Kallisti MUD
Mprog/Oprog Handbook, Version 1.0
September 2019

* * *

www.kallistimud.com

Contents

I OVERVIEW

MOBPROGS are an OLC (on-line-coding) feature that allows builders to create and assign small, or complex, programs to mobs, without needing to write actual code, or find an available and willing coder. Individual mobs can have several mobprogs, or none. (All single examples of mobs with a shared Vnum have the same set of mobprogs). (Not everything that a coder can write can be duplicated by mobprogs, but many common features can easily be created, and mobprogs can be quite complicated, and can also be much easier to understand, and troubleshoot, than pages of code.)

Mobprogs or mprogs allow a mob to react to events on the mud, such as being bribed, one or more players entering their room, spells being cast on them, being in a fight, someone saying the secret word, what time it is, being slapped or farted at or tickled, being low on hits, dying, and so forth. Similarly, object progs or oprogs allow objects to interact with players. The events that TRIGGER a mprog or oprog are versatile and can be used in clever ways once they are understood. The **HEADER** tells the mud what type of event to react to (for example: speech, act, or fight), what specific example of the type (in some cases, for example: an actprog that only responded to tickling, or a speechprog that reacted to 'heh'), what % of the time to react to the trigger, and other information specific to the type of trigger. Triggers are explored more fully in the section VI for Header Options and in help **MPTRIGGERS** and **OPTRIGGERS** in the mud.

Once a mprog is triggered, it executes an action or series of actions. Mobs can execute almost any action a player can, many godcommands, and other actions that a players cannot. These actions are explored more fully in **MPCOMMANDS**.

Maybe you don't want the mobprog to happen every time the trigger condition is met. Perhaps the event occurs too often, or you want the mob to react unpredictably, or rarely. Most mobprogs can be assigned a % chance of triggering, (part of the **HEADER**) and any mobprog can be assigned various limits (part of the **BODY**).

(The mobprog's **HEADER** can be edited.) The **BODY** (explored more fully in **MPCOMMANDS** and **MPIFCHECKS**) tells the mud what to do after the trigger condition is met, and what additional conditions to check for. (The mobprog's BODY can be edited.)

Maybe you don't want to react to everyone that enters your room, or fights you, or says the secret word, or bribes you, though. Maybe it only works for elves, or monks, or remorts, or females, or when dispel magic is cast? This is where **IFCHECKS** come into play. **IFCHECKS** allow the mobprog to test for certain conditions and react only if the conditions are met. (This includes a Random ifcheck that allows the prog to be limited to only some % of the time the mptrigger condition is met). One mobprog can contain more than one ifcheck, with different consequences, and ifchecks can also be combined using AND and/or OR, and can use ELSE, and BREAK, and can also be nested. (Logical thinking and good design become very important.)

In many cases, the mobprog can perform actions, but needs to know who the action is to be performed 'on.' Obvious choices include 'everyone in the room,' 'the person who triggered the mobprog,' 'the mob itself,' 'a nearby door,' 'a random player in the room,' 'some object in the mob's inventory,' or 'somebody else.' This is where **MPVARIABLES** come into play. (Syntax is very important). Variables are used to identify the target of an action and are also used within messages generated by the mobprog.

Both the HEADER and BODY of a mobprog can be edited. In addition, you can re-organize mobprogs, add or insert new ones, delete them, review them, and (this is important!) save them. This is explored more fully in **MPEDITING**.

It is worth noting and remembering that mobprogs and ifchecks use % differently. For example, the ifcheck 'if rand(40)' means 40% of the time, and the ifcheck 'if hitpercent(60)' means when current hits go below 60% of maxhits, BUT, some mobprogs accept a number from 1 to 100 that is not the 'actual percentage,' but instead generates a number in the 1 to 1000 range, for example, mobprog 'rand_prog 70' means 'checks every pulse against a 586/1000 chance.' And some mobprog types do not use a % at all (but can still be limited within the body of the prog by using an 'if rand(%)' check).

II TERMINOLOGY

Here are some common terms used in building and in this handbook:

Mob	Mobile. A monster. Can also refer to the mob file.
MobProgs	Mobile programs. Similar to Spec Procs but can be edited directly on the mob rather than compiled in the in mud's code itself. Also called MProgs.
NPC	Short for 'non-player character.' Used synonymously with 'mob.'
ObjProg or Oprog	Object programs. Similar to Spec Procs but can be edited directly on the object rather than in the mud's code itself. Also called OProg.
PC	Short for 'player character.'
Spec Proc	Short for 'special procedure' the special routines that the MUD should use with a mob, obj or room.

III OLC

Kallisti uses the Online Creator (OLC) for almost all building work. This is because OLC is "more visual" to some people; but more importantly, it offers immediate feedback. You can build your mob or object and create your mprog immediately. It also eliminates the need to remember the correct formatting.

The basic olc commands are:

MPEDIT	This command allows you to edit a specific mob's mprogs.
MPSTAT	This command shows you a mob's mprogs without you having to be in the editor.
MPTEST	The command tests something.
MPSAVE	This command saves all the mobprog changes.
MOBPROGPERCENT	This command gives you the actual percentage chance of an mprog firing depending on the number provided.
OPEDIT	This command allows you to edit a specific object's oprogs.
OPSTAT	This command shows you an object's oprogs without you having to be in the editor.
OPTRIGGER	This command triggers an oprog to run.

MPEDIT

The mpedit command allows you to edit a specific mob's mobprogs. You must be mobedit-ing the mob (Mobedit <Vnum>). You can Mpedit save OR Msave to save all changes and update the progs on your mobs.

The mobprog consists of the **HEADER**, which identifies the type of event that triggers the mobprog, and necessary parameters, and the **BODY**, which identifies what other conditions to check and what actions to perform.

HEADER

Usage: `mpedit <command> [parameters]`
Note: The parameters, and the number and order of them, vary according to the command.

`Usage: mpedit <list | add | delete | insert | edit | save | copy | header | done>`

To create or edit an mprog you need to be in `medit` on the mob.

- Use `mpedit list` to list all mprogs for the mob you are medit-ing.
- Use `mpedit add <progtype> [parameters]` to add a mprog to the end of the mprog list. The [parameters] depend on the progtype chosen
- Use `mpedit delete <prognumber>` to delete an mprog from the list.
- Use `mpedit insert <prognumber> <progtype> [parameters]` to insert a mprog in the list. The [parameters] depend on the progtype chosen.
- Use `mpedit edit <prognumber>` to edit the command list of an existing mprog.
- Use `mpedit copy <vnum to copy from> <prognumber of the copy mob>` to copy an mprog from another mob. Adds this mprog to the end of the list of the mob you are editing.
- Use `mpedit header <prognumber> <progtype> [parameters]` to replace an mprog header

completely.

When you are done creating and editing your rooms:

- Use `mpedit save` to save the current mprog changes to the mob file and update progs on mobs.
- Use `mpedit done` to exit from the mpeditor without saving.

Note: After exiting the editor buffer, you must still mpedit save, (or msave), to save the prog to the .mob file. This also updates all mobs with the prog. You can save an 'empty' mobprog to a prognumber, then access it later with mped edit.

BODY

After entering the editor, (mpedit edit <prognumber>. You must first be medit-ing the mobVnum.), you can manipulate the body of the mobprog.

Note: The editor will overwrite the line you are on.

```
Begin entering your text now (/? = help /s = save /c = clear /l = list)
(The line below is 78 characters - press ENTER before exceeding that length)

/?               display this help list
/l [from] [to]   list buffer (or part of it), with line numbers
/  [from] [to]   list buffer (or part of it), with line numbers
/n [from] [to]   list buffer (or part of it), without line numbers
/c               clear entire edit buffer
/d <line #>      delete specified line or current line if none specified
/g <line #>      goto line specified
/<line #>        goto line specified
/i <line #>      insert line before line specified or before current line if none
specified
/r <old> <new>   global search and replace
/a               abort editing without saving
/! <command>     execute external command outside of editor
/s               save buffer and exit editor
```

Note: After exiting the editor buffer, you must still mpedit save, (or msave), to save the prog to the .mob file. This also updates all mobs with the prog.

