The CircleMUD Builder's Manual

Jeremy Elson, `jelson@circlemud.org`

Revision 1.14, 30 September 1996, for CircleMUD 3.00

This document describes how to create CircleMUD areas, and specifies the file formats for worlds, mobiles, objects, shops, and zones, as well as providing examples of each. All information necessary to build worlds can be found in this document. The intended audience is world builders interested in creating new worlds, implementors who need to decode existing world files, and coders extending the current world specification. Thanks to Jeff Fink (Teker) for documenting the shop format (as well as for submitting the shop code itself!), and Alex Fletcher (Furry) for writing parts of the Introduction. More information about CircleMUD, including up-to-date versions of this documentation in ASCII and Postscript, can be found at the *CircleMUD Home Page* `<http://www.circlemud.org/>` or *FTP site* `<ftp://ftp.circlemud.org/pub/CircleMUD/>`.

Contents

1 Introduction

1.1 Your Job as a Tinyworld Architect

As a *Tinyworld Architect* or *Builder*, your job is to create the virtual world in which players can roam around, solve puzzles, find treasures, and gain experience. A Builder creates the rooms, objects, and monsters with which players will interact, defining their textual descriptions, stats, abilities, and other special properties. A Builder should not be confused with the MUD's *Coder*, whose job it is to modify the C code that makes the CircleMUD server actually run. A Builder does not need to be a programmer, and building is not programming; building is done by writing data files in a particular format, which this document describes in detail.

There is a standard world included with the CircleMUD distribution which is intended to serve as a stepping stone and a basic guide to demonstrate what kind of worlds you can build for your MUD. To *only* use the standard CircleMUD world, without adding any original work, is to guarantee your MUD will be boring and largely ignored. MUDs are judged in many ways, but one of the most important is the number and quality of areas available. The areas are what tend to make a MUD original. For example, one MUD could be based upon a magic-rich world and the code and areas would reflect this, while another could revolve around cities and thieves. Each of these MUDs would have its areas designed in such a way to flesh out this theme. In essence, building an area is like writing a book. It needs to have a plot, needs to be descriptive, and needs to be populated with memorable people and places.

Writing an area requires inspiration and imagination before all else. Ideas for areas often come from literature; for example, an area that traces Alice's adventures through Wonderland or Dante's trip through the Inferno. Areas usually start out on paper long before they reach a computer; a general map of the region can help to solidify the idea and a specific map of each individual room is absolutely required so that the rooms can be linked together in a way that makes sense geographically. Taking notes on ideas for which monsters should be encountered in the area, their descriptions, and in what location the monsters should appear can also help when planning an area.

1.2 Game Balance

"Game Balance" is a term that brings a different thing to mind for every person that hears it. What is most important about game balance is to keep in mind for whom each area is designed–for example, high level players, newbies, or small groups. The objects and monsters found in the area should match the level, abilities, and needs of the players expected to use the area. Most players do not like to be given vast treasure with no difficulty in getting it, but on the other hand, nobody likes to fight the most difficult mobile on the MUD and get nothing for doing it. The job of the chief builder of a MUD and the authors of the individual areas is to find a happy medium between these two extremities. The process of finding that medium on your MUD is what makes MUDs original.

The main factor that affects game balance is the areas that make up the MUD. Because of this, each area should be checked against the others to ensure that one area is not impossibly hard or absurdly easy or rewarding relative to the rest of the world. Each area that comes with the MUD or is added later should be checked by one or more implementors or builders, and the characteristics of the mobiles and objects should be changed to suit to the balance of the MUD. Each new area that becomes part of the world should not be added until it has been similarly balanced to the implementors' satisfaction. Understandably, builders want their zones to be popular, but they sometimes attempt to achieve this goal by purposefully making their zone unbalanced, adding powerful weapons or armor with no harmful side-effects or mobiles that are easy to kill yet give massive numbers of experience points. Such zones are destined both to become very popular and invariably to bring about the death of your MUD's game balance.

An area's balance should be an integral part of the design process, not something to be tacked on as an afterthought. Too often, an area will be designed with outrageously good weapons and armor which throws off the balance of the game. Naturally, after such zone is added, players complain bitterly if it is ever removed or toned down. Also, because the rent system saves hitrolls, damrolls, and ac-apply's, veteran players will be able to hold on to their old, spectacular equipment unless it is explicitly taken from them, even after the area has been changed. This does nothing but generate bad feelings on all sides. Therefore, the wise implementor will always carefully check a zone for balance *before* it is added to the production MUD. It is generally not a good idea to "let the players balance the area" by unleashing an unbalanced area on them and watching to see where the hordes of players go.

