# Three's Dictionary and Encyclopedia

These docs were written with the help of Three, who wrote probably the best library I've seen. Contained herein are her docs, and some examples that I've written. -- *Chrys*

---

## Table of Contents



---

## Three's Unabridged Dictionary of Commands

DROP

`drop <object>`

Drops <object>. Dropping a thing in the temple sacrifices it. Otherwise, a dropped thing is relocated to the current room, unless its STICKY flag is set, or the room has a DROP-TO. Unlinked exits can only be dropped in rooms you control. 'throw' is the same as 'drop'.

EXAMINE

`examine [object]`

Displays all available information about <object>. <object> can be specified as <name> or #<number>, or as 'me' or 'here'. You must control the object to examine it. Wizards can examine objects in other rooms using #<number> or *<player>

GET

`get <object>`

Picks up <object>. <object> can be a thing or an unlinked exit. 'take' is the same as 'get'.

GIVE

`give <player>=<pennies>`

Gives player the specified number of pennies. The only thing you can give is pennies. You can't give someone pennies if their new total would be greater than 10000. Wizards can give as many pennies as they wish, even negative amounts, without affecting their own supply, and can give pennies to things to change their sacrifice values.

GOTO

`go[to] <direction>`
`go[to] home`

Goes in the specified direction. 'go home' is a special command that returns you to your starting location. The word 'go' may be omitted. 'move' is the same as 'go'.

GRIPE

`gripe <message>`

Sends <message> to the system maintainer.

HELP

`help`
This displays a short help message.

INVENTORY
`inventory`
Lists what you are carrying. This can usually be abbreviated to inv.

KILL
`kill <player> [=<cost>]`
Attempts to kill the specified player. Killing costs either <cost> or 10 pennies, whichever is greater. The probability of success is <cost> percent. Spending 100 pennies always works (except against wizards, who can never be killed). Players cannot be killed in rooms which have been set HAVEN.

LOOK
`look [object]`
Displays the description of <object>, or the room you're in if you don't specify one. <object> can be a thing, player, exit, or room, specified as <name> or #<number> or 'me' or 'here'. 'read' is the same as 'look'. Wizards can look at objects in other rooms using #<number> or *<player>

MOVE
See GO.

NEWS
`news`
Displays the current news file for the game. Must be typed in full.

PAGE
`page <player>`
This tells a player that you are looking for them. They will get a message telling them your name and location. This costs 1 penny. If a player is set HAVEN, you cannot page them, and they will not be notified that you tried.

QUIT
`QUIT`
Log out and leave the game. Must be in all capitals.

READ
See LOOK.

ROB
`rob <player>`
Attempts to steal one penny from <player> The only thing you can rob are pennies.

SAY
`say <message>`
Says <message> out loud. You can also use '"<message>'. Another command is ':<message>'. This is used for actions, ex. if your name was Igor, and you typed ':falls down.', everyone would see "Igor falls down." See also WHISPER.

SCORE
`score`
Displays how many pennies you are carrying.

TAKE
See GET.

THROW
See DROP.

WHISPER
`whisper <player>=<message>`

Whispers the message to the named person, if they are in the same room as you. No one else can see the message. Wizards can whisper *<player>=<message> to whisper to players in other rooms.

WHO

`WHO [<player>]`

List the name of every player currently logged in, and how long they have been inactive. If given a player name, it displays only that name and idle time. Must be in all capitals. There are two player flags that pertain to the formatting of WHO, REVERSE_WHO and TABULAR_WHO. See also FLAGS.

@BOOT

`@boot <player>`

Disconnects the player from the game. Only Wizards can use this command.

@CHOWN

`@chown <object>=<player>`

Changes the ownership of <object> to <player> Only wizards may use this command. Players can't be @chowned; they always own themselves. See also the new @CHOWN.

@CREATE

`@create <name> [=<cost>]`

Creates a thing with the specified name. Creation costs either <cost> pennies or 10 pennies, whichever is greater. The value of a thing is proportional to its cost. To be exact, value= (cost/5)-1.