MPSTAT

This command can give you information about an existing mob's mprogs using `mpstat <mob name> | <mob #>`. An MPSTAT of a beastly fido, mob number 3062 will look like this:

```
Name: a beastly fido  Vnum: [3062]
Short description: a beastly fido
Hp: 11/11  Mana: 21/21  Move: 81/81
Lv: 1  Class: MOB (0)  Align: -200  AC: 9  Gold: 0  Exp: 5
1> act_prog pats 100
em wags his tail happily.
```

MPSAVE

Used when you are done editing the mob's mprogs and want to save them.

MOBPROGPERCENT

```
Usage: mobprogpercent <number>
```

Supply a number from 1-100 and I'll tell you the actual percentage chance of firing in a mobprog.

SPECIAL NOTE: The mud command mobprogpercent <#> will return the actual mobprog percent value for a given # (1-100), so you don't have to do the math or consult a chart. The formula is [Y=X^1.5].

OPEDIT

The opedit command allows you to edit a specific object's progs. You must be oedit-ing the object (oedit <Vnum>). You can Opedit save OR Osave to save all changes and update the progs on your mobs.

The objprog consists of the **HEADER**, which identifies the type of event that triggers the objprog, and necessary parameters, and the **BODY**, which identifies what other conditions to check and what actions to perform.

HEADER

Usage: `opedit <command> [parameters]`
Note: The parameters, and the number and order of them, vary according to the command.

```
Usage: opedit <list | add | delete | insert | edit | save | copy | header | done>
```

To create or edit an mprog you need to be in `medit` on the mob.

- Use `opedit list` to list all oprogs for the mob you are oedit-ing.
- Use `opedit add <progtype> [parameters]` to add a oprog to the end of the oprog list. The [parameters] depend on the progtype chosen
- Use `opedit delete <prognumber>` to delete an oprog from the list.
- Use `opedit insert <prognumber> <progtype> [parameters]` to insert a oprog in the list. The [parameters] depend on the progtype chosen.
- Use `opedit edit <prognumber>` to edit the command list of an existing oprog.
- Use `opedit copy <vnum to copy from> <prognumber of the copy obj>` to copy an oprog from another mob. Adds this oprog to the end of the list of the obj you are editing.
- Use `opedit header <prognumber> <progtype> [parameters]` to replace an oprog header completely.

When you are done creating and editing your rooms:

- Use `opedit save` to save the current oprog changes to the obj file and update progs on objs.
- Use `opedit done` to exit from the opeditor without saving.

Note: After exiting the editor buffer, you must still opedit save, (or osave), to save the prog to the .obj file. This also updates all objs with the prog. You can save an 'empty' objprog to a prognumber, then access it later with oped edit.

BODY

After entering the editor, (opedit edit <prognumber>. You must first be oedit-ing the obj Vnum.), you can manipulate the body of the objprog.

Note: The editor will overwrite the line you are on.

```
Begin entering your text now (/? = help /s = save /c = clear /l = list)
(The line below is 78 characters - press ENTER before exceeding that length)

/?               display this help list
/l [from] [to]   list buffer (or part of it), with line numbers
/  [from] [to]   list buffer (or part of it), with line numbers
```

```
/n [from] [to]   list buffer (or part of it), without line numbers
/c               clear entire edit buffer
/d <line #>      delete specified line or current line if none specified
/g <line #>      goto line specified
/<line #>        goto line specified
/i <line #>      insert line before line specified or before current line if none
specified
/r <old> <new>   global search and replace
/a               abort editing without saving
/! <command>     execute external command outside of editor
/s               save buffer and exit editor
```

Note: After exiting the editor buffer, you must still opedit save, (or osave), to save the prog to the .obj file. This also updates all objs with the prog.

OPSTAT

This command can give you information about an existing object's oprogs using `opstat <obj name> | <obj #>`. An OPSTAT of a water dragon, mob number 22517 will look like this:

```
Name: a statue of a water dragon  Vnum: [22517]
Short description: a statue of a water dragon.
Long description: A stone replica of what seems like a sea monster is
constructed here.
1> greet_prog 100
mpechoat $n Sensing the presence of a life form, the spirit the water speaks to
you from its statue.
silently mpmload 22526
```

OPTRIGGER

This command triggers an oprog to run. Usage: `optrigger <vnum> <prognumber>`

VI HEADER OPTIONS – TRIGGERS

Triggers are the various choices of types of mud events that initiate mobprogs and objprogs. Examples could include speaking, telling something to the mob, performing a certain action or social, manipulating an item, a certain hour of the day, being in a fight, getting hit by a certain spell, and many others. Some triggers respond when an event occurs, some check for the event at each pulse, some check at each round of fighting, etc.

MPTRIGGER types:

Act	reacts to action messages.
Allgreet	reacts when a PC or an NPC enters the mob's room.
Attack	reacts when a mob begins fighting a new target.
Birth	reacts when the mob is loaded into the mud.
Bribe	reacts when the mob is given coins.
Cast	reacts if a PC casts the assigned spell in the mob's room.
Command	reacts to commands typed by players.
Death	reacts when the mob dies.
Entry	reacts when the mob enters a room.
Fight	reacts during fights.
Give	reacts when the mob is given an item.
Greet	reacts when a PC enters the mob's room.
Hitpercent	reacts when below the assigned % of maxhits.
Pulse	reacts at an assigned pulse.
Random	reacts randomly, according to the assigned %.
Speech	reacts to speech messages.
Time	reacts at an assigned hour.
Position	Not implemented. Reacts to a mob position. Pos_prog.
Qcomplete	reacts when the quest is finished

OPTRIGGER types:

Attack	triggers when the object is used to attack
Birth	triggers when the object is loaded into the mud
Command	see mprog help for command triggers (same syntax)
Drop	triggers when you drop the item
Fight	checks every combat pulse
Get	triggers when you get the object
Greet	triggers when someone enters the room
Pull	triggers when you pull/push/turn/press the item
Random	triggers at a random time
Remove	triggers when you remove the item
Sac	triggers when you sacrifice the object
Speech	see mprog help for speech triggers (same syntax)
Wear	triggers when you wear the item
Act	Not implemented. see mprog help for act triggers (same syntax)
Close	Not implemented. triggers when an object is closed. Close_prog
Damage	Not implemented. triggers when an object takes damage. Damage_prog
Enter	Not implemented. triggers when an object is entered. Enter_prog
Examine	Not implemented. triggers when look or examine the item. Exa_prog
Open	Not implemented. triggers when the object is opened. Open_prog
Repair	Not implemented. triggers when you repair the object. Repair_prog
Use	Not implemented. triggers when you use the item. Use_prog

ACT_PROG/act_prog

Act: responds to action messages sent by the mud (same as the messages sent to players), that contain the specified keyword or text string. Mainly used for poses, socials, spell messages, and combat text. (Does not respond to say, tell, ask, whisper, or emote which are the domain of speechprog. It also does not respond to communication channels.)

MPROG

Actprogs trigger when the mob is the victim of a message, so you can make a mob react to things like, 'licks you', 'You notice the spells wearing off too fast to notice', 'critically vaporizes you', and so on.

`< > mpedit add act <text string>` (Note: No assigned %.)

Examples:

`act_prog licks you` `tickle $r`	The mob will tickle a random player in the room if you lick him.
`act_prog licks` `giggle`	The mob will giggle if anyone in the room licks someone else, or the mob, or themselves.
`act_prog debris from the large tremor` `mpforce $n flee`	The mob will force a player to flee if the player casts earthquake. Note: This player must be in the room with the mob, visible to the mob, and lower level than the mob for mpforce to work.

`act_prog critically vaporizes you`
`if rand(50)`
`say That time it hurt.`
`Endif`
	About half of the times anyone critically vaporizes the mob, it will comment on it.

ALLGREET_PROG/allgreet_prog

Allgreet: Tests the prog when a PC or an NPC enters the mob's room. (Will not fire if the mob is in combat).

MPROG

`< > mpedit add allgreet <#>` (# ranges from 1-100 but represents a range of 1-1000.)

Example:

`allgreet_prog 100` `say Welcome to the Dwarven Digs, $N!`	Every time a mob or a player enters this mob's room, he will welcome them by name.

