# Appendix A – Summary of Commands

This appendix details various commands that are available on MOOs. It includes commands available on the player classes and feature objects provided with LambdaCore, plus others such as those that are available on `$room`.

A note about the syntax specifications: Text enclosed in angle brackets "`<>`" must be specified at the time it is typed in, e.g. instead of "`<name>`" you must supply an actual name, without the angle brackets. Text enclosed in square brackets "`[]`" is optional to a command. If you include it, don't type the square brackets. A vertical bar "`|`" separates either-or cases. For example, `@notedit <object | object>.<property>` means you must either supply an object, or an object and a property name separated by a period (notice that the period isn't in the angle brackets). Verb names with an asterisk in them (*) may be abbreviated to the part that comes before the asterisk.

The form of each entry is as follows

<command> <arguments>

<location/source>

<Usage notes>

Commands are listed alphabetically, ignoring punctuation. Object numbers for command locations will vary from MOO to MOO, so I only give them for commands that are LambdaMOO-specific. "LambdaMOO" is abbreviated as "LM".

**`@abort`**
Embedded with certain `$command_utils` verbs.

When prompted for text input or a yes-or-no answer, you can type `@abort` to exit a task entirely.

**`@abort`** `[<object>[:<verb>]]`
LM player class `#7069`

Causes queued tasks to be aborted. You may specify either all tasks associated with a particular object, or only tasks associated with a particular verb on an object.

**`@addalias`** `<alias>[,...,<alias>] to <object>`
**`@addalias`** `<alias>[,...,<alias>] to <object>:<verb-name>`
**`@addalias#`** `<alias>[,...,<alias>] to <object>:<verb-number>`
`$player`

The first form adds one or more aliases to an existing object. Note, player aliases may not have spaces in them.) The second form adds one or more aliases to an existing verb on an object. The third form unambiguously adds one or more aliases to a particular verb on an object when there are two or more verbs with the same name. (See also `@verbs` and `@list#`.)

**@adddict** <one or more words>
$frand_class
Adds a word or words to $spell, if present. (Many MOOs have recycled $spell because it is cumbersomely large.) Only wizards and players in $spell.trusted may use this verb.

**@addfeature** <feature object>
**@add-feature** <feature object>
$player
Adds a feature object to your list of features. See also @rmfeature. A feature is an object that provides additional commands that you can use.

**@addlag**
LM feature object #26787
Turns off the lag reduction FO and any features similar to it. (Re-)enables @gag, @check-full, and any other commands that utilize a player's :tell verb to filter and/or otherwise pre-process text before it is displayed to your screen.

**@add-notify** me to <player>
**@add-notify** <requestor> to me
$mail_recipient_class
This verb, used cooperatively between two players, allows one player to be notified when the other player receives MOOmail. The first form sends a MOOmail message to <player> indicating that one wishes to receive notification. The second form adds the requestor to a person's .mail_notify property. This might be useful if several players shared a group character (such as LambdaMOO's Grand_Master, for example) and wanted to be notified if the group character received mail, so as to be able to respond to it in a timely manner.

**@add-owned** <object>
$builder
Adds an object to your .owned_objects property in the highly unlikely event that it wasn't added upon creation of the object.

**@addr*oom** [<name>] [<place>]
**@addr*oom** [<place>] [<name>]
$frand_class
Adds a room to the list of rooms you know "by name". See also @rooms. If <name> is not specified, then the room is remembered by its actual name (as opposed to a nickname you provide). If <place> is not specified, then the current room is remembered.

**@addword** <word or words>
$frand_class
Add a word or words to your personal dictionary, if one is kept. (This may be disabled on some MOOs that do not use $spell.)

**@age** [<player>]
$player
Tells a person's MOO age, i.e. how long it has been since e first connected. On LambdaMOO, the difference between a person's actual MOO age and eir official MOO age is because system down time doesn't count for aging of people. This arose from a ballot to "stop the clock" on legislative issues (including the determination of voting age, etc.).

**@answer** [<message-number>] [on <mail-recipient>] [sender | all | followup] [include | noinclude]
$mail_recipient_class
See @reply.

**@arb** [all]
LM player class #322
Displays a list of the connected members of the LambdaMOO Architecture Review Board. If all is specified, then displays all members, whether connected or not.

**@arb-ballots**
LM player class #322
Lists open ballots for the LambdaMOO Architecture Review Board.

**@arb-nominate** <player>
LM player class #322
Obsolete. See @nominate.

**@arb-petitions** [all]
LM player class #322
Displays a list of all petitions nominating candidates to the office of LambdaMOO Architecture Review Board, except those you may have declined. Using the argument all shows all ARB nominating petitions.

**@args** <object>:<verb name> [<dobj> [ <prep> [<iobj>]]]
**@args#** <object>:<verb number> [<dobj> [ <prep> [<iobj>]]]
$prog
Changes the argument specifiers for an exisiting verb. Any omitted argument specifiers remain unchanged. If no arguments are given, then this prints out the current argument specifiers for the indicated verb. The second form (@args#) is used to specify a verb by number rather than by name, and is useful if an object has two verbs with the same name.

**@at** <object>
$frand_class
Prints a brief list of connected players either with or in <object> (depending on whether the specified object is a player, some other kind of object, or a room).

**@audit** [<player>] [for <string>] [from <object number>] [to <object number>]
$builder

Shows a list of objects that you own or that a specified player owns, with their object numbers. You may optionally specify a string, and see a list of objects with names or aliases that begin with that string. (Note, if the string has a space in it, then you must enclose it in double quotes.) You may optionally restrict the listing to a range of object numbers.

**@ballots** [all | open | closed | passed | failed | defeated]
LM player class #322

Prints a list of ballots. If no arguments are provided, prints a list of open ballots, if any.

**@ban*** <object>
**@ban!** <object>
LM player class #322

Prevents the specified object from entering any room that you own. If used with the exclamation point, prevents the specified object *and any descendents of it* from entering any room that you own. See also @unban.

**@banned**
LM player class #322

Prints a list of all objects you have banned from rooms you own using the **@ban** command.

**boring** [on | off]
LM player class #5803

Boring on makes you impervious to food fights. Boring off enables you to participate again. Boring with no argument toggles the setting and tells you what the new setting is. In addition, you will not lose things from your inventory into the couch cushions if you are boring.

**@boot** <guest>
LM player class #322

Disconnects a guest player's connection and disallows new connections from that guest's site for the following one hour. You must be at least four months old to use the command, and must give a reason, which is posted to *boot-log. The command must be seconded by another player.

**@bug** [<text>]
$mail_recipient_class

Sends MOOmail containing <text> to the owner of the room you're in as a bug report. If you do not specify text in the command line, then you are moved to the mail room to compose your message (presumably at greater length).

**@build-o*ptions** [<option> | <option setting>]
Also: **@buildo*ptions @builder-o*ptions @buildero*ptions**
$builder
Used without arguments, this command displays your current builder-options (settings that modify various builder commands). Used with a single option, displays the current setting of that option. Used with an option setting, modifies the specified option.