1.3 Making Your Areas Interesting

An interesting area will always attract more players than a bland one. There are many ways to make an area interesting. Try to be as descriptive as possible; don't hold back on writing extra descriptions. Players are so accustomed to not having richly described areas that finding an extra description can often be a real treat. Also, one oft forgotten thing to describe are the door exits. Describing all of these can give a feel of standing out in a field and looking off to the north and seeing something like:

```
The fields stretch off towards the large hills on the horizon.  Far to
the north you see what appears to be a plume of smoke.
```

With door descriptions like these, an area will feel more fleshed out to the player. Many players (both experienced and first timers) read the descriptions carefully the first time they walk through an area, and having many extra descriptions helps them fill out their idea of what things actually look like.

One thing that should never be done is to have generic room descriptions like "You stand in a big room. It is very dark." Descriptions like these detract in general from the rest of the world, and if they are found room after room can bore a player to tears. Such a description could be changed to:

```
     You stand in a room of very large size.  Shadows cower along the
walls and almost seem to be moving as you look around yourself.  The floor
is made of heavy stones which are very dark in color.  The ceiling is
quite some distance above you, but you can still make out objects hanging
from it, ruining the smoothness that is characteristic of the rest of the
room.
```

Another way to make an area interesting is to create some sort of plot line for it, or a coherent theme, rather than a collection of haphazardly related rooms. The plot can be complex like infiltrating a castle to garner the war plans of the evil Lord Zygol, simple like ridding the caves of goblins, or anything in between. Often the plot in an area can be advanced by some fairly simple puzzles or descriptions. With the help of special procedures written in C by the MUD's coder, involved puzzles of Zork-like complexity can be readily created.

Not all mobs have to be designed to be killed, nor does every shopkeeper have to buy or sell something–they could just be created so that they refuse to trade with any player characters. The players will then wonder why the shopkeeper exists. Perhaps giving him a jewel will make him more friendly. In this way, an area can be made infinitely more exciting by coding some special procedures for it. Perhaps random teleporters throughout the area, perhaps some procedures that have mobiles respond to questions from players.

All in all, the best way to make an area interesting is to use variety, intelligence, and imagination in building. Try to imagine what it would be like for you to walk through and what you might try looking at or doing, and then try to incorporate that into your area. Show your area to others and take their advice. By taking all of this extra effort in creating your area, you will be rewarded by leaving a lasting memory of your area in the minds of many players.

1.4 Using World-Building Programs

In the old days, the only tool that was used (or required) to write a MUD area was a simple text editor. However, in the course of time, various people have written programs to help builders create worlds without having to understand the complex details of the world file format. World- building programs are becoming more popular, especially the fancy graphical builders that run under Microsoft Windows. You may prefer to use one of them rather than trying to use a simple text editor and understanding the format on your own. New world-builders are constantly being written and released, so any attempt to describe them here will almost certainly be obsolete by the time you read it. However, some of them can be found in the `contrib` section of CircleMUD's *official FTP site* `<ftp://ftp.circlemud.org/pub/CircleMUD/contrib>`.

2 The Mechanics of World Building

2.1 Overview of the MUD World

CircleMUD's world is divided into distinct sections called *zones*. Each zone typically consists of a single, modular, geographically coherent region of the MUD's virtual world with a consistent storyline. Each zone can define its own physical space (the *world*), monsters (usually called *mobiles*), *objects* (such as armor, weapons and treasures), and *shops*, in which a mobile in a particular room can buy and sell objects to players.

A single zone typically contain less than 100 rooms, 100 monster definitions and 100 object definitions, but a large region can be subdivided into several zones at the author's discretion. For example, the City of Midgaard is divided into two zones, one for the main city and one for the southern residential area. A zone can also use mobiles and objects defined by another zone, but this practice is discouraged because it makes zones less modular, meaning that it becomes more difficult to add or remove one zone without affecting another zone.

Each room, mobile and object within a zone is given a unique number called a *Virtual Number* or *Vnum*. The Vnums for the rooms, mobiles and objects are independent, so there can be both a room number 3001 and an object number 3001. When defining and referencing parts of a zone, the zone author always refers to each entity by its Vnum and *never* by name. Vnums are normally not seen by players. Each zone itself also has a Vnum. A common convention is to number the zone with the Vnums of its component rooms, divided by 100. For example, Midgaard is zone 30, consisting of rooms 3000 to 3099. Mobile and object numbering follows the same convention.

The author of the zone can define aspects of each room such as the terrain type, special properties like whether the room has its own light source or is a death trap, and other parameters. A very important aspect of each room is the position of other rooms in relation to it; for example, from room 3014, one can go north to reach room 3005, east to room 3015, etc. Great care should be given to making the room links logical and consistent. A player who moves east and then immediately west should find herself back in the same room in which she started.

