Chapter 8 – LambdaMOO-Specific Reference Information

LambdaMOO, the original and largest MOO in existence, has been extended by its users in a marvelous variety of ways. This chapter attempts to document some of the highlights. It is divided into three sections: Feature Objects, Player Classes, and a detailed description of LambdaMOO's political system and how to use it.

A Short Compendium of LambdaMOO Feature Objects

There are around a thousand feature objects on LambdaMOO, which makes a thorough survey impractical. There are some, however, that I've come to think of as "standards of the jazz repertoire", in a manner of speaking, either because of their popularity or usefulness or both, and it is those that I choose to present here.

It's not uncommon for a feature object to have a variety of unrelated verbs, the theme between them being merely the author who wrote them. I have written about those verbs that inspired me to choose these feature objects as examples; you should read the help text for a particular feature object (FO) to see what other commands it offers.

The section is divided into three main categories: popular social feature objects, informational feature objects, and utilities.

Popular Social Feature Objects

#30203 (Stage-Talk Feature)

Stage talk is also known as *directed say*. This FO lets you direct speech to a particular person. To use it, precede a player's name with a dash. If I type:

```
-Plaid_Guest Hi.  Welcome to LambdaMOO.
```

then everyone in the room will see:

```
Yib [to Plaid_Guest]: Hi.  Welcome to LambdaMOO.
```

This FO also provides a way to simulate pointing to yourself and saying something. If I type:

```
<In a silly mood, today.
```

then everyone in the room will see:

```
Yib <- In a silly mood, today.
```

The Stage-Talk feature is provided as part of LambdaCore.

#40842 (Social Verb Core and Feature Object)

This is a very popular set of "short cut" verbs for frequently used gestures. The verbs can be used either alone or directed at a player or other object in the room. If I type:

```
wave eep
```

then I see:

```
You wave to eep.
```

eep sees:

```
Yib waves to you.
```

The other people in the room see:

```
Yib waves to eep.
```

The social verbs provided by this FO are:

comfort	cry	nod	hug
wink	poke	grin	kiss
yawn	shrug	smile	french
wave	blush	laugh	bow
cackle	cringe	sigh	
giggle	smirk	chuckle	

You can type `social` to see a list of these verbs online. Several of the verbs have text which modifies them, for example `nod` depicts you as nodding solemnly.

The help text explains how to customize these verbs if you don't like the modifiers provided by the FO itself.

Versions of this feature object are frequently found on other MOOs, but are not standardized.

#21132 (Antisocial Feature)

This verb is a natural follow-on to the social verb core. The syntax is basically the same, except that you can't use the verbs by themselves – you have to designate a target. On the other hand, you can specify more than one target, for example:

```
eye Klaatu Kirlan Boo
You eye Klaatu, Kirlan, and Boo warily.
```

The available anti-social verbs are:

`eye`	`pat`	`eyeball`
`grump`	`paperwork`	`mess`
`feh`	`poke`	`wake`

rtfm	pout	sycophant
ignore	pave	smoke
waggle	BillyBragg	salad
Blake	bop	report
glare	EPOXY	Snideley
toy	waffle	sledge
pound	postage	tile
silly	mmt	wither
gag	rael	alpo
frown	cough	crush
prod	ice	deconstruct
unplease	flame	smell
WhiteRabbit	roll	stern
unamused	divine	shoebox
peer	disembowel	twist
snooty	Buddy	thwap
face	defenestrate	whuggle
growl	neuter	Wholeflaffer
tongue	handcuff	Sapphos
ridicule	hoop	value
bury	blame	reverse-@eject
sigh	dist	boot

I recommend trying them out with a good-natured friend.

#4572 (APHiD's Socializing Feature Object)

This verb is an extension of `emote`. It enables you to direct any gesture to a particular person (or to everyone in the room) rather than having to rely on those provided by other feature objects or having to customize a message in advance. The command begins with a period (`.`) followed by the first person singular of any verb (no space in between) followed by the rest of your text. If I type:

```
.hit Nim up for some chocolate.
```

I will see:

```
You hit Nim up for some chocolate.
```

Nim will see:

```
Yib hits you up for some chocolate.
```

Everyone else will see:

```
Yib hits Nim up for some chocolate.
```

If you have this feature and type `@social-option +all`, then you can use the syntax:

```
.wave to everyone
```

Each person will see a message listing everyone in the room except that the word "you" will be substituted for eir name in the message.

There are other Feature Objects that offer variations of this verb, which is sometimes referred to as *posing*.

#5490 (Dancing Feature Object)

I've included this feature object in my compendium because it's fun and because it's one that people frequently ask about. It provides a selection of dances which you can perform either solo or with a partner. It provides a verb, `@polite`, which lets you be polite (ask, first) or more impulsive when you dance with someone. You can type `@dances` to see the choices available.

```
Werebull hands Yib a rose which she places between her
teeth.  Then Werebull leads Yib through a rhythmic tango,
stepping across the floor and ending with Werebull holding
Yib in a low dip.
```

This feature object was originally programmed by APHiD.

Informational Feature Objects

People are naturally curious about the world around them. Or nosy, depending on your point of view. The properties and verbs on just about everything are readable, and I am continually amazed at the number of ways that programmers find to combine disparate data and draw unexpected conclusions. There is a slowly evolving tug of war between the snoopers and the snooped-on, as well. As verbs query ever more extensively, counter-verbs are written to detect and/or deflect various queries. As detection verbs become known and come into more widespread use, snooping verbs become more stealthy in their methods, and on it goes.