@DESCRIBE

`@describe <object> [=<description>]`

<object> can be a thing, player, exit, or room, specified as <name> or #<number> or 'me' or 'here'. This sets the description a player sees when they use the command 'look <object>'. Without a description argument, it clears the message. It can be abbreviated @desc.

@DIG

`@dig <name>`

Creates a new room with the specified name and displays its number. This costs 10 pennies.

@DUMP

`@dump`

Only wizards may use this command. Saves the database from memory to disk. Automatically occurs every hour, and when @shutdown is used.

@FAIL

`@fail <object> [=<message>]`

<object> can be a thing, player, exit, or room, specified as <name> or #<number> or 'me' or 'here'. Sets the fail message for <object>. The message is displayed when a player fails to use <object> Without a message argument, it clears the message. See also @OFAIL.

@FIND

`@find [name]`

Displays the name and number of every room, thing, or player that you control whose name matches <name> Because the command is computationally expensive, this costs 1 penny.

@FORCE

`@force <player>=<command>`

Only wizards may use this command. Forces the game to act as though <player> had entered <command>

@LINK

`@link <object>=<number>`

```
@link <object>=here
@link <dir>|<room>=home
```
Links <object> to room specified by <number>. For things and players, sets the home room. For rooms, sets the drop-to room. For exits, sets the target room; exits must be unlinked, and you must control the target room unless its LINK_OK flag is set. Linking an exit costs 1 penny. If the exit was owned by someone else, the former owner is reimbursed 1 penny. Wizards can @link objects in other rooms using #<number> or *<player>

@LOCK
```
@lock <object>=<key>
```
Locks <object> to a specific key(s). <object> can be specified as <name> or #<number>, or as 'me' or 'here'. Boolean expressions are allowed, using '&' (and), '|' (or), '!' (not), and parentheses ('(' and ')') for grouping. To lock to a player, prefix their name with '*' (ex. '*Igor'). See the examples section.

@NAME
```
@name <object>=<new name> [<password>]
```
Changes the name of <object>. <object> can be a thing, player, exit, or room, specified as <name> or #<number> or 'me' or 'here'. For a player, it requires the player's password.

@NEWPASSWORD
```
@newpassword <player> [=<password>]
```
Only wizards may use this command. Changes <player>'s password, informing <player> that you changed it. Must be typed in full.

@OFAIL
```
@ofail <object> [=<message>]
```
The @ofail message, prefixed by the player's name, is shown to others when the player fails to use <object>. Without a message argument, it clears the message. <object> can be specified as <name> or #<number>, or as 'me' or 'here'. See also @FAIL.

@OPEN
```
@open <dir>[;<other dir>]* [=<number>]
```
Creates an exit in the specified direction(s). If <number> is specified, it is linked to that room. Otherwise, it is created unlinked. You or anyone else may use the @link command to specify where the unlinked exit leads. Opening an exit costs 1 penny. If you specify <number>, linking costs 1 more penny.

@OSUCCESS
```
@osuccess <object> [=<message>]
```
The @osuccess message, prefixed by the player's name, is shown to others when the player successfully uses <object>. Without a message argument, it clears the @osuccess message. It can be abbreviated @osucc. <object> can be specified as <name> or #<number>, or as 'me' or 'here'. See also @SUCCESS.

@PASSWORD
```
@password <old password>=<new password>
```
This changes your password.

@SET
```
@set <object>=<flag>
@set <object>=!<flag>
```
Sets (or, with '!', unsets) <flag> on <object>. See FLAGS in the encyclopedia.

@SHUTDOWN
```
@shutdown
```
Only wizards may use this command. Shuts down the game. Must be typed in full.

@STATS

```
@stats [player]
```

Display the number of objects in the game. For wizards, also lists a breakdown by object types. Wizards can supply a player name to count only objects owned by that player.

@SUCCESS

```
@success <object> [=<message>]
```

Sets the success message for <object>. The message is displayed when a player successfully uses <object>. Without a message argument, it clears the message. It can be abbreviated @succ. <object> can be specified as <name> or #<number>, or as 'me' or 'here'. See also @OSUCCESS.

