Glossary of Terms

`@` – By convention, the first character of a command that transcends the MOO's virtual reality. One of many descriptions of a MOO is "a text-based virtual reality". MOOs have themes, or general motifs, such as "a large mansion and its grounds". Within this text-based virtual reality, players can talk with one another, gesture (emote), move from room to room, pick up and drop various objects, and so on. These are sometimes referred to as *VR actions*. There are also several commands that one can issue that break with or transcend the virtual reality: `@who` lists all the players who are logged on, for example, and `@join` moves you to someone else's location on the MOO. I call these commands *meta-VR*.

`#` – Every object on a MOO has a unique number, which is indicated by the "#" sign. There is much you can do without paying attention to object numbers, but an object can always be referred to by its number. If you are not holding an object or in the same room with it, then (with a few exceptions) you *must* refer to it by number for the system to figure out which object you mean.

`ARB` – The Architecture Review Board. In December, 1991 the LambdaMOO wizards created the Architecture Review Board to assist them in assessing who should receive more quota to build with. In July, 1993, a ballot passed that made the ARB an elected body.

`alias` – Every valid object in the MOO has a name. Objects may have additional aliases, which are other names (often shorter) that can also be used to refer to an object. An object named "`a big, black, hairy spider`" might have the alias "`spider`", for example. One way to see an object's aliases is to `examine` it. You can add an alias with the `@addalias` command; you can remove one with the `@rmalias` command. See also `help @rename`.

`arguments` – arguments are pieces of information that are provided to a command, program, verb or subroutine so that it can do its job. If you look at the room you are in by typing the word `look` by itself, we say that you have invoked the `look` verb with no arguments. If you type `look hat`, then the word "hat" is an argument to the `look` command. If you type `look rabbit in hat`, then the words "`rabbit`", "`in`" and "`hat`" are arguments to the `look` command. In such a case, "`rabbit`" is the direct object, "`in`" is the preposition, and "`hat`" is the indirect object. Some commands always take the same fixed number of arguments. For example, the command `@go` always takes one argument, the room to which you wish to travel. Other commands can take an arbitrary number of arguments. For example, `go` needs at least one argument, but is able to take several. `go north` will move you from your current location through the exit named "`north`", if there is one. If you are in the Living Room on LambdaMOO, typing `go north east east up east north` will move you through successive exits until you arrive at the library.

`argument specifiers` – When a programmer first creates a verb on an object, e must, as part of the command that creates the verb, specify what arguments the verb takes if any. This is done by typing a word that stands for the direct object of the command, a word that stands for a preposition, and a word that stands for the indirect object. If I am creating a trophy, and want to write a verb to award it to someone, I might type the following:

```
@verb trophy:award this to any
```

"`Award`" is the name of the verb itself. "`This`" means that when someone types the award command, the trophy ("`this`") will be in the position of the direct object. "`To`" is the preposition, signifying that the trophy will be awarded *to* someone, not from or about em, for example. Last, "`any`" indicates that the trophy can be awarded to anyone or anything. The words "`this to any`" are the argument specifiers.

`background task` – A foreground task is a task that executes "while you wait". There are some things that the computer does independent of a person typing in a command and waiting to see the result. Suppose I have a MOO timer, and I type the command `set timer for 5 minutes`. The computer might print to my screen, "`Timer started`" and that exchange represents a completed foreground task. I proceed to chat with my friends on the MOO. Each invocation of `say` and `emote` also represents a (short) foreground task. Meanwhile, the timer program is counting off the seconds, up to five minutes, without tying up my screen. That is, I don't have to wait the five minutes before I can type something else. The counting off of the seconds is said to be accomplished "in the background", or is described as a *background task*. When the five minutes are up, the timer prints the line, `DING! 5 minutes are up!` to my screen, and the background task is ended. (See also **task**.)

`bash` – An offline party or other gathering of MOOers. Sometimes bashes are given a qualifying prefix. "NYE-bash" would be a New Year's Eve bash. "Sushi-bash" would suggest some MOOers going out for sushi together. Usually the term "bash" implies that any MOOer who is told the time and place is welcome to attend.

`'bot` – 'Bot is short for "robot", and it typically refers to something that acts like a player but isn't a player. There is a further refinement, which is whether the bot is an actual *player object* or not. Some people have written programs that log on to LambdaMOO (using a player name and password). These programs maneuver the player-object through the MOO, and are programmed to recognize and react to conversation and perhaps other text generated by human typists. Another kind of 'bot is a non-player object within LambdaMOO that is controlled to a greater or lesser extent by a human typist but which is not in fact a player object. These are sometimes also called *puppets*. This kind of 'bot is easier to identify, because if you examine it, you'll see that it is not its own owner. Real players own their player objects; puppets and other automata do not.