This section is divided into three parts, based on the kinds of things the FO's provide information about: People (players), places, and things.

Feature Objects that Primarily Provide Information About People

#24222 (login watcher)

The login watcher informs you when other players connect to or disconnect from the MOO. It provides a way for you to designate individual players as "interesting", and then you will see messages like the following:

```
< connected: Tartan_Guest.  Total: 192 >
< disconnected: Elephant_Ears.  Total: 186 >
```

The `@wwho` verb on this feature object shows you an `@who` listing of only those players you consider interesting (instead of everyone on the MOO).

Your @interesing list is technically private (the property where it's stored is unreadable), but the *act* of adding someone to or removing someone from your @interesting list can be detected, as can the use of @wwho. (See also help #14141:@sint.)

#24262 (Fast & Dangerous Info FO)

This is the most comprehensive informational FO that I know about, though it gets occasional criticism for not being stealthy enough.

```
@fbi
```

will give you a detailed report with lots of information about a player.

```
@kgb
```

will do essentially the same thing but with less chance of triggering someone's detection verbs, if e has any.

Both of these verbs are resource hogs, and the FO provides a large suite of more specialized verbs that you can use if you only want to know a particular thing about someone and not eir entire life story. These are:

@@ <player or players>	Gives quick information about one or more players' location(s).
@aka <player>	Lists someone's aliases.
@morphs <player>	Lists someone's morphs.
@nohelp <object>	If an object doesn't have formal help text per se, this verb will list all the help for the individual verbs.
@xref <player>	Shows some relational information about <player>, for example whether you own kids of eir generics or use feature objects e owns, etc.
@bdays	Shows you the names and ages of players whose birthday is today. (Uses info from the birthday machine in the Living Room.)
@cwo	Lists @interesting players who **c**onnected and disconnected **w**hile you were **o**ut.
@enemies <player>	Prints a list of people with whom <player> "might have strained diplomatic relations". This might include persons that <player> has @gagged, @refused or @banned, for example.
@fields <topic>	Prints a list of specific fields known to the helpful person finder, about which you may then make further inquiry to identify a helpful person. Example: @fields newbie

`@fusers <feature object>`	Prints a list of players who use the specified feature object.
`@holding <player>`	Prints an easy-to-read list of the items the specified player is carrying in eir inventory.
`@horniness <player>`	Offers information about the kind of sex that `<player>` seems to be horny for, if any. This is determined by `<player>`'s `.horny` property (if present), which can be either a string giving the object number of a player (e.g. "`#50222`"), any string (e.g. "vanilla"), or a one-letter code. The codes are as follows: "S"(traight), "G"(ay), "B"(I), "F"(arm, i.e. animals), or "X" (bondage, discipline, sadism or masochism). A person will only be reported by this verb as being horny if e is registered with the birthday machine as being 18 years of age or older.
`@intruders` `@intruders <player>`	Lists people who are in rooms you own or in rooms that `<player>` owns.
`@kusers <object>`	Shows all kids of the specified object and their owners.
`@spy <player>`	Shows a description of the room `<player>` is in, plus who's with em.
`@log` `@log <player>`	Shows recent login and logout activity for the specified player.
`@oc`	Gives an **o**nline **c**ount of how many players are connected and maximum new logins allowed.
`@on`	Shows a list of everyone online. Uses shortest aliases and tries to fit it all on one screen.
`@opals` `@opals <player>`	Gives a list of **o**nline **p**als of the specified player. (This verb is useful because different player classes have different ways of designating pals.)
`@reapable`	Shows you whether any of your pals or anyone you have designated as `@interesting` is up for reaping soon.
`@recon <room>`	Prints information about a specified room, including its description, the names and genders of connected players in the room, and some information about the room's security setting, if available.
`@sen <player or players>`	Provides information about players' seniority based on MOO ages.

`@status <player>`	Shows abbreviated information about a player's status on the MOO. See `help #24262:@status` for an explanation of the various fields and abbreviations.
`@subscribers`	Shows a list of players who subscribe to a list with notification. (The identities of players who subscribe to a list without notification are not available to this FO.)
`@using #24262`	Gives online information about all the verbs on this FO, including the syntax for each verb.
`@wf` `@wf <location>`	Think ("**w**here **f**rom") Lists players from a specified location. Uses `#5365` (Real Life Registry). (See `help #5365`.)
`@xpr`	Lists the "e**xp**e**r**ience levels" of all connected non-guest players sorted into five MOO age categories, "dinos", "fogies", "oldies", "middies", and "newbies". See `help #24262:@xpr` for definitions of these terms.
`@youths` `@youths!`	Prints a list of players believed to be 16 years old or younger (according to the birthday machine). The second form lists both connected and unconnected players.
`@geezers` `@geezers!`	Prints a list of players believed to be as old as or older than Haakon (#2) (according to the birthday machine). The second form lists both connected and unconnected players.
`ii <player>`	Prints a player's inventory, with object numbers.
`wias with <string>`	**W**h**i**ch (player) **a**liases **s**tart with a specified string? Useful for screening possible names you might want to add.
`woas with <string>`	**W**h**o**se **a**liases **s**tart with a specified string? This version is much spammier than `wias`, giving each player's name and number and the alias in question that e goes by.

#36714 (Carrot's Social Interaction Feature)

This feature object shows information about players' MOO ages, offline ages (as provided to and by the birthday machine), and also provides some verbs that list players' friends and cliques.

#1283 (Karma FO)

This FO is strictly for fun, but is popular. It gives you the same information about yourself that you can get by sitting and meditating in the Japanese Garden, and also will give you similar information about other players. There is a note on the reference shelf in the library that explains how one's karma is determined.