@TELEPORT

```
@teleport [<object>=] <room>
```

Teleports <object> to <room> <object> must be a thing. (Wizards can also teleport players.) You must be able to link to the destination, and either control the object or its current location. You can only teleport objects into a room, not into someone's inventory. If the target room has a drop-to, <object> will go to the drop-to room instead. Wizards can teleport things into players' inventories.

@TOAD

```
@toad <player>
```

Only wizards may use this command. Turns the player into a slimy toad, destroying their character. Must be typed in full.

@UNLINK

```
@unlink <dir>
@unlink here
```

Removes the link on the exit in the specified direction, or removes the drop-to on the room. Unlinked exits may be picked up and dropped elsewhere. Be careful, anyone can relink an unlinked exit, becoming its new owner (but you will be reimbursed your 1 penny). See @LINK.

@UNLOCK

```
@unlock <object>
```

Removes the lock on <object>. See @LOCK.

@WALL

```
@wall <message>
```

Only wizards may use this command. Shouts something to every player connected. Must be typed in full.

---

# Three's Encyclopedia of the TinyWorld

BEING KILLED

Getting killed is no big deal. If you are killed, you return to your home, and all things you carry return to their homes. You also collect 50 pennies in insurance money (unless you have >= 10000 pennies). See MONEY.

BOGUS COMMANDS

Bogus commands can be made using exits. For example, to make a 'sit' command, one could "@open sit", then "@link sit=here" (because unlinked exits can be stolen), "@lock sit=me&!me" (impossible to be both at once, therefore always fails), and "@fail sit=You sit on

the chair."; "@ofail=sits on the chair.". Since nobody can go through it, it always fails. The @fail message is displayed to the player, and the @ofail message (preceded by the player's name) to everyone else.

CONTROL

There are 3 rules to controlling objects:

1. You control anything you own.
2. A wizard controls everything.
3. Anybody controls an unlinked exit, even if it is locked.

Builders should beware of 3, lest their exits be linked or stolen.

COSTS

Costs:

- kill: 10p (or more, up to 100p).
- page: 1p.
- @dig: 10p.
- @create: 10p (or more, up to 505p), sacrifice value=(cost/5)-1.
- @find: 1p.
- @link: 1p (if you didn't already own it, +1p to the previous owner).
- @open: 1p (2p if linked at the same time).

Wizards don't need money to do anything.

DESTROYING

Nothing can be destroyed. However, everything can be reused. You can give an object a new name with @name, redescribe it with @describe, and set new success and fail messages for it. Exits can be @unlink'd and picked up and dropped elsewhere, so you can pick up an extra exit and use it in another room.

DROP-TOs

When the @link command is used on a room, it sets a drop-to location. Any object dropped in the room (if it isn't STICKY) will go to that location. If the room is STICKY, the drop-to will be delayed until the last person in the room has left.

FAILURE

You fail to use a thing when you cannot take it (because it's lock fails). You fail to use an exit when you cannot go through it (because it's unlinked or locked). You fail to use a person when you fail to rob them. You fail to use a room when you fail to look around (because it's locked). See STRINGS, and in the dictionary, @FAIL and @OFAIL.

FLAGS

The flags are displayed as letters following an object's ID number. Flags are set with the @set command. The flags are:

- A(bode)
- [B(uilder)]
- C(hown_OK)
- D(ark)
- H(aven)
- J(ump_OK)
- L(ink_OK)
- S(ticky)
- T(emple)

- W(izard)
- and the gender flags
  - M(ale)
  - F(emale)
  - N(euter)

The WHO list also uses REVERSE_WHO and TABULAR_WHO, but they do not show up in the ID number. Some systems also use B(uilder). See TYPES, GENDER, and individual flag names.

WIZARD

If a person is WIZARD, they are a wizard, unkillable, subject to fewer restrictions, and able to use wizard commands. It is only meaningful for players. Only another wizard can set this flag. In general, WIZARDs can do anything using #<number> or *<player> Only player #1 can set and unset the WIZARD flag of other players. No WIZARD can turn their own WIZARD flag off.