`built-in function` – A program is a collection of commands which are executed in a particular order. These commands can either be other programs (often called "subroutines"), or any of a subset of commands that are intrinsic to the system (in other words, commands that are provided by the server rather than written in MOO code). Some examples of built-in functions:

```
length(<item>)
players()
connected_players()
```

`byte` – A byte is a unit of computer storage space. Typically, one letter of the alphabet takes up one byte of space.

`call` – Suppose you are making chocolate soufflé. Recipes have several steps, and sometimes refer to other recipes, e.g., "Make a béchamel sauce (see page 257)." Similarly, commands (verbs) can – and usually do – consist of several steps, often referring to other verbs. These other verbs are sometimes called *subroutines*, and when a verb asks the computer to execute one, we say that the verb *calls* the subroutine.

`channel` – Many MOOs have a special communications facility called channels. A fair analogy might be to compare channels to Citizens' Band radio. One connects to a particular channel, and can then listen to everyone who is talking on the channel and can transmit on the channel without being co-present in the same room(s) as others who are also connected to the channel.

`character` – A *character* is another name for a *player*, which is an object that represents a typist within the context of the MOO. Sometimes the two terms are combined: *player-character*. The distinguishing feature of a player-character object, unlike every other kind of object, is that the built-in system function `is_player(<object>)` returns 1.

`child` – With the exception of the root object (#1), every valid object has all the same properties and verbs as another object, said to be its parent. An object is said to be a *child* of that other object. An object can have many children but only one parent..

`class` – When an object is intended solely to serve as the parent of other objects rather than being used itself, it is referred to as a *generic* or as a *class*. Things that have that object as a parent or ancestor are said to be of that object's class.

`command` – A command is a word, sometimes with accompanying arguments, that a typist enters with the intention of obtaining some result or causing some effect. All commands are verbs, but not all verbs are commands. This is because some verbs are only meant to be called from within other verbs and not directly by a typist. Examples of commands are:

```
look me
@who
put rabbit in hat
```

`command line` – When you type a command, you are said to type it *at the command line*. Some verbs (e.g. `say`, `emote`, `page`, `@join`) are intended to be invoked as commands, and these are sometimes called *command line verbs*. Other verbs are meant to be called only from within other verbs. These are called *subroutines*.

`connected` – A room is said to be *connected* if you can get to it without having to teleport.

`contents` – Every valid object in a MOO has a property that designates its location (given in terms of an object number), and another property that designates its contents: things whose location is in turn the object in question. If A contains B, then B appears in A's `.contents` property, and the value of B's `.location` property is A. A refinement to this concept is that the notion of containment extends to contents of contents. Think of nested boxes. If A contains B and B contains C, then even though C doesn't appear directly in A's `.contents` property, C is said to be in A's *containment hierarchy*. Objects cannot contain themselves, nor can two objects contain each other simultaneously or otherwise violate the containment hierarchy. (The MOO has no formal way to designate sizes of things, so a tiny little jewel box can easily contain a hippopotamus!)

`core` – Also referred to as the *core database*. A brand new MOO that is just up and running consists of two major pieces, the server and the core database. The server is the actual program that runs on the host computer. The core database is that set of objects within the MOO that every MOO starts with. The core database includes several objects that are sets of utility programs, written in the MOO programming language, which are used in making more complex objects and MOO programs. The core also includes #2 (the player that is the ArchWizard), the system object, the generic room, the generic exit, and various other items. The core includes a verb, `#0:core_objects`, that defines the list of objects that comprise the core database. This book confines itself to MOOs based on LambdaCore. Other available cores include JHCore and EnCore.

`data types` – The MOO programming language recognizes six different kinds or *types* of data: integers, decimal numbers (also called *floating point numbers*), character strings, objects, error codes, and lists.

`database` – When used in the context of a MOO, the *database* refers to the collection of *all* the existing objects along with their associated properties and verbs. The server loads the database to run the MOO.

`defined` – Every verb is associated with an object, either a player, a player class, a place (room), or a concrete object. It is associated with this object when it is first programmed. (In the case of rooms or players or articles, the verb usually manipulates that object in some way.) We say that a verb is *defined* on the object to which it was attached at the time it was programmed. We care about which object a verb is defined on when we want to list it to see how it works, or perhaps fix a bug (if the verb is defined on an object that we own).