Feature Objects that Primarily Provide Information About Places

#23824 (Compass Rosette Feature Object)

This is an *extremely* useful FO for exploring – a very good adjunct or alternative to the `@ways` command. The `@rose` command can be customized in a variety of ways. This feature object has been ported to many MOOs, with the author's permission.

#46278 (Carrot's Viewing Feature)

This FO provides the `@dview` command, which capitalizes the names of details in a room's description or a detail itself. One might argue that this is cheating, but on the other hand, it might lead people to look at some details that they might otherwise not have bothered even to look for.

#41975 (Obvious Features Object)

This feature object offers a potpourri of verbs. One that is very useful for exploring is `@lrs` (also `@lrs <room>`) which stands for "long range scan". It presents a tiered list (three levels) of exits and destinations accessible from the room you're in or the room you specify.

Feature Objects that Primarily Provide Information About Objects

#10218 (Mazer's Object Utilities Feature Object)

This FO is a nice collection of commands for investigating the object hierarchy. It includes `@ancestors`, `@descendents`, `@descendents_suspended`, `@branches`, `@branches_suspended`, `@leaves`, and `@leaves_suspended`.

#113366 (LambdaMOO Museum Search FO)

The number of generic objects on LambdaMOO has become almost mind-bogglingly large. The `@msearch` command helps you narrow your search. You could, for example, type `@msearch puppet`, and after the search was complete you

would be directed to specific displays in the museum showing objects that have the word “puppet” in their name.

Utility Feature Objects

#35353 (Pasting Feature)

This feature object is included with the LambdaCore. You can use the `@paste` command to display multiple lines of text to everyone in the room you’re in, or `@pasteto <player>` to display multiple lines of text to one person. In both cases, the system prompts you to type in lines of text, followed by a period on a line by itself. In lieu of typing, of course, you can paste in text from another window. Note, `@paste` is sensitive to lag and sometimes the text doesn’t appear immediately. Output from the pasting feature object looks like this:

```
----------------------Private message from Frebblebit----------------------
This text was generated with the
@pasteto command.
-------------------------------end message--------------------------------
```

#25552 (Multi-communications feature)

This is the feature object that enables players to talk on channels. It has extensive help text. The basic commands are:

`@channels`	Lists all channels.
`@xcon <channel>`	Quit one channel and start listening to another.
`@xsw <channel>`	Continue listening to your current channel, but talk on and listen to a new channel.
`@xsilence <channel>`	Quit listening to and talking on a channel.
`@xm <text>` `xm <text>`	Broadcast `<text>` on your current channel.
`@xm: <text>` `xm:<text>`	Emote `<text>` on your current channel.

`@xwho`
`@xwho <channel>`
`@xwho all`
	Shows who is listening on your current/designated channel, or all channels.

#65000 (Quota-Transferral Feature)

In 1994, `*Ballot:Quota_Transfers` passed, enabling players to transfer quota to one another. This feature object is the mechanism by which quota transfer is accomplished. The syntax is:

```
@qt <amount> to <recipient> <optional reason>
```

You must be at least one month old to transfer quota. Transfers are posted to the public mailing list `*Quota-Transfer-Log`.

Popular LambdaMOO Player Classes

The section on player classes in Part I summarized all the commands on all the player classes that are provided as part of LambdaCore. This section outlines the commands available on the most popular LambdaMOO player classes.

LambdaMOO has over sixty player classes to choose from. For this section, I have chosen classes from the player class hierarchy that are used by about 70% of the LambdaMOO population.

Details of the syntax and usage of each command are given in Appendix A. Simple summaries are presented here to give a sense of what each of these player classes has to offer.

Generic LambdaMOO Citizen

All players on LambdaMOO are members of the Generic LambdaMOO Citizen player class. People who run other MOOs might want to consider putting a citizen player class between Frand's player class and generic builder.

Some of these commands were mandated by ballot, others are utility commands relating to structures that are specific to LambdaMOO, (e.g. the RPG system).

`@tutorial` – Transports you to the tutorial, where you can practice a variety of basic commands.

`@ban`, `@ban!` – Universally bans an object from all rooms that you own. The second form also bans any children of the banned object.

`@unban` – Stop banning a person or thing from rooms you own.

`@banned` – List those players and things you have `@banned`.

`@gms` – Prints a list of RPG Game-Masters

`@make-petition` – Create a petition object.

`@petitions` – List existing petitions.

`@ballots` – List ballots. With no arguments, lists open ballots. Can also take any of "`passed`", "`failed`" or "`all`" as an argument.

`@petition-options` – Displays and/or sets various options relating to petitions, including whether or not you want to be automatically informed about open ballots.

@nominate – A command to nominate a player to elected office. The elected boards are the Architectural Review Board (ARB), the reapers, and the registrars.

@arb-petitions, @reaper-petitions, @registrar-petitions – List petitions of people currently nominated for the Architectural Review Board, reapers, and registrars, respectively.

@arb-ballots, @reaper-ballots, @registrar-ballots – List the specified open ballots.

@boot – A command which permits players to boot a guest off the system. Two players must act in concert, and a reason must be given. The offending guest's site is blocked for 1 hour.

@witness – A set of commands (see `help @witness`) to reliably log an exchange between players, view such logs, and publish such logs to a mailing list.

@age – Displays a person's age, with deductions for system down time. (The deductions are an enhancement to the generic player's `@age` command.)