STICKY

If a thing is STICKY, it goes HOME when dropped. If a room is STICKY, its drop-to is delayed until the last person leaves Only meaningful for things and rooms.

LINK_OK

If a room is LINK_OK, anyone can link exits to it (but still not from it). It has no meaning for people, things, or exits. See @LINK in the dictionary.

DARK

If a room is DARK, then when people besides the owner 'look' there, they only see things they own. If a thing or player is DARK, then "look" does not list that object in the room's Contents:. Only wizards can set players or things dark.

TEMPLE

If a room is TEMPLE, you can sacrifice things for pennies by dropping them there. It has no meaning for players, things, or exits. Only wizards can set this flag.

GENDER

```
@set me=unassigned|male|female|neuter
```

Default unassigned. If a player's gender is set, %-substitutions will use the appropriate pronoun for that player. Only meaningful for players. See SUBSTITUTIONS.

HAVEN

```
@set here=haven
@set me=haven
```

If a room is HAVEN, you cannot kill in that room. If a player is set HAVEN, he cannot be paged.

ABODE

```
@set here=abode
```

If a room is set ABODE, players can set their homes there, and can set the homes of objects there. (LINK_OK is now used only for exits, and ABODE is for players and objects.)

REVERSE_WHO

```
@set me=reverse_who
```

If this flag is set, the WHO list will be displayed in reverse order, with newest players listed last. This flag can only be set on players.

TABULAR_WHO

```
@set me=tabular_who
```

If this flag is set, the WHO list will be displayed in a tabular form. This flag can only be set on

players.

GOAL

There is no ultimate goal to this game, except to have fun. There are puzzles to solve, scenery to visit, and people to meet. There are no winners or losers, only fellow players. Enjoy.

HERE

The word 'here' refers to the room you are in. For example, to rename the room you're in (if you control it), you could enter "@name here=<new name>".

HOMES

Every thing or player has a home. This is where things go when sacrificed, players when they go home, or things with the STICKY flag set go when dropped. Homes are set with the @link command. A thing's home defaults to the room where it was created, if you control that room, or your home. You can link an exit to send players home (with their inventory) by "@link <dir>=home". Drop-tos can also be set to 'home'. See @LINK.

LINKING

You can link to a room if you control it, or if it is set LINK_OK or ABODE. Being able to link means you can set the homes of objects or yourself to that room if it is set ABODE, and can set the destination of exits to that room if it is LINK_OK. See @LINK.

ME

The word 'me' refers to yourself. Some things to do when starting out:

1. Give yourself a description with "@describe me=<description>", then look at yourself with "look me".
2. Prevent anyone else from robbing you with "@lock me=me".
3. Set your gender, if you wish it known, with "@set me=male" or "@set me=female" (or "@set me=neuter" to be an 'it').

MONEY

Building and some other actions cost money. How to get money:

1. Find pennies.
2. Sacrifice (drop) things in the temple.
3. Get killed.
4. Be given money.
5. Rob someone.

Once you reach 10000 pennies, it becomes difficult to acquire more. See COSTS and SACRIFICING. Wizards don't need money to do anything.

ROBBERY

When you rob someone, you succeed or fail to use them (See SUCCESS and FAILURE). You can protect yourself from being robbed by entering "@lock me=me" (See ME, and in the dictionary, @LOCK). If you lock yourself to yourself, you can rob yourself and set off your @success and @osuccess messages. Try entering "@osucc me=is goofy." and robbing yourself in a crowd. See ROB in the dictionary.

SACRIFICING

You sacrifice a thing by dropping it in the temple. Sacrificing an object gives you the value of an object. You can't sacrifice something you own. If you have >= 10000 pennies, all sacrifices are worth only 1 penny. The sacrifice value of a thing is set at creation by "@create frob=cost", by the formula value=(cost/5)-1. Only a wizard can change the value of an object, once created.

STRINGS

Objects have 6 strings:

1. A name.
2. A description.
3. A success message (seen by the player).
4. A fail message (seen by the player).
5. An osuccess message (seen by others).
6. An ofail message (seen by others).

