you are looking to highlight. An example of RegExp highlighting is included within Genie's default highlights:

```
^(\\w+) begins to lecture you on the proper usage of the (.+) skill
```

This highlights the line if you begin to listen to a lesson.

In addition RegExp highlights allow you to use brackets to precisely control what text is highlighted. If you use parenthesis in a RegExp highlight, only text matched inside the parenthesis will be highlighted.

So in our previous example, even though the % sign was part of the highlight, it is not highlighted because it was not within parenthesis.

Regular Expression highlights are very resource intensive, so it is preferable to use anchors that designate the beginning or end of a line.

For more examples of [regular expressions](#), see the appropriate section of this help file.

10.3.2. Case Sensitive

If this box is checked the highlight will take the case of letters into account when matching. For example, having 'you' highlighted with Case Sensitive checked would highlight you but not You.

10.3.3. Colors

Genie provides a set of default colors for use in highlights. These can be accessed by clicking on the paintbrush icon (text color) and the background icon (background color) within the highlights section of the configuration window. The 48 default colors are listed under Basic colors; in the Color pop up window.

Genie also allows you to use hexadecimal numbering which allows you to precisely control the quantities of red, green and blue within any color. If you know the number for a color you can enter it directly in the Color field. Alternatively you can access the Color window by clicking on the paintbrush icon (text color) or the background icon (background color) within the highlight tab of the configuration window.

The color palette allows you to drag the cross-hairs until you select a color you are happy with. The slider on the right allows you to select the brightness of that color.

Any color you select can be added to a set of 16 custom colors by clicking on the Add to custom colors button. These are saved for your use by Genie and the feature is useful for colors that you reuse frequently.

11. Plug Ins

11.1. What is a Plugin?

A plugin uses a new window to display specific information that is not readily available in any of the default Genie windows and which, for various reasons, justifies its own separate window. Plugins have advantages over highlights, aliases, substitutes and other methods of displaying game text as they physically locate information in a new window which can be moved, resized and customized at will. Plugins can display a wide range of information from the status of the moons to a real-time experience window that updates as your experience ticks through to a list of all living enemies in your current room.

11.2. How Do I Install A Plugin?

You can find the various plugins available for Genie in the forums.

<http://www.genieclient.com/bulletin/index.php?showforum=30>

To install a plug in you first need to download the files to your computer. The majority of plug ins contain a single .dll file which is zipped for convenience. Extract the contents of the archive and place the files it contains in your Genie3/Plugins folder. When you next start Genie it will automatically load the new plug ins.

Make sure to keep track of updates to the game and to Genie itself, both of which can effect plug in performance. The latest versions of all plug ins will be available on the Genie forums.

11.3. Configuring Plug Ins

Plug ins can be configured directly from Genie's main menu. Select the Plugins option and Genie will display a list of active plugins. Clicking on any of these plugins will open the configuration options for that specific plugin. These vary in each case and full details of how to configure your specific plugins can be found in the forum thread you used to download the plugin.

11.4. Writing a Plug In

If you are interested in writing a plug in for Genie 3, the Plugin development forum is a good place to start. This can be found at

<http://www.genieclient.com/bulletin/index.php?showforum=32>.

11.5. Useful Plugins

Here are some of the most useful plugins, though many more are available in the forums.

Experience tracker and circle calculator

<http://www.genieclient.com/bulletin/index.php?showtopic=2263>

Room display

<http://www.genieclient.com/bulletin/index.php?showtopic=1746>

Empath Crutch

<http://www.genieclient.com/bulletin/index.php?showtopic=2155>

12. Status Bar

12.1. What is the Status Bar?

The statusbar is a customizable bar that has ten slots for displaying game information of your choosing. This can range from any variables to specific text of your choosing. The statusbar can be displayed at the top or bottom of your screen according to your preference or left off altogether. For details on how to do this see the [Bars and Panels](#) section.

12.2. How do I add something to the statusbar?

Two types of information can be output to your statusbar slots, variables and specific text. The syntax for outputting is #statusbar followed by the number of the slot you wish to use between 1 and 10 and finally, the information to be output.

For example if you wished to output the name of your current character to your statusbar you could go about this in two ways. Firstly you could enter the information yourself as follows:

```
#statusbar 2 Bob
```

The other method is to use a variable instead of typing out the name by hand, in this case the \$charactername variable. So if you entered the following:

```
#statusbar 2 $charactername
```

The output would be identical if you used the same character. However, by using the variable you do not have to change your entry as you change characters. In other words, the same command displays the name of any character you may play.

Obviously the statusbar would be of limited value if you had to remember and type out such long strings of text every time you wanted to display something on it. However, [Triggers](#) work perfectly with the statusbar and allow you to automatically send the statusbar command whenever a specific event occurs.

For example let us suppose you want a statusbar entry that displays the lesson you are currently being taught, If you set up the following as a trigger

Trigger

```
^You begin to listen to (\w+) teach the (.+) skill
```

Action

```
#statusbar 10 Listening to $1 teaching $2
```

Say a character called Gandalf teaches you Brawling, once you begin to listen your 10th statusbar

entry would read as follows.

Listening to Gandalf teaching Brawling

If the teacher, the lesson or both change your statusbar will automatically update itself.

You can use any variable, whether it is a global one like \$righthandnoun or a self-created one in a statusbar entry.

You can incorporate the #statusbar command into macros in a similar way, which is particularly useful for displaying stances in combat.

12.3. EVAL Triggers

Eval triggers give you a way of monitoring a specific variable and updating your statusbar when that variable changes. For more details on how to set up an eval trigger see the appropriate section of the triggers help file.

Lets suppose that you want a statusbar entry that displays the number of favors that your character has. This involves two triggers. Firstly the initial trigger that captures the number of favors and saves it as a variable

Trigger

```
^You have (\d+) TDPs
```

Action

```
#var tdp $1
```

Secondly the eval trigger. This will update the statusbar every time this variable changes.

Trigger

```
$favor
```

Action

```
#statusbar 1 Favors: $favor
```

Don't forget to designate the second trigger as an eval trigger by checking the Eval box.

12.4. Advanced Statusbars

More advanced use of the statusbar would enable you to utilize Genie functions like #math to set up counters. For example you might keep track of how many monsters you have killed in a specific hunting session. This involves two triggers. The first saves a variable that records a running count of the mobs you have searched:

Trigger

```
You search the
```

Action

```
#math crittercount add 1
```

The second is an [eval trigger](#) which outputs this variable to your statusbar every time it changes:

Trigger

```
$crittercount
```

Action

```
Enemies Killed: $crittercount
```

13. Automapper

13.1. What is the Automapper?

The Automapper is a window that displays a graphical representation of MUD rooms. This not only allows you to gain an overview of locations by seeing how rooms connect to one another but also allows you to navigate more easily. You can travel via the automapper by clicking on a room and Genie will automatically navigate your character to that specific room.

The automapper requires map files to work. Many of these have already been recorded by Genie users. You can find the most up to date pack of maps for DragonRealms on the forums here: <http://www.genieclient.com/bulletin/index.php?showtopic=1888>

To install these maps, download the archive and unzip it to your Genie3/Maps folder.

There is also a script in development that uses these maps to travel. You can find it here: <http://www.genieclient.com/bulletin/index.php?showtopic=2134>

Genie cannot use automapper functionality in areas which are not yet mapped. Any maps you record can be submitted in the map pack thread on the forums and will be incorporated into the next release, allowing all Genie users to take advantage.

13.2. How to use the Mapper

Genie will automatically detect if your character moves into a room that has been mapped and will load the corresponding map for you. If you have just logged on you will need to move one room so that Genie can identify which room it is currently in.

The automapper window will display itself on top of the Genie window. This is designed for second monitors, so you can have the automapper window open on a second screen with the game itself on the first.

To move to a room simply right click on the appropriate node on the map. Genie will then move your character to that room by the shortest route.

Nodes with Blue borders represent links to adjacent maps and clicking on one of these will load the specified map, which is usually labelled. It is not yet possible to travel across multiple maps using the automapper, but it will be in the future.

The Genie maps use the following key. Red for shops. Green for points of interest. Blue for water and Blue Edged for linking nodes. Your characters current position is indicated by the pink node.

13.2.1. Levels of a Map

Complex maps, especially those of cities and towns, are often divided into levels. The levels can be accessed by using the up and down arrows at the top right of the automapper window. The text to the left of the arrows indicates which level you are currently on. For example the map of the Crossing has three levels. Level 2 for residential areas. Level 1 for the interiors of shops and guilds and Level 0 for everything else. This keeps the map from becoming cluttered and allows you to make efficient use of the available space, preventing the map from becoming too large and difficult to read. Genie will automatically navigate between levels as you move between them.

13.2.2. Zoom

You can zoom in and out by using the magnifying glass icons to the far top right of the automapper window. This allows you to view complex areas in more detail or zoom out to take in the whole of an area if you wish.

13.2.3. #goto

Each room in a map is assigned a specific room id number. You can discover this number by hovering your mouse over the room on the map. The id number and name of the room will be displayed in the bottom left hand corner.

You can navigate directly to any room without using the automapper screen provided you know the id number for your destination room and the area you are in has been mapped. The syntax for this command is as follows:

13.2.4. #goto <room id number>

So for example if you were in the Crossing, entering `#goto 551` would take you to the bank teller. This can be incorporated into [scripts](#) and is very useful for travelling within an area.