`descended from` – An object is said to be descended from another object if the other object is in its chain of parents.

`expression` – An expression is a combination of characters that, when evaluated as a piece of MOO code, generates a value.

`exit` – An exit is a special kind of object that doesn't exist in any particular place per se (its actual location is usually `#-1 ($nothing)`, but that is associated with a room, its *source*. When you are in a room, there are what are called *obvious exits*. When you type the name of an obvious exit, you are transported to that exit's destination, and that is referred to as *invoking* an exit.

`fertile` – Every valid object on LambdaMOO has a property that indicates which object is its parent. An object's initial parent is specified when it is created. You may create a child of an object you don't own if and only if the (potential) parent object is fertile. For an object to become fertile, its owner must make it fertile using the `@chmod` command.

`flag` – A flag is a kind of variable whose value is either TRUE or FALSE.

`foreground task` – A task that executes "while you wait", typically the result of typing in a command. See also **background task**.

`forked task` – This is another name for **background task**.

`fork bomb` – A fork bomb is a program that generates more and more forked tasks until the system is overloaded. The system becomes terribly lagged, then grinds to a halt. A fork bomb is usually viewed by a MOO's wizards as a "denial of service attack" and its perpetrator, when found, may have eir programming privileges revoked. On rare occasions, a new programmer will generate a huge number of forked tasks inadvertently. If the wizards believe this to be the case, there is usually no punishment beyond disabling the offending verb and asking the programmer to be more careful in the future.

`gag` – To gag someone is to indicate to the system that you do not wish to see any text that results from the gagged player's typing anything. Other people will still see the gagged player's output. See `help @gag`.

`game master` – In the LambdaMOO RPG, one who has programming privileges. (See also **`Grand_Master`**.)

`generic` –An object that exists solely to be a parent of other objects.

`Grand_Master` – A character on LambdaMOO who owns all RPG objects and has access to statistics otherwise unreadable to other players (except wizards). One or more typists may have and use this character's password. Grand_Master is used only for RPG administrative purposes and occasional debugging. (See also **`game master`**.)

`gurst` – In LambdaMOO parlance, a gurst is a guest whose typist has a registered player-character. Some players log on as guests in order to circumvent noise abatement measures. Some do so to rediscover the joys of truly anonymous MOOing or simply for a change of pace. Some do so because they have for some reason temporarily or permanently lost the ability to connect as their regular player-character. Opinions differ on whether unruly behavior or an intent to deceive are necessary in order for a guest to be considered a gurst.

`idle` – To remain connected without interacting. It is used as both an adjective and a verb: AcidHorse is idle. AcidHorse is idling. It is not unusual for a person to emote `:idles` before paying attention to another window or leaving eir keyboard while still connected to the MOO.

`input` – information that goes into something, usually a computer program. A program that sorts numbers, for example, might display the prompt, `Enter some numbers to sort:` The numbers that the user types in would be the input. (The sorted list of numbers would be the program's output.)

`invalid` – MOO objects are said to be either valid or invalid. An object is valid if someone has created it or recreated it (with the `@create` or `@recreate` commands respectively). An object is invalid if it has been recycled, or if it has never been created in the first place. (E.g. object #99999999999999999999 is not a valid object (as of this writing).) There are a few objects that exist but are not valid. These usually have negative object numbers (e.g. #-3), and are used by the system to designate various error conditions.

`inventory` – Those objects that a player is holding or carrying.

`invoke` – To cause a command to be executed by typing its name. If there is an exit in your vicinity named "north" for example, you are said to invoke the exit when you type `north.`

`lag` – A (usually unexplained) delay in the system's response time. Normally when you type something in, especially if you're simply saying or emoting something, the associated text prints out on your screen right away, and if you're in a room full of active players, text appears at a fairly steady rate. There are a variety of reasons why someone else's text might be delayed in its appearance: e might have been called away from the keyboard unexpectedly, e might be thinking about eir response, or typing in a long line, or multi-tasking, or it might just be because of lag. A sure sign of a lag storm is when your *own* text is delayed in appearing. Sometimes lag isn't on the MOO itself, but is in the network somewhere between the MOO and a typist's computer. This is usually referred to as *net lag*. It is characterized by some MOOers experiencing lag and others not.

`LambdaCore` – A core database derived from LambdaMOO. (See also **core**.)