**@check-chp*arent** <object> to <new parent>
$builder
An object cannot be changed to a new parent if that object or any of its descendents defines a property that is also defined on the intended new parent. This command prints out all instances of conflicting properties that would interfere with @chparent in this manner.

**@ch*eck** <number of lines> [[!]<player>[,…,[!]<player>]]
$player
Prints a list of "best guesses" about where a line or lines of text originated, looking for "distrusted" players. By default, you and all the wizards are trusted, but you may specify additional players to be trusted (<player>) or not to be trusted (!<player>). You must have @paranoid on for this to work; LambdaMOOers will have to type @rmlag for @paranoid to work.

**@ch*eck-full** <number-of-lines> | <search string>
$player
Used to identify the source of text that is of dubious origin. @check-full prints out information about all the verbs responsible for a line of text displayed to your screen. You may specify either a number of lines or a string of text whose origin you wish to know more about. You must have @paranoid on for this to work; LambdaMOOers will have to type @rmlag for @paranoid to work.

**@check-p*roperty** <object>.<property name>
$prog
Prints a list of all descendents of <object> that define <property name>. See also @check-chparent.

**@chmod** <object> [+|-]<any substring of "rwf">
**@chmod** <object>.<property> [+|-]<any substring of "rwc">
**@chmod** <object>:<verb> [+|-]<any substring of "rwxd">
**@chmod#** <object>:<verb number> [+|-]<any substring of "rwxd">
$prog
Sets or changes permission flags. Objects – and also objects' individual properties and verbs – have *permission flags* that control whether non-owners can or cannot: read (objects, verbs, and properties), write (objects, verbs, and properties), execute (verbs), manually setting the value of properties, and make children (objects) (i.e. is the object fertile?). The "c" flag determines whether the owner of an object may change the value of a property that is defined on a parent or ancestor. (There is a longer explanation of the "c" flag and its use beginning on page 165.) If used with the "+" or "-" signs, it incrementally sets or clears the specified values. If used

without the “+” or “-” signs, it sets the permission flags to the specified values (clearing values as necessary). See also `help @chmod`.

**`@chparent`** `<object> to <new parent>`
`$builder`
Changes the parent of `<object>` to `<new parent>`. The object now has all the new parent’s properties and verbs, and all the new parent’s ancestors’ properties and verbs.

**`@cl*asses`**
`$builder`
Prints a list of object classes that the wizards have identified as “useful”. (This information is stored in `#0.class_registry` as a list of sublists. Each sublist consists of: `{<category>, <one-line description>, {<objects>}}`. The list is maintained manually.)

**`@clearp*roperty`** `<object>.<property>`
**`@clprop*erty`** `<object>.<property>`
`$prog`
Clears the value of the specified property on an object. This means the property will henceforth inherit its value from the object’s parent and will change as the value of the propety on the parent changes. The property will remain clear until it is set or changed on the child object itself.

**`@clear-tell-filter*-hook`**
LM player class #33337
Removes any tell-filter that is in use. (See `@set-tell-filter`.)

**`@comment`** `[<text>]`
`$mail_recipient_class`
Sends MOOmail containing `<text>` to the owner of the room you’re in as a comment. If you do not specify text in the command line, then you are moved to the mail room to compose your message (presumably at greater length).

**`@complete`** `<beginning of a word>`
`$frand_class`
Lists all the words in the dictionary (if present) that begin with the text you supply. E.g. `@complete silh` will give “silhouette”. (This may have been disabled in MOOs that don’t support `$spell`.)

**`connect`** `guest | <specific guest name>`
**`connect`** `<player name> [<password>]`
`$login`
Connects you to the MOO. Guest connections do not require a password. Omitting the password for a player connection will provide a separate prompt for your password, so that it will not be displayed on your screen.

**`@contents`** `[<object>]`
`$builder`
Gives a definitive list of `<object>`’s contents. If `<object>` is not specified, then lists the contents of the room you’re currently in.

**@copy** <object>:<verb> to <target object>[:<new verb>]
**@copy-x** <object>:<verb> to <target object>[:<new verb>]
**@copy-move** <object>:<verb> to <target object>[:<new verb>]
$prog

Copies a verb from one object to another, or to a new verb on the same object. It's better to have an object inherit a verb from a parent object than to copy verbs to objects directly, but occasions arise when copying a verb is the only way to get something done. @copy-x copies the verb without its "x" (executable) flag set, and would be used to archive a verb before making modifications to the working verb. @copy-move deletes the original verb after the copy is complete.

**@count** [<player>]
$builder

Tells you how many objects <player> owns and the total number of bytes used by those objects. <player> defaults to yourself.

**@countDB** [<player>]
$builder

This verb is related to @count, differing only in the counting method. @count inspects a player's .owned_objects property. A very few system characters (notably Hacker) do not participate in the object ownership system. To count these players' objects, it is necessary to consider every object in the database and see if <player> is its .owner. In large databases, this takes a long time and hogs system resources. Use @count instead, whenever possible.

**@create** <parent object> named <name>[,<alias>,…,<alias>]
$builder

Creates an object with the specified parent, name, and aliases. The object's parent can be changed at a later time with the @chparent command. The name and/or aliases can be changed with the @rename command. Aliases can be added or removed with the @addalias and @rmalias commands respectively. For rooms and exits, it's better to use @dig than @create.

**@cspell** <any number of words> | <object>.<property> |
<object>:<verb>
$frand_class

For those MOOs that utilize $spell, this command will check for misspelled words. It tends to run slowly.

**@db*size**
$prog

Reports the number of valid objects and allocated objects in the database.

**decline** <petition>
LM #55266 (Generic-Petition)

Removes <petition> from the list you see when you type @petitions.

**@define** <variable> as <value>
LM player class #8855
This command will probably be of interest only to programmers. It lets you predefine a value for use in a subsequent call to `eval`. See also `@listdefs` and **@undef**.

**@denewt** <player> [<comment>]
$wiz
Reverses the effect of `@newt` or `@temp-newt`.

**@describe** <object> as ["]<description>["]
$player
Sets the description of the specified object. If you omit the quotation marks, then sentences will be separated by a single space only, regardless of how many spaces you use to separate them when you type the description in. For multi-line descriptions, edit the `.description` property with the note editor.

**@detail** me with <detail name>[,<alias>,…<alias>] as <detail description>
**@detail** me with <detail name> is
LM player class #6669
Let's you add a detail to your description. Use the null string to remove a detail. Use the second form to display a detail. See `help #6669:@detail` for more detailed examples.

**@details** me | <nearby player> | <player object number>
LM player class #6669
Shows what details the specified player has defined. Note, this does not match on players' names the way many verbs do (e.g. `page`). If the player you want to know about isn't in the same room, you will have to use eir object number.