SUBSTITUTIONS

@osuccess and @ofail messages may contain %-substitutions, which evaluate to gender-specific pronouns if the player's gender is set. They are:

- %s (subjective) = Name, he, she, it.
- %o (objective) = Name, him, her, it.
- %p (possessive) = Name's, his, her, its.
- %n (player's name) = Name.

Capitalized pronouns are also available with %S, %O, and %P. If you need a '%', use '%%'. Example: '@ofail teapot=burns %p hand on the hot teapot.' See GENDER.

SUCCESS

You successfully use an object when you take it. You use an exit successfully when you go through it. You successfully use a person successfully when you successfully rob them. You successfully use a room when you look around. See STRINGS, and in the dictionary, @SUCCESS and @OSUCCESS.

TYPES OF OBJECTS

There are 4 types of objects:

1. *Things* are inanimate objects that can be carried.
2. *Players* are animate objects that can move and carry.
3. *Exits* are the means by which objects move.
4. *Rooms* are locations that contain objects and linked exits.

The first letter following an object's ID number indicates the type:

- P(layer)
- E(xit)
- R(oom)
- (otherwise) thing.

---

# Examples

Igor is a new player. He sets his description by typing:

```
@desc me=Igor is a ferret with an evil glint in his eye.
```

He has guarded himself from being robbed, and set some fail messages on himself that people will see when they try to rob him. He typed:

```
@lock me=me
@fail me=Igor chomps you on the knee with his little sharp teeth.
@ofail me=howls in pain as Igor bites them.
```

Now, here is what happens if Murf tries to rob Igor:

```
Murf types:   rob igor
Murf sees:    Igor chomps you on the knee with his little sharp teeth.
all else see: Murf howls in pain as Igor bites them.
```

'them' as a pronoun isn't to specific, and so Igor should do this:

```
@ofail me=howls in pain as Igor bites %o.
```

So if Murf robs Igor, this is what everyone else will see:

```
Murf howls in pain as Igor bites him.
```

This is assuming that Murf did a '@set me=male'. If not, it would have printed:

```
Murf howls in pain as Igor bites Murf.
```

Igor wants to set a message that he will use a lot, so he sets his @osucc:

```
@osucc me=runs around the room nipping at everyone's heels.
```

Now, if he wants to display that message:

```
Igor types:   rob me
Igor sees:    You stole a penny.
              Igor stole one of your pennies!
all else see: Igor runs around the room nipping at everyone's heels.
```

Igor wants to make an object called 'Ferret chow'. He types:

```
@create Ferret Chow
@desc Ferret Chow=This is a big bag full of yummy ferret chow.
@succ Ferret Chow=You tear into the end of the bag, stuffing yourself.
@osucc Ferret Chow=tears into the Ferret Chow bag, eating greedily.
```

Now Igor decides that he wants to be the only one who can pick up the bag.

```
@lock Ferret Chow=me
@fail Ferret Chow=It's icky Ferret Chow. It would probably taste gross.
@ofail Ferret Chow=decides Ferret Chow is icky.
```

If Igor picks up the bag:

```
Igor types:   get Ferret Chow
Igor sees:    You tear into the end of the bag, stuffing yourself.
all else see: Igor tears into the Ferret Chow bag, eating greedily.
```

Igor is now carrying the bag. He must drop it if he wants to see the messages again. If Murf picks up the bag:

```
Murf types:   get Ferret Chow
Murf sees:    It's icky Ferret Chow. It would probably taste gross.
all else see: Murf decides Ferret Chow is icky.
```

Because the bag was locked to Igor, Murf cannot get the bag.