`$limbo` – The location to which a player is returned when e logs off if any of the following apply: eir home is `$player_start`, eir home won't accept em as a resident, or eir home is invalid.

`list` – A list is a particular way of representing a set of things in the MOO programming language. The elements of a list may be numbers, strings of characters, objects, other lists, or any combination. Lists are designated using curly braces "`{}`", and their elements are separated by commas. Example: `{"This", "is", "a", "list", "of", "strings", "."}` The empty list is a meaningful construct in the MOO programming language, and is designated with just the curly braces: `{}`.

`lurk` – To read a mailing list without posting, to stay in a room without saying, emoting or otherwise contributing anything, or to listen to a channel without speaking on it.

`matching` – Matching refers to the system associating a name you type in with an object's unique number. If you page a particular player (i.e. `page mockturtle Don't you ever sleep?`), the code for the "`page`" command matches `mockturtle` to mockturtle's object number, and forwards the message accordingly. If you are holding a rock and type `drop rock`, the system will match the word "rock" with the rock you are holding and move the rock from you to the room you are in. If you are holding more than one rock, the system will be unable to match the word "rock" to a unique object, and you would see, `I don't know which "rock" you mean.` If you weren't holding a rock at all, The system would display, `I see no "rock" here`. (The first case is referred to as an *ambiguous match* and the second is a *failed match*.) In general, in order for the system to match a name that you type with a unique object, you either have to be holding the object or in the same room as the object. Exceptions to this include many of the commands that refer to players (e.g.

`page` and `@join`) since player names are unique, and also the `@go` command, which can look for the name of a room in a player's `.rooms` database.

`mav` – According to the FAQ found at http://www.mudconnect.com/mudfaq/mudfaq-p1.html#q30, Mav was a TinyMUDder who would sometimes accidentally emote something to an entire room when he meant to whisper or page it. The word has come to mean any mix-up between `say`, `emote`, `remote-emote`, `whisper`, `page`, etc.

meta-VR – Actions or commands that "break" the virtual reality and call attention to the fact that you are using a computer program and not a real (i.e. tangible) mansion, castle, cave system, what-have-you. By convention, most non-VR commands on MOOs begin with the "`@`" sign, although there are exceptions to this. Some examples of non-VR commands would be `@who`, `@join <person>`, `@send <person-or-mailing-list>`.

`morph` – A morph is an alternate presentation that a player-character may adopt, while temporarily storing eir original name, gender, and description. An alter ego. Did you ever notice that you never see Super Man and Clark Kent together at the same time? They might be morphs of the same person. While a player's name, description, gender, and messages may change when e morphs, eir object number (and password) remain the same.

`multiple characters` – Many people choose to have more than one character on a MOO. Multiple characters are different from morphs in that there are two (or more) separate player objects with different object numbers; it is more difficult for players to determine that multiple characters are controlled by the same typist than it is with multiple morphs. LambdaMOO permits multiple characters, but they must be registered as such. Among other things, only one of a typist's multiple characters may participate in the political system there.

`multi-tasking` – This term refers to a computer that is executing more than one task at the same time. Informally, a person is said to be multi-tasking if eir attention is divided between two more simultaneous activities.

`newt` – The act of metaphorically turning a player into a newt or, a player who has had this done to em. The `@newt` command, available to wizards only, blocks a player's access to the system, either for a specified or indefinite period of time. This action is typically taken when the wizards believe a player might be a threat to the system. It is occasionally done for punitive reasons or for noise abatement. It is possible for a player to effectively newt emself, using the `boot_player()` built-in function within a custom `:confunc` verb. (See also **toad**.)

`non-VR` – See **meta-VR**.

`object` – Objects are the fundamental building blocks of an object-oriented system. On a MOO, every object has a unique number (prefixed by the "#" sign), a name, an owner, a location, and a property listing its contents.

`options package` – An options package is a set of commands and data values that enables you to customize some aspect of your MOOing experience. There are many options packages that govern how different things work for you, specifically. Examples of these include `mail-options`, to customize various aspects of sending and receiving mail, `edit-options`, which customize certain aspects of how the editors work, `builder-options`, and `programmer-options`. Options packages can be associated with player classes, generic rooms, feature objects, or any other kind of object. There is no single definitive way to list all the options packages available to you, but they are usually referenced in other objects' help texts. `Help options` will give you a list of some of them.

`output` – The result of (usually) a computer program, which is displayed to one or more users. Sometimes one refers to a person's output, meaning that which the person produces, either manually or with the aid of a program.