**@dig** <new room name>
**@dig** <exit>[,<aliases>][|<return-exit>[,<aliases>]] to <new room name> | <existing room object number>
$builder
Creates a new room, or exits to (and optionally back from) either a new room or a specified existing room. Note, the vertical bar "|" separating exits to and from a room is actually part of the command, rather than the meta syntax. Example:

```
@dig north,n | south,s to The Mosh Pit
```

**@disown** <object> [from <parent>]
Also: **@disinherit**
$prog
Contrary to what the name might suggest, this command does not alter an object's ownership. Rather, it alters an object's parentage, changing an object's parent to its grandparent. This command would be used if you did not own the object, but owned its parent, and no longer wanted the object to be a child of that parent. Changing the permission flag on the parent to `-f` (see `@chmod`) will prevent people from using that object as a parent in the future.

**@d*isplay** <object>[.][,][<property>]
**@d*isplay** <object>[:][;][<verb>]
**@d*isplay** <object>[.|,][:|;]
$prog

This command displays <object> and/or <object>.<property> and/or <object>:<verb> ownership, permissions, and values (for properties). The period and colon indicate information about properties or verbs defined on the object itself; the comma and semi-colon indicate information about inherited properties and verbs. See `help @display` for a more detailed explanation.

**@display-o*ptions** [<option> [<setting>]]
Also: **@displayo*ptions**
$player

Used without arguments, this command displays your current `@display` options (settings that modify various aspects of the output from `@display`). Used with a single option, displays the current setting of that option. Used with an option and setting, modifies the specified option.

**@dump** <object> [with [id=#<new object number>] [noprops] [noverbs] [create]]
$prog

Prints out all of an object's verbs and properties. If you specify `with create`, it will print it out in a form that can be used for porting an object to another MOO rather than merely investigating it on the source MOO. You can optionally omit properties and/or verbs, and can ask it to use a different object number in its output (this, too, can be useful for porting an object to another MOO).

**@edit** <object>[.<property>]
**@edit** <object>:<verb> [<dobj> <prep> <iobj>]
$player

Invokes the note editor or verb editor as appropriate. If no property is specified, then it defaults to `.text` if <object> is a descendent of `$note` or `.description` for any other kind of object.

**@edit-o*ptions** [<option> [<setting>]]
Also: **@edit-o*ptions**
$player

Used without arguments, this command displays your current `@edit` options (settings that control various aspects of the in-MOO editors). Used with a single option, displays the current setting of that option. Used with an option setting, modifies the specified option.

**@egrep** <regular expression> in <object> | <list of objects>
$prog

Searches the specified object or list of objects for verbs containing a substring matching <regular expression>. A regular expression is a template for expressing generalized strings. See `help regular-expressions`. See also `@grep`.

**@eject** <player or other object> [from <location>]
**@eject!** <player or other object> [from <location>]
**@eject!!** <player or other object> [from <location>]
$player

The usual way to move something is with the @move command, and it's considered polite to try to @move a thing before ejecting it. If the object won't move and you own the object's location (this includes yourself), then you should use @eject. With no exclamation points, this moves an object to its .home if indicated and possible. With one exclamation point, moves the offending object to the location $nothing, but notifies the object that it's being moved. With two exclamation points, moves the object to $nothing but doesn't notify the object.

**eprint** <expression>
**eprint<n>** <expression>
LM player class #5803

This command is useful for printing out complicated MOOcode expressions with indentation intended to make them easier to read and understand. For example if you had a complicated conditional clause and wanted to sort out what was actually being checked:

```
eprint (caller == this && args[2] ||
this.tally_board.registry:primary_char(dude) in
this.tally_board.public_access || (caller != this &&
$local.second_char_registry:trust(caller_perms())))
```

would yield:

```
((caller == this) && args[2])
    || ((   this.tally_board.registry:primary_char(dude)
         in this.tally_board.public_access)
    ||  (   (caller != this)
         &&
$local.second_char_registry:trust(caller_perms())))
```

The output can optionally be restricted to <n> columns. There is no space between eprint and <n>, e.g,

```
eprint10 ((ticks_left() < 3000) && suspend(0)).
```

**eval** <MOO-code>
**eval-d** <MOO-code>
**;**<MOO-code>
$prog

Evaluates a line of text as if it were MOO-code. Eval-d prints errors as values rather than generating a traceback. See help eval.

**exam*ine** <object>
$player

Provides more information about an object than you can get just by looking at it, including its full name, aliases, object number, owner, description, and any

obvious verbs that you can use on it. Unlike @examine, its output can be modified by the object's owner.

**@exam*ine** <object>
$player
Provides the same information that examine does, except that its output can't be controlled by the object's owner. This has advantages and disadvantages. The advantage is that you may see more information. The disadvantage is that the information might not be printed as nicely or might not be relevant to you (e.g. "obvious verbs" that are really only intended to be used by the object's owner).

**@features** [<name>] [for <player>]
$player
Lists all of a player's features matching <name>, or, if <name> is not supplied, all features for that player. Lists your own features if no player is specified.

**@find** <object number> | <player name or alias> | .<property name>
| :<verb name> | ?<help topic>
$frand_class
Prints the location of the specified thing. This is especially useful for verbs and properties because it prints out *all* instances that it finds in your vicinity (including your known objects, see @remember).

**@flush-cache**
LM player class #322
When you use a feature object, the server must look through all your feature objects to find the appropriate verb. This takes time. In order to help reduce lag, the system records your most recently used feature object verbs and checks those first. The place where your most-recent-usage information is stored is called a *cache*. This command clears out the cache of your recently-used feature object verbs.

**follow** <player>
LM player class #8855 (This command is also provided on some other MOOs, but not is included with LambdaCore.)
Causes you to follow <player>. See also unfollow, stop-following, @list-followers and lose.

**@forget** <object>
LM player class #26026
Removes an object from the list of those you keep track of using @remember.

**@forked** [<player>]
$prog
Displays a list of all your suspended and forked tasks with their respective task_id numbers. Only a wizard may specify a player other than emself. If a wizard invokes this command without specifying a particular player, then this command will display all forked tasks in the system. This command is particularly useful for identifying tasks that you may want to @kill.

**@forked-v*erbose** [<player>]
$prog
This command displays the same information as @forked, except that for tasks that are suspended rather than forked off, shows the full callers() stack.

**@forward** <msg> [on *<recipient>] to <recipient>[,<other recipients>]
$mail_recipient_class
Forwards a MOOmail message to the designated recipient(s). See help @forward for a discussion of the nuances of this command.

**@gag*!** <player or object>
$player
This command prevents you from seeing any text emanating from the specified player or object. (Note, this does not include posts to mailing lists or MOOmail. See @refuse.) If an object has children, then you must use the exclamation point; this will have the effect of @gagging the object and all its descendents. It is not possible to @gag a parent object only.

**@gaglist** [all]
$player
The first form, with no arguments, displays a list of players and objects that you are gagging. The second form looks for and displays players who are gagging you. The second form is slow to run and adds to lag, so it should be used sparingly.

**@gag-site** <guest> for <duration>
LM player class #322
Prevents you from seeing any text from any guest connecting from the same site as the designated guest, for the specified duration of time.