@arb, @reapers, @registrars – Display a list of current Architecture Review Board members, current reapers, or current registrars, respectively.

@flush-cache – Delete one's feature cache. Feature caching was mandated by ballot #71687, although an inspection of the code suggests it may only have been partially implemented.

@will – A command to specify how you would want various objects you own to be distributed, if you were to leave and never come back.

@email – LambdaMOO has an email policy which prohibits the use of solely free anonymous email addresses. Existing users with such accounts must provide an *identified* email address (one whose owner's name is known or knowable), either their own or a relative's, in order to gain full access to LambdaMOO. `@email` is the command by which such users can provide a valid email address. (See `help @email` for more information.)

@gag-site <guest> for <duration> – Gag a guest and all guests from the same site as the specified guest.

@ungag-site – Stop gagging all guests from a particular site

@gag-sites – List the guests whose sites you have gagged.

Generic Player Class With Additional Features of Dubious Utility(#7069)

This player class was programmed by Gary_Severn (#15), who was also the wizard Dukas. There is no help text for it, per se. I presume it was programmed before adding help text to player classes and generic objects was as established a convention as it is today.

seek – This command tries to combine the convenience of teleporting with the appearance of walking. If you `seek` a person, it tries to find an entrance into the

room e's in and move you through it. It only sort of works, but the concept is good. (There is an updated version of this on FO #27325:@seek.)

`Another feature of #7069 that isn't a command per se:`

The title verb affixes an idle status to your name when someone looks at a room you're in.

Experimental Guinea Pig Class with Even More Features of Dubious Utility (#5803)

(See also help 5803-index.)

`@ss/@sshow` – A short version of `@show` that doesn't list properties or verbs.

`ways` – Very like `@ways`. Shows a short list of obvious exits. Unlike `@ways`, it doesn't let you specify a location, but only shows you exits from the room you are in at the time. My speculation is this: This command existed first, then the more flexible `@ways` command was added to Frand's player class at a later time and `#5803:ways` was rendered obsolete but no-one thought to remove it.

`!` – A polite spoof verb. `!<string>` announces an arbitrary string to your current location, except that if the string doesn't contain the your name somewhere as a distinct word, your `.spoof_attribution` property (with the substitutions of player.name for `%n` and `%` for `%%`) is appended and if the resulting string *still* doesn't contain your name, then your name is appended anyway.

`@lastlog` – This verb takes two forms:

```
@lastlog <player> <player> <player>
```

is just like `@who`, except that it shows disconnected players before connected players.

```
@lastlog for <number> <day/week/month>[s]
```

(e.g. `@lastlog for 2 days`) lists all players who have logged on during the specified interval.

`boring` – `boring on` makes you immune to food fights; `boring off` allows you to participate again. Subsequent to the creation of this command, at least one other object was modified to respect the `.boring` property: If you are `.boring`, then things will not fall out of your pockets down between the Living Room couch cushions.

`@pedit` – A property editor for properties that are other than strings or lists of strings. The verb documentation describes it as "experimental".

`party` – Looks for various rooms with more than one player, tells you who's there, how long they've been idle, and asks if you'd like to go there.

`heartbeat` – Starts up a task to print out the time every `<n>` minutes from now on unless you've been active during that minute. Good for people who like to stay

idle for long periods of time and who want to have some means of time stamping occurrences that show up in their client buffers. (This verb is actually not callable from the command line; one has to write a separate verb to call it, or use `eval`.)

+ – This is the remote-emote command. It is kind of like a regular emote in that you specify your actions in the third person, but you specify the player who will see it, and you don't have to be in the same room as em. If I am in Yib's Study, and I type `+boo beans you with a water balloon`, then Boo would see: `(from Yib's Study) Yib beans you with a water balloon.`

eprint – This is a command that will mostly be of interest to programmers. Type `eprint <long-horrible-complicated-moocode-expression>`, and it will format said expression to fit within `.linelen` columns in a way that's (usually) more readable.

@prettylist – This command lists a verb with indentations that are intended to make it easier to read. Its author (Rog) acknowledges that it is slow.

@nprop – The author of this verb (Rog) writes:

> The only difference between `@nprop` and `@prop` is that `@nprop` does a full eval of its value argument, whereas `@prop` only accepts literals; e.g.,
>
> ```
> @nprop foo.bar x:contents() rc Wizard
> ```
>
> It may also be that I never moved this to #217 because I decided that doing a full eval wasn't a good idea anyway (if you need to do it, then you may as well just do `;foo.bar = <whatever>`).

@nlist – The author of this verb (Rog) writes:

> `@nlist` seems to differ from `@list` in accepting an `upload` argument, which causes the listing to appear in a form that can then be edited and then blasted back by a client. This has been superceded by local editing (i.e., `@edito +local`, `@edit foo:bar`) and so is likewise not terribly useful anymore.

page – This is a fancier version of the original page command that enables you to page more than one player at a time. If you do, you must enclose the list of player names in quotes:

```
page "Klaatu Nim Bartlebooth" Party at Club Dred!
```

Player Class hacked with eval that does substitutions and assorted stuff (#8855)

This player class has a hodgepodge of commands, some of which are strictly of interest to programmers, but others of which are of a more general interest. See also `help pcs-index`.

follow – This command is not to enable you to follow someone, but to allow someone else to follow you, although since most LambdaMOO players are descended from this player class, pretty much anyone can follow anyone.

stop-following – A person would use this command to stop following you.

@list-followers – List the people who are following you.

lose – `lose <name>` will cause `<name>` to cease following you. `lose everyone` will cause everyone to cease following you.