`parameters` – Limits, usually numeric, that are set in advance. One speaks of "working within a set of parameters." Parameters are not the same as arguments: If one had a program to sort numbers, the arguments would be the numbers to sort. A parameter might be the maximum number of arguments the program could or would accept.

`parent` – See **child**.

`parser` – When you type a line of text, the system has to figure out which segment of what you typed constitutes the command, and identify the verb to run, the direct object, the preposition, and indirect object, if present. The part of the system that does this is called the parser.

`player, player-character, player object` – An object on a MOO that represents the VR embodiment of a human typist.

`player class` – An object on a MOO that serves as a repository for additional commands that a player might choose to use. A player adopts a player class by changing eir parent to the player class. Implicit in this act is adopting all ancestors of the selected player class as well. There is a calculated risk in adopting a player class which is that player class owners could theoretically intercept private communications, and are able to change some of a player's fundamental attributes. Such incidents are rare, but one should know the risks going in. (See the section on player classes beginning on page 43 for more about this.)

`$player_start` – The location where guests, new players, and players without an otherwise-valid home find themselves when they log on.

`port` – (Think "transport".) To record all the particulars of an object on one MOO and use the information to recreate it as exactly as possible on another MOO. One should ask permission of an object's author before porting it.

`primary character` – On some MOOs, human typists are permitted to have more than one player character. On LambdaMOO, only one of these has the rights of citizenship (authoring and signing petitions, voting, etc.). The one with the voting rights is referred to as one's primary character. Others are referred to as *secondary characters*. MOOs that support multiple characters sometimes have a registry to differentiate primary and secondary characters, which only wizards can access. By convention, it is the prerogative of the typist and no one else to disclose secondary character information. If someone shares such information, it behooves one to treat it as privileged.

`program` – A sequenced set of instructions that a computer follows slavishly. Programs can be very simple or quite complex or anything in between. On a MOO, the terms *program* and *verb* are frequently used interchangeably. (One programs a verb, but does not verb a program, however.)

`programmer` – A person who programs computers. In a MOO, wizards grant what is called a *programmer bit* which changes a player's `.programmer` property from 0 to 1. The system then recognizes the player as empowered to write MOO programs. When contrasted with the term *user*, this term more specifically refers to the person who wrote the program that the user is using.

`property` – A property is a named piece of data associated with an object. Within the system, properties are designated by the object number followed by a period (`.`) followed by the property's name. When speaking of a property independent of the object it might be associated with, the object number is sometimes omitted. For example, "Every valid object has the following properties: `.name`, `.location`, `.contents`, `.owner`." Players may change the values of many properties on themselves and on objects they own. Programmers may add new properties to objects they own. Properties are used to store data that are needed or wanted after a verb has finished executing, or for data that are needed by more than one verb. (See also **`variable`**.)

`puppet` – An object that may impersonate a player, whose "hearing" may be monitored by a player and whose responses may be controlled by a player, but which is not in fact a player. Butlers and bartenders at various venues are likely to be puppets.

`queue` – A queue is a list of tasks that are scheduled to run at a later time.

`reap` – To expunge a player-character from a MOO, typically because e has been inactive for a long time.

`reaper` – Traditionally, only MOO wizards have the power to reap a player. On LambdaMOO, certain non-wizard players are entrusted with this task. LambdaMOO reapers are elected.

`response latency` – The delay between a real time communication to a player and that player's response. Reasons for a lengthy response latency might be thinking before answering, composing and typing in a long response, being away from the keyboard, or system **lag** (which see).

`return value` – When a verb is called or an expression is evaluated, it always returns a value. The *return value* of the expression `2 + 2` is `4`, to give a simple example. The return value is not always the entirety of the result – sometimes a verb may also have a *side effect*. In many cases, the return value is either 1 or 0, signifying, "The operation was successful," or, "The operation was not successful," respectively. The operation in question is whatever the verb is supposed to do.

`room` – An object that is descended from `$room`. More loosely, an object that players can enter and which looks, sounds, smells, and feels like a room (i.e. it might as well be a room).

`.rooms database` – A property on a player that is a list of object numbers and names or abbreviations for rooms. One's `.rooms` database is used in conjunction with the `@go` command, so that you can type `@go library`, for example, without having to remember the library's object number. You can type `@rooms` to see your own `.rooms` database. You can modify your `.rooms` database with the `@addroom` and `@rmroom` commands.