**@gag-sites**
LM player class #322
Displays a list of all guests whose sites you have gagged, along with when you gagged the site and how much time is remaining before the site-gag expires.

**@gender** [m | f | n | <other>]
$player
Sets your gender (and pronouns) to male, female, neuter, etc.. Without an argument, displays your current gender setting and other available genders to

**@gethelp** [<topic> [from <db or dblist>]]
$prog
Locates and prints out the raw text of a help topic in a form that can be cut, modified, and pasted back in (like @dump). With no argument, gets the blank ("") help topic.

**@gms** [all]
LM player class #322
Prints an @who listing of connected (or all) LambdaMOO RPG game masters.

**@go** <location>
$frand_class
Teleports you to the specified location, which can either be the object number of a room or the name of a room in your .rooms database.

**go** <direction>
$room
Moves you in the specified direction (e.g. north). You may specify more than one direction, in which case you will go in those directions in sequence. Go north east north would move you first north, then east, then north again.

**@grep** <string> in <object> | {object list}
$prog
Searches the specified object or list of objects for verbs containing <string>. See also @egrep.

**@gripe** [<text>]
$mail_recipient_class
Moves you to the mail editor, ready to send a mail with subject heading <text> to the mail recipient(s) specified by the wizards in $gripe_recipients.

**heartbeat**
Syntax: ;me:heartbeat(<n>)
LM player class #5803
Starts up a task that prints a time stamp to your screen every <n> minutes. For those who idle for long periods of time, this can help identify when someone paged you. Note, this command can only be used by programmers (because it has to be started up using eval).

**help** [<topic>]
$player
Displays online help text for the specified topic. If no topic is specified, then it displays a list of some of the topics for which help text is available.

**home**
$player
Moves you to your home, or to the default player starting place if your home is invalid or won't accept you for some reason.

**@idea** [<text>]
$mail_recipient_class
Sends MOOmail containing <text> to the owner of the room you're in as an idea suggestion. If you do not specify <text> in the command line, then you are moved to the mail room to compose your message (presumably at greater length).

**inv*entory**
$player
Prints a list of things you are holding, with their object numbers.

**@join** <player>
$frand_class

Teleports you to the specified player's location. You can specify <player> by object number, name, or any alias.

**@keep-m*ail** [<message sequence>]
Also: **@keepm*ail**
$mail_recipient_class

Prevents the designated mail message(s) from expiring (i.e. being automatically deleted after a certain amount of time). See help @keepmail.

**@kids** <object>
$builder

Prints out a list (with object numbers) of all an object's children. (Note, not all descendents, just children.)

**@kill** <task id> | <object>:<verb> | soon <number of seconds> | all
| %<trailing id>
Also: **@killq*uiet**
$prog

Kills one or more background tasks (see @forked). The second form, @killquiet, is better for killing large numbers of tasks, as it prints a summary of the number of tasks killed rather than a line for each one (especially useful if you've accidentally created a chain of forked tasks, each of which is sending text to your screen). Task id numbers tend to be large. You can use the %<trailing id> form to abbreviate the number to its last few digits: Instead of typing @kill 2053554299, you can type @kill %299, and the system will kill all tasks in your queue that end in the numbers 299.

**@known*_objects**
Also**: @known*-objects**
LM player class #26026

Prints a list of objects you've made note of with @remember.

**@last-c*onnection** [all ]
$player

Reports your most recent connection information (when and from where) or, if all is specified, your last ten connection times and sites.

**@lastlog** [<player>[,...,<player>]]
$player
LM player class #5803

Shows the last disconnect time of the specified player or players. If called with no arguments, shows the last connection times of *all* players.

**@linelen*gth** [<number>]
$player

This command is used in conjunction with @wrap, to cause the MOO to perform word-wrapping for you. Without arguments, informs you of your current setting,

along with whether word-wrapping is currently turned on or off. With a number as an argument, sets your line length to that number of characters.

**@list*#** <object>:[<verb name> [<dobj> <prep> <iobj] | <verb number>] [with | without parentheses | numbers] [all] [<start>..<end>]
$prog
Lists the MOO code associated with the specified verb. Normally, this command lists only the code found either on the specified object itself or on its nearest ancestor. The optional argument `all` causes the corresponding code on the object and *all* ancestors to be displayed. By default, lines are numbered and show only those parentheses necessary to the meaning of the code. You can specify a range of line numbers to list if you know you only want to see part of the verb. These defaults can be changed with the `@programmer-options` command.

**@listdefs**
LM player class #8855
Lists variables you have defined using the `@define` command.

**@list-followers**
LM player class #8855
Prints a list of people who are programmatically following you. (See also `follow`.)

**@listgag** [all]
$player
See @gaglist.

**@location*s** <object>
$builder
Prints out the names and numbers of all objects that contain the specified object.

**@lock** <object> with <key expression>
$builder
This command is used to specify (via `<key expression>`) locations to which an object may be moved (and by extension, locations to which an object may *not* be moved). See `help keys`.

**lose** <player> | all
LM player class #8855
This command causes `<player>` (or everyone) to stop following you programmatically. See `follow`.

**@mail** <message-sequence> [on <recipient>]
$mail_recipient_class
Displays headers of the specified mail messages. (See the section on reading mail that begins on page 51 for a detailed explanation.)

**@mail-all-new*-mail**
$mail_recipient_class
Displays the headers of all unread mail messages on yourself and lists to which you are subscribed.

**@mail-o*ptions** [<option> | <option setting>]
Also: **@mail*ptions**
$player
Used without arguments, this command displays your current @mail options (settings to customize various aspects of the mail system). Used with a single option, displays the current setting of that option. Used with an option setting, modifies the specified option.

**@make-petition** <name>[,<alias>,...,<alias>]
LM player class #322
Creates a LambdaMOO petition with the specified name and aliases.

**@measure** <object>
**@measure** summary [<player>]
**@measure** new [<player>]
**@measure** breakdown <object>
**@measure** recent [<number of days>] [<player>]
$builder
For MOOs that use byte-based quota, objects are measured approximately once a week by a background measurement task. The various forms of the @measure command provide a way to update the measurement records on an incremental basis, when needed. @measure <object> measures the size of an object on demand. This is appropriate if an object's size is known to have undergone a recent significant change. @measure summary updates the summary information displayed by the @quota command. @measure new measures all of a player's objects that have never been measured. (This might be needed if one were creating a large number of small objects in a short time span, as one is only permitted to have a fixed number of unmeasured objects at a time.) Use @measure breakdown if you need to find out what part(s) of an object are taking up large amounts of space. @measure recent measures those things which have not been measured automatically with the specified number of days.

**@mess*ages** <object>
$player
Lists all the messages on the specified object, and their values.

**@mode** [brief | verbose]
$player
Sets your viewing mode. If brief, then only the name of a room will display on your screen when you enter that room. If verbose, then a room's name and description will display when you enter it. The default is verbose.