13.2.5. How to Record a Map

To record a new map, click on the new map icon on the top left of the Automapper window title bar. Press the red record button in the middle of the title bar. Genie will now record your characters movements and assign each new room they enter a unique room id for that map.

To start the map move your character around the area, taking care to travel into and out of every room. Many rooms have different entry and exit commands and as such it is necessary to travel through them in both directions to capture this information.

Once you have entered and exited every room in the area you wish you map, you can save your map

by clicking on the save icon in the top left of the automapper window title bar.

It is worth noting that the Genie Dragonrealms maps follow Ranik's numbering system, so it is a good idea to use this. Ranik's maps can be found at Elanthipedia here:

http://www.elanthipedia.com/wiki/Ranik_Maps

and are a very useful reference tool when creating your own for Genie.

13.2.6. How to Edit a Map

Unless you are very lucky, your map will probably look a little strange. Genie allows you to edit your map by adjusting node placement, adding labels and colors and even adjusting the arcs (the paths between nodes) to suit your preferences.

13.2.6.1. Moving Nodes and Labels

To move a node on a map, simply click the Move Nodes/Labels Compass Icon on the Automapper window button bar. This is a toggle. When it is selected it is surrounded by a blue border and you can move nodes and labels by simply clicking and dragging them. You can select multiple nodes by clicking and selecting as many as you wish while holding down the left mouse button. In this way you can move a block of nodes to a new position without moving them automatically.

The Snap to Grid button makes moving nodes a much simpler process. It underlays your map with a grid, and each node automatically positions itself on the corner of a grid square. This is useful for quickly getting a uniform look to your map, particularly in cities and towns.

13.2.6.2. Labels

Labels are free floating text that adds information about specific areas to a map. They can be used to indicate the name of the next map, districts in a town or even monsters found in a hunting ground.

To add a label click on the edit button on the automapper window toolbar. The editing window will appear at the bottom of the automapper window. Click the New Map Label button and enter your text in the Text field.

You can specify co-ordinates for the label in the Position field by entering X Y and Z co-ordinates. Alternatively you can leave them as 0,0,0 and move the label as described in the [moving nodes and labels](#) section.

13.2.6.3. Deleting Nodes

You can delete a node from a map by selecting the node you wish to delete. Click on the edit button on the automapper window title bar and the details of your selecting node will appear. Clicking on

the Remove eraser icon will delete the selected node from your map.

13.2.6.4. The Edit Panel

In addition to adding labels and deleting nodes the editing window is a powerful tool for adding functionality to your maps. With a node selected it allows you to edit any of the information recorded for that node.

The Name and Description fields should be captured as you record a map. They correspond to the name of the room and the descriptive text output by the game when you enter it.

You can edit the node color by clicking on the painbrush icon in the bottom left hand corner. The Genie maps use the following key. Red for shops. Green for points of interest. Blue for water and Blue Edged for linking nodes.

The Key/Linked Map field allows you to either enter additional descriptive text to a node or to link another map to it. Examples of descriptive text would be adding the name of shops or guilds to be found in that node. For example 'Rangers Guild'.

To link a map to a node you are editing, simple enter the map's filename in the Key/Linked Map field. For example entering Map456_GMTeaParty.xml will change the specified node to a linking node, and link map 456 to your current map.

Finally you can edit the arcs that connect nodes on your map. For example you may want a specific arc to go West even though the exit it represents is an Easterly exit. To esit an arc click on the Edit Arcs button in the edit panel.

This will display a list of the arcs (exits) from the node you are editing. Select the arc you wish to edit by clicking on it. Below this list are several fields. To Node ID indicates the room number of the node that the specified arc connects your current node to. Name is the identification for this specific arc. The Direction drop down menu allows you to customize the direction of the arc on the map. Choose the direction you want to arc to appear in and select it, then click apply. You may need to do this for both the entry and exit arcs depending on the map in question.

14. Layout

14.1. What are layouts?

The Layout tab controls the appearance of the various windows used by Genie to display text. You can create and delete windows and alter the color, font and size of the text. For more information see the [#window](#) and [#layout](#) command sections of this help file.

14.2. How to Add a Window

To add a default window that is no longer visible simply click on the Windows option on Genie's main menu. Clicking on any of the Windows listed in the drop down menu will bring that window to the top of your screen.

To add a new window of your own creation, load the Configuration window by clicking Edit on the Genie Main Menu bar and selecting Configuration. Under the Layout Tab you will find the Windows and Settings tabs. In the Windows tab select 'Add' and enter a name for your window in the Title field. You can select the background and text colors by using the two icons to the right of the Color box.

Some [Plugins](#) will require you to create specific windows to display the new information they gather. More details can be found in the specific thread where the plug in is available for download.

14.3. How to Remove a Window

To remove a window right click it and select Close Window. The Window can still be displayed by selecting it from the Windows menu as described in the [How to Add a Window](#) section.

14.4. How to Reposition a window

Simply clicking and dragging the edges of a window will enable you to resize it. When it nears the edge of another Window, Genie will automatically align your windows for convenience.

14.5. Changing Font

To change the font displayed by a window, select the window in the Windows tab within the Layout tab of the Configuration menu. Click the Font button and select a font, style and size that suits you. You will be able to view a sample of your new font in the Sample window. Should you wish to use a non standard character set you can use the Script drop-down menu to select one providing it is

installed on your computer.

14.5.1. Fixed Width Fonts

A fixed width font uses the same amount of space for each character. Certain [Plugins](#) require fixed width fonts to display correctly and some users may prefer a fixed-width font for general use. Several of these fonts are available for download on the Genie site.

14.6. Time Stamp

When the Time Stamp Output box in the Windows tab is checked, Genie will add a time stamp to each separate line of text. This is particularly useful for speech based windows.

You can remove the time stamp quickly by right-clicking on a Window and selecting the Time-Stamped option.

14.7. Layouts

A Layout is a way of saving the arrangement of your windows and bars so that you can reload that particular arrangement. Genie has a default layout which is loaded when it starts. You can overwrite this layout by selecting Layout from genie's main menu and clicking 'Save Default Layout'. In a similar manner you can load the default layout by selecting 'Load Default Layout;' from the same menu.

14.7.1. Named Layouts

These are layouts that do not occupy the default slot. At any time you can save your current layout as a named layout by selecting Layout from Genie's main Menu and clicking 'Save Layout As.' These named layouts can be loaded by selecting 'Load Layout' from the same menu.

This allows you to have several different layouts for different characters, computers or situations, controlling exactly which windows are visible at specific times.

Note:

Be careful to save your changes after altering your windows or layout, either in the default position or a named save, as if you don't they will be lost when you exit Genie.

14.7.2. Bars and Panels

The appearance of Genie's bars and panels is also controlled from the Layout option on Genie's Main Menu and are saved as part of your default and named Layouts. The options are:

14.7.2.1. Icon Bar

This bar contains the Roundtime display, the exit compass and the contents of the Left and Right hand.

The docking options are 'Dock Top' and 'Dock Bottom.'

14.7.2.2. Script Bar

This bar displays all currently active scripts and allows you to pause, resume and cancel them.

The docking options are 'Dock Top' and 'Dock Bottom.'

14.7.2.3. Health Bar

This bar displays your Health, Concentration, Stamina and Spirit.

The docking options are 'Dock Top' and 'Dock Bottom.'

14.7.2.4. Magic Panels

This option adds a Mana bar to your Health Bar and your currently prepared spell to your Icon Bar.

The options are 'Display' or 'Do Not Display.'

14.7.2.5. Status Bar

This allows you to display text or variables of your choosing. For more information see the [status bar](#) section of this help file.

The options are 'Display' or 'Do Not Display.'

15. File Menu

This section of the help file describes the various options of the File Menu.

15.1. Connect

Clicking on 'Connect' brings up the Simutronics Game Connect window. This allows you to connect to the game.

Account Name is the name of the Account you wish to connect to

Password is the password assigned to this account

Character Name is the name of the Character you wish to play

Game is the Simutronics game that you are connecting to.

You can choose to save the password in your Character profile. See the [Profile](#) section for more.

15.2. Auto Log

While this is selected Genie will automatically log the contents of your Game window. These logs can be found in your Genie3/Logs directory. Unselecting this option will stop Genie logging.

15.3. Open Log In Editor

This will open the active log in your default text editor.

15.4. Auto Reconnect

With this option selected, Genie will automatically attempt to reconnect should the initial attempt fail.

15.5. Ignores/Gags Enabled

With this option selected, Ignores and Gags will be applied. The option means you can quickly turn them on and off should you wish. For more information on setting up [Ignores](#) and [Gags](#), see the respective sections of this help file.

15.6. Triggers Enabled

With this option selected, Triggers will be used. The option means you can quickly turn them on and off should you wish. For more information on setting up [Triggers](#), see the appropriate section of this help file.

15.7. Plugins Enabled

With this option selected, Plugins will be loaded. The option means you can quickly turn them on and off should you wish. For more information on [Plugins](#), see the appropriate section of this help file.

15.8. Automapper Enabled

With this option selected, the Automapper will be enabled. The option means you can quickly turn the Automapper on and off should you wish. For more information on the [Automapper](#) see the appropriate section of this help file.