Igor wants to build a few rooms. He can only build off of a place where he can get a link. He needs to ask around to find one of these if he is just starting to build. Murf is going to give Igor a link named 'n;north'. That means that both 'n' and 'north' activate that exit. Igor digs a room, and links the exit to it. He types:

```
@dig Igor's House
```

At this point, the program will respond "`Igor's House created with room number xxxx`". We'll pretend it gave the room number as 1234.

```
@link n;north=1234
```

The program will respond with "`Linked`". Now Igor sets a few messages on the exit. He types:

```
@desc n=North is Igor's House.
@succ n=You crawl into Igor's House.
@osucc n=crawls into Igor's House.
```

These messages work just the same way they work on object, like the Ferret Chow. Next, Igor goes in the room, and creates an out exit. Murf has been nice enough to not only give Igor the n;north exit, but to set his room to L(ink_ok). Murf's room number is 623. Igor types 'n' or 'north' to go in the room, then types:

```
@open out;back;s;south=623
```

The program will respond with "`Opened. Trying to link... Linked`". Igor now has a south exit back to Murf's room. Murf can now set his room to !link_ok, so no one else can link to it. Igor puts some messages on the south link as well. He decides he wants to describe the room, so he types:

```
@desc here=This is Igor's home. It is a small room, lined with paper
shreds. Over in the corner is a small hole.
```

Now Igor wants to dig a small room that the hole connects to. He types:

```
@dig Igor's Hidden Room
```

The program tells him that the room is number 1250. He then types:

```
@open hole=1250
@lock hole=me
@desc hole=This is a small hole, just the size of Igor.
@fail hole=You can't fit.
@ofail hole=can't fit through the hole.
@succ hole=You slip into the hole.
@osucc hole=slips into the hole.
```

This creates and links the exit called 'hole' to Igor's Hidden Room. He locks the exit to him, so only he can go through the exit. When he uses the exit, the success and osuccess messages will be displayed. When someone else tries to use the exit, the fail and ofail messages will be displayed. Since Igor owned the room that he was linking from, he had to use @open to create the link first. He now types

'hole' to go in the room, and types '@open out=1234' to create and link an exit called 'out' that leads to his House. If Igor wants everyone BUT Murf to be able to go 'hole', he types:

```
@lock hole=!*murf
```

This locks the hole against the player Murf. If he wants a person to be able to go through 'hole' only if they have the bag of Ferret Chow, he types:

```
@lock hole=Ferret Chow
```

If he wants himself to be able to go in the hole, even if he doesn't have the Ferret Chow, he types:

```
@lock hole=Ferret Chow | me
```

If he wants to lock everyone out except for himself and Murf if Murf has the bag of Ferret Chow, he types:

```
@lock hole=(*murf & Ferret Chow) | me
```

You can get more and more complicated with locks this way. Igor is done building his home, and wants to set his home to it, so when he types 'home' he will go there instead of Limbo(#0RDLA). He goes in his house, and types:

```
@link me=here
```

The program will respond with "`Home set`". Now Igor can go 'home', and QUIT and not worry about his inactive body cluttering up the landscape.

Creating whole houses and adventures can be easy if you understand the way the @ commands work. When you build a room, you should have a very thorough description. Every thing listed in the description should be given a bogus exit (see entry) to detail the place. For example, here is the description of a room built by Three.

```
Three's flat(#5400)
Red wall-to-wall carpeting covers the floor. A cushy brown leather
couch sits across from a wide-screen TV with a VCR and video disc
player stacked on top.  Escher prints hang on the walls, hilited by
track lighting. Papers protrude from a roll-top desk to one side,
adjoining an imposing stereo whose controls rival those of 747 cockpits.
The kitchen lies north, the foyer south, and the bedroom beyond a
short passage east.
Contents:
Flitterby Award for Comprehensive Building
```

Now, you noticed the desk in the room. A 'look desk' will show:

```
Every drawer and cubby is overflowing with papers, envelopes, flyers,
leaflets, folders, booklets, binders, quick reference cards, and
other paper products. A Compaq luggable sits in a small canyon of
paper. Atop the desk stands a framed photo. Under the desk sits a
back stool.
```

Now, since this was done with a exit to create a bogus command, you might try going through the

exit, so you will get the fail message. A 'desk' will show:

```
You rummage thru the desk drawers, finding nothing of interest.
```

Here is an examine of the bogus command, to show you how it was done:

```
desk(#5395E)
Owner: Three  Key: Three(#5370PTF)&!Three(#5370PTF) Pennies: 0
Every drawer and cubby is overflowing with papers, envelopes, flyers,
leaflets, folders, booklets, binders, quick reference cards, and
other paper products. A Compaq luggable sits in a small canyon of
paper. Atop the desk stands a framed photo. Under the desk sits a
back stool.
Fail: You rummage thru the desk drawers, finding nothing of interest.
Ofail: rummages thru the desk drawers.
Destination: Three's flat(#5400R)
```

In this way, a highly detailed room can be built, and greatly increase the atmosphere of the place. Take a walk around and look at the place first, before deciding to build. Then sit down and think carefully about what you want to build. Careful planning has made several very interesting places.