**@more** [rest | flush]
$player

If you have @pagelength set and the system has produced more lines of output than will fit on your screen, you will see a message of the form

```
*** More ***  <n> lines left.  Do @more [rest | flush] for
more.
```

@more without arguments prints sufficiently many lines to fill your screen, or all that remain, if they will fit. @more rest will print all of the remaining lines, regardless of whether they will fit or not. @more flush discards all remaining lines instead of displaying them on your screen.

**@move** <object> to <location>
$frand_class

Teleports the specified object to the specified location.

**mu*rmur** <person> <text>
LM player class #33337

This command does the same thing as whisper, except that the syntax is such that you don't need to enclose the whispered text in quotation marks.

**news** [all | new | contents | archive]
$player

Read the contents of the newspaper, which is a subset of messages on the *news mailing list that the wizards have designated as being of current interest or relevance to the entire MOO. news new will display news items that you have not yet read. news all will display all news items. news contents will display headers of all news items. news archive will display all messages on the *news mailing list.

**@netforw*ard** [<message-sequence> [on <mail-recipient>]]
$mail_recipient_class

Forwards the designated message(s) to your registration email address. Defaults to the current message on your current folder.

**@newmess*age** <message-name> [<message-text>] [on <object>]
$builder

In general, only programmers can add new properties to objects. This command lets non-programmer builders add message properties to objects they own.

**@newt** <player> [<reason>]
$wiz

A wizard-only command. Inhibits the specified player's ability to connect to the system. A MOOmail message is automatically sent to *site-locks. See also @denewt, @temp-newt.

**@next** [<how-many>] [on <mail_recipient>]
$mail_recipient_class

Prints out the next <how-many> mail messages on your current folder or the designated mail recipient (yourself or a mailing list). Defaults to one message.

**@nominate** <player> for <office>
LM player class #322
Nominates a person for public office on LambdaMOO. This can only be done during a two-week nominating period before any election. The offices are: ARB, Reaper, and Registrar.

**@notedit** <note-object> | <object>.<property>
$player
Moves you to the note editor, working either on the text of the specified note or on the text in the designated property on the designated object.

**@nprop*erty** <object>.<prop-name> [<initial-value> [<perms> [<owner>]]]
LM player class #5803
This is just like @prop*erty, except that <initial-value> is evaluated, first.

**@owner** <object>
$player
This command shows you who the owner of an object is. (It was added to LambdaMOO in July, 2000, and may or may not be available on other MOOs.)

**page** <player> <text>
$player, LM player class #5803
Sends <text> to <player> as if you were paging em from a distance. The fancier version on LM player class #5803 lets you page more than one player simultaneously; you must enclose the players' names in quotation marks: page "<player1> <player2> <player3>" <text>.

**@pagelen*gth** [<number>]
$player
This command is used in conjunction with @more to control the display of lines on your screen. When a number is specified, it sets your page length to a number of rows (of text). The system will prompt you with a message to type "@more" if there is more text about to display than will fit at one time. Without an argument, it will show you your current setting. To turn off page buffering and see all the lines of text at once, set your page length to zero. This would be an appropriate choice if you were switching from telnet to a client that lets you scroll back, for example.

**@paranoid** [off | immediate | <number>]
$player
This command is used to record and investigate lines of text that print to your screen. If invoked with immediate as the argument, it will prepend each line you see with the name of the player it thinks is responsible for generating that line. If invoked with <number>, then that number of lines is stored for later inspection with @check or @check-full. On LambdaMOO, players must first type @rmlag to disable the lag-reduction FO for @paranoid to work.

**@parent** <object>
LM player class #8855
Tells you the name and object number of an object's immediate parent.

**@parents** <object>
$builder
Displays the names and object numbers of an object's parent and ancestors.

**party**
LM player class #5803
This command prints a list of rooms and occupants in order of decreasing crowd size and increasing idle time (i.e. the liveliest parties first). For each, it comments on the security arrangements and asks if you want to go there. You may discontinue the listing at any time by typing @abort.

**@password** <old-password> <new-password>
$player
Changes your password.

**@paste**
Pasting Feature Object, LM player class #8855
Prompts for lines of text (terminated by a period on a line by itself) then displays the text to the entire room.

**@pasteto** <player>
Pasting Feature Object
Prompts for lines of text, then displays them (privately) to the specified player.

**@pc-news**
LM player class #33337
The author of the Politically Correct Featureful Player Class Created Because Nobody Would @copy Verbs to 8855 provided himself with a way to broadcast news items to users of his player class in a manner similar to the MOO-wide news command. This command is used to read those news items.

**@pc-options**
LM player class #33337
This player class provides an options package that works the same way that @mail-options does. Type @pc-options to list these options, and help @pc-options for additional information on setting them.

**@pedit** <object>.<property>
LM player class #5803
This command moves you to the property editor, described as "highly experimental" by its author. If you are editing a property whose value is a string or list of strings, you are probably better off using the note editor, instead, but this facility might be useful for editing properties with a more complex structure. See help @pedit for more detailed information.

**@petition-options**
LM player class #322
Lists options that pertain to various aspects of the LambdaMOO petition and ballot system. Use @petition-option +noannounce to suppress announcements of open ballots every time you log in.

**@petitions** [all | public | signed | vetted]
LM player class #322
Lists all or some petitions, as specified. The default is to list signed petitions. You can use @petition-options to customize the order in which petitions are presented.

**@prettylist** <object>:<verb>
LM player class #5803
Prints a verb with line breaks and indentations intended to make it easier to read.

**@prev*ious** [<how-many>] [on <mail_recipient>]
$mail_recipient_class
Prints out the previous <how-many> mail messages on your current folder or the designated mail recipient (yourself or a mailing list). Defaults to one message.

**@prog*ram** <object>:<verb> [<dobj> <preposition> <iobj>]
**@program#** <object>:<verb-number>
$prog
These commands put you into a line-reading mode. The lines you type in are saved as the content of the designated verb on the designated object, if that verb exists (otherwise the lines of text are still read, but are ignored).

**@progo*ptions**
Also: **@prog-o*ptions @programmero*ptions @programmer-o*ptions**
$prog
Lists a set of options available to programmers to customize certain system behaviors related to programming, e.g. whether line numbers print out when you list a verb. The @programmer-options package works like the @mail-options package.

**@prop*erty** <object>.<property-name> [<initial value> [<permission-flags> [<owner>]]]
$prog
Adds a property to an object. The initial value defaults to 0. The permission flags default to "rc", but this can be changed as one of the @programmer-options. A wizard may specify an owner other than emself.

**@pros*pectus** <player>
$prog
This command is like @audit, but provides additional information about each object, such as whether it has kids, how many verbs are defined on it, etc. See help @prospectus for more detailed information.

**@quickr*eply** <msg> [on <recipient> ] [sender | all | followup]
Also: **@qreply**
$mail_recipient_class
This command lets you reply to a mail message without actually going to the mail editor. It prompts you for lines of input and then sends them directly.