'<player> <text> – A shortcut for paging someone. Append the name of the player you want to page directly after the `'` character:

```
'Klaatu Hi!  It's been ages!  How have you been?
```

@parent – Tells you an object's direct parent (as opposed to a list of its ancestors, which is what `@parents` does).

@define, @undefine, @listdefs – For those who program, the `@define` command lets you pre-define one or more variables, which are then in place when you run the `eval` command (i.e. globals). `@undefine` lets you remove definitions, and `@listdefs` lists what you currently have pre-defined.

Politically Correct Featureful Player Class Created Because Nobody Would @Copy Verbs To 8855 (#33337)

This player class enables you to refuse a couple of additional actions from players. **@refuse flames from <person>** inhibits your reading posts from said person on public mailing lists. Instead of the body of the message, you will see something like:

```
[ Flame from ReaDinG (#61050) skipped: 133 lines. ]
```

If curiosity gets the better of you, you can either `@unrefuse flames` from the person, OR use `@peek` to look at the message and see what it actually says.

mu – This stands for "murmur". It's an alternate form of whisper, whose syntax is like page. `mu Yib Shall we withdraw to the drawing room?` is the same as `whisper "Shall we withdraw to the drawing room?" to Yib`. The advantage of the first over the second is that it's shorter to type, and you don't have to put the text to be whispered in quotes.

@watch – This command tells you when an idle player becomes active, again. When the player does become active, the system prints eir name enclosed in square brackets on a line by itself:

```
[Yib's_Assistant]
```

It is possible for a player to detect when you start watching em (by modifying eir `:title` verb).

goto – This command has been disabled, but is documented here as an item of historical interest. One could type `goto <location>` and be caused (if possible) to walk to the location using exits rather than by teleporting. The verb has a comment indicating that it was disabled because it was "too spammy".

'<player> <text> – This is an enhanced version of the shortcut page verb. It records the last person you paged, so that subsequently you can omit eir name:

```
'Kirlan Fee, Fie, Fo
' Fum!
```

Kirlan would see both lines as a regular page from you.

@pc-news – This was a way for the author of this player class to simulate a new news item, but only for members of this player class.

@set-tell-filter, @unset-tell-filter, @tell-filter – A tell filter is an object which can intercept text that you are about to see and modify it in some way before you see it. One might use one, for example to prepend the name of the person typing, if it doesn't appear in the text itself, or to add a time stamp to every line. `@set-tell-filter` designates such an object. `@unset-tell-filter` ceases to use your current tell filter. `@tell-filter` tells you what tell filter you are currently using.

@pc-options – This player class uses an options package similar to the mail-options package. The `@pc-options` command shows you what the options are and how you have them set. In particular, you can choose to set a crosspost limit (ignore messages on mailing lists if they've been cross-posted to more than a specified number of lists). You can also have the system ask you for confirmation before displaying a long mail message.

Here's an odd note: The Schmoo class, #4803, has a shout verb. #33337 isn't Schmoo class, but was able to intercept Schmoo shouts. The verb `:who's_infidel_slime` fingered those non-Schmoos who intercepted Schmoo shouts. See also `help pc-index`. I don't know why it's called "Politically Correct".

Generic Super_Huh Player (#26026)

This player class offers the very helpful facility to "note down" object numbers for future reference. In *some* cases you can refer to these objects by name instead of number. For example, if you type `@remember #2031`, which is Yib's Guide to Interesting Places, you can type `read guide` at any time and read the current page, even if the guide is on its shelf in the library. Unfortunately, this only works some of the time (because of how the parser makes decisions about what object you really mean when you type a command).

@remember – This is how you "note down" an object's number for future reference.

@known – Lists those objects that you have opted to remember

@forget – Remove an object from your list of known objects.

Detailed Player Class (#6669)

The detailed player class lets you define a detail on yourself, so that if your description mentions a snazzy belt buckle, you can make it so that the command `look buckle on <you>` tells the person looking some details about the belt buckle.

`@detail` – This command lets one add, change, or remove a detail. Type `help #6669:@detail` for the various syntaxes.

`@details` – This command lists the details you have defined on yourself.

Other Player Classes

Here we come to the major branch between general player classes. Most people describe the division as being between `SSPC (#49900)` (a descendent of `Sick's Sick of Spam player class (#59900)`) and `Super-Schmoo (#4803)` (a descendent of the `Piping Player Class (#20781)`). Different players find different advantages in each. (The reason why an object can be a descendent of an object with a higher number than itself is because object numbers are recycled and re-used.)

Feature objects were a later invention than player classes, so in the early days the commands available to you were mostly a function of what player class you chose. This all happened before my time, but by many accounts there was some strong competition between player class authors to garner users. The two branches provide several of the same functions but do so in different ways. These include morphing, keeping a list of pals so that you can see which of your pals are logged on, etc., and answering machines (a much later addition). The Schmoo player class also provided the facility to define various descriptions associated with various states of undress, and has what is called “Schmoo shouting”, enabling users of the Schmoo player class to broadcast remarks to one another.

The major drawback to fancy player classes, but particularly ones that provide morphing, is that they take up a lot of quota. Aside from that, it’s probably just a matter of taste regarding the syntax of the commands provided. Most of the functionality is the same, now, between the two branches, and with few exceptions most new commands are provided via feature objects, instead.

An Overview of LambdaMOO’s Political System, and How to Use It

This section outlines the history of how LambdaMOO has been governed from its inception to the present, and gives a detailed description of how to use the mechanisms of the ballot system that we currently have in place. (For those who are

curious about what kinds of issues arise to vote on, a compendium of all closed ballots (as of May 10, 2003) is provided in Appendix E.)