Note: The state of the above options will be displayed in the Game window when you load Genie.

15.9. Mute Sounds

With this option selected, all sounds will be muted.

15.10. Show XML Raw Data

This shows the raw XML Data received from Simutronics, and can be useful for diagnosing bugs or errors with Genie.

16. Variables

16.1. What Is A Variable?

A variable is a piece of information stored by Genie for use in [triggers](#), [scripts](#), [macros](#) and other functions. There are two types of variable.

Global variables are stored globally. When you exit Genie they are saved and can be called on by [scripts](#), [triggers](#), [macros](#) etc... Several Global variables are built into Genie. There is a list of these in the [Global Variables](#) section. Examples of these include \$righthand which captures the name of the item in you character's right hand and \$roomdesc which store the description of the room you are currently in.

Local variables are used in [scripts](#). They are defined at the start of the script and are only available for use within that script. For more on local variables see the section on [Script Variables](#) within this help file.

16.2. How to Set / Edit a Variable

The syntax for setting a variable is

```
#var <variable name> <variable value>
```

Note: This section assumes you know how Regular Expressions can be used while capturing variables. For more on how Regular Expressions capture information and how you can call that information using \$1, \$2 etc... see the [Regular Expression](#) section of this help file.

There are several ways to define a variable. You can use a trigger. For example you might set a variable that describes who is teaching you.

Trigger

```
^You begin to listen to (\w+) teach the (.+) skill
```

Action

```
#var teacher $1
```

You might also use the same trigger to set two variables, the first describing who is teaching you, the second describing the lesson

Trigger

```
^You begin to listen to (\w+) teach the (.+) skill
```

Action

```
#var teacher $1;#var lesson $2
```

You can set a variable in a script. To do this simply use the command #var [variable name] [variable

value] in your script. So to set a variable called combat to 'Yes' when you enter combat you might enter the following into your script

```
put advance
#var combat yes
```

Note: For more detailed instructions on setting variables in scripts see the [scripting](#) section of this help file.

16.2.1. Macro

To set a variable via a macro, simply bind #var <variable name> <variable value> to your desired key. So for example if you wished to have a variable that describes your current stance after you shift into it you would bind the following

```
put stance shield;#var stance Shield
```

16.3. Built in Variables

Certain variables are rendered as either 1 or 0. If this is the case 1 = On and 0 = Off. So for example \$hidden. If it is 1, you are hidden, if it is 0 you are not.

\$bleeding	Whether you character is bleeding or not.
\$dead	Whether your character is alive or not.
\$diseased	Whether your character has a disease or not. (Currently not supported in Dragonrealms)
\$hidden	Whether your character is hidden or not.
\$invisible	Whether your character is invisible or not.
\$joined	Whether your character is in a group or not.
\$kneeling	Whether your character is kneeling or not.
\$poisoned	Whether your character is poisoned or not. (Currently not supported in Dragonrealms)
\$prone	Whether your character is prone or not.
\$sitting	Whether your character is sitting or not.
\$standing	Whether your character is standing or not.
\$stunned	Whether your character is stunned or not.
\$webbed	Whether your character is webbed or not.
\$charactername	The name of your character.

\$connected	Whether you are connected to the game or not.
\$gamename	The name of the game you are connected to.
\$concentration	Your concentration expressed as a percentage.
\$health	Your health expressed as a percentage.
\$mana	Your mana expressed as a percentage.
\$spirit	Your spirit expressed as a percentage.
\$stamina	Your stamina expressed as a percentage.
\$gametime	The time 'ticks' in the game you are playing.
\$date	The date.
\$datetime	The date and time.
\$time	The time.
\$roomdesc	The description of the room your character currently occupies.
\$roomexits	Exits from the current room.
\$roomname	The name of the room your character currently occupies.
\$roomobjs	Objects in the room your character currently occupies.
\$roomplayers	Other characters in the room your character currently occupies.
\$monstercount	The number of monsters in the room your character currently occupies. For more see #config monstercountignorelist .
\$monsterlist	A list of monsters in the room your character currently occupies.
\$roomid	Automapper room id.
\$zoneid	Automapper zone id.
\$zonename	Automapper zone name.
\$down	Whether there is a down exit from the current room or not.
\$east	Whether there is an east exit from the current room or not.
\$north	Whether there is a north exit from the current room or not.
\$northeast	Whether there is a northeast exit from the current room or not.
\$northwest	Whether there is a northwest exit from the current room or not.
\$out	Whether there is a out exit from the current room or not.
\$south	Whether there is a south exit from the current room or not.

\$southeast	Whether there is a southeast exit from the current room or not.
\$southwest	Whether there is a southwest exit from the current room or not.
\$up	Whether there is an up exit from the current room or not.
\$west	Whether there is a west exit from the current room or not.
\$lastcommand	Last command sent to the game
\$lefthand	The item in your character's left hand. "Empty" when empty.
\$lefthandnoun	The noun of the item in your character's left hand. "" when empty.
\$righthand	The item in your character's right hand. "Empty" when empty.
\$righthandnoun	The noun of the item in your character's right hand. "" when empty.
\$preparedspell	Your currently prepared spell. "None" when none prepared.
\$spelltime	The amount of time in seconds the spell has been prepared.
\$prompt	
\$roundtime	Current roundtime value.

17. Command Line

This section lists all the commands that are available within Genie and gives an example of the appropriate syntax. Command arguments listed in [] are optional.

17.1. Command Line Reference

17.1.1. #beep

About

This command sends a single beep to the windows sound system.

Syntax

```
#beep
```

Examples

This will create a trigger that beep when your character get stunned.

```
#trigger {eval $stunned = 1} {#beep}
```

Related commands

#flash

#play

17.1.2. #clear

About

This command clears the window text buffer of the named window.

Syntax

```
#clear <window>
```

window - The name of the window to clear.

Examples

This will clear the 'Game' window.

```
#clear Game
```

This will clear the 'Thoughts' window.

```
#clear Thoughts
```

Related commands

#window

17.1.3. #colors

About

This command provides you a list of available simple name colors.

Syntax

```
#colors
```

Related commands

#echo

#highlight

17.1.4. #comment

About

This command puts a comment in the named window title.

Syntax

```
#comment <window> <text>
```

window - The name of the target window.

text - The text to place in the window title.

Examples

This will put "Hello world!" in the 'Game' window title.

```
#comment Game Hello world!
```

This trigger will put the current room name in 'Game' window title.

```
#trigger {eval $roomname} {#comment Game $roomname}
```

Related commands

#window

17.1.5. #connect

About

This command connects you to the game.

Syntax

```
#connect
```

```
#connect <profile>
```

```
#connect <account> <password> <character> <instance>
```

profile - The profile name to connect to.

account - Account name.

password - Account password.

character - Character name.

instance - Game instance short name (DR, DRF, DRP, DRT)

Examples

This command will re-connect to the game using the last known login details.

```
#connect
```

This command will connect using the GandalfDR profile. You must create the profile in advance.

```
#connect GandalfDR
```

This command will connect to 'DR' instance with character 'Gandalf'.

```
#connect MYACCOUNT mypw Gandalf DR
```

Related commands

#profile

17.1.6. #echo

About

This command sends Text to the 'Game' window by default.

The command can also be used to send colors, using hex code, or the simple color name.

It can also be used to send text to other windows.

(Triggers will not react on this information.)

Syntax

```
#echo [>window] [color[,bgcolor]] <text>
```

text - The text you want to echo.

window (optional) - The window you want to echo to. Default is 'Game'.

color (optional) - Foreground color. Either use hex code or simple name. Use #colors for a list of the simple names.

bgcolor (optional) - Background color.

Examples

This command will echo "Hello cruel world!".

```
#echo Hello cruel world!
```

This command will echo "Hello cruel world!" in Yellow.

```
#echo Yellow Hello cruel world!
```

This command will echo "Hello cruel world!" in a pink shade.

```
#echo #ff50ff Hello cruel world!
```

This command will echo "Hello cruel world!" in a White text with Blue background to the 'Log' window.

```
#echo >Log White,Blue Hello cruel world!
```

This command will echo a random number between 1 and 10 in Green.

```
#echo Green {#random 1 10}
```

This command will echo "101" in Red.

```
#echo Red {#evalmath 100+1}
```

Related commands

#colors

17.1.7. #edit

About

This command is a shortcut to edit a script.

(You may change the text edit application using #config editor option.)

Syntax

```
#edit <script>
```

script - The file name of the script you want to edit.

Examples

This command will edit or create a script called myscript.cmd.

```
#edit myscript
```

Related commands

#config

17.1.8. #eval

About

This command evaluates an expression.

For more see the section on [Expressions](#) and their functions in this help file.

Syntax

```
#eval <expression>
```

expression - The expression you want to evaluate.

Examples

This command will echo "test".

```
#echo {#eval {tolower("TEST")}}
```

This command will echo "2".

```
#echo {#eval {count("Abba", "b")}}
```

This command will save "test" to the 'myvar' variable.

```
#var myvar {#eval {tolower("TEST")}}
```

Related commands

#evalmath

#if

17.1.9. **#evalmath**

About

This command evaluates a math expression.

For more see the section on [Math Expressions](#) in this help file.