`RPG` – Role Playing Game. Some MOOs have within them a role playing game akin to the early adventure/dungeon games from which social MOOs arose. On LambdaMOO, this game is referred to as an "area", though various sets of rooms might be tightly or loosely connected. Briefly, one becomes "initiated", a process by which one acquires a surrogate (called a "doll" or "voodoo doll") on which are recorded one's victories and defeats over various and sundry RPG opponents. You train to increase skill and then venture forth to seek treasure, fight various monsters, and so forth.

`secondary character` – Some MOOs permit a typist to have more than one player character. Usually the first or oldest of these is designated as one's primary character, and others are referred to as secondary characters. On LambdaMOO it matters especially because only primary characters have the rights of citizenship (authoring petitions, voting, etc.).

`server` – The program, written in the C programming language (as it happens) that, when running, is the MOO. This program accepts connections from players logging on, reads what players type in, and responds accordingly. When you type something in, part of the program that's running (the parser) figures out what command you've typed and which object(s) you're trying to manipulate, and then causes the appropriate function or verb to be executed.

`shouting` – There are two usages of the term "shouting". One is to say or emote something in all capital letters. The other is to broadcast something to everyone who is logged on, even though they aren't in the same room. An appropriate use of the latter would be for a wizard to shout that e is about to reboot the system for some reason.

`side effect` – Some verbs just return a value: an arithmetic calculation, the name or location of an object, etc. But other verbs do things in addition to returning a value, such as announcing text to a room, or changing something in the database. These additional things are called *side effects.*

`spam` – Copious amounts of unwanted text whose volume is so great it renders its content useless or pointless.

`spoof` – To cause unattributed text to appear on other people's screens, or the unattributed text itself. There are three general forms. In one, no player's name is included: "The chandelier falls to the floor with a crash!" In another, the name of the player perpetrating the spoof does appear in the text, but not at the beginning, and another player's name might appear at the beginning instead. The classic form of this is, "Werebull causes Yib to fall down laughing," (with Yib causing this text to appear, not Werebull). Some players vehemently object to this form of spoof; others take it in stride. It is in fairly common use. The third form is particularly offensive and considered officially unmannerly on most MOOs, and this is text that depicts a player doing or saying something, which text the depicted player did not emself type in: "Yib produces a previously unseen puke green wiffle bat and proceeds to bash herself several times over the head with it." (Where some unnamed stinker caused this text to appear and Yib didn't.) There is no programmatic way to prevent players from spoofing, but there are a few different ways to detect it, including tell-filters and the combined commands `@paranoid` and `@check-full`.

`string` – A sequence of letters, numerals, or punctuation marks or any combination thereof. When depicted, a string is enclosed between double quotation mark characters, e.g., `"chocolate souffle".`

`subroutine` – Some verbs are called not from the command line but from within other verbs. These are called subroutines. Suppose you wanted to make a chocolate soufflé. The recipe might begin, "Make a béchamel sauce (see page 257)." You would turn to page 257, follow the directions for making a béchamel sauce, then return to your place in the soufflé recipe. Executing a subroutine is like making the béchamel sauce.

`syntax` – A generalized expression of the correct usage of a command or subroutine.

`system` – This is a general term that is used to refer to a program that is running or a set of programs working together. It can mean the MOO itself, as in, "What did the system respond when you typed `@parents me`?" or it can refer to the operating system on the machine on which the MOO is running, as in, "The system will be shut down in two hours for its ritual Saturday night bath."

`system character` – A player object that does not actually have a human typist associated with it. System characters are typically used to serve as the owner of record of objects associated with one or another project. On LambdaMOO, for example, the system character "Petitioner" owns all petitions, ballots, and related objects.

`task` – Many players can use a MOO at once. The system receives text that a player types, processes it in some way, and then (usually) prints text to the player's screen in response. Whenever a player types in a command and the system executes it, that is called a task, and more specifically, it is called a *foreground task*. Other tasks run "in the background", which is to say that the player who initiated this task is free to type in another command (thereby starting another task) before the *background task* is finished. Many objects with "delayed reaction" behavior utilize background tasks. Here's an example. In the LambdaMOO Living Room, there is a fireplace. You can pile logs in the fire place, and then light the fire. While the logs are burning, the fire hisses and crackles and pops, but meanwhile you are free to continue conversing with others present. It is a background task that causes the fireplace noises to appear periodically. Every background task has a unique numerical identification number called its `task_id`.

`teleport` – Moving to a room in a way that is inconsistent with the Virtual Reality, e.g. using the `@go` or `@join` command.