---

# TinyMUCK mods

This document describes the differences between TinyMUCK and TinyMUD v1.5.2. It is *not* an introduction to TinyMUD, nor a tutorial on TinyMUCK. It is a reference guide to the changes which have been made.

The main additions in TinyMUCK are object-local and player-local commands, multi-destination links and player @chown.

## Object-Local and Player-local Commands

In order to allow players to do interesting things with objects, TinyMUCK adds the ability to attach 'actions' to them. Actions are much like 'exits' on a room, and in fact use the same type in the database (TYPE_EXIT). In order to use these actions, an object must either be in the room, or in the player's possession.

Player-local commands can only be used by the player who owns them.

The two new commands, @action and @attach, are used as follows:

@ACTION

```
@action <action-name>=<source-object>
```

@action creates a new action named <action-name>, and attaches it to <source-object>. <source-object> can also be 'here' or 'me'.

@ATTACH

```
@attach <existing-action-name>=<source-object>
```

@attach re-attaches an existing action to a new source object.

To get an action to do something, you link it like a regular exit.

> Notes: Exits and actions are interchangeable, so @action foo=here is equivalent to @open foo. You can no longer 'get' and 'drop' actions. Use @attach foo=me and @attach foo=here, respectively.

## TinyMUCK Links

The @link command in TinyMUCK works a little differently than in TinyMUD.

@LINK

> `@link <action-name>=<dest1> <dest2> <dest3> ...`
> Where <action-name> can also be an exit name, and <destn> are the numbers of the objects to link to.

If you @link an action to an object, using that action will summon the indicated object. If someone else is holding the object, it will immediately disappear from their inventory.

In the case of an action attached to an object, an object-summon will also send the source object home. To prevent this, set the action STICKY. Room destinations on things and players teleport the player to that room.

If TinyMUCK is compiled with `#define TELEPORT_TO_PLAYERS`, <destn> can also be another player, provided that they are LINK_OK. Using this action will cause you to teleport to that player's location. To provide some degree of privacy, that player must also be JUMP_OK for the action to work. The default for new players is !JUMP_OK. Summary: player must be LINK_OK at link-time, and JUMP_OK at use-time.

You can also link an action to a room you control (like a regular exit), or another action. The latter I have termed a *meta-link*. Note that meta-links override the locks on all actions below the initial one. A recursive check is performed to make sure that an action destination does not cause a loop. Room or player destinations in actions which are linked to will be ignored in a meta-link.

Meta-links can be used, for example, to restore the objects in an adventure to their starting places, or to move objects around in rooms other than the one that the player is in.

## The JUMP_OK Flag

In order to restrict the use of teleport-to-player and teleport- to-room destinations on movable things (like players and objects), TinyMUCK has the JUMP_OK flag. If you create an action on an object or yourself, it will always work in rooms that you own.

You can only use these actions in other rooms if the other rooms are JUMP_OK.

When new rooms are created, their JUMP_OK flag is set to whatever the player's JUMP_OK flag is. This allows the puzzle builder (for example) to create a set of rooms which are !JUMP_OK, or a social-area-builder to create a set of JUMP_OK rooms.

JUMP_OK also serves the purpose (as mentioned above) of stopping players from teleporting to you.

### Player @chown

In TinyMUD, the @chown command allows Wizards to change ownership of objects. In TinyMUCK, the @chown can be used by players to transfer ownership of other players objects to themselves. The object to be transferred must be a THING, it have the CHOWN_OK flag set, and it must be in the player's inventory.