Historical Overview

The birth of LambdaMOO is generally recognized as the date on which Haakon (#2), the ArchWizard, first connected, which was October 31, 1990. At the very beginning, LambdaMOO consisted of just a few friends, and the differences that arose were simply worked out informally. As the MOO grew in size, it ceased to be a place where everyone knew everyone else. When differences arose, people tended to turn to the wizards to help them sort out their squabbles and/or make an arbitrary decision, and the wizards tended to try to assist. Eventually, the wizards asked Haakon to get them out of the "discipline/manners/arbitration" business, and in December of 1992, Haakon posted "LambdaMOO Takes A New Direction", (see Appendix C) in which he decreed that the wizards would henceforth serve strictly as systems programmers and that players at large were now free (whether they wanted to be or not) to sort out their own differences.

Some time later a character going by the name of Mr_Bungle depicted some other characters as brutally sodomizing themselves in the Living Room. This was decried by many as abusive and intolerable, and there was much discussion on public mailing lists (mostly `*Social-Issues`) of what to do. In this particular instance, a wizard eventually took it upon emself to `@toad` Mr_Bungle, but the fact that the populace had no *mechanism* with which to govern themselves suddenly became clear. Haakon then created a petition and ballot system, intended to provide a way for the citizenry to express their collective views to the wizards and compel them to take specific technical actions intended to have social consequences. (See `help petitions-motivation` for some additional details about the particulars of how the petition system was implemented.) Haakon's original vision was that the petition/ballot system as given would be used by the populace to bootstrap a more sophisticated system. This has not yet come to pass.

The first petition to pass as a ballot was `*Ballot:Arbitration`. It called for the creation of a system that players could use to resolve disputes among themselves, including a system for volunteer arbitrators and a system for making "minor" changes to the arbitration system itself without having to go through the larger petition/ballot process. Every petition is subject to vetting by the wizards as one of the requirements it must meet before being promoted to a ballot, and today that petition would probably be denied vetting for being "insufficiently precise for the wizards to know how to implement it." `*B:Arbitration` was authored by a wizard, though, and everyone was optimistic, and no one foresaw the pitfalls that were in store. As it turned out, the arbitration system was fraught with difficulties, and often used for "playing games with the system" rather than in a sincere effort by players to resolve their differences with one another. `*B:Arbitration` was repealed in February of 1999; as of May 10, 2003, no conflict resolution system has yet been legislated to take its place.

The wizards tried very hard to restrict themselves to purely technical actions and not take any actions that would have social consequences, but with LTAND Haakon had, in fact, made a promise on the wizards' behalf that was impossible to keep. Many technical decisions have social consequences, and a choice not to execute some technical action can also have social consequences. Too often the wizards found themselves in a position of "damned if they did and damned if they didn't," and the damnations finally became more than the wizards were willing to bear. In May of 1996, the LambdaMOO wizards collectively authored a document titled, "LambdaMOO Takes Another Direction", commonly known as LTAD (see Appendix D for the full text). This document acknowledged that the line between technical and social decisions often is not a clear one, acknowledged that the wizards had (of necessity) made decisions in the past that had social consequences, and acknowledged that they would continue to do so. LTAD formally reintroduced wizardly fiat. In an effort to counterbalance the reintroduction of wizardly fiat, the wizards also announced the creation of a special "standing" petition, `*P:Shutdown`. Should it pass as a ballot, the wizards pledge to shut LambdaMOO down for good. `*P:Shutdown` has come to ballot (and failed) on two occasions. This special petition is pre-vetted, and furthermore requires only a simple majority to pass. The reason for requiring only a simple majority is that most of the wizards felt that if more than half didn't want LambdaMOO to keep going, they'd just as soon pack it in, themselves. (The author of this book was a LambdaMOO wizard at the time and participated in the crafting of LTAD.)

A Citizen's Guide to the LambdaMOO Petition System

But First, For Those Who Just Don't Want to be Bothered

LambdaMOO's petition system has a variety of mechanisms in it to ensure that players are adequately notified of open ballots and current elections. For some, this is too much of a good thing. Those who do not wish to be notified or lobbied should type the following commands:

```
@petition-option +noannounce
@petition-option +noannounce_ARB
```

New ballots and ARB elections are announced in *news. In addition, each time a player logs in, e is notified of ballots and/or ARB ballots on which e has not yet voted. These two petition options suppress these notifications. (You will still see the *news entry.)

```
@refuse politics for <duration, e.g. 100 years>
```

Refusing politics is an advisory act rather than a prohibitive act in that people are supposed to refrain from lobbying you if you have `@refused` politics, but this is not programmatically enforced. An older system depended on players adding a `.apolitical` property or `:apolitical` verb to themselves, and some players still utilize this method. See `help apolitical`.

To see if someone else has refused politics, type:

```
@refusals for <player>
```

The Particulars

Only primary, non-guest, non wizard characters may participate in the petition system. The voting age is 30 days from the time of first connection.

A player must be at least one year old to run for the Architecture Review Board.

A player must be at least four months old to run for the office of reaper.

A petition must acquire at least 10 signatures before it may be submitted for vetting.

The number of signatures required for a petition to be promoted to a ballot is 10% of the average of all votes for and against all previous closed ballots, not to be fewer than 50. The number of signatures required to promote a nomination petition to a ballot is 50. For a nomination petition to be promoted to a ballot, the nominee must be one of the signatories (indicating acceptance of the nomination).