`tell-filter` – Every player has a verb on emself called "`:tell`". (Some non-player objects have `:tell` verbs, too.) This verb receives as input a string of text, and prints that text on the player's screen. A tell filter pre-processes text before it is displayed. For example, it might inspect the text and prepend name of the player who initiated it, maybe in angle brackets. So instead of seeing, "Jack causes Jill to fall down laughing," you might see, "<Jill> Jack causes Jill to fall down laughing."

`tick` – A tick is a unit of computation. Just as it takes most people less effort to add 2 + 2 than to multiply 13 by 8, different tasks take different amounts of computing power, and these amounts are measured in units called *ticks*. When you type something in, 30,000 ticks are allotted to the task. (This is the default. The actual number may vary from MOO to MOO). Programmers of commands can, if necessary, ask the system to "take a breath" (metaphorically speaking) and then resume with an additional allotment of ticks, though this means that a command will take longer to complete, both in terms of absolute time (seconds) and ticks. Why do we care about ticks? Because each task gets only its allotted number of ticks before the system switches to allow the next task some ticks to compute. The computer can only do so much at once before it starts to get bogged down. When the system bogs down, everyone experiences lag (increased response time). Good programmers try to write code that uses a minimum of ticks without sacrificing clarity for future readers or maintainers.

`tiny scenery` – Objects (especially rooms) that have descriptions only and are not in any way interactive, or items that are mentioned in a room's description for which there is no corresponding object.

`toad` – The act of metaphorically turning a player into a toad, or, a player who has had this done to em. The `@toad` command, available to wizards only, removes the flag by which the system recognizes the player-object as a player. This action is a natural part of the reaping process (but does not constitute all of the reaping process). On infrequent occasions it is done for punitive reasons. It is impossible for a player to toad emself. Contrary to popular belief, toading can be undone, as long as the player object has not yet been recycled.

`toad scar` – When a player is `@toaded`, one of the side effects is that eir object number is removed from the list of players returned by the built-in function `players()`. If a player is reinstated, eir player number is appended to the *end* of the list of players, thus appearing out of numerical sequence, and this appearing out of sequence is what is meant by the phrase *toad scar*. To quote Nosredna, a LambdaMOO wizard, "The difference between toading and newting is that toading leaves a scar and newting doesn't."

`troll` –Trolls are players who log on and make inflammatory remarks or send inflammatory posts to mailing lists, caring more about riling people up than the actual substance of their utterances. Contrary to what one might think, trolls are not universally reviled. Some players actively enjoy challenging trolls about their alleged views, and believe that both they and the trolls realize that there is a sort of "dance" going on. (N.B. The name comes not from unruly fairies, but rather the act of dragging bait through the water, hoping that fish will bite or chase it.)

`typist` – The human being who is sitting at the computer keyboard typing. A single typist may have one or more player-characters.

`user` – A person using a computer program. This term is sometimes contrasted with *programmer,* the person who wrote the program that the user is using.

`valid` – A *valid* object is one that can be used within the MOO in certain conventional ways. There are certain pieces of information that are attached to every valid object without exception. These pieces of information include the object's owner (identified by object number), its location (identified by object number), its contents (a list of one or more object numbers or the empty list), and its parent (identified by object number). An object that has a number but doesn't have these pieces of information associated with it is not a valid object, by definition. Invalid objects exist, though, and are used in a number of ways. One of these is `$nothing` (`#-1`), which is where rooms are conventionally located. Other so-called invalid objects signal an error condition, specifically a failed or ambiguous match.

`variable` – A named piece of data that is used within a verb, but which does not exist before the verb runs or after the verb has finished executing. Variables are used to store intermediate results while a verb is in the process of running but which are needed neither at a later time nor by another verb. (Contrast **`property`**, which is used to store a result or state for later re-use.)

`verb` – A verb is a named, ordered sequence of commands that the server can interpret and execute. Whenever you type a command, for example `@who` or `put rock in box`, you are asking the computer to do something. You are using a verb. Some verbs are not intended to be used directly by someone typing at a terminal, but are intended to be called by other verbs. These verbs generally either produce some side effect – such as changing some data somewhere, or return some intermediate result to the verb that called it (such as 3, or an object number) – or both. The beauty of a MOO is that ordinary players can create new objects and write new verbs on them, thus extending the richness and variety of the environment.

`VR` – Virtual Reality. In particular, *VR* refers to actions that conform to the virtual reality of the MOO you are using. Examples would be saying things to people in the room with you, "walking" (i.e. using conventional exits and modes of transportation) as opposed to teleporting, etc. (See also **`meta-VR`**.)