@CHOWN
: `@chown <object>[=<player>]`
Changes the ownership of <object> to the player, or to the specified <player>. Only wizards may change the ownership of an object to someone else. Players can't be @chowned; they always own themselves.

CHOWN_OK
: If an object is CHOWN_OK, anyone can @chown it to themself. See @CHOWN in the dictionary.

This allows two things:

1. Players can give each other gifts.
2. Players can trash old objects, and put them in public-accessible places for re-use (until real recycling is implemented).

For example, a player might create a note, lock it to the appropriate player, do @set note=CHOWN_OK, and then leave it in their mailbox. The receiving player could then @chown note, transferring ownership to themselves, and use it to reply, or leave it CHOWN_OK and drop it to a junkpile someplace.

The `#ifdef PLAYER_CHOWN` must be used when compiling to allow this feature to be implemented.

### Miscellaneous

`#ifdef VERBOSE_EXAMINE` This causes the examine command to print a verbose list of the flags on an object (for those who can't remember what PWDLSJ stands for).

---

## My own mods

These two fields are similar to @succ, @osucc, @fail, and @ofail. When applied to things, it gives the message when a person drops something. When applied to links, the @drop tells the player entering, and the @odrop tells others that he has entered. When applied to people, the @odrop tells others how you died, and the @drop tells the person who killed you how you died.

@DROP
: `@drop <object> [=<message>]`
Sets the drop message for <object>. The message is displayed when a player drops <object>.

Without a message argument, it clears the message. <object> can be specified as <name> or #<number>, or as 'me' or 'here'. See also @ODROP.

@ODROP

`@odrop <object> [=<message>]`

The @odrop message, prefixed by the player's name, is shown to others when the player drops <object>. Without a message argument, it clears the @odrop message. <object> can be specified as <name> or #<number>, or as 'me' or 'here'. See also @DROP.

For instance. A thing called "desk"

```
@drop desk=You drop the desk with a sigh.
@odrop desk=drops the desk with a sigh.
```

When you "`drop desk`":

```
you see:    You drop the desk with a sigh.
others see: Whoever drops the desk with a sigh.
```

A link/exit.

```
@drop out=You slide down the chute and end up in a dark place.
@odrop out=slides down the shute and enters from above, looking
surprised.
```

When you enter the room:

```
you see:    You slide down the chute and end up in a dark place.
others see: Whoever slides down the shute and enters from above,
            looking surprised.
```

For people.

```
@drop me=Whoever drops over, flops, and dies.
@odrop me=Whoever drops over, flops, and dies, with blood sprouting from
all over.
```

When a Wizard types, "`kill Whoever=100`":

```
He sees:      Whoever drops over, flops, and dies.
Everone else: Wizard killed Whoever!  Whoever drops over, flops,
              and dies, with blood sprouting from all over.
[You yourself will see the usual messages...]
```

## 4/21/90 --- Entering beta-test for property lists.

Property lists (p-lists) are an extension to objects, players and rooms for TinyMUCK. Each object will now have a p-list that can be checked against in locks and the like. p-lists are basically string pairs, and can be anything. Properties stay on until removed, and an object can have as many properties as it likes.

The GENDER flag will now be a part of p-lists instead of being part of the flags.

Syntax for setting properties:

@SET

`@set <thing>=[property:[type]]`
Set the *property* of <thing> to *type*. Without a *type* argument *property* is removed from <thing>. Without the *property* argument, **all** properties are removed from <thing>.

Examples:

```
@set me=sex:male
@set me=hair:red
```

Locks use the same syntax

```
@lock out=me | sex:male
```

Locks check the person's properties and the properties of everything she's carrying at the moment.

## Page can now include a message

PAGE

`page <player>[=message]`
Page the named <player>. If a *message* is present, they will receive that information along with any other page output.

The ABODE and HAVEN flags are optional and may change from game to game depending on the player.

Abode is independent from link, so you can set home to here even though it's not link_ok.

## Examine

Examine now shows you the owner of a place if you don't control it.

---

Return to the TinyMUCK Page

---

*Page created by Bolo, and maintained by Tugrik d'Itichi.*
*Comments/Questions/Flames to: FMPages@furry.com*