Syntax

```
#evalmath <expression>
```

expression - The math expression you want to evaluate.

Examples

This command will echo "101".

```
#echo {#evalmath 100+1}
```

This command will echo "11".

```
#echo {#evalmath round(10.5)}
```

Related commands

#eval

#if

#var

17.1.10. **#event**

About

This command sends a delayed command.

(Using this command without arguments lists the event queue.)

Syntax

```
#event
```

```
#event <delay> <command>
```

delay - The delay defined in seconds.

command - The command to execute.

Examples

This command will echo "Hello world!" in 10 seconds.

```
#event 3 {#echo Hello world!}
```

This command will echo "PIZZA IS READY!" in 10 minutes.

```
#event 600 {#echo PIZZA IS READY!}
```

Related commands

#send

#queue

17.1.11. #flash

About

This command flashes the current genie application in the windows task bar.

Syntax

```
#flash
```

Examples

```
#trigger {eval $stunned = 1} {#flash}
```

Related commands

#beep

17.1.12. #help

About

This command displays text-documents from your Genie Help/ folder.

Syntax

```
#help  
#help <section>  
#help edit <section>
```

section - The section keyword you want to display.

Examples

This command will show the index, if it exists.

```
#help
```

This command will show the section "echo", if it exists.

```
#help echo
```

17.1.13. #if

About

This command evaluates an expression for a true or false result value.
For more see the section on [Expressions](#) in this help file.

Syntax

```
#if <expression> <>true command> [false command]
```

expression - The expression you want to evaluate.

true command - The command to execute if the expression evaluates to true.

false command (optional) - The command to execute if the expression evaluates to false.

Examples

This command will echo "true".

```
#if {1 = 1} {#echo true} {#echo false}
```

This command will echo "false"

```
#if {1 = 0} {#echo true} {#echo false}
```

This command will send "stow right" if your right hand is not empty.

```
#if {"$righthand" != "Empty"} {#send stow right}
```

Related commands

#eval

17.1.14. #keys**About**

This command lists the available key names used for macros.

Syntax

```
#keys
```

Related commands

#macro

17.1.15. #log**About****Syntax**

```
#log [>name] <text>
```

name (optional) - The log name to write to. Default is 'CharacternameInstance_Date.log'.

text - The text you want to write to the log file.

Examples

This command will save "Hello world!" to the current log file.

```
#log Hello world!
```

This command will save "Remember to remember." to the 'Notes_Date.log' log file.

```
#log >Notes Remember to remember.
```

Related commands

#echo

17.1.16. #math**About**

This command does simple math operations on a variable.

(For more complex math operations use `#evalmath` command.)

Syntax

```
#math <variable> add/subtract/set/multiply/divide/modulus <number>
```

variable - The variable name you want to use.

number - The decimal number to alter it with.

Examples

This command will add 1 to the variable 'myvar'.

```
#math myvar add 1
```

This command will add the value of 'myothervar' to 'myvar'.

```
#math myvar add $myothervar
```

Related commands

`#evalmath`

17.1.17. #parse

About

This command sends output data to the main windows as if it came from the GAME.

This is a good way to test triggers.

Syntax

```
#parse <text>
```

text - The text to simulate.

Examples

This command will parse "You've gained a new rank in first aid.".

```
#parse You've gained a new rank in first aid.
```

Related commands

`#trigger`

17.1.18. #play

About

This command plays a sound file with the .wav extension.

Syntax

```
#play stop
```

```
#play <filename>
```

filename - The filename you want to play.

Examples

This command will play the alert.wav sound file from the Genie Sounds/ folder, if it exists.

```
#play alert
```

This command will stop any sound currently playing.

```
#play stop
```

Related commands

#beep

#playsystem

17.1.19. #playsystem

About

This command plays any of the built in sounds in your Microsoft Windows Operating System. (Advanced users may look up available options in their Windows Registry.)

Syntax

```
#playsystem <alias>
```

alias - SystemAsterisk, SystemDefault, SystemExclamation, SystemExit, SystemHand, SystemQuestion, SystemStart, SystemWelcome, and more ...

Examples

This command will play the Microsoft Windows Exit sound.

```
#playsystem SystemExit
```

Related commands

#play

17.1.20. #put

About

This command sends commands directly to the game without waiting for Roundtime.

Syntax

```
#put <command>
```

Examples

This command will send 'look' to the game.

```
#put look
```

Related commands

#send

17.1.21. #queue

About

This command sends a delayed command.

(Same as #event command, but it will automatically wait for any runtime to end.)

Syntax

```
#queue clear
#queue <delay> <command>
```

Examples

This command will clear any commands currently in the queue.

```
#queue clear
```

This command will send unhide after 3 seconds (or 3 seconds plus the size of any runtime at that time)

```
#queue 3 unhide
```

Related commands

#event

#send

17.1.22. #script

About

This command allows you to directly control scripts.

This is also where you can set a debug level for troubleshooting your script.

Syntax

```
#script abort/pause/pauseorresume/resume/trace/vars <script>
#script abort/pause/pauseorresume/resume/trace/vars all
#script abort/pause/pauseorresume/resume all except <script>
#script debug <level> [script]
#script explorer
```

script - The name of the script you want to target.

level - Available debug levels:

0 = off

1 = goto, gosub, return, labels

2 = pause, wait, waitfor, waitformove

3 = if evaluations

4 = variables, math, evalmath, counter

5 = actions

10 = shows all script rows

Examples

This command will abort all scripts.

```
#script abort all
```

This command will pause all scripts except for 'myscript'.

```
#script pause all except myscript
```

This command will show a trace of script 'myscript'.

```
#script trace myscript
```

This command will set debug level 2 for script 'myscript'.

```
#script debug 2 myscript
```

This command will show all script variables for 'myscript'.

```
#script vars myscript
```

This command will create a macro that brings up the script explorer window when you press 'F5'.

```
#macro {F5} {#script explorer}
```

Related commands

#edit

17.1.23. #send

About

This command sends text/commands to the game, first putting the text/command in the queue. (You may also use the #send shortcut '-'.)

Syntax

```
#send [delay] <command>  
-[delay]<command>
```

Examples

This command will send 'hide' to the game.

```
#send hide
```

This command will send 'unhide' to the game.

```
-unhide
```

This command will send 'hide' to the game, then 'unhide' after 3 seconds (plus the size of any runtime left after the 3 seconds.)

```
#send hide;#send 3 unhide
```

Related commands

#queue

17.1.24. #statusbar

About

This command allows you to display text at the bottom of the Genie window in the statusbar.

Syntax

```
#statusbar <index> <text>
```

index - There are 10 available text slots.

text - The text you want to display.

Examples

This command will display "Hello world!" in the statusbar.

```
#statusbar Hello world!
```

This command will display "Hello world!" in the 5th slot of the statusbar.

```
#statusbar 5 Hello world!
```

This trigger will show current monster count in statusbar.

```
#trigger {eval $monstercount} {#statusbar Monster count: $monstercount}
```

Related commands

```
#comment
```

17.2. Configuration Commands**17.2.1. #alias****About**

This command is to create/update an 'ALIAS'. An alias is a shorthand for a set of commands. (Use \$0 \$1 \$2 ... for user arguments in the command.)

Syntax

```
#alias
#alias <alias> <command>
#alias clear/load/save/edit
```

Examples

This command will display currently loaded aliases.

```
#alias
```

When using this alias the command "sscim" would send "sheath scimitar in harness" to the game.

```
#alias {sscim} {sheath scimitar in harness}
```

With this alias the command "af Gandalf head" would send "aim Gandalf" and "fire Gandalf head" to the game.

```
#alias {af} {aim $1;fire $0}
```

17.2.2. #class**About**

Class determines if a setting is active or inactive. The class command is used to create/update and activate/inactivate classes.

Syntax

```
#class
#class <name> on/off
#class all on/off
#class +<name> -<name> ...
#class clear/load/save/edit
```

Examples

This command will list current classes and their status.

```
#class
```

This command activates the class "Swimming".

```
#class Swimming on
```

This command activates "Swimming", "Climbing" and inactivates "Hunting".

```
#class +Swimming +Climbing -Hunting
```

17.2.3. #config**About**

This command updates the configuration settings.

Warning: Do not alter these unless you know what you are doing.

Syntax

```
#config
#config <setting> <value>
#config load/save/edit
```

Examples

This command will list all settings and their current value.

```
#config
```

This command will mute all sounds.

```
#config {mute} {1}
```

This command will change the default editor to UltraEdit (if the given path is correct)

```
#config {editor} {C:\Program Files\IDM Computer
Solutions\UltraEdit\uedit32.exe}
```

17.2.4. #gag**About**

This command sets text In Game to be ignored thus not sent to your Game Screen.

Please note that the pattern is ALWAYS in RegEx format.

Warning: Using this feature in the wrong way may render lag and cause Genie to be unusable.

Syntax

```
#gag
#gag <pattern> [class]
```

pattern - The pattern to ignore.

Examples

This command will list all loaded gags.

```
#gag
```

This gag would hide all origami exhales.

```
#gag {^\w+ exhales into (his|her) .+, then with a tremendous "BANNG!"
```

```
smacks it flat between (his|her) hands before tossing away the .+
tattered remains\.$}
```

This gag would hide this scavenger troll message.

```
#gag {^The scavenger troll examines its equipment, gauging its value\.$}
```

17.2.5. #highlight

About

This command sets the text you want highlighted.