Each mobile is given characteristics such as number of hit points, bare hand damage capability, strength, and special skills and abilities. Objects have parameters such as weight, value, and magical properties. The author can also choose how these three pieces of the world are combined to form the initial state of the zone: for example, the number of each mobile that exist and in which rooms they stand, the equipment that each mobile uses, objects which might be on the floor, and the doors which may be initially locked or unlocked.

When the CircleMUD server runs the zone, it sets each zone to its initial state as defined by the author, and then makes the zone "come alive" by randomly making mobiles wander through the zone and, if desired, attack players. While the players are using the zone (killing the mobiles and picking up equipment) the server periodically resets the zone to its initial state (a *zone reset*) to prepare the zone for the next group of players.

2.2 Learning By Example

Before descending into the details of MUD building, it should be noted that the formats of the world files are sufficiently complex that it is probably not possible to gain a complete understanding of them merely by reading this documentation. This document is designed to be a reference manual and therefore may not serve as a particularly good tutorial. While there are examples provided at the end of each section, they are only meant to be representative and are not comprehensive examples of all possible ways to use the features that will be described. The most effective way is to learn by example: examine some of the areas that come with CircleMUD and try to figure out the meanings of the numbers in different rooms, objects, mobiles, and zone files, using this manual as a guide. Once you're proficient at reading world files, you'll find that creating them is a much easier task.

2.3 CircleMUD World Files

Each CircleMUD zone is defined by five types of files: world files, mobile files, object files, shop files, and zone files. World files (*.wld) define the actual rooms and the links from one room to another. Mobiles (*.mob) are the monsters which inhabit the MUD. Objects (*.obj) are the weapons, armor, treasure, and other objects manipulated by players and monsters. Shop files (*.shp) define the MUD's shopkeepers, controlling what they buy, sell, and say. Finally, Zone files (*.zon) bring all the previous elements together to define the initial state of the zone, describing how monsters should be equipped, where monsters should be placed, where objects on the ground should be, which doors should be locked, etc. These five types of files are collectively referred to as "the world," or sometimes the "tinyworld files."

CircleMUD uses *split world files* to make the world easier to manipulate. Instead of all the rooms being lumped together in a single, cumbersome file, the rooms are split into many different files, one file for each area of the world. All five types of files are split in a similar manner. Circle has one directory for the room files (`lib/world/wld`), one directory for the object files (`lib/world/obj`), and so forth.

Circle doesn't care how the world files are split or what the names of the files are, but certain conventions have developed to make management of the world easier. Each file typically contains information for only a single zone and the filename is typically the zone number, with an extension indicating one of the 5 file types.

For example, the file `30.wld` contains rooms 3000 to 3099 of zone 30; `42.mob` contains mobiles 4200 to 4299 of zone 42, etc.

Also in each of these directories is a file called "`index`" that tells the server which files from that directory should be loaded when the server boots and a file called "`index.mini`" which (minimal) set of files should be loaded when the server is booted with the `-m` option.

Every world file used by Circle (including the index files) must be terminated by the dollar sign ($) to tell the server that the file has ended. Without the dollar sign, the server will not boot properly.

The `split` utility that comes with CircleMUD can be used to divide a large file into a number of smaller files; for example, if you have a large zone that you'd like to break into several smaller zones. See the *CircleMUD Utility Manual* `<http://www.circlemud.org/~jelson/circle/cdp/utils>` for more information on how to use `split`.

2.4 Using Bitvectors

When learning about the formats of CircleMUD world files, you'll frequently see references to "bitvectors." A bitvector is a group of flags which each can be either on or off. Bitvectors and their flags are used in many ways within CircleMUD, such as to define the personality of mobiles, the characteristics of rooms, etc. Understanding how to use bitvectors is essential if you want to build a CircleMUD world.

At every point where this document says a bitvector is required, it will be accompanied by a table describing the flags which you can use with that bitvector. The table will look something like this:

1	a	DIRTY	The room is dirty.
2	b	STINKY	The room stinks.
4	c	MUSHY	The floor of the room feels mushy.
8	d	SWAMPY	The room resembles a swamp.

Note there are four columns in the table. The first column contains the numeric value of the flag. The second contains the alphabetic representation of the flag. The third is the name of the flag, and the fourth is a description of what the flag does.

There are two ways you can construct a bitvector with the table above: the numeric method and the alphabetic method. The numeric method is to select all flags you'd like to activate, take the numbers of those flags as listed in the first column of the table, and add them all up. The resulting sum will be the bitvector. The alphabetic method is much easier: just write down all the letters of the flags you'd like to use with no spaces in between. For both numeric and alphabetic bitvectors, use "0" to indicate a bitvector where none of the flags are set.