**@quick*send** <mail-recipient> [subj = "text"] [<one-line-message>]
Also: **@qsend**
$mail_recipient_class
Sends a message to a player or a list without moving you to the mail editor. If you do not specify a one-line message, it prompts you for lines of input, then sends them directly.

**@quit**
$player
Disconnects you from the MOO. Your player object is automatically moved to its home a short time later.

**@quota** [<player>]
$builder
Prints out your current quota and measured usage, or the quota of a specified player.

**@ranm**
$mail_recipient_class
See @read-all-new*-mail.

**@read** [<message-sequence> [on <mail-recipient>]]
$mail_recipient_class
Reads the specified messages on the specified mail recipient (yourself or a MOO mailing list). If no message sequence or recipient is specified, reads your current message on your current folder. Updates your current message pointer.

**@read-all-new*-mail**
$mail_recipient_class
Reads all new messages on all mailing lists to which you are subscribed. Prompts you at the end to verify that you got all the information, and if you answer yes, updates your current message pointer. If the system crashes or you somehow disconnect before being able to answer the prompt, then your current message pointer is not updated, and these messages will still appear as new messages next time you log on. This command can be abbreviated as @ranm.

**@reaper-ballots**
LM player class #322
Lists ballots for the office of LambdaMOO Reaper.

**@reaper-petitions**
LM player class #322
Prints a list of all petitions nominating candidates to the office of LambdaMOO Reaper, except those you may have declined. Using the argument all shows all reaper nominating petitions.

**@reapers** [all]
LM player class #322
Prints a list of connected (or all) LambdaMOO Reapers. Reapers recycle players who have not logged on for a certain amount of time, and oversee the distribution of their owned objects if deemed appropriate. See help reaping.

**@recreate** <object> as <parent> named <new-name>
$builder
Takes an existing object and totally recreates it as a new kid of <parent> as if with **@create**. Verbs and properties on the object are stripped off, and inherited properties are reset to be clear.

**@recycle** <object>
$builder
Destroy an object irretrievably. If you have your @builder-options set to –bi_create (the preferred setting), the object will be turned into a kid of $garbage for re-use the next time someone invokes @create. Players may not be recycled unless they have first been made non-players with the @toad command or an equivalent.

**@refusal-r*eporting** [on | off]
$frand_class
If set to on, notifies you when someone whom you are refusing attempts a refused action while you are connected ("so that you can thumb your nose," says the documentation). If invoked without an argument, displays whether refusal reporting is currently on or off. Refusal reporting works for page, whisper, and mail, but doesn't work for move, join, or accept.

**@refusals** [for <player>]
$frand_class
Lists players and actions that you are refusing or that the specified player is refusing.

**@ref*use** <action(s)> [from <player>] [for <duration>]
$frand_class
The MOO provides a way to refuse certain actions, either universally or from a specified player. The actions that can be refused are page, whisper, mail, move, join (only works in certain rooms that support it), accept, flames, politics (LambdaMOO only), and all of the above. See also help @refuse.

**@registerme** [as <email-address>]
$player
Displays your current MOO registration email address, or changes it to a new one. If you are changing it, a new password is generated and mailed to the new address. You can change the new password back again with the @password command.

**@registrar-ballots**
LM player class #322
Lists ballots for the office of LambdaMOO Registrar.

**@registrar-petitions** [all]
LM player class #322
Prints a list of all petitions nominating candidates to the office of LambdaMOO Registrar, except those you may have declined. Using the argument all shows all registrar nominating petitions.

**@registrars** [all]
LM player class #322
Prints a list of connected (or all) LambdaMOO Registrars. Registrars assist the wizards in creating new players. They have access to players' email addresses.

**@remember** <object>
LM player class #26026
Remember an object's number. You can see a list of these objects by typing @known. See also @forget.

**@remove-feature**
See **@rmfeature**.

**@rename** <object> to "<new-name>"[,"<alias>",...,"<alias>"]
$player
Rename an object, with or without additional aliases. See help @rename for some detailed examples.

**@renumber** [me | <mail-recipient>]
$mail_recipient_class
Renumbers, from 1 to the total number of messages, all MOOmail messages on yourself or on a mailing list you own. Renumbering a public mailing list is inadvisable because it disrupts other players' current message pointers to that list. No messages are actually lost, but @rn will show that there are new messages on the list while @nn will say that there are no new messages. See also help zombie-messages.

**@repl*y** [<message-number>] [on <mail-recipient] [sender | all | followup] [include | noinclude]
$mail_recipient_class
Takes you to the mail editor and sets up a reply to the specified message. Specify sender to reply to the sender only, all to send your reply to all recipients who received the original post, or followup to send your reply to the first non-player recipient (i.e. a list). Specify include or noinclude to include or omit (respectively) the text of the original message. If these options are omitted, the defaults are sender and noinclude, but these can be changed with @mail-options.

**@request** <character-name> for <email-address>
$guest
This is the command used to request a player-character on a MOO.

**@resend** <message-sequence> [on <mail-recipient>] [to <recipient(s)>]
$mail_recipient_class
This is like @forward, except that it keeps the original body of the forwarded message intact and modifies the header to indicate that you resent it.

**@resident** <player-or-object>
**@resident** !<player-or-object>
**@residents**
$room
The first form adds a player or object to a room's list of allowable residents. If a player, that player may then set eir home to that room. The second form removes a player or object from a room's list of residents. The third form displays a room's current list of residents.

**@rmalias** <alias> from <object>
**@rmalias** <alias> from <object>:<verb-name>
**@rmalias#** <alias> from <object>:<verb-number>
$player
Removes an alias from the specified object or verb. @rmalias# is for unambiguously identifying a verb when an object may have more than one verb with the same name.

**@rmdict** <word>
$frand_class
Remove a word from $spell, if present. (Many MOOs have recycled $spell because it is cumbersomely large). Only wizards and players in $spell.trusted may use this verb.

**@rmfeature** <feature-object>
$player
Remove a feature from your .features list. Feature objects are used to extend the set of commands available to a player. See also @add-feature.

**@rmlag**
LM feature object #26787
Turns on the lag reduction FO and any features similar to it. Disables @gag, @check-full, and any other commands that utilize a player's :tell verb to filter and/or otherwise pre-process text before it is displayed to your screen.

**@rmm*ail** [<message-sequence>] [from <recipient>]
$mail_recipient_class
Removes one or more MOOmail messages from yourself or a specified mail recipient (mailing list). See also @unrmmail.

**@rmprop*erty** <object>.<property-name>
$prog
Removes the named property from the specified object.

**@rmr*oom** <name>
$frand_class
Remove the named room from the list of rooms you remember by name. See also @addroom and @rooms.

**@rmverb** <object>:<verb-name> [<dobj> <prep> <iobj>]
**@rmverb#** <object>:<verb-number>
$prog
Remove the specified verb from the specified object. If there are two or more verbs with the same name, removes the most recently defined one. If the argument specifiers are provided, then it removes the most recently defined one matching both verb name and argument specifiers. The second form, @rmverb#, is used to unambiguously remove a verb as specified in the (1-based) list given by the built-in function verbs(<object>).