ATTACK_PROG/attack_prog

Attack: Tests the prog once, when a mob is first attacked or drawn into a fight.

MPROG

Attack reacts when a mob begins fighting a new target.

`< > mpedit add attack <#>` (# ranges from 1-100 but represents a range of 1-1000.)

Examples:

`attack_prog 100`
`if level($n) < 51`
`mptransfer $n 3001`
`endif`
	If a player level 50 or under attacked this mob, by any means, they would find themselves back in the Temple before the first round of combat.

`attack_prog 63` `yell Mom!! $N is going to hit me!`	About 50% of the time, this mob will yell, before it has even been hit.

Note: The prog is actually tested when the mob is assigned a new fight target, and before the first combat pulse, so, it is possible to intercept a combat before it begins. The most common reason a mob is assigned a fight target is that a player

attacked the mob, but it also happens, and this progtype fires, when a mob switches, when its target is rescued, or when an aggressive mob initiates combat. Thus, the prog does not fire when a group member joins combat, nor if a second group or player attacked a mob that was already fighting a group or player.

OPROG

Attack triggers when the object is used to attack

`< > opedit add attack <#>` (# ranges from 1-100 but represents a range of 1-1000.)

BIRTH_PROG/birth_prog

Birth: Tests the prog when the mob is loaded into its zone.

MPROG

Birth reacts when the mob is loaded into the mud.

`< > mpedit add birth <#>` (# ranges from 1-100 but represents a range of 1-1000.)

Examples:

`birth_prog 100` `yell I'm back!!!`	This prog always alerts anyone in the area that this mob has repopped
`birth_prog 100` `mpsetskill 'bash' 95`	This prog enables the mob to use the skill ' bash ' with a 95% chance of success.
`birth_prog 63` `cast 'sanctuary'`	About half the time, when this mob repops, it sancs itself.

.

Note: If this mob has the 'cleric' disposition, it will know the sanc spell, but if it doesn't, you would have to enable the spell, using mpsetskill, for it to work. A birthprog can also be used to adjust the assigned skills/spells in a given disposition.

Note: Enabling skills and spells can be done in any mobprog, but are usually done in a birthprog, so that it only needs to be checked/assigned when the mob is loaded into the mud. Also note that you can limit the chance of the mob knowing a spell (by putting a % chance on the birthprog) as well as limiting the mob's chance of using the spell succesfully.

OPROG

Birth triggers when the object is loaded into the mud

`< > opedit add birth <#>` (# ranges from 1-100 but represents a range of 1-1000.)

BRIBE_PROG/bribe_prog

Bribe: Tests the prog if a player gives the mob the specified number of coins, or more.

MPROG

`Bribe reacts when the mob is given coins.`

`< > mpedit add bribe <#>` (# is the number of coins.) (Note: No assigned %.)

Example:

bribe_prog 50000 say This is a lot of money. I'm going to put it in a safe place. Show yourself out. Don't go north while I'm not looking, either, there are plenty more guards in there. east mpdelay $i 20 west	If given more than 50000 coins, this mob (we are assuming that he is a blocker for the path north) will warn you off, then leave the door unguarded for 20 pulses.

Note: A similar way to temporarily allow players to get past a blocker mob would be to have him take a short nap.

Note: The coins don't actually go to the mob for his use. The coins go into the mob's inventory as an object. If you don't want that, have him silently sacrifice coins. If needed, you can use mpcoins to load coins on a mob.

CAST_PROG/cast_prog

Cast: Triggers when the specified spell is cast by a player.

MPROG

Cast reacts if a PC casts the assigned spell in the mob's room.

< > mpedit add cast <spell name> (Note: No assigned %.)

Examples:

cast_prog protection from death if rand(62) say He'll still get you. Endif	About half the time that anyone casts the spell 'protection from death' in this mob's room, he will issue this warning.
cast_prog sanctuary if actortarget($t) cast 'dispel magic' $n endif	When someone casts the spell 'sanctuary' in this mob's room, the mprog checks to see if they cast it on themselves, and if so, the mob tries to dispel them.
cast_prog drain if mobtarget($t) cast 'harm' $n endif	If anyone casts drain in the room with this mob, the mprog checks whether the spell was targetted at the mob. If it was, the mob casts 'harm' at the caster who tried to drain him.

Note: This assumes that the mob knows these spells.

Note: The mprog triggers when the named spell is cast. Spells that fail because the target already has that affect, for example, will trigger the mprog. A 'lost concentration' will not.

COMMAND_PROG/command_prog

Command: Triggers on the specified command typed by a player.

MPROG

Command reacts to commands typed by players.

< > mpedit add command <type> <command> <#>

- <command> is the actual commands typed by players, such as look, get, kill, turn, emote, say, poke, cry, cast, north, etc. It can also be used to react to 'commands' that aren't actual mud commands.
- <#> ranges from 1-100, but represents a range of 1-1000.
- <type> parameter (unlike any other mptrigger).
    - Type 0: the mud executes all commands, whether they match the trigger and fire the prog or not.
    - Type 1: the mud executes the command if it does NOT match the trigger and set off the prog and eats or swallows or 'does not execute' the command if it does match the trigger.
    - Type 2: the mud never executes the command.

SPECIAL NOTE: Type 2 commandprogs can be very risky. Unless options inside the prog allow, or cause, or force things to happen, (or the prog doesn't fire 100% of the time), NOTHING that is typed in the room will or can happen. This calls for cautious and well thought out design, which can usually be achieved in other, safer ways.

SPECIAL NOTE: Command_progs do not work in room 1200. This provides a 'safe room' to try to deal with a rogue or runaway prog.

Examples:

command_prog 0 look 100 say What are you looking at, Buster? slap $n	This prog is at 100%, so it always fires, if a player types the command ' look '. Any player who looks at anything will be taunted and then slapped. Because it is a type ' 0 ' the prog will not eat the command, so the player will also see the description of whatever/whoever it looked at.
command_prog 1 look 63 emote pokes you in the eyes!	About half the time, this prog will fire if a player types ' look '. When it doesn't, the player will see whatever it looked at. When it does fire, being a type 1, it will eat the command, and the player will see the emote instead of the description of what it looked at. (The room will also see the emote.)
command_prog 1 look 100 if cmdarg(0) == mobsname mptransfer $n 26204 endif	Since it always fires and is type 1 and eats the command this prog will never send any results to a player from a look command, but if the player typed the exact phrase, 'look mobsname ' they would be transferred to room 26204
command_prog 0 say 50 mpecho The parrot says, 'Bwuaaaak! $c!'	About 35% of the time, the mob with this prog will send an echo to the room that includes an exact quote of what a player said in the room. (It would obviously make more sense if a parrot was in theroom.)
command_prog 0 * 100 $C	(Note: * is supported as a 'wild card' for any command.) This mob would mimic players in the room, performing (or trying to perform) every action the player does.
command_prog 1 burrow 100 if cmdarg(0) == north mpechoat $n You burrow into the rubble and find a way through it. mptransfer $n 20834 endif	If a player typed the phrase 'burrow north' they would see the message and be transferred to room 20834, just as if they had found an exit leading to that room. Because it is a type 1, the command will not be executed. In this case, that means no 'Unknown Command' message would be sent to the player.
command_prog 1 * 100 if cmd($n) == escapeword mptransfer $n 1200 endif	This shows an example of leaving yourself a way to get out of a prog you are building and testing. Any word or phrase you typed in this room would be 'eaten,' godcommands and editing commands included, except for escapeword, which would transfer you out of the prog's room
command_prog 2 look 63 mpechoat $n There is a blinding flash!	The command 'look' will never be executed in this room. About half the time it is typed, the player will see the echo.

SPECIAL NOTE: Commandprogs can eat the command that is typed, or not, and then test for other conditions, and react, or not. On the Big Wish List is a way to have a commandprog suppress the command, but then execute it (force or allow the player to execute it) if none of the ifcheck conditions are met, or in other ways at the builder's discretion. So far, all attempts to do this send the command back to the room, where it is caught by the commandprog again, causing a loop. Maybe someday.