When using RegEx you may include () to only highlight what is inside.

Syntax

```
#highlight
#highlight line/string/beginswith/regex <color> <pattern> [class]
#highlight clear/load/save/edit
```

Examples

This command will list all loaded highlights.

```
#highlight
```

This highlight would make the text matched inside the () Purple.

```
#highlight {regex} {Purple} {^You reach out with your senses and see (.+)
available for (.+)\.$}
```

This is same as the example above except it belongs to the "Magic" class.

```
#highlight {regex} {Purple} {^You reach out with your senses and see (.+)
available for (.+)\.$} {Magic}
```

17.2.6. #layout

About

This command load and save window layout.

Syntax

```
#layout load/save <filename>
```

Examples

This command would save the current window layout as "Config/Layout/laptop.layout".

```
#layout save laptop
```

This command would load the file "Config/Layout/laptop.layout".

```
#layout load laptop
```

17.2.7. #load

About

This command is simply a short-cut to using each of configuration settings to load from disk.

Syntax

```
#load all
#load type type type ...
```

type - vars, aliases, classes, triggers, config, macros, subs, gags, highlights, names, presets, layout, profile

Examples

This command would load all settings.

```
#load all
```

This command would load variables, aliases, macros and presets

```
#load vars aliases macros presets
```

17.2.8. #macro**About**

This command allows you to set combination keys to send commands.

Syntax

```
#macro
#macro <keys> <command>
#macro clear/load/save/edit
```

keys - Type #keys to see a list of available key names. Combine them with the ',' character.

command - The command you want to execute when the keys are pressed.

Examples

This command will list all macros and their commands.

```
#macro
```

With this macro, pressing 'F4' key would send "sheath" to the game.

```
#macro {F4} {#send sheath}
```

With this macro, pressing 'F5' will bring up the script explorer window.

```
#macro {F5} {#script explorer}
```

With this macro, pressing 'Shift' + 'Escape' keys would pause all scripts.

```
#macro {Shift, Escape} {#script pause}
```

With this this macro, pressing '0' on the numpad would send "sneak down" if hidden, and "down" if not hidden.

```
#macro {NumPad0} {#if {$hidden = 1} {#send sneak down} {#send down}}
```

17.2.9. #name**About**

This command sets your list of names to be highlighted.

Syntax

```
#name
#name <color> <name>
```

```
#name clear/load/save/edit
```

Examples

This command will list all names and their colors.

```
#name
```

This would highlight "Gandalf" as Red whenever you see the name in the game.

```
#name {Red} {Gandalf}
```

17.2.10. #preset

About

This command configures the built in Colors.

Syntax

```
#preset  
#preset <setting> <color>  
#preset clear/load/save/edit
```

Examples

This command will list all available presets.

```
#preset
```

This command will make your health bar SpringGreen with Black background.

```
#preset {health} {SpringGreen,Black}
```

17.2.11. #profile

About

This command will load or save a profile. For more see the section on [Profiles](#) in this help file. You must be connected to the game to be able to save a new profile.

Syntax

```
#profile load/save
```

Examples

If you are connected to the game this will save your character unique profile.

```
#profile save
```

17.2.12. #save

About

This command is simply a short-cut to using each of configuration settings to save to disk.

Syntax

```
#save all  
#save type type type ...
```

type - vars, aliases, classes, triggers, config, macros, subs, gags, highlights, names, presets, layout, profile

Examples

This command would save all settings.

```
#save all
```

This command would save variables, aliases, macros and presets

```
#save vars aliases macros presets
```

17.2.13. #subs, #substitute

About

This command configures what you want substituted.

Please note that the pattern is ALWAYS in RegEx format.

Warning: Using this feature in the wrong way may render lag and cause Genie to be unusable.

(Use \$0 \$1 \$2 ... to capture match groups.)

Syntax

```
#subs <pattern> <replace with> [class]
```

pattern - The pattern to match.

replace with - What you want to replace it with.

Examples

This command will list the loaded substitutes.

```
#subs
```

This substitute would replace "Roundtime: 10 seconds." with "RT: 10s".

```
#subs {^Roundtime: (\d+) seconds\.$} {RT: $1s}
```

17.2.14. #trigger

About

This command will configure your triggers. A trigger is a command you want to perform when a RegEx pattern is matched.

Please note that the pattern is ALWAYS in RegEx format.

(You may also use 'Evaluate' patterns to trigger on variable changes.)

Syntax

```
#trigger
#trigger <pattern> <command> [class]
#trigger eval <evaluation> <command> [class]
#trigger clear/load/save/edit
```

Examples

This command will list the current triggers.

```
#trigger
This will echo "Gandalf attempted to steal from you." if you catch Gandalf stealing from you.
#trigger {^You catch (\w+) making a grab for your pockets\!$} {#echo >Log $1 attempted to steal from you.}
This will echo "Battled dance ended." when your Barbarian dance ends.
#trigger {^You feel your inner fire cool, as the adrenaline pumping effect of your battle dance ends} {#echo >Log $1 Battled dance ended.}
This will create an evaluate trigger that beep when your character get stunned.
#trigger {eval $stunned = 1} {#beep}
```

17.2.15. #un*

About

These commands are used to remove settings from memory.

Syntax

```
#unalias <name>
#unclass <name>
#ungag <pattern>
#unmacro <keys>
#unname <name>
#unsubs <pattern>
#untriggr <pattern>
#unvar <name>
```

Examples

This command would remove your "Gandalf" name highlight.

```
#unname Gandalf
```

17.2.16. #var, #var, #variable

About

This command creates/updates a 'VARIABLE'

Syntax

```
#var
#var <name> <value>
#var clear/load/save/edit
```

Examples

This command will list the current global variables.

```
#var
```

This will create a variable named "container" and place the value of "backpack".

```
#var container backpack
```

17.2.17. #window

About

This command is used to control the Genie windows.
Use #layout save command to commit changes to the disk.

Syntax

```
#window add/remove/show/hide <name>
```

name - The window name you want to target or create.

Examples

This will show the Thoughts window.

```
#window show Thoughts
```

17.3. Automapper Commands

17.3.1. #goto

About

This command will find the closest path to room id or label and launch the automapper.cmd script to travel to it. The current room must be known.

Syntax

```
#goto <label or room id>
```

Examples

```
#goto bank
```

Related commands

#mapper

17.3.2. #mapper

About

This command is used to control the Auto Mapper.

Syntax

```
#mapper  
#mapper delete [room id]  
#mapper find <room name[|room description]>  
#mapper label <text>  
#mapper load/save/clear/record/lock/snap/allowduplicates/show/hide  
#mapper save [filename]
```

```
#mapper roomid <id>
#mapper select <room name>
#mapper timeout <milliseconds>
#mapper zoneid [id]
#mapper zonename [name]
#config automapper on/off
```

Examples

This command will bring up the Auto Mapper Window.

```
#mapper
```

This command will set the current map zone id to "7b"

```
#mapper zoneid 7b
```

This command will set the current map zone name to "Arthe Dale"

```
#mapper zonename Arthe Dale
```

17.4. Evaluate

This is relevant for #if, #eval, and the script commands if and eval.

17.4.1. Keywords

```
and
or
not
true
false
```

```
<
```

Less than

```
>
```

More than

```
=
```

Equal

```
!=
```

Not equal to

```
<>
```

Not equal to

17.4.2. Functions

```
contains(source text, search text)
count(source text, search text)
element(source text, index)
endswith(match text)
```

```
indexof(source text, search text)
instr - see contains()
instring - see contains()
lastindexof(source text, search text)
len(source text)
length - see len()
match(source text, match text)
matchre(source text, pattern)
replace(source text, replace text)
replacere(source text, pattern, replacement text)
startswith(match text)
substr(source text, start index, end index)
substring - see substr()
tolower(source text)
toupper(source text)
trim(source text)
```

17.5. Evaluate Math

This is relevant for #evalmath and script evalmath command.

17.5.1. Keywords

```
e
pi
```

17.5.2. Functions

```
cos(number)
sin(number)
tan(number)
floor(number)
ceiling(number)
max(number1, number2)
min(number1, number2)
arcsin(number)
arccos(number)
arctan(number)
sqrt(number)
log(number)
log10(number)
abs(number)
round(number[, decimals])
ln(number)
neg(number)
pos(number)
```

18. RegExp

Reg Exp, short for Regular Expressions, are a pivotal part of Genie and have applications in many of the program's functions. Triggers, highlights, scripts, gags, substitutes and more can all be made considerably more powerful and useful when you begin to use Regular Expressions.

They can be confusing at first but the following examples should help you gain a basic understanding of how Regular Expressions work. Further examples can be found in the forums here:

<http://www.genieclient.com/bulletin/index.php?showforum=25>

18.1. What is a Regular Expression?

A Regular Expression is a pattern of characters that describe or represent text. These characters can be used to match certain ranges of text from a perfect character match, though a range of alpha or numeric characters. These Expressions can be used in Triggers, highlights, scripts and more to enable you to match the desired text more efficiently or to match multiple variations within a single Trigger or Highlight.

For example

.