**@rmword** <word>
$frand_class
Remove a word from your personal dictionary (stored in a player's .dict property).

**@rn**
$mail_recipient_class
Lists a summary of new messages on mailing lists to which you are subscribed, similar to that displayed when you log in.

**@rooms**
$frand_class
Displays a list of rooms you have remembered using the @addroom command.

**seek** <player>
LM player class #7069
Tries to move you to the designated player's location using an exit, thus simulating walking (as opposed to teleporting). (See also help #27325:@seek).

**@send** [<recipient> [<recipient(s)>] [subj[ect]="<subject>"]
$mail_recipient_class
Moves you to the mail editor and prepares you to compose a MOOmail message to the designated recipient(s). If no recipient is specified, resumes an earlier mail editor session, if there was one.

**@setenv** <environment string>
$prog
Sets a string that is evaluated before the eval command evaluates anything else. Example: @setenv me=player;here=player.location

**@sethome**
$player
Tries to set your home to your current location. Some rooms permit players to set their homes there, while others do not. This is at the discretion of the room's owner. If you are a room owner and want to make it so that anyone may set eir home there, @set <room>.free_home to 1. To permit an individual player to set eir home to a room you own, use the @resident command.

**@set*prop** <object>.<property name> to <value>
$builder
Changes the value of an existing property on an object to the specified new value.

**@set-tell-filter*-hook** <tell-filter-object>
LM player class #33337
A tell-filter is an object that intercepts text which is about to be displayed to your screen and may (or may not) modify that text in some way before it is actually displayed to you. One possible example would be to put a special symbol before text that was generated with the emote verb. A tell filter is usually custom programmed by the player intending to use it – if you use a tell-filter owned by someone else, its owner would theoretically be able to see most of the text that you see if e chose to. (See also @clear-tell-filter and help tell-filter.)

**@s*how** <object> | <object>.<property> | <object>:<verb>
$prog
This command is very similar to @display, but the information it displays about objects, properties and verbs is in a different format and more detailed. See also @ss and @display.

**@skip** <mail recipient>
$mail_recipient_class
Skip to the end of a mailing list, as if you had read all the messages. (This resets your current message pointer for that list.)

**@sort-owned*-objects** object | size
$builder
The @audit command displays a list of objects you own in the order that you created them. This command lets you change that so that your objects are sorted by object number or by size. Once this is done, however, there is no way to go back to having them sorted by when they were created.

**@spell** <any number of words> | <object>.<property> |
<object>:<verb>
$frand_class
Checks the spelling of a sequence of words, the words in an object's property (the property must be a string or list of strings), or the quoted parts of a verb. (This command may be disabled on some MOOs.)

**@spellm*essages** <object>
$frand_class
Checks the messages (all properties whose name ends in _msg) on the specified object for correct spelling. See also help spelling.

**@spellp*roperties** <object>
$frand_class
Check all properties on the specified object for correct spelling. Properties that are not a string or a list of strings will be ignored. See also help spelling.

**@spurn** [!]<object>
$frand_class
Prevent an object or any of its descendents from entering your inventory. Used with the exclamation point, this command removes an object from your list of spurned objects.

**@spurned**
$frand_class
Displays a list of spurned objects.

**@ss*how** <object> | <object>.<property> | <object>:<verb>
LM player class #5803
A short version of @show.

**stop-following** <player>
LM player class #8855
Cease programmatically following <player> wherever e goes. See also **follow**.

**@subscribe** [<mailing list>]
**@subscribe*-quick** [<mailing list>]
$mail_recipient_class
Subscribes you to a mailing list. If you type @subscribe without specifying a mailing list, then the system will print out all the lists to which you are not subscribed, along with their descriptions. @subscribe-quick, without specifying a mailing list, will print out only the names of the mailing lists to which you are not subscribed, i.e. the lists' descriptions are omitted.

**@subscribed**
$mail_recipient_class
Displays a list of all mailing lists to which you are subscribed, whether or not they have new messages on them. See also @rn.

**@suggest*ion** [<text>]
$mail_recipient_class
Sends MOOmail containing <text> to the owner of the room you're in as a suggestion. If you do not specify text in the command line, then you are moved to the mail room to compose your message (presumably at greater length).

**@sweep**
$player
This verb searches your local environment for objects that might be relaying information. It omits objects and verbs owned by yourself or by a wizard. Programmers wishing to customize what is displayed by their objects when someone uses the @sweep command should add a :sweep_msg verb.

**@teleport**
See @move.

**@tell-filter**
LM player class #33337
Displays information about the tell-filter object in use, if any. See @set-tell-filter.

**@toad** <player> [graylist | blacklist | redlist] [<comment>]
**@toad!** <player> [graylist | blacklist | redlist] [<comment>]
**@toad!!** <player> [graylist | blacklist | redlist] [<comment>]
$wiz
A wizard-only command. Deactivates a player object's status *as* a player, but does not recycle the object. The player's owned objects are left in the database as orphans, so it's a good idea to @audit em first and @recycle the objects listed. If used with one exclamation mark, the victim is also @blacklisted. If used with two exclamation marks, the victim is @redlisted. The optional comment is included in a post to *site-locks. See also help @toad and help @blacklist.

**@tutorial**
LM player class #322
Starts a tutorial of basic MOO commands. Type quit at any time to discontinue it.

**@typo** [<text>]
$mail_recipient_class
Sends MOOmail containing <text> to the owner of the room you're in, to report a typographical error. If you do not specify text in the command line, then you are moved to the mail room to compose your message (presumably at greater length).

**@unban** <player or object> | everyone
LM player class #322
Cease banning someone or something from all rooms you own. See also @ban, @banned.

**@undef*ine** <label>
LM player class #8855
Remove a definition for eval. See @define.

**unfollow** <player>
LM player class #8855
Stop following <player> wherever e goes. See also follow.

**@ungag** <player or object>
$player
Cease @gagging a player or object. You will once again see text originating from em or it. See @gag.

**@ungag-site** all | last | <guest> [<date>]
LM player class #322
Cease @gagging a site associated with a guest. See @gag-site, @gag-sites. Specify the date if you used @gag-site for guests with the same name on different occasions.

**@unlock** <object>
$builder
Clear any lock you may have placed on the object. See @lock, and help locking.

**@unmess*age** <message-name> [from <object>]
$builder
Remove a message property from an object you own (defaults to yourself). (Normally only programmers can add and remove properties. But anyone can add or remove a message.) See @newmessage.

**@unread** <msg> [on <recipient>]
$mail_recipient_class
Reset your message pointer, as if you haven't yet read the specified message on the specified mailing list.

**@unrefuse** <actions> from <player> | <actions> | everything
$frand_class
Cease @refusing specified actions. See @refuse.