OPROG

Command see mprog help for command triggers (same syntax)

`< > opedit add command <type> <command> <#>`

- <command> is the actual commands typed by players, such as look, get, kill, turn, emote, say, poke, cry, cast, north, etc. It can also be used to react to 'commands' that aren't actual mud commands.
- <#> ranges from 1-100, but represents a range of 1-1000.
- <type> parameter (unlike any other mptrigger).
  - Type 0: the mud executes all commands, whether they match the trigger and fire the prog or not.
  - Type 1: the mud executes the command if it does NOT match the trigger and set off the prog and eats or swallows or 'does not execute' the command if it does match the trigger.
  - Type 2: the mud never executes the command.

DEATH_PROG/death_prog

Death: Tests the prog when the mob dies.
Deathprogs clear any waitstates so any deathprog commands will be executed immediately before the mob dies.

MPROG

Death reacts when the mob dies.

`< > mpedit add death <#>` (# ranges from 1-100 but represents a range of 1-1000.)

Example:

`death_prog 63`
`mpset $i position 12`
`mpoload 3010`
`silently drop bread`
`mpset $i position 0`
	50% of the time, when this mob is killed, it will load a loaf of bread (Vnum 3010) in the room. (The item loads in the mob's inventory. The 'silently' command allows the mob to drop the item in the room without any message sent to the room.)

Note: Most mpcommands will work in a deathprog, but many regular commands will not, (because the mob is dead). This example changes the mob's position to 'standing,' to allow him to drop the bread, then returns him to position 'dead.'

DROP_PROG/drop_prog

Drop: Triggers when an item is dropped.

OPROG

Drop <%> triggers when you drop the item

`< > opedit add drop <#>` (# ranges from 1-100 but represents a range of 1-1000.)

ENTRY_PROG/entry_prog

Entry: Tests the prog when the mob enters a room and any player is in the room.

MPROG

```
Entry          reacts when the mob enters a room.

< > mpedit add entry <#>            (# ranges from 1-100 but represents a range of 1-1000.)
```

Example:

`entry_prog 100`
`if islevel($n) <= 50`
`mptransfer $n 3001`
`mpat 3001 mpecho And stay out!!  I mean it, $N!!`
`else`
`mpechoat $n Take me to your leader!`
`Endif`
	When this mob enters a room that has any players in it, it tests each one's level, and any non-nobles are transferred to the Temple (Vnum 3001) and a stern warning (by name), is echoed to the Temple. Noble players are (individually) sent the 'leader' message.

FIGHT_PROG/fight_prog

Fight: Tests the prog each combat round, when the mob is fighting.

MPROG

```
Fight          reacts during fights.

< > mpedit add fight <#>            (# ranges from 1-100 but represents a range of 1-1000.)
```

Example:

`fight_prog 12`
`mpecho The spoiled brat lights a firecracker and throws it behind you.$\ The firecracker explodes, injuring you slightly from the flying debris!`
`mpareadamage 2d7+18`
	About 42 out of 1000 combat pulses, the mob will send a message to the room, just as if it had performed an action, then every player in the room takes the amount of damage rolled.

Note: the special character $\ inserts a carriage return in the text string, so players in the room see this message as 2 lines of text, immediately following each other.

OPROG

Fight <%> Not implemented? checks every combat pulse

```
< > opedit add fight <#>            (# ranges from 1-100 but represents a range of 1-1000.)
```

GET_PROG/get_prog

Get <%> triggers when you get the object

OPROG

Get <%> triggers when you get the object

`< > opedit add get <#>` (# ranges from 1-100 but represents a range of 1-1000.)

GIVE_PROG/give_prog

Give: Tests the prog if a PC gives the mob the specified item.

MPROG

Give reacts when the mob is given an item.

`< > mpedit add give <Vnum / text / "textstring" / * >` (Note: No assigned %.)

Examples:

`give_prog *`
`say Just what I needed, but not very much.`
`drop $o`
`sneer`
	(Note: * is supported as a 'wild card' for any object.)
	This prog will always trigger and will trigger on any object given to the mob, who will comment on it, drop it, and then sneer.

`give_prog bread`
`if rand(30)`
`eat $o`
`say Thanks, I sure was hungry.`
`else`
`drop bread`
`say Dork, I'm not hungry.`
`Endif`
	This prog triggers if the mob is given any item that responds to the keyword 'bread.' About 30% of the time, the mob eats the bread and thanks you, the rest of the time the mob drops the bread and insults you.
	Note: The prog assigns the text 'bread' to the mpvariable $o, and responds to either one.

`give_prog pipeweed` `eat bread`	This prog triggers if the mob is given any item that responds to the keyword 'pipeweed.' Note: If an item existed named 'pipeweed' that wasn't bread, and you gave it to this mob, it would try to eat bread from its inventory.
`give_prog pipeweed` `eat $o`	This prog triggers if the mob is given any item that responds to the keyword 'pipeweed.' Note If an item existed named 'pipeweed' that wasn't bread, and you gave it to this mob, it would try to eat the item.
`give_prog 3012` `eat pipeweed`	If you give this mob pipeweed bread (Vnum 3012) it eats it.
`give_prog 3012` `eat $o`	If you give this mob pipeweed bread (Vnum 3012) it tries to eat 'bread,' because the prog has assigned the text 'bread' to the mpvariable $o, because 'bread' is Vnum 3012's first keyword. The pipeweed bread you just gave it will be the first bread it finds in inventory, however. If you give it any other bread, nothing will happen, because it only triggers on the Vnum specific to pipeweed bread.

`give_prog *`
`if vnum($o) == 3012`
`eat pipeweed`
`say Thanks!`
`else`
`give $o $n`
`say I don't want this.`
`endif`
	If you give this mob pipeweed bread (Vnum 3012) it eats it. If you give it any other item, it gives it back to you and tells you why.

`give_prog <pipeweed bread>` `drop $o`	This text string option is supported but not recommended. This example prog will never fire, because it is searching for the 'exact match' for 'pipeweed bread' in the object file's aliases, and object 3012's first alias is bread.

SPECIAL NOTE: These examples were valid when mobprogs were coded, and should still be valid training examples, but there is NO EXCUSE for dropping untested mobprogs into the mud!

GREET_PROG/greet_prog

Greet: Tests the prog when a PC enters the mob's room. (Won't fire if the mob is in combat).

MPROG

Greet reacts when a PC enters the mob's room.

`< > mpedit add greet <#>` (# ranges from 1-100 but represents a range of 1-1000.)

Examples:

`greet_prog 100`
`if isevil($n)`
`and race($n) == elf`
`and class($n) == cleric`
`say Wow, an evil elf cleric! I thought I was the only one!`
`endif`
`if race ($n) != elf`
`and race($n) != half-elf`
`tell $n This is elven territory. You really should leave.`
`endif`
	This tests every player who enters the mob's room, first to see if they are evil AND an elf AND a cleric, comments to the room about it if all 3 are true, then tests to see if each player is not an elf AND not a half-elf and warns away each of those who is neither.
	
	Note: The if/and combination only requires a single endif (unlike the nested ifchecks).
	
	Note: In this case, one combination of ifcheck is tested after the other combination of ifchecks has been tested. (They aren't nested). Each ifcheck or combination of them requires its own endif.

`greet_prog 55`
`if isevil($n)`
`or isgood($n)`
`say Santa tells me you've been bad or good!`
`Endif`
	40% of the times when a player enters the room the mob is in, it tests whether the player is good OR is evil, and comments on it if either is true.
	
	Note: The if/or combination only requires the single endif (unlike the nested ifchecks)
	
	Note: If a group of 9 players entered this mob's room and none of them were neutral-aligned, each of them would see the mob 'say' this text string 9 times. This would probably not be considered good design.

OPROG

Greet <%> triggrs when someone enters the room

`< > opedit add greet <#>` (# ranges from 1-100 but represents a range of 1-1000.)

HITPERCENT_PROG/hitprcnt_prog

Hitpercent: Tests the prog if the mob goes below the assigned % of maxhits.

MPROG

`Hitpercent` `reacts when below the assigned % of maxhits.`

`< > mpedit add hitpercent <#>` (# ranges from 1-100 but represents a range of 1-1000.)