This matches any single character. For example the regular expression `r.t` would match `rat`, `rut`, `rot`, but not `root` as there are two characters in the final word between the `r` and `t`. In a similar way the Regular Expression `r..t` would match `root` but not `rat`, `rut` or `rot`.

By default a Regular Expression match is case sensitive. For case-insensitive matching, wrap the expression with `"/"` and `"/i"`. For example:

`/you/i`

18.2. Beginners Guide to the Essential Regular Expressions

We have already looked at how the Regular Expression `.` works. There are several other basic Regular Expressions that are very important to learn. This section will detail them with examples.

^

This Regular Expression matches the beginning of a line. For example, the regular expression `^When in` would match the beginning of the string "When in the course of human events" but would not match "What and When in the".

If we were using a game-specific example, the Regular Expression `^You whisper to` would match your whispers, but not if someone said to you:

'I saw You whisper to that man! What did you say?'

\$

In a similar way to `^` matching the beginning of a line, `$` matches the end of one. For example, the regular expression `weasel$` would match the end of the string "He's a weasel" but not the string "They are a bunch of weasels" or "That's one fat weasel you have there Bill."

`\b`

Like the previous Regular Expression, `\b` matches the end of something, in this case a word rather than a line. `/b` matches a word boundary, that is, the position between a word and a space. For example `er\b` matches the "er" in "never" but not the "er" in "verb".

Understanding how to use these three Regular Expressions allows you to define the beginning and end of words and lines and therefore control exactly what is being matched. Let's have a look at how we might use these Expressions within the game.

18.3. How do I use a Regular Expression?

So, now you have a basic understanding of how Regular Expressions work, let's look at how to incorporate them.

Regular Expressions are meant to be combined. By adding together various combinations you can control exactly what you match. Let's look at an example.

We have already seen how the Regular Expression `.` matches any character. In addition the Regular Expression `*` matches zero or more occurrences of the character preceding it.

So if we put them together as a new Regular Expression `.*` then our new Regular Expression matches any number of any character. In other words it matches any number of characters.

We can continue to build on this example.

One of the most useful functions of Regular Expressions is the ability to store certain parts of the string and save them as variables for later use. Anything within parenthesis will be stored in nine temporary memory slots which can be recalled using `$1` to `$9`. Let's look at an example.

If we take our previous regular expression of `.*`, which matches any number of characters, and enclose it in parenthesis we would have `(.*)`. Not only would this match any character as before, but it would also store that information in the first temporary memory slot, `$1`. Let's see how we can use this in a [Trigger](#).

Say we set up the following as our Trigger

```
^You begin to advance on (.*).\.
```

This will match any string of text beginning with **You begin to advance on** followed by any amount of any character. Anything following the word on will be saved in the first memory slot, `$1`.

To save that information as a **variable**, we would use the following as the action for the trigger

```
#var target $1
```

This saves the information in memory slot 1 as the variable called target. So say you advanced on a Ship's rat. The game would displaying:

```
You begin to advance on a ship's rat.
```

Your Trigger would match this string and it would save '**a ship's rat**' in the first temporary memory slot, then save it as the variable target. If you subsequently advanced on a musk hog, the game would display the following:

```
You begin to advance on a musk hog
```

and overwrite your target variable with '**a musk hog**'.

In this way Regular Expressions can be used to capture text which, although it occurs in the same place in a sentence within the game, is constantly changing.

18.4. Other Regular Expressions

This section will provide a useful list of Regular Expressions with matching examples.

```
.
```

Matches any single character. For example the regular expression **r . t** would match the strings rat, rut, r t, but not root.

```
^
```

Matches the beginning of a line. For example, the regular expression **^When in** would match the beginning of the string "When in the course of human events" but would not match "What and When in the".

```
$
```

Matches the end of a line. For example, the regular expression **weasel\$** would match the end of the string "He's a weasel" but not the string "They are a bunch of weasels" or "That's one fat weasel you have there Bill."

```
*
```

Matches zero or more occurrences of the character immediately preceding. For example, the regular expression **. *** means match any number of any characters.

```
\
```

This is the quoting character, use it to treat the following character as an ordinary character. For example, **\\$** is used to match the dollar sign character (\$) rather than the end of a line. Similarly, the expression **\.** is used to match the period character rather than any single character.

[]

Matches any one of the characters between the brackets. For example, the regular expression **r[aou]t** matches rat, rot, and rut, but not ret.

Ranges of characters can be specified by using a hyphen. For example, the regular expression **[0-9]** means match any digit.

Multiple ranges can be specified as well. The regular expression **[A-Za-z]** means match any upper or lower case letter.

To match any character except those in the range, the complement range, use the caret as the first character after the opening bracket. For example, the expression **[^269A-Z]** will match any characters except 2, 6, 9, and upper case letters.

\< \>

Matches the beginning (\<) or end (\>) of a word. For example, **\<the** matches on "the" in the string "for the wise" but does not match "the" in "otherwise".

()

Treat the expression between (and) as a group. Also, saves the characters matched by the expression into temporary holding areas. Up to nine pattern matches can be saved in a single regular expression. They can be referenced as \$1 through \$9. For example, **^You begin to listen to (\w+) teach your (.+)** would save the name of your teacher to \$1 and the name of your lesson to \$2, though you would have to set these as variables if you wanted to capture the information permanently.

|

Or two conditions together. For example **(him|her)** matches the line "it belongs to him" and matches the line "it belongs to her" but does not match the line "it belongs to them."

+

Matches one or more occurrences of the character or regular expression immediately preceding. For example, the regular expression **9+** matches 9, 99, 999.

?

Matches 0 or 1 occurrence of the character or regular expression immediately preceding.

{i}

Match a specific number of instances or instances within a range of the preceding character. For example, the expression **A[0-9]{3}** will match "A" followed by exactly 3 digits. That is, it will match A123 but not A1234. The expression **[0-9]{4,6}** any sequence of 4, 5, or 6 digits.

\b

Matches a word boundary, that is, the position between a word and a space. For example **er\b** matches the "er" in "never" but not the "er" in "verb".

\B

Matches any character not at the beginning or end of a word. For example **ea*r\b** matches the

"ear" in "never early".

```
\d
```

Matches a digit character. Equivalent to [0-9].

```
\D
```

Matches a non-digit character. Equivalent to [^0-9].

```
\s
```

Matches any white space including space, tab, form-feed, etc. Equivalent to "[\f\n\r\t\v]".

```
\S
```

Matches any non white space character.

```
\w
```

Matches any word character including underscore.

```
\W
```

Matches any non-word character. Equivalent to "[^A-Za-z0-9_]".

18.5. Advanced Regular Expression Examples

Here are a few ways to incorporate Regular Expressions. Once you can understand why these work you will be well on your way to mastering probably the most important part of maximizing Genie's potential.

Gweth Substitute. This replaces the 'Your mind hears XXX thinking' text with simply XXX: What they say.

```
^Your mind hears (\S+) thinking, (.*)$
$1: $2
```

Whisper Trigger. This pair of triggers echoes your whispers to the log window.

```
^(whis|whisp|whisper)\s+((?!to \w+) (\w+))\s+(.*)$
#var Whisper.Text $4
```

```
^You whisper to (.*)\.$
#echo >Log #94DD75 You whisper to $1, "$Whisper.Text";#unvar
Whisper.Text
```

New Rank Trigger. This echoes New Rank: Whatever Skill to a window named Ranks whenever you gain a new rank in a skill.

```
^You've gained (?:a new|\d+) (?:rank|ranks) in\s?(?:your ability (?:to|
with)|using|maneuvering in|your)?\s?(.+)\s?(?:use|area)?\.$
#echo >Ranks Yellow New Rank: $1.
```

As you can see, Regular Expressions can appear complex and baffling, but once you master the principles and learn the uses of each character they unlock a much greater part of Genie's functionality and enable you to create far more useful Triggers, Substitutes, Scripts and more.

18.6. Regular Expression Language Elements

These links details some further information on the set of characters, operators, and constructs that you can use to define regular expressions:

[http://msdn.microsoft.com/en-us/library/az24scfc\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/az24scfc(VS.71).aspx)

<http://regexhero.net/reference/>

18.7. Testing your Regular Expression

To confirm that your RegEx patterns is working as intended it might be a good idea to test them. To do this you can use the [#parse](#) command from inside Genie.

Creating a small script might help you. For example:

```
action echo MATCH: $1 when ^([A-Z][a-z]+) just arrived\.$
pause 0.1
put #parse Somedude just arrived.
```

Note that because Genie is multi threaded we need the pause to give it time to add the action.

You may also use this page:

<http://regexhero.net>

I also recommend the RegxBuddy software:

<http://www.regexbuddy.com>

19. Scripts

19.1. What is a Script?

A script is a set of commands that executes actions within the game. Scripts are designed to eliminate the need for large amounts of repetitive typing when performing mundane tasks and they can range from extremely simple, perhaps opening your backpack, getting an item and then closing it again, through to very complex scripts that hunt for you once you have set them up correctly. The more advanced scripts resemble small programs in their complexity.

It is best to think of a script like a story. You enter in a series of commands and whenever you execute the script, Genie will perform them in sequence like it is reading through your script.