A regular ballot requires a 2/3 majority of yes/no votes to pass. There is no minimum number of votes, i.e. no quorum.

For elected offices, those players who receive more "yes" votes than "no" votes are eligible, and ballots are ranked, in decreasing order, by the difference between the number of "yes" votes and the number of "no" votes. Of those, the top <n> are selected, where <n> is the number of vacancies on the board for which elections are being held. (See `help @ARB-ballots`.)

Votes are secret. Upon closure of a ballot, the object numbers of those who voted yes or no are stored on a randomized list, though which way a person voted is not stored. The list of those who voted is readable only by wizards, and is kept only for reference should there be a case involving accusations of multiple-character voting.

Time limits for petitions and ballots:

From first creation to author's signature	No expiration.
From author's signature until submission for vetting (10 or more signatures required)	14 days.
During vetting	No expiration. Signatures may not accrue while a petition is waiting to be vetted.
From vetting to acquiring enough signatures to promote a petition to a ballot	90 days.
From promotion to a ballot to ballot closure	14 days.

Should the system crash, the clock is considered to be stopped. Petitions and ballots have the duration of the down time added to their expiration times; players have the down time subtracted from their age for purposes of participating in the political system. (This is why the `@age` command gives a second (younger) age "for official purposes".)

Vetting Criteria

Before a petition may become a ballot, it must be vetted by the wizards. The vetting criteria are as follows:

A petition must be:

- Appropriate according to the guidelines given in `help petitions` (see below).
- Sufficiently precise and detailed in its description of the desired effect or facility that the wizards can understand how to implement it. (It is best, however, if you do not specify a particular implementation.)
- Technically feasible for the wizard to implement.
- Not likely, in the wizards' opinion, to jeopardize the functional integrity of the MOO.
- Not likely, in the wizards' opinion, to bring the wizards, Pavel Curtis, Stanford University, or Placeware, Inc. into conflict with any real-world laws or regulations.
- Consistent with passed ballot #55018, which states, "No petition may call for any change which results in differential treatment between those who sign it and those who do not."

The following is set forth in `help petitions`:

> Petitions and ballots are for proposals that the wizards perform some set of purely technical actions that only wizards can perform, where those actions are intended to address some social goal. For example, the actions might be intended
>
> - To directly achieve some social goal (such as banishing some individual from the MOO or removing some person from the list of wizards).
> - To grant some piece of wizardly power to the general MOO in some form that's intended to be used to address social problems in the future (such as creating a truly escape-proof jail or giving players a way to temporarily banish each other from the MOO).
> - To restrict some or all players in some way that is believed will help to achieve some social goal (such as keeping guests from logging in anonymously or making it impossible for players to print out messages containing certain words.
> - To modify the petitions and ballots mechanism itself in some qualitative way (such as changing it to require fewer signatures on petitions or only a simple majority on ballots).

Petitions and ballots are not for simple requests that the wizards fix some bug in the core nor, at the other end of the spectrum, are they for proposals that involve the wizards' having to exercise some kind of social judgement role (such as demanding that the wizards banish anyone who's being abusive).

Participating as a Voting Member of the Populace

Petitions and ballots are objects, owned by the system character, `Petitioner (#4)`. As such, each has its own object number, by which it can always be referenced, but one may also reference a petition or ballot as follows:

```
*Petition:<name or alias>
*P:<name or alias>
*Ballot:<name or alias>
*B:<name or alias>
*ARB-Petition:<player name>
*ARB-P:<player name>
*Reaper-Petition:<player name>
*Reaper-P:<player name>
```

The generic ballot is a child of the generic petition, and so all commands that can be used on or with petitions can also be used on or with ballots, though some may not apply and are disabled as appropriate. (You can't sign a ballot, for example, but you can list signatures on petitions *or* ballots.)

A petition's or ballot's status is recorded in its description. This includes the number of signatures acquired and required, whether it is up for vetting, expiration date, etc. As per the naming conventions listed above, you can type `look <petition or ballot>` to get some basic information about it.

To see a list of petitions or ballots, type one of the following:

```
@petitions [all | public | signed | vetted]
@ballots [all | open | closed | passed | defeated]
@arb-petitions
@arb-ballots
@reaper-petitions
@reaper-ballots
```

Use the commands `decline <petition>` and `undecline <petition>` to control which petitions display when you use the `@petitions` command.

You can type `read <petition or ballot>` to view its text. You can mail a petition or ballot's text to yourself by typing `mailme <petition or ballot>`. Wizards sometimes add implementation notes to petitions they have vetted. You can read these notes (if present) by typing `impl <petition or ballot>`.

Petitions and ballots are also mailing lists. You can read mail on them and send mail to them just as you would any other mail recipient.

To sign a petition, type `sign <petition>`. If you have a feature object that depicts you holding up a sign with text on it, you may need to use one of the alternate forms of this command, `@sign <petition>` or `psign <petition>`. You can remove your signature from a petition at any time by typing `unsign <petition>`.

If you are one of the first ten people to sign a petition, i.e., if it has not yet been submitted for vetting, then ideally you are signing to indicate not only that you think the measure deserves to be brought before the LambdaMOO populace as a ballot, but also that you have reviewed it and believe that it meets the vetting criteria.

To vote on a ballot, type one of:

```
vote yes on <ballot>
vote no on <ballot>
abstain on <ballot>
```

You may change your vote as many times as you wish, up until the time a ballot closes. Note, you must manually cast your vote even if you have signed the petition – a vote is not automatically cast for you.