Example:

`hitpercent_prog 99`
`flee`
`mprestore $i`
	If this mob lost more than 15 per 1000 of its maxhits, it would flee and fully restore itself. (Unless you killed it in one round or enjoyed chasing it from room to room all over the mud, it would probably not be considered good design.) Note: This is only checked while fighting.

PULL_PROG/manip_prog

Pull: Triggers the prog when the items is manipulated.

OPROG

Pull <%> triggers when you pull/push/turn/press the item

`< > mpedit add pull <#>` (# ranges from 1-100 but represents a range of 1-1000.)

Example:

`manip_prog 100` `mpechozone 333 A large icy capsule rises from the center of the cathedral and hovers over the northwest tower.`	All the time, when the item is manipulated (push, pull, turn, press, etc) it sends this message to all players in the zone.

PULSE_PROG/pulse_prog

Pulse reacts at an assigned pulse.

MPROG

Pulse reacts at an assigned pulse.

`< > mpedit add pulse <#>` (# ranges from 1-100 but represents a range of 1-1000.)

QCOMPLETE_PROG/qcomplete_prog

Qcomplete reacts when the quest is copleted.

MPROG

Pulse reacts at an assigned pulse.

`< > mpedit add pulse <#>` (# ranges from 1-100 but represents a range of 1-1000.)

RANDOM_PROG/rand_prog

Random: Tests each pulse against the assigned % chance of firing.

MPROG

Random reacts randomly, according to the assigned %.

`< > mpedit add rand < # >` (# ranges from 1-100 but represents a range of 1-1000.)

SPECIAL NOTE: The mud command mobprogpercent <#> will return the actual mobprog percent value for a given # (1-100), so you don't have to do the math or consult a chart. The formula is [Y=X^1.5].

Note: A mobprog rand_prog 100 will fire every pulse. This is commonly used to test for specific ifcheck conditions, that don't fall into the other mobprog types. The Random ifcheck can be used internally to set frequency limits.

Examples:

`rand_prog 100`
`if playerspresent($i) >= 2`
`grumble`
`mpecho $I mutters, 'I want to be alone.'`
`endif`
	The mud checks every pulse to see if 2 or more players are in the room with the mob. If there are, the mob complains out loud about it.

`rand_prog 40`
`if playerspresent($i) >= 2`
`hug $r`
`if rand(60)`
`kiss $r`
`endif`
`endif`
	This prog fires 25% of the time. When it does, it tests to see if 2 or more players are in the room with the mob. If there are, the prog picks one player at random and the mob hugs it. Then, 60% of the time, the mob kisses the same player.
	Note: This is a nested ifcheck, and each ifcheck requires its own endif.

OPROG

Random <%> triggers at a random time

`< > opedit add rand < # >` (# ranges from 1-100 but represents a range of 1-1000.)

REMOVE_PROG/remove_prog

Remove <%> triggers when you remove the item

OPROG

`< > opedit add remove < # >` (# ranges from 1-100 but represents a range of 1-1000.)

SAC_PROG/sac_prog

Sac <%>: triggers when you sacrifice the object

OPROG

`< > opedit add sac < # >` (# ranges from 1-100 but represents a range of 1-1000.)

SPEECH_PROG/speech_prog

Speech: responds to speech, including say, tell, ask, whisper or emote, containing the keyword or text string. (Does not respond to communication channels).

MPROG

```
Speech         reacts to speech messages.

< > mpedit add speech <text string>        (Note: No assigned %.)
```

Example:

`speech_prog Open Sesame!`
`open gate`
`if sex($n) == male`
`whisper $n I wouldn't go in there,`
`bud!`
`endif`
	If the text string matches exactly, the mob opens the gate, then whispers a warning to the player if he's male.
	
	Note: The speech_prog argument (Open Sesame!, in this example) is not case sensitive, but all punctuation does count. In this example, if a player says 'Open Sesame!' the prog will trigger. If a player emotes, 'emote I think we have to emote open sesame! or something.', the prog will also trigger. If a player whispers, Open sesame the prog will not trigger, because the ! is missing.

OPROG

Speech: see mprog help for speech triggers (same syntax)
`< > opedit add speech <text string>` (Note: No assigned %.)

TIME_PROG/time_prog

Time: Tests the prog at a specific hour of the day.

(SPECIAL NOTE: Timeprog is very limited. I have specific ideas about improving it, there just hasn't been time to work on it yet.)

MPROG

```
Time  reacts at an assigned hour.

< > mpedit add time < # / * >        (Note: No assigned %.)
```

is the hour of day or * is a wild card for any hour.

Examples:

`time_prog 24`
`if iszoneempty($i)`
`shout Midnight, and all's well!`
`endif`
	This prog fires at midnight, then always tests to see if any players are in the same zone as the mob, and shouts if there are none.

`time_prog *` `say Do you hear bells??`	This mob whispers every hour. (Note: * is supported as a 'wild card' for any hour.)

WEAR_PROG/wear_prog

Wear <%> triggers when you wear the item

OPROG

< > opedit add wear < # > (Note: No assigned %.)

VI BODY OPTIONS

MPVARIABLES

MPVARIABLES

```
   $i    mob doing the mobprog
   $n    person causing the trigger
   $o    object causing the trigger
   $r    random PC in the room
   $t    target PC
   $p

Operators used in ifchecks for STRINGS:
  ==   checks for an exact string match
  !=   checks for not equals, or no match
   /   checks to see if the specified string is contained in the target string
  !/   returns true if there is no match in the target string

Operators for MATH comparisons:
  ==   equality
  !=   not equals
  <=   less than or equal to
  >=   greater than or equal to
  >    greater than
  <    less than
  &    bitwise AND
  |    bitwise OR
```

IFCHECKS

MPIFCHECKS

If-checks are used in MOBPROGS to test certain targets for certain conditions, such as age, alignment, sex, position, vnum, and many others. Common targets include $i, the mob itself; $o, an object; $n, the player who triggered the prog, and $r, a random player in the room. **MPVARIABLES** determine the target of the ifcheck, as well as the target of the consequences. (Numerical and String operators are also listed in Mpvariables.)

Typical usages:

```
if iswhatever($X)
if iswhatever($X) >= <#>
if iswhatever($X) / <text string>
```

Examples:

```
if isgood($r)
if hitpercent($i) <= 35
 if name($n) !/ dragon
```

Multiple ifchecks:
You can use OR and AND in your ifchecks. They must appear on a new line.
Example:

```
if dex($n) >= 25
or level($n) >= 58
```

```
bow $n
endif
```

Note: Each ifcheck, or combination of ifchecks, must have an endif.

IFCHECKS

rand	cansee	isnpc	ispc	isimmort
isgood	isneutral	isevil	isfighting	ischarmed
isfollowing	isgrouped	ismounted	isoutlaw	ispthief
race	sex	position	class	name
inroom	inzone	vnum	numfollowers	hitpercent
manapercent	movepercent	gold	qpoints	age
level	size	clannum	legendpoints	nobpoints
str	dex	con	wis	int
cha	luk	hp	mp	vp
maxhp	maxmp	maxvp	objtype	objval(0-5)
playerspresent	mobspresent	iszoneempty	mobhere	objhere
carriesobj	wearsobj	hasobj	timehour	timeday
timemonth	timeyear	actortarget($t)	mobtarget($t)	cmd($n)
cmdarg				

ACTORTARGET, MOBTARGET, OBJTARGET

actortarget($t): checks to see if assigned spell was targeted at the caster (i.e. did the player cast that particular spell on himself, in the room with the mob).
`Usage: if actortarget($t)`

mobtarget($t): checks to see if the assigned spell was targeted at the mob
`Usage: if mobtarget($t)`

objtarget($o): checks to see if the target is an object

SPECIAL NOTE: The ifchecks actortarget($t), and mobtarget($t) are used in cast_progs. These are explored more fully in the cast_prog examples under MPTRIGGERS. The variable $t is not currently assigned in any other progtype. Do NOT use $t as an independent variable under any circumstances.

AFFECTEDBYSPELL

affectedbyspell(#): checks to see if the mob with the mprog is affected by a certain spell. The number is the spell number. See the builder's handbook for a list of the spells and their numbers.