Scripts can be extremely intelligent, accounting for all sorts of conditions and variables, you just have to know the correct commands to use. This section of the help file will guide you through the basics and provide you with examples of all the scripting commands available within Genie.

Before writing any scripts be sure to familiarize yourself with Simutronics' policy regarding afk scripting.

19.2. How do I write a Script?

Scripting can be imposing for the novice. The best way to learn is by getting stuck in, starting simple and gradually making more and more complex scripts. It is a little like learning new phrases in another language. At first they seem strange but very soon they become familiar and you are using them to learn even more complex phrases.

If you have any questions or queries that are not covered by this help file, be sure to ask them on the forums here:

<http://www.genieclient.com/bulletin/index.php?showforum=24>.

If you have written or even used a script in another front end, Storm Front for example, the principle is very similar in Genie and learning to script in Genie will involve learning which commands are different and the range of extra options Genie's scripting option gives you over your old front end. You might wish to review the section on [Differences with the Genie Scripting Engine](#).

If you have never written a script before, start with the [Beginners Guide to Scripting](#) section of this help file, which will guide you through a couple of basic scripts to get you started.

19.3. Creating / Editing A Script

To create a new script, select Script from Genie's main menu. Select Script Explorer to bring up the

Scripts Window. This will list all your current scripts.

To create a new script, select 'New Script'. You will be able to enter the name for your script and once you have chosen one, you can enter your script in your default text editor.

To edit an existing script, select the script from the list in the Script Explorer Window. Once you have selected the script you wish to edit, click the Edit Script button at the top of the Script Explorer Window. The script will load in your default text editor.

To delete an existing script, select the script from the list in the Script Explorer Window. Once you have selected the script you wish to delete, click the Delete Script button at the top of the Script Explorer Window.

19.4. Differences with the Genie Scripting Engine

Genie's scripting engine has several functions not available in other front ends. If you are already an experienced scripter it is worth looking at [Actions](#), [GoSubs](#) and [Comparisons](#), none of which are available in other front ends.

In addition it is important to remember that labels and variables in Genie are case sensitive. Many Storm Front scripts will work in Genie, but if your script does not work it is often worth quickly checking the case.

19.5. Multiple Scripts?

Genie is different from other front ends because it can run scripts simultaneously. This allows for more powerful use of scripts and means that there is no need to create one 'do it all' script.

In addition to running multiple scripts, you can pause and resume any of your active scripts at any time. Simply click on the name of the script you wish to pause on your [Script Bar](#). You will notice the icon for that script changes from a Play triangle to a Pause icon. You can resume this script at any time by pressing the icon again.

You can cancel all active scripts by pressing the 'Escape' key at any time.

19.6. How to Download and Run Scripts

A great source of information is the Genie forums. You can find help on all aspects of scripting here:

<http://www.genieclient.com/bulletin/index.php?showforum=24>.

One of the best ways to learn how to script is to examine other people's scripts. You can find numerous scripts written for Genie here:

<http://www.genieclient.com/bulletin/index.php?showforum=26>.

To use a script from the forums, download it from the appropriate thread. It will normally come as a zip file. If you extract this to your Genie3/Scripts folder the script will be available for use within Genie. It is worth checking the top of the thread, as most scripts will be written for certain conditions, be they classes, items, locations etc. Some of the more advanced scripts will have variables that you need to set to customize the script to you. Details of these will be included in the thread or within the script itself. For more on this see the section on [Script Variables](#) in this help file.

To run a script within the game simply type the name of the script, preceded by a period. So to run a script called forage you would type `.forage` and press Return.

19.7. Script Variables

A Genie script can designate certain variables for use by the script. These are separate from the [Global Variables](#) that are saved in the Variables part of the [Configuration Window](#). They designate pieces of information that the script will use to carry out tasks, and usually customize a script in some way. For example, a skinning script might have the name of the pelt to be skinned as a variable, so you can use the same script, no matter what you are skinning.

There are two ways to set variables for a script. The first is to enter them after the name of the script when executing it. You can add up to 9 variables in this manner, which are designated %1-%9. For example, if you were running a script called skin and wanted to set the variable as goblin skin, you would type

```
.skin goblin skin
```

then press return. Goblin skin would then be usable within the script as the first variable by calling %1.

The second method of defining local variables for a script is at the head of the script itself. The syntax for this is `var`. So to set the same variable for our skinning script, we would have the line

```
var skin goblin skin
```

at the top of our script.

Each method is more or less useful, depending on the circumstances. For scripts where the variable is less likely to change, say for example if you are designating a container for storage, then the second method is preferable as you do not need to type it out constantly.

However, for variables that change frequently, like the name of the pelt in our last example, it is better to use the first method to save having to edit the script manually every time you wish to run it.

To call a local variable within a script you use the % character.

If the variable has been designated in the command line, you call it by using %1 through %9, depending on the number of the variable.

If the variable is designated by var at the top of the script, you call it by using % followed by the name of the variable. So in our previous examples you would call the variable from the first example by using %1 and the variable from the second example by using %skin.

If the line in question involved getting the pelt from your backpack, it would look like this in each case:

```
put get %1 from my backpack
```

and

```
put get %skin from my backpack
```

Local variables are not available for use outside of a script, but you can save them as global variables if you wish. To do this we would use the #var syntax followed by the name we wish to use for our global variable and then our local variable.

So for example to save out %skin variable globally we would add the following line

```
#var skin %skin
```

19.8. Global Variables

Global Variables can be utilized by any script. To call a Global Variable you use the \$ character followed by the name of a variable. So for example to stow whatever is in your right hand the script line would look like this

```
put stow $righthandnoun
```

In addition a script can save variables globally. The syntax for this is put #var followed by the name of the variable and then value. You can use this to create your own variables within scripts. So to continue our last example, we might save the name of the item in our right hand before we stow it. This allows us to carry out other actions that might use our characters hands, then we can retrieve the weapon afterwards. To do this our script would look like this:

```
put #var itemright $righthandnoun
put stow $righthandnoun

[other actions performed by the script]

put get $itemright
```

To delete a variable in a script we use the #unvar command in the same way. So to continue our example, we might want to delete the item variable after we had retrieved our item. When we add

this in our script would look like this:

```
put #var itemright $righthandnoun
put stow $righthandnoun

[other actions performed by the script]

put get $itemright
put #unvar itemright
```

For information on local variables and how they can combine with global variables, see the [Local Variables](#) section of this help file.

19.9. Essential Commands

The following commands are essential to even the most basic scripts.

```
put
```

Any command that you want sent to the game must be preceded by the 'put' command.

```
send
```

In a similar way to put, send will send a command to the game. However, it places this command in a queue. These actions will be resolved when runtime is finished and will be send in the order they were received, oldest first.

```
pause
```

This command pauses the script for the specified amount of seconds (eg pause 5) or for a single second if no value is specified. It then waits for runtime to continue.

19.10. Beginners Script

In this section we will construct a few simple scripts by using the commands from the [Essential Commands](#) section so that you can see how they work. Our first script will get your armor from your rucksack and wear it.

Let us say that your armor consists of a suit of Field Plate, a pair of Mail Gloves and a Full Plate Helm. The first thing your script needs to do is open your backpack. Don't forget the put command!

```
put open my backpack
```

Then you need to get your armor and wear it. These actions require separate lines. You can't just keep adding put commands as the game can only handle two type-ahead lines. In this case we can use pauses to break up our 'put' commands. Our script would now look something like this:

```
put open my backpack
pause
put get my Field Plate
put wear my Field Plate
pause
```

Now, if we add in the remaining items in a similar manner we end up with the following:

```
put open my backpack
pause
put get my Field Plate
put wear my Field Plate
pause
put get my Mail Gloves
put wear my Mail Gloves
pause
put get my Full Plate Helm
put wear my Full Plate Helm
pause
put close my backpack
```

This script will open your backpack, get your armor and wear it, piece by piece, then close your backpack. We can use variables to improve this script. Currently, you would have to change the script if you changed your armor, but if we use variables for the armor names this becomes much easier.

To set a local variable we use the var command at the head of the script. In this case, we will need one variable for each piece of armor.

```
var armor Field Plate
var gloves Mail Gloves
var head Full Plate Helm
```

Now instead of using the name of the item each time, we can simply use %armor or %head. SO our new script would look like this:

```
var armor Field Plate
var gloves Mail Gloves
var head Full Plate Helm

put open my backpack
pause
put get my %armor
put wear my %armor
pause
put get my %gloves
put wear my %gloves
pause
put get my %head
put wear my %head
pause
```

```
put close my backpack
```

19.11. Intermediate Scripting

This section will continue to build on the basic script from the Beginner's Script section. It will demonstrate how to use gosubs and how to save variables for later use.

A gsub is a small section of a script that is used for a repeated action. You can use the gsub every time you perform that action, and it prevents you having to type out the same command numerous times in your script. For example you might have a gsub that stows. This means that every time you want to stow an item, you don't have to include the 'put stow [item] command, as you can simply direct the script to the gsub instead. For more on gosubs and some examples see the [gsub](#) section of this help file.

Another useful function for intermediate scripting is matching. This involves the script waiting for a specific piece of text to be output by the game. To continue our previous example, we might match 'You put you' when your item is stowed. This ensures that the script does not continue until the action you wanted performed has been executed.