For example, imagine you want to create a room that is dirty, mushy, and resembles a swamp, but does not stink. Using the numeric method, you'd look up the numbers of those three flags (1 for dirty, 4 for mushy, and 8 for swampy), and add them up to get 13. Using the alphabetic method, the bitvector would simply be "acd". Bitvectors are case-sensitive; "acd" is very different from "Acd" and "ACD".

At every point where the CircleMUD format requires a bitvector, you can write either a numeric bitvector or an alphabetic bitvector. They are completely interchangeable. However, be forewarned that if you use alphabetic bitvectors, your area will not be compatible with MUDs based on the original DikuMUD. Alphabetic bitvectors are a CircleMUD enhancement and may not be supported by MUDs based on Gamma Diku.

In some bitvector tables, you will see values whose descriptions say "Reserved for internal use" or "Do not use". You should never set those flag values in your world files.

2.5 Adding new areas to the MUD

After an area is written, there are three steps required to add it to the MUD for testing: copying the files into the proper directories, adding the new filenames to the appropriate index files, and running the MUD in syntax-check mode to make sure the new area is formatted correctly.

All world-related files go in the directory `lib/world`. In this example, we will imagine that your new area is zone number 57 (which should consist of rooms, objects and mobiles numbered 5701-5799). Your zone probably has 5 files: `57.wld`, `57.mob`, `57.obj`, `57.shp`, and `57.zon`. The first step is to copy each of these files into their appropriate subdirectory: `57.wld` should be copied to the directory `lib/world/wld`; `57.mob` should be copied to the directory `lib/world/mob`, and so forth.

The next step is to add the name of the newly copied world files to the `index` file contained in each of the world subdirectories. Note you will need to change 5 index files: one for each of the world files that you copied in the previous step. Adding the filenames to the index files tells CircleMUD that the files should be loaded; they will *not* be loaded simply by virtue of being in the correct directory. First, edit the file `lib/world/wld/index`; you should see a list of the current world (room) files. Add a single line that says `57.wld` in the correct numeric order. Next, add a similar line in the other index files: add `57.mob` to `lib/world/mob/index`; add `57.obj` to `lib/world/obj/index`, etc.

Now you can try to boot the MUD with the new world. If you're adding a new area which hasn't been debugged yet, it's usually a good idea to run Circle in its syntax-checking mode first. From Circle's root directory, type `bin/circle -c` to run Circle's syntax checker. If the check runs with no SYSERR messages, the syntax of the area is probably correct and the MUD can be safely booted. Otherwise, check the *CircleMUD SYSERR List* `<http://www.circlemud.org/~jelson/circle/cdp/syserr>` for more information on how to correct the formatting errors. Also, see the *CircleMUD Administrator's Guide* `<http://www.circlemud.org/~jelson/circle/cdp/guide>` for more information on how to run CircleMUD or how to use the syntax checking mode.

3 World (Room) Files

3.1 The Format of a Room

The format of a room is:

```
#<virtual number>
<room name>~
<Room Description>
~
<zone number> <room bitvector> <sector type>
{Zero or more Direction Fields and/or Extra Descriptions}
S
```

There can be between 0 and 6 Direction Fields. There should not be more than one Direction Field for a particular direction. No Extra Descriptions are required but an unlimited number are allowed. Each room is terminated with the literal letter `S`.

Virtual Number

This number is critical; it is the identity of the room within the game. All other files will use this number to refer to this room. From within the game, this number can be used with "goto" to go to this room. The virtual numbers must appear in increasing order in the world file.

Room Name

This string is the room's title, which is displayed before the room description when players look at the room, or displayed alone if players are using "brief").

Room Description

The description of the room seen when they type "look," or when they enter the room with brief mode off.

Zone Number

This number is obsolete and no longer used. Historically it contained the zone number of the current room but it is currently ignored. It is maintained as part of the format for backwards compatibility.

Room Bitvector

A bitvector (see section 2.4), with the following values:

```
1     a  DARK          Room is dark.
2     b  DEATH         Room is a death trap; char ``dies´´ (no xp lost).
4     c  NOMOB         MOBs (monsters) cannot enter room.
8     d  INDOORS       Room is indoors.
16    e  PEACEFUL      Room is peaceful (violence not allowed).
32    f  SOUNDPROOF    Shouts, gossips, etc. won´t be heard in room.
64    g  NOTRACK       ``track´´ can´t find a path through this room.
128   h  NOMAGIC       All magic attempted in this room will fail.
256   i  TUNNEL        Only one person allowed in room at a time.
512   j  PRIVATE       Cannot teleport in or GOTO if two people here.
1024  k  GODROOM       Only LVL_GOD and above allowed to enter.
2048  l  HOUSE         Reserved