AGE

age: returns age of target

CANSEE

cansee: returns true if the mob can see the target victim or object (for mobs who don't see invis, hidden, dark, etc.)

CARRIESOBJ, WEARSOBJ, HASOBJ

carriesobj: checks for a specific object in inventory
wearsobj: checks for a specific object in equipment

hasobj: checks for a specific object in inventory and equipment
Examples: Note: <Vnum> or <text>

```
if hasobj($i) == 3005
if wearsobj($n) == Dispater
```

CLANNUM, CLANRANK

clannum: returns clan number for PC's

clanrank: returns the PC's clan rank number

CLASS

class: returns class of target

```
if class($n)!= <text>
```

CMD, CMDARG, CMDQUEUELEN

cmdarg: checks extra command arguments entered when the prog was triggered
`Usage: if cmdarg(#) <string-operator> <text>`
Note: ' # ' represents which argument you want to test. ' 0 ' is the FIRST, ' 1 ' is the SECOND, etc.

Examples:

```
if cmdarg(0) / sanc
if cmdarg(2) !/ $n
if cmdarg(0) == 'dispel
if cmdarg(1) / magic
```

cmd($n): checks the command entered when the prog was triggered
`Usage: if cmd($n) <string-operator> <text>`

Examples:

```
if cmd($n) == cast
if cmd($n) !/ look
```

cmdqueuelen($i): checks the command queue to see how many commands are left
`Usage: if cmdqueuelen($i) <string-operator> <text>`

Example:

```
if cmdqueuelen($i) < 6
```

SPECIAL NOTE: The ifchecks 'cmdarg' and 'cmd($n)' are ONLY available for use in command_progs. They are explored more fully in the command_prog examples under MPTRIGGERS.

EXPLANATION:
For use in commandprogs (only), anything a player types is temporarily recorded, in the format: cmd($n) cmdarg(0) cmadarg(1) cmdarg(2) cmdarg(3)... These stored commands and arguments can be tested (in commandprogs only). For example, if a player typed cast 'protection from dea' doopw, the ifcheck if cmdarg(1) == from would return true. The ifcheck if cmdarg(2) == death would return false, but the ifcheck if cmdarg(2) / de would return true, because 'de' is contained in 'dea'. The ifcheck if cmdarg(0) == protection would return false, because the apostrophe counts as part of the match. The entire text string cast 'protection from dea' doopw would be recorded in the mpvariable $C and the entire argument text string 'protection from dea' doopw would be recorded in the mpvariable $c.

SPECIAL NOTE: commandprog/cmd/cmdarg are useful for specific tests like a mob responding to having 'dispel magic' cast at it, but they are also highly vulnerable to careless design and unpredicted results. BUILD WELL! TEST!! TEST!! TEST!! TEST!! TEST!! TEST!! TEST!! There is NO EXCUSE for dumping untested mobprogs into the mud.

EXTRA SPECIAL NOTE: commandprog, and these examples, were developed before castprog was coded. Castprog is much better for handling spells!

DRUNK, HUNGRY, INSANE, STONED, THIRSTY, WIRED

isdrunk: true if drunk greater than 0

ishungry: true if hungry is at 0

isinsane: true if insane greater than 0

isstoned: true if stones greater than 0

isthirsty: true if thirsty is at 0

iswired: true if wired greater than 0

GOLD

gold: returns gold of target

HIT, MAX HIT, HIT %, MANA, MAX MANA, MANA %, MOVE, MAX MOVE, MOVE %

hp, mp, vp: returns current h/m/v for target

maxhp, maxmp, maxvp: returns max h/m/v for target

hitpercent:
manapercent:
movepercent: returns current h/m/v percent of target

```
if hitpercent($n) >= 75
```

INOBJ, INROOM, INZONE

inroom: returns current room vnum of target

```
if inroom($n) == 3001                    Note:  Numeric only.
```

inzone: returns current zone number of target

```
if inzone($i) == 32                      Note:  Numeric only.
```

inobj: returns if the item is inside another item

```
if inobj(#) == 32      Is item # inside item 32?      Note:  Numeric only.
```

ISCHARMED

ischarmed: true if target is charmed

ISEXITCLOSED, ISEXITLOCKED

isexitclosed(#): true if the exit in the direction indicated is closed. The number is based on the direction number. 0 for north, 1 for east, etc. See the builder's manual under Exits for direction numbers.

isexitlocked(#): true if the exit in the direction indicated is locked. The number is based on the direction number. 0 for north, 1 for east, etc. See the builder's manual under Exits for direction numbers.

ISFIGHTING

isfighting: true if target is in combat

ISFOLLOWING, ISLEADER

isfollowing: true if the mob is following the target $n

```
if isfollowing($n)
```

isleader: true if the mob checked ($n) is solo or the leader of a group

```
if isleader($n)
```

ISGOOD, ISEVIL, ISNEUTRAL

isgood: true for good aligned targets

isneutral: true for neutral aligned targets

isevil: true for evil aligned targets

ISGROUPED

isgrouped: true if the target is a group member.

ISMOUNTED

ismounted: true if target is mounted

ISNPC, ISPC, ISIMMORT

isnpc: only true for mobs

ispc: only true for real players

isimmort: true if target is a god PC

ISOUTLAW

isoutlaw: true for outlaws

ISQUESTEDBYMOB

isquestedbymob: true if the PC is on a quest and the mob specified is the quest mob

ISPTHIEF

ispthief: true for pthieves

ISZONEEMPTY, MOBSINZONE

iszoneempty: returns true if the zone has no PC's present

```
if iszoneempty($i / # )
```

($i) specifies the zone the mob is in
(#) specifies a zone number

mobsinzone: returns true if the specific mob is in the zone

```
if mobsinzone(#)
```

(#) specifies a specific mob number

LEGENDPOINTS or HEROPOINTS, NOBPOINTS or NOBLEPOINTS

legendpoints: returns current number of legend or hero points for PC's

nobpoints: returns current number of nobility points for PC's account
Note: nobpoints for a PC below level 50 always returns 0 (zero).

LEVEL

level: returns level of target

MOBHERE, MOBZONE

mobhere: checks for a specific mob in the room Note: <Vnum> or <text>
Example:

```
if mobhere(13201)
say Hey, its the meta sage guard!
endif
```

mobzone: returns current zone number of the mob

```
if mobzone($n) == 32
```

Note: Numeric only.

MOBSPRESENT

mobspresent: returns the number of NPC's in the room (the mob counts itself)
Supports ($i) for the mob's current room, or (roomVnum) for a specified room

MOBNEXIST, OBJNEXIST

mobnexist: checks for a specific mob in the world Note: <Vnum> or <text>
Example:

```
if mobnexist(13201) > 0
```

objnexist: checks for a specific obj in the world Note: <Vnum> or <text>
Example:

```
if objnexist(13201) > 0
```

NAME

name: returns name of object or person

```
if name($n) / <text>
```

Note: The / operator returns true if the text is part of the target's name.

Example:

```
if name($n) / dragon
```

```
smirk $n
endif
```
This mob would smirk at a player named Pendragon, Dragonsbane, RedDragonBelt, etc.

NUMFOLLOWERS

numfollowers: returns the player's total number of followers
`if numfollowers($n) <= 9`

Note: This returns 'followers,' not 'group-members.' A player might have a number of ungrouped charmed beasts or familiars technically following him, but not grouped. This would count them.

OBJHERE, OBJSINZONE

objhere: checks for a specific object in the room Note: <Vnum> or <text>
Example:

```
if objhere(identify)
say Who left an identify scroll here?
endif
```

objsinzone: returns true if the specific object is in the zone
`if mobsinzone(#)` `(#)` specifies a specific object number

OBJTYPE

objtype: returns object type number Note: Numeric only
`if objtype($o) == 1` true if object is a LIGHT (object type 1)

See Builder's Handbook for object type numbers.

OBJVAL 0 TO 5

objval0 - objval5: returns val0 through val5 of target object
`if objval2($o) <= 0` Note: Numeric only

See Builder's Handbook for object values.