There is an `@petition-options` package for petitions that works along the same lines as `@mail-options`, `@edit-options`, and `@display-options`:

`@petition-options`	Display all your current petition-option settings.
`@petition-option <option>`	Display the current setting for a particular petition option.

`@petition-option petition_order=created`
`@petition-option petition_order=written`
`@petition-option petition_order=vetted`
`@petition-option petition_order=written`
`@petition-option petition_order=stages`
	Controls the order in which petitions are displayed.

`@petition-option signature_order=signing` `@petition-option signature_order=name`	Controls the order in which petition or ballot signatures are listed.
`@petition-option +noannounce` `@petition-option –noannounce`	Controls whether or not you will see an announcement when you log in of open ballots on which you have not yet voted.
`@petition-option +no_announce_ARB` `@petition-option -no_announce_ARB`	Controls whether or not you will be notified when you log in of ARB ballots on which you have not yet voted.

`@petition-option subscribe=query`
`@petition-option subscribe=always`
`@petition-option subscribe=never`
	Controls whether you will always or never be subscribed to a petition's mailing list when you sign it, or asked if you wish to subscribe.

`@petition-option subscribe_ARB=query`
`@petition-option subscribe_ARB=always`
`@petition-option subscribe_ARB=never`
	Same as above, except for ARB nominating petitions.

The LambdaMOO mailing list `*committee` receives automated notifications of new petitions and changes to the status of existing petitions and ballots.

Creating Petitions

This section describes the sequence of steps for creating a petition and getting it promoted to a ballot.

The first step is to make a petition object using the `@make-petition` command. The syntax is similar to `@create`:

```
@make-petition <name>,<alias>,<alias>, ... , <alias>
```

This will create a petition object, owned by Petitioner (#4). The petition and its associated mailing list do not come out of your quota. You may `@rename` your petition at any time without losing signatures. (Note, this is not the same as giving your petition a new title (see below), which *does* erase all signatures.) I recommend giving your petition a short name (even an abbreviation) because the is the name by which people will reference it. You will have a chance to put a longer line of descriptive text in the title.

You may only have one petition at a time.

At any time up until the petition is promoted to a ballot, you may type `burn <petition>` and it will be recycled.

Next, give your petition a descriptive title. Use the retitle command even when you are giving your petition a title for the first time:

```
retitle <petition> as <descriptive title>
```

The first time you give your petition a title, it won't have any signatures on it. You can retitle it later, if you choose to, but all signatures on it will be erased.

Next, edit the text of your petition by typing `@notedit <petition>`. Editing the text will also erase all signatures. Many petition authors tend to sign their petition at this point and start collecting feedback *and* signatures (no one may sign a petition until its author has signed it). I recommend against this, because then every time someone makes a good suggestion and you change the petition, all the signatures are lost and you have to collect them all over again. One or two rounds of

this isn't bad, but more than that and people tend to lose enthusiasm. Instead, I recommend that you type:

```
post <petition>
```

This will cause it to be listed whenever someone invokes the `@petitions` command, even though you haven't signed it, yet. You can type `unpost <petition>` at any time if you change your mind.

After you have gotten as much feedback as you think you're going to at this early stage, and incorporated as many changes into the petition's text as you choose to, then is the time to sign it and start asking others to sign it, too. Once a petition's author has signed it, the clock starts: You must get at least 10 signatures in order to submit it for vetting, and you have fourteen days in which to do this. If you can't get ten signatures (including your own) within that time, the petition will expire and automatically be recycled.

Once you have ten signatures, you may then submit the petition for vetting by typing:

```
submit <petition>
```

A mail message indicating that you have requested vetting will automatically be sent to the petition mailing list and to `*wizards`. At this point, the clock stops: Petitions do not expire while awaiting vetting. Neither can they gather more signatures.

Once the petition is vetted, the clock starts again. The next thing to do is gather the remaining signatures needed to promote your petition to a ballot. The number usually hovers around 50. You can find out exactly how many more signatures it needs by looking at it. It's important to remember that not everyone is open to being lobbied to read and sign pending legislation. Before approaching someone, you should type `@refusals for <player>` and not approach em if e is refusing politics. If you are using a program to lobby people, it should include a call to:

```
$code_utils:verb_or_property(<player>, "apolitical");
```

and not bother em if you get a truth value. You have 90 days to obtain the requisite number of signatures to promote the petition to ballot status, or else it will expire and automatically be recycled. (As with the number of signatures, you can see how much time remains on the petition by looking at it.)

Once your petition is promoted to ballot status, you are then free to create a new petition.

In addition to the `@petition-options` outlined above, there are a couple of options that petition authors can set on petitions themselves:

`@options on <petition>`	Display current option settings.
`@set-option notify on <petition>` `@unset-option notify on <petition>`	If this is set, you will be notified whenever someone signs or unsigns the petition.

`@set-option autosubmit on <petition>` `@unset-option autosubmit on <petition>`	If this option is set, the petition will automatically be submitted for vetting when it acquires 10 signatures.

Nomination Petitions

Elections are announced on `*news`. To nominate someone, type:

```
@nominate <player> for <office>
```

The possible offices are ARB, Reaper, or Registrar.

Nomination petitions do not require vetting. They are promoted to ballot status automatically when they get 50 signatures, but only if the nominee emself has signed it, indicating that e accepts the nomination.

Keeping Up with it All

The mailing list `*committee` provides a daily update that includes (as applicable) new petitions, burned or expired petitions, changes to a petition's status (e.g. submitted for vetting), promotions to ballot, and signature changes.