**@unrmm*ail** [list | expunge] [on <recipient>]
$mail_recipient_class
When you remove a MOOmail message from yourself or a mailing list using @rmm, it isn't really deleted from the database, but rather is saved in a sort of limbo as a zombie message. The main purpose of the @unrmm command is to undo the removal of a message, restoring it to yourself or a mailing list. You can also use this command to view a mailing list's associated zombie messages, or to expunge any zombie messages so that they are well and truly gone forever.

There are a few idiosyncrasies of this verb that the formal syntax, while correct, doesn't make very clear. First, notice that while we remove messages from a list, we unremove them on a list. Second, unremove is an all-or-nothing proposition – you can't specify a message sequence. For a specified recipient, you list *all* zombie messages, restore *all* zombie messages, or expunge *all* zombie messages. See also help @unrmmail and help zombie-messages.

**@unsend** [<message-sequence>] from <player>
$mail_recipient_class
This command enables one, in some circumstances, to retract a post that one has sent to a player. There are several exceptions. See help @unsend. It was added to LambdaMOO in 1999, and therefore is not present in older versions of the core database.

**@unset-tell-filter*-hook**
LM player class #33337
Removes any tell-filter that is in use. (See @set-tell-filter.)

**@unsubscribe** [<list or lists>]
$mail_recipient_class
Unsubscribes you from the specified mailing lists, or your current mailing list if no mailing list is specified.

**@unsubscribed**
**@unsubscribed-quick**
$mail_recipient_class
Prints out the names and descriptions of all mailing lists to which you are not currently subscribed. The quick version prints out names only.

**@uptime**
$player
This command displays the amount of time since the last system restart.

**@users**
$player
Lists names of all connected players, in alphabetical order. See also @who.

**@verb** <object>:<verb name(s)> [<dobj> <preposition> <iobj>
[<permission flags> [<verb owner>]]]
$prog
Adds a new verb to an object. If more than one alias is given to the verb, then the names should be separated by spaces and all enclosed in double quotes, e.g., @verb me:"fee fie fo fu*m". Default argument specifiers can be specified with @prog-option. The verb owner defaults to yourself; only wizards can specify a different verb owner. You must own an object or be a wizard in order to add a verb to it.

**@verbs** <object>
$prog
Prints a concise list of the verbs defined on an object. See also @display.

**@verify-owned**
$builder
Verifies that your owned objects are all, in fact, owned by you. (A situation where this might not be the case could occur when an inexperienced wizard tried to change an object's ownership by manually setting its .owner property rather than using the wizard-only @grant command.)

**@version**
$player
Prints out the version number of the currently-running MOO server, and the date that the database was extracted from the core.

**@watch** [<player> | none | off]
LM player class #33337
This verb notifies you when the watched player ceases to be idle, i.e., when e types something. With no arguments, tells you whom you are watching. With none or off, turns watching off.

**@ways** [<room>]
$frand_class
Lists a room's obvious exits and their aliases.

**ways**
LM player class #5803
Lists all of the obvious exits from your current location.

**@web** me is [<web information>]
$frand_class
This verb lets you specify web information about yourself (e.g. your home page) for others to view. If <web information> is omitted from the command, it shows you your current web information. Programmers may access others' web information either via a player's .web_info property or :web_info verb. This verb and its associated properties are not included with the LambdaCore (even though $frand_class is included).

**where*is** [<player> [<player(s)>]]
**@where*is** [<player> [<player(s)>]]
$player
Lists the name(s), object number(s), location(s) and location object number(s) of the specified player or players. If no argument is given, then lists all players.

**wh*isper** "<text>" to player
$player, $frand_class, LM player class #7069
Communicates <text> to the specified player. Other players in the room do not see the exchange. The version on $frand_class permits @refusal of whispers. The version on LM player class #7069 stores the identity of the person who most recently whispered something to you for use with its respond verb, **=**. See also murmur, @refuse, **=**.

**@who** [<player> [<player(s)>]]
$player
Shows the names, numbers, idle durations and locations of the specified players, or all players if none are specified. Many different versions of this verb have since been written; some player classes provide ways to select which version of @who you prefer to use. See also @users.

**@will** recycle <item designator>
**@will** bequeath <item designator> to <player>
**@will** refuse <item designator> to <player>
**@will** keep <item designator>
**@will** display
**@will** forget <item designator>
**@will** clear
LM player class #322

This command provides a variety of ways for one to specify how one wishes one's objects to be disposed of in the event that one is reaped. See `help @will` on LambdaMOO for more detailed information.

**@witness** [on]
**@witness** off
**@witness** show [<number>]
**@witness** display [<number>]
**@witness** delete <number>
**@witness** email <number>
**@witness** publish <number>
LM player class #322

This command provides a way to log conversations. Witness logs cannot be modified, even by the person who is doing the logging. See `help @witness`.

**@wizards** [all]
$player

Lists connected (or all) wizards.

**@wrap** [on | off]
$player

If the words you see get cut off at the right edge of the screen, this means that you are either using telnet, or, for some other reason, you don't have word-wrap. `@wrap on` causes the MOO to perform word-wrapping for you. This command is used in conjunction with the `@linelen` command, which tells the MOO how long a line may be before it is wrapped to the next line. `@wrap off` discontinues this behavior; you may need to do this if you switch from using telnet to using a client program. Typing @wrap with no arguments will tell you whether word-wrapping is currently on or off.

'<player> <text>
LM player class #8855, LM player class #33337

This command is a short cut for the `page` command. Though not part of the core, it has been ported to various other MOOs. (The version on #33337 permits you to omit <player> to respond to the person who paged you most recently.)

**?** [<topic>]
$player

This is a short cut for the help command.

!<text>
LM player class #5803

This command lets you vary the forms of your statements if you get tired of beginning everything with your name. If your name is included anywhere in <text>, then <text> is displayed as you typed it in. Otherwise, your name is appended to the end as an attribution. See help !. Here are a couple of examples (with Yib doing the typing):

```
!Two thumbs up!
```

displays:

```
Two thumbs up!      --Yib
```

```
!A coconut cream pie sails into the room and smashes into
Yib's face!
```

displays:

```
A coconut cream pie sails into the room and smashes into
Yib's face!
```

#<string>[.<property>|.parent] [e*xit | p*layer | i*nventory] [for <code>]
$prog

Prints information about the object named by <string>. This is a very powerful shortcut for some of the things that the eval command does. In particular, it enables you to look at properties of an object without having to know its object number in advance. Properties can be chained in sequence. Here are a few examples:

```
#rock.color_list
```

displays the .color_list property of rock.

```
#yib p
```

displays Yib's name and object number.

```
#yib.description p
```

displays Yib's description – foils look-detection.

```
#yib.location p
```

displays Yib's location.

```
#yib.location.description
```

displays a description of Yib's location.

```
#yib.location.owner p
```

displays who is the owner of Yib's location.

See help #.

**+**<player> <text>
LM player class #5803

This is the remote-emote command. It lets you display <text> to <player> as if it were an emote, but you do not have to be in the same room with em.

**=**[<text>]
LM player class #7069

Responds with <text> to the player who most recently paged you. Used with no argument, displays the player to whom you are ready to respond.