PLAYERSPRESENT

playerspresent: returns the number of PC's in the specified room, or the mob's current room

Usage:

`if playerspresent($i) >= <#>`	Returns true if more than ' # ' PC's are in the mob's current room
`if playerspresent(Vnum) == <#>`	Returns true if exactly ' # ' PC's are in the specified room

POSITION

position: returns position of target
`if position($i) == <text>`

QPOINTS

qpoints: return qpoints of target

RACE, SEX

race: returns race of target
```
if race($n) ==<text>
```

sex: returns sex of target
```
if sex($n)!=<text>
```

RAND, RANDOMNUM

rand: it rolls a random number 1-100 and if the number is less than the specified number it triggers the if statement.

```
if rand(75)        true 75% of the time
```

While Randomnum is an option it doesn't appear to do anything. Use rand when wanting something to trigger a certain percentage of the time.

SIZE

size: returns size of target obj, player, or mob

STRENGTH, DEXTERITY, CONSTITUTION, WISDOM, INTELLIGENCE, LUCK, CHARISMA

str, int, wis, dex, cha, con, luk: returns the chosen stat value

Example:
```
if luk($n) >= 6
```

TIMEHOUR, TIMEDAY, TIMEMONTH, TIMEYEAR

timehour:
timeday:
timemonth:
timeyear: checks for a specified year/month/day/hour
Usage:
```
if time(X) <numerical operator> <#>
```

Examples:
```
if timemonth > 6
if timehour != 12
and timehour >= 8
if timehour == 2
```

VAR

var: returns the mprogvar number
```
if var($i) == 8
```

Use this with MPVAR to set the mpprogvar.

VNUM

vnum: returns vnum of object or mob
```
if vnum($o) == 3000                    Note:  Numeric only.
```

MPCOMMANDS

MPCOMMANDS

mpareadamage	mpasound	mpat	mpcoins	mpdamage
mpdamagearound	mpdelay	mpdoor	mpecho	mpechoaround
mpechoat	mpechozone	mpforce	mpflush	mpgive
mpgorandom	mpgoto	mpjunk	mpkill	mpmload
mpoload	mpopen	mppurge	mpqend	mpqstart
mprestore	mproom	mpset	mpsetskill	mpstat
mptransfer	mptrigger	mpvar	mpvpurge	mpzreset
opstat	optrigger	silently		

These are the special 'prog only' commands that you can give your mobs within mprogs or objects within oprogs. Mobs can also execute any normal player commands (bow, give, put, pose, drop, fart, south, etc.)

MPASOUND

mpasound Sends text to all adjacent rooms

```
Usage:  mpasound <text string>
```

MPAT

mpat allows the mob to perform an action in a different room

```
Usage:  mpat <roomVnum> <command> [parameters]
```

Examples:

```
mpat 12472 mpecho You hear screeching in the distance.
mpat 3001 yell Where am I?
mpat 8702 pull lever
mpat 16028 mppurge
```

MPCOINS

mpcoins loads a specified number of coins on the mob

```
Usage:  mpcoins <#>
```

MPDAMAGE, MPAREADAMAGE, MPDAMAGEAROUND

mpdamage damages a target (damage must be in the exact form: XdY+Z)

```
Usage:  mpdamage <$X / text> XdY+Z
```

Examples:

```
mpdamage $n 12d40+112
mpdamage guard 4d6+12
```

mpareadamage damages every PC in the room

```
Usage:  mpareadamage XdY+Z
```

mpdamagearound damages every PC in the neighboring rooms

```
Usage:  mpdamagearound XdY+Z
```

MPDELAY

mpdelay delays the target for the specified number of pulses

```
Usage:  mpdelay <$X / room> <#>
```

Examples:

`mpdelay $i 2`	delays the mob for 2 pulses.
`mpdelay $n 20`	delays the player who triggered the prog 20 pulses.
`mpdelay room 4`	delays the entire room.

MPDOOR

mpdoor manipulates a door in a certain direction

```
Usage:  mpdoor <$X / room> <direction> <open/closed/unlocked/locked/wizlocked>
```

MPECHO, MPECHOAT, MPECHOAROUND, MPECHOZONE

mpecho sends text to everyone in the same room as the mob

```
Usage:  mpecho <text string>
```

Note: Mpecho with no text can be used to insert a blank line in the output.
Note: Similarly, the variable $\ will insert a carriage return, so that your output moves to the next line.

mpechoat sends text to a specific person

```
Usage:  mpechoat <$X / text> <text string>
```

mpechoaround sends text to everyone in a room except the mob and target

```
Usage:  mpechoaround <$X / text> <text string>
```

mpechozone sends text to everyone in a zone except the mob and target

```
Usage:  mpechozone <$X / text> <text string>
```

MPFLUSH

mpflush flushes all pending mob commands from its command queue

```
Usage:  mpflush
```

Note: This can be useful if you want a mob to immediately stop what it's doing and do something else, particularly if it has a lot of commands with delays.

MPFORCE

mpforce allows the mob to force another mob or a player to do something

```
Usage:  mpforce <$X / text> <command> [parameters]
```

Examples:

```
mpforce $n give 1000 coins $i
mpforce king order followers kill $n
mpforce ringbearer mpjunk all
mpforce $r sneeze
```

Note: The target must be in the same room as the mob, lower level, and visible.

MPGIVE

mpgive gives an item to victim

```
Usage:  mpgive <item> <name>
```

MPGOTO, MPGORANDOM

mpgoto lets a mob goto a different room

```
Usage:  mpgoto <roomVnum>
```

mpgorandom lets a mob goto a random room from a list

```
Usage:  mpgorandom <roomVnum> <roomVnum>
```

MPJUNK

mpjunk sacrifices an item, or 'all' items

```
Usage:  mpjunk <text / all>
```

Note: This works on worn gear as well.

MPKILL

mpkill attacks the specified target

```
Usage:  mpkill <$X / text>
```

Note: This fails if the mob is already fighting, or when trying to attack its master if the mob is charmed.

MPMLOAD, MPOLOAD

mpmload loads a mob with the specified vnum

```
Usage:  mpmload <vnum> <maxexist>
```

Note: If there are already maxexist number of this mob in the game, it will not load. It will always load if no maxexist is specified but be limited to 5 max.

mpoload loads an object with the specified vnum

```
Usage:  mpoload <vnum> <maxexist>
```

Note: If there are already maxexist number of this object in the game, it will not load. It will always load if no maxexist is specified.

Note: The object loads in the mob's inventory. To load an object in the room, have the mob silently drop <item / vnum>.

MPOPEN, MPCLOSE

mpopen opens a door as if an immortal (through wizlock)

mpclose close and wizlock a door/object

MPPURGE, MPVPURGE

mppurge purges an existing mob or object in the room, or 'all' mobs and items

```
Usage:  mppurge <text / all>
        Mppurge $i
```

Note: mppurge all does not purge the mob itself.

mpvpurge purges a mob or object in the room by item number

```
Usage:  mpvpurge <mob / obj> <#>
```

MPQSTART, MPQEND

mpqstart starts a quest

mpqend end the quest

MPRESTORE

mprestore restores the target

```
Usage:  mprestore <$X / text>
```

Examples:

```
         mprestore $n
         mprestore doopwinkle
```

Note: The target must be in the room with the mob.

MPROOM

mproom allows the mob to do commands similar to rset.

Example:

```
      mproom exit up connect 2 1200
      mproom exit w delete
```

MPSET

mpset sets a target's h/m/v, gold, or stats

```
Usage:  mpset <$X / text> <h/m/v/stat> <#>
```

Examples:

```
mpset $n hitpoints 6
mpset $i gold 1000
mpset SirAngel movepoints 0
mpset guardian dex 4
mpset $n str 29
```

Note: The target must be in the room with the mob.
Note: Set stats for PCs are temporary, until their next logout/in. Even so, this should be used cautiously, if at all.

mpset position <#>	
mpset defaultpos <#>	sets a mob's default position
mpset hitpoints <#>	sets hit points
mpset manapoints <#>	sets mana points
mpset movepoints <#>	sets movement points
mpset drunk <#>	sets drunk
mpset insane <#>	sets insane
mpset sex <#>	sets the sex.
mpset addact <#>	sets a specific action flag
mpset remact <#>	removes a specific action flag
mpset addaff <#>	sets a specific affect
mpset remaff <#>	removes a specific affect
mpset str <#>	sets strength
mpset dex <#>	sets dexterity
mpset con <#>	sets constitution
mpset int <#>	sets intelligence
mpset wis <#>	sets wisdom
mpset cha <#>	sets charisma
mpset luck <#>	sets luck
mpset gold <#>	sets gold

```
Usage:  mpset position <#>
        mpset defaultpos <#>                          Note:  Numeric only.
```

(Special note: You must use the position values from OLC, or from the Builder's Handbook v2.0 or later, (or the latest version of the handbook, when this reference becomes obsolete or dated).