There are two types of match. Match, which waits for a match before continuing, and matchreturn or matchre, which is very useful in gosubs. When your script moves to a gsub section, it remembers its place, like a bookmark. If you use matchre, assuming the match is successful it will return to your place. It is easiest to see this in practice:

Our stow gsub will look like this:

```
stow:  
pause .2  
matchre RETURN You put your  
put stow %s  
matchwait
```

This will pause briefly and begin watching for the text 'You put you'. When it is output by the game, the script will return from the gsub to your previous place. The gsub will then attempt to stow your item.

You will notice I have used the variable %s. This is a special 'slot' available, where you can temporarily save a variable within a script. This is how we differentiate which specific item is to be stowed. We save the name of the item to be stowed in the %s slot, then send our script to the gsub section. This is a much more efficient way of handling actions, especially in larger scripts such as movement or stealing scripts.

Another feature that is useful is splitting your script up into sections. These are designated by a : after the name, and help you control the flow of your script. For example you might have a start: section then an end: section. It also means you can repeat section, to save duplicating text. The command to move to a new section is goto [name of section]

Remembering that the command for sending the script to a gosub section is gosub [name of gosub], our armor stowing script, incorporating all of the examples mentioned above, would look like this.

```
var armor field plate
var head cowl
var hands gloves
var container duffel bag

start:
put open my %container
pause
save %armor
gosub get
gosub wear
save %gloves
gosub get
gosub wear
save %head
gosub get
gosub wear
pause
put close my %container

get:
pause
matchre RETURN You get your
put get my %s
matchwait

wear:
pause
matchre RETURN You work your way into|You slip|You put
put wear my %s
matchwait
```

This saves little space in such a short and basic script. However, learning how to use these features will pay dividends as your scripting becomes more complex.

19.12. Script Command Reference

19.12.1. action

The action command creates script triggers.

Syntax

```
action <command> when <pattern>
action <command> when eval <evaluation>
action remove <pattern>
```

command – The script commands to perform. You may define several commands by using the “;” character.

pattern – The regex pattern to look for.
evaluation – The evaluation expression to trigger on.

Note: If you use GOTO command in an action, it will automatically clear the GOSUB return history.

Actions are script-local triggers. This command will perform an action when a regular expression or variable evaluation is matched.

Example:

```
action var myvar $1 when ^(\S+) gives you a gentle poke in the ribs  
action goto bleeding when eval $bleeding = 1
```

```
action echo HIDDEN = $hidden when eval $hidden
```

19.12.2. action classes

```
action (class) <command> when <pattern>  
action (class) on  
action (class) off
```

Use () to group actions into classes. Classes can be turned on and off using the above syntax.

19.12.3. counter

```
counter add/substract/set/multiply/divide/modulus <number>
```

Similarly to the math command, counter manipulates the default counter variable. The counter is referenced as the variable %c.

Example:

```
counter set 10  
counter add 1.5  
echo %c
```

19.12.4. debug

```
debug <level>
```

This command sets the script debug level. Debug is text output to show what a script does.

Available debug levels:

- 0 = off
- 1 = goto, gosub, return, labels
- 2 = pause, wait, waitfor, waitformove
- 3 = if evaluations
- 4 = variables, math, evalmath, counter
- 5 = actions
- 10 = shows all script rows

19.12.5. delay

```
delay <seconds>
```

Functions similarly to pause, but does not wait for runtime.

19.12.6. echo

```
echo <text>
```

Echo text to Game window output. Use #echo command if you want to color the output.

Example:

```
echo Hello World!
```

19.12.7. eval

```
eval <variable> <expression>
```

Evaluates an expression or function and assigns the return value to the specified variable.

Example:

```
eval myvar tolower("HELLO WORLD")
```

```
echo %myvar
```

19.12.8. evalmath

```
evalmath <variable> <expression>
```

Evaluates a math expression (including math functions) and assigns the return value to the specified variable.

Example:

```
evalmath myvar 10 * (200 + 1) / 10
```

```
echo %myvar
```

```
evalmath myvar floor(19.4) + floor(5.8)
```

```
echo %myvar
```

19.12.9. exit

```
exit
```

Ends the script immediately.

19.12.10. gosub

```
gosub <label> [argument argument argument ...]
```

```
gosub clear
```

Moves to a label. You may also supply arguments that will be placed in \$0 \$1 \$2 \$3 ... variables. The \$0 variable contains all the arguments.

Use the return statement to return to the line immediately following the gosub.

Example:

```
gosub mylabel hello world
```

19.12.11. goto

```
goto <label>
```

Moves to a label.

19.12.12. if

```
if <expression> then <command>
```

Evaluates an expression, executing the then statement or script block if true. The expression may include eval functions, similar to the when clause of the action statement.

Optionally followed by an else block which will be executed if the expression evaluates to false.

19.12.13. if_X

Checks that the script start arguments exist. Arguments are stored in %1, %2, ... %9, and are checked by if_1, if_2, ... if_9, respectively.

This test may be followed by an else block which will be executed if the argument does not exist.

19.12.14. else

```
else
```

19.12.15. include

```
include <file name>
```

19.12.16. match

```
match <label> <text>
```

```
match clear
```

Creates a match table for the matchwait command. In most cases, a match table will consist of several match statements that associate various labels with different possible messages that may occur in response to a given game command. When the matchwait command is issued, the script waits until it receives one of the strings or regular expressions in the match table, then will goto the associated label.

If parentheses are used in a matchre statement, the enclosed pattern is stored in the \$1, \$2, ... variables after the label.

Match tables can consist of match and matchre statements intermingled.

19.12.17. matchre

```
matchre <pattern>
```

See match.

19.12.18. matchwait

```
matchwait [timeout]
```

Pauses the script to wait for a match in the match table. When a match is found, moves to the label associated with the match string or expression. Waits for runtime to end.

Optionally, you may specify a timeout in seconds after which the script will stop waiting for a match and proceed to the next line.

19.12.19. math

```
math <variable> add/subtract/set/multiply/divide/modulus <number>
```

Performs basic math operations on the variable's value and saves the results back into the variable.

set overwrites the variable's current value with <number>

add adds <number> to the variable's current value

subtract subtracts <number> from the variable's current value

multiply multiplies the variable's value by <number>

divide divides the variable's value by <number>

modulus divides the variable's value by <number> and assigns the remainder to the variable

Example:

```
math myvar set 10
```

```
math myvar add 1.5
```

```
echo %myvar
```

19.12.20. move

```
move <command>
```

Sends a command to the game then waits for a room description.

Example:

```
move north
```

```
move go bank
```

19.12.21. nextroom

```
nextroom
```

Waits for the next room description.

Example:

```
put climb embankment
```

```
nextroom
```

19.12.22. pause

```
pause <seconds>
```

Pauses the script for the specified number of seconds, or for one second if none specified. Waits for runtime to end before continuing.

Example:

```
pause 2  
pause 0.5
```

19.12.23. put

```
put <text>
```

Sends text to the game.

You may also execute the built in game commands by starting with a # character or launch a script by starting with a . character.

Example:

```
put hide  
put #var myglobal 1
```

19.12.24. random

```
random <min> <max>
```

Generates a random number between the minvalue and maxvalue seed values. The random value is referenced by %r.

Example:

```
random 1 10  
echo %r
```

19.12.25. return

```
return
```

Returns to the point at which gosub was called.

19.12.26. save

```
save <text>
```

Stores a value in the default save variable, referenced by %s.

Example:

```
save WORLD  
echo HELLO %s!
```

19.12.27. send

```
send <command>
```

```
send [delay] <command>
```

This command is basically the same as the put command, but commands are entered into the queue. If no number is added for the number of seconds you want to wait the queue will wait for runtime to end before sending the command. See #send command sections for more information.

Example:

```
send hide
send 3 unhide
```

19.12.28. shift

```
shift
```

Shifts all script arguments to the left. The value of %2 becomes %1, %3 becomes %2, and so on.

19.12.29. timer

```
timer start/stop/setstart/clear
```

Manipulates the built-in timer. The timer value is referenced by %t.

clear clears the value of %t

start starts the timer

stop stops the timer

setstart sets the timer to start at the specified date and time. Acceptable formats for datetime

include:

```
09/15/08 18:00
```

```
09/15/08 18:00:00
```

```
09/15/2008 18:00
```

```
09/15/2008 18:00:00
```

```
09/15/08 6:00 PM
```

```
09/15/08 6:00:00 PM
```

```
09/15/2008 6:00 PM
```

```
09/15/2008 6:00:00 PM
```

Example:

```
timer start
```

```
pause 3
```

```
echo Timer is now %t
```

19.12.30. var

```
var <name> <value>
```

Initializes a script variable. Once set, script variables are referenced by %variablename

19.12.31. unvar

```
unvar <name>
```

Removes a variable from script memory.

19.12.32. wait

```
wait
```

Waits for a game prompt. It also waits for roundtime end.

19.12.33. waitfor

```
waitfor <text>
```

Waits for a specific text string from the game output.

19.12.34. waitforre

```
waitforre <pattern>
```

Functions similarly to waitfor, but instead waits for a regular expression match from the game output.

19.12.35. labels

```
label:  
scripterror:  
%lastlabel
```

Labels are used to mark section starts for goto, gosub and match tables.

19.12.36. script blocks

```
{  
}
```