`wheel` – An influential person. As in, "big wheel".

`wizard` – A wizard is a player on a MOO with special powers not available to ordinary players, among them the power to create new players, `@newt` and `@toad` existing players, read otherwise-unreadable properties on any object, read otherwise-unreadable verbs on any object, read any message on any mail recipient (including players' private MOOmail, though wizards generally do not exercise this power), view all background tasks, and kill any background task. Wizards are usually hand-picked by the person who owns or is responsible for the system as a whole (this person is referred to as the ArchWizard). In general, one cannot become a wizard simply by reaching a certain specified level of proficiency, although proficiency is usually one of the criteria for selecting wizards, along with trustworthiness. Wizards are expected to use their powers with discretion. If you do not trust the wizards on a particular MOO to do so, you should not participate on it.

Conversational Typing Abbreviations

addy – address
afaik – as far as I know
afk – away from keyboard

atm – at the moment
bbl – be back later
bcnu – Be seeing you.
bf – boyfriend
brb – be right back
btw – by the way
f2f – face to face
fdl – falls down laughing
filfre – feel free (to do something or other)
gf – girlfriend
ianal – I am not a lawyer
iirc – if I recall correctly
imho – in my humble opinion
imnsho – in my not so humble opinion
imo – in my opinion
istr– I seem to recall
j/k – just kidding
l8r – later
lol – laughs out loud
ltns – long time no see
oic – Oh, I see.
otoh – on the other hand
pov – point of view
ppl – people
qooc – quoted out of context
rotfl – rolling on the floor laughing
rtfm – read the manual
R U M or F? – Are you male or female? (This phrase is now eschewed by experienced players except to make fun of inexperienced players.)
stfu – shut the fuck up
tmi – too much information
ttfn – Ta ta for now
ttyl – Talk to you later
wrt – with regard to
wrte – we regret the error
ymmv – your mileage may vary

Addendum to the Glossary

The glossary entries for **`call`** and **`subroutine`** draw an analogy between a program calling a subroutine and a cookbook referring to one recipe from within another recipe.

The following two recipes are provided as an adjunct to those entries, *just in case* someone actually decided to check my cross-references.

Béchamel Sauce

2 Tablespoons butter	Salt
2 Tablespoons flour	Freshly ground pepper
1 Cup milk, heated	

Melt the butter in a small shallow pan. Stir in the flour and cook, stirring constantly, until it bubbles a bit, but don't let it turn brown. 2-3 minutes.

Add the milk a little bit at a time, stirring to incorporate each addition completely before adding more. You should have a smooth paste. Bring just to a boil, add salt and pepper to taste, then lower the heat and simmer for 2-3 minutes more. Remove from heat.

This can be stored for later use. After it has cooled somewhat, place a piece of plastic wrap directly on the surface to prevent a skin from forming.

Chocolate Soufflé

Contrary to popular belief, a soufflé is not difficult to make, though it takes a while and must be served immediately. Most of the work can be done ahead of time.

2 1/2 ounces unsweetened chocolate	3/4 Cup whole milk
5 Tablespoons sugar	3 eggs, separated
2 Tablespoons butter	1 teaspoon vanilla
2 Tablespoons flour	1 pint vanilla ice cream (for the sauce)
1/8 teaspoon salt	

Preheat the oven to 325° F. Butter a 1 1/2 quart soufflé dish and dust with granulated sugar. Set aside.

Put the chocolate, 2 Tablespoons of the sugar and 2 Tablespoons of hot water in a small pan and heat slowly, stirring occasionally, until the chocolate is melted and smooth. Remove from the heat and set aside.

Follow the procedure for béchamel sauce (see page 257) using 2 tablespoons butter, 2 tablespoons flour and 3/4 cup milk, but salting only lightly and leaving out the pepper. Blend in the chocolate mixture.

Beat the egg yolks well. Stir a little of the hot sauce into the yolks, then add the yolks to the remaining sauce. Stir well, then set aside to cool.

This much can be done in advance.

With clean beaters, beat the egg whites until foamy, then slowly add the remaining 3 tablespoons of sugar, and continue beating until stiff but not dry. Stir about 1/4 of the whites into the chocolate mixture, then fold in the remainder. Stir in the vanilla.

Pour all into the soufflé dish, sprinkle the top with sugar, and bake for 35 minutes. Meanwhile, set out the ice cream to melt at room temperature.

Serve immediately with a “cold vanilla sauce” made from the melted ice cream.