MPSETSKILL

mpsetskill enables specific skills/spells you want the mob to be able to use

Note: Mob disposition enables certain skills/spells for certain classes. This allows you to add almost any skill/spell to those, allows cross-class skills/spells, and allows mobs with no disposition set to use skills/spells that you choose.

Note: This can be done in any mobprogtype, but is typically done in a birthprog, so that the mud only executes this when the mob is loaded.

Note: If your mob has no mana, it won't cast assigned spells very efficiently.

```
Usage:  mpsetskill < 'skill name' > < % >
```

Note: 'skill name' identifies the skill/spell by text name, the apostrophes are needed. % is the mob's actual chance of casting/performing the spell/skill.

Note: The mobprogmob assigns skills/spells to itself. There is no other option.

Examples:

```
mpsetskill 'first aid' 92
mpsetskill 'sanctuary' 45
mpsetskill 'protection from death' 84
```

MPSTAT, OPSTAT

mpstat helps identify the mob and gives its basic condition. Shows mobprograms that are set.
Returns: programs set, name, v number, short description, hit, max hit, mana, max mana, stam, max stam, level, class, alignment, AC, gold, and experience.

opstat helps identify the object and gives its basic condition. Show oprogs that are set.
Returns: programs set, name, v number, and short description

MPTRANSFER

mptransfer transfers a specified player or mob to a specified room

```
Usage:  mptransfer <$X / text / mobVnum> <roomVnum | here>

Text can be a name, allplayers or group.
```

Note: mptransfer <text> means a one-word name, text strings are not supported.

Note: With <text> or <vnum>, the prog looks for a target in the room that matches, then looks for a match anywhere in the mud, starting with the most recently loaded mob. With a common name, (such as 'guard'), this could produce unpredictable results.

MPTRIGGER, OPTRIGGER

mptrigger this command triggers a particular mprog of another mob
optrigger this command triggers a particular oprog

```
Usage:  mptrigger <mobVnum> <prognumber>
```

Note: You can think of this as one mob forcing some other mob to do something very complicated.

Note: You can mptrigger a rand_prog 0 prog, and the mob will perform the checks and execute the actions listed in it. There would be no other way for that prog to be triggered, since it fires zero percent of the time. Progs that are only triggered are easier to spot while debugging if they fire at 0%.

Note: Mptrigger is very rarely the best way to accomplish something.

SPECIAL NOTE: Mptrigger also works as a godcommand, for debugging and testing purposes. (Same syntax.)

MPVAR

mpvar sets mprogvar to some value

For use with the ifcheck var

MPZRESET

mpzreset this command resets a particular zone

```
Usage:  mpzreset <zone #>
```

Note: This should not be used carelessly or on the zones of others.

SILENTLY

---

silently allows mobs to execute commands without sending text to the room

```
Usage:  silently <command> [parameters]
```

XIV QUICK REFERENCE

MPTRIGGERS

```
Act          reacts to action messages.
Allgreet     reacts when a PC or an NPC enters the mob's room.
Attack       reacts when a mob begins fighting a new target.
Birth        reacts when the mob is loaded into the mud.
Bribe        reacts when the mob is given coins.
Cast         reacts if a PC casts the assigned spell in the mob's room.
Command      reacts to commands typed by players.
Death        reacts when the mob dies.
Entry        reacts when the mob enters a room.
Fight        reacts during fights.
Give         reacts when the mob is given an item.
Greet        reacts when a PC enters the mob's room.
Hitpercent   reacts when below the assigned % of maxhits.
Pulse        reacts at an assigned pulse.
Random       reacts randomly, according to the assigned %.
Speech       reacts to speech messages.
Time         reacts at an assigned hour.
Position     Not implemented. Reacts to a mob position. Pos_prog.
Qcomplete    reacts when the quest is finished
```

OPTRIGGERS

```
Attack       triggers when the object is used to attack
Birth        triggers when the object is loaded into the mud
Command      see mprog help for command triggers (same syntax)
Drop         triggers when you drop the item
Fight        checks every combat pulse
Get          triggers when you get the object
Greet        triggers when someone enters the room
Pull         triggers when you pull/push/turn/press the item
Random       triggers at a random time
Remove       triggers when you remove the item
Sac          triggers when you sacrifice the object
Speech       see mprog help for speech triggers (same syntax)
Wear         triggers when you wear the item
Act          Not implemented. see mprog help for act triggers (same syntax)
Close           Not implemented. triggers when an object is closed. Close_prog
Damage       Not implemented. triggers when an object takes damage. Damage_prog
Enter        Not implemented. triggers when an object is entered. Enter_prog
Examine      Not implemented. triggers when look or examine the item. Exa_prog
```

```
Open        Not implemented. triggers when the object is opened. Open_prog
Repair      Not implemented. triggers when you repair the object. Repair_prog
Use         Not implemented. triggers when you use the item. Use_prog
```

IFCHECKS

rand	cansee	isnpc	ispc	isimmort
isgood	isneutral	isevil	isfighting	ischarmed
isfollowing	isgrouped	ismounted	isoutlaw	ispthief
race	sex	position	class	name
inroom	inzone	vnum	numfollowers	hitpercent
manapercent	movepercent	gold	qpoints	age
level	size	clannum	legendpoints	nobpoints
str	dex	con	wis	int
cha	luk	hp	mp	vp
maxhp	maxmp	maxvp	objtype	objval(0-5)
playerspresent	mobspresent	iszoneempty	mobhere	objhere
carriesobj	wearsobj	hasobj	timehour	timeday
timemonth	timeyear	actortarget($t)	mobtarget($t)	cmd($n)
cmdarg				

MPCOMMANDS

mpareadamage	mpasound	mpat	mpcoins	mpdamage
mpdamagearound	mpdelay	mpdoor	mpecho	mpechoaround
mpechoat	mpechozone	mpforce	mpflush	mpgive
mpgorandom	mpgoto	mpjunk	mpkill	mpmload
mpoload	mpopen	mppurge	mpqend	mpqstart
mprestore	mproom	mpset	mpsetskill	mpstat
mptransfer	mptrigger	mpvar	mpvpurge	mpzreset
opstat	optrigger	silently		

MPVARIABLES

```
 $n    person causing the trigger
 $i    mob doing the mobprog
 $r    random PC in the room
 $t    target PC
 $o    object causing the trigger
```

```
Operators used in ifchecks for STRINGS:
  ==   checks for an exact string match
  !=   checks for not equals, or no match
   /   checks to see if the specified string is contained in the target string
  !/   returns true if there is no match in the target string

Operators for MATH comparisons:
  ==   equality
  !=   not equals
  <=   less than or equal to
  >=   greater than or equal to
  >    greater than
  <    less than
  &    bitwise AND
  |    bitwise OR
```

MPEDITING

```
Editing the HEADER
list     list all mprogs for the mob you are medit-ing
save     save current mprog changes to .mob file
done     exit the mpeditor and the meditor without saving
edit     edit the command list of an existing mprog
delete   delete a mprog from the list
copy     copy a mobprog from another mob
insert   insert a mprog in the list
add      add a mprog to the end of the mprog list
header   replace a mobprog header completely

Editing the BODY
 /?      on-line editor help (inside the editor)
```

XV CREDITS

Upgrades and Improvements

DATE	VERSION	EDITED BY	DATE	VERSION	EDITED BY
Sept 1, 2019	1 (Kallisti)	Ivy			

This version of the Mprog Handbook is created for Kallisti MUD/Legends of Kallisti, version 5.0.

If there are any questions about this document, please email the Admin list: imms@kallistimud.com or contact the administration.


```
html special chars - php
cross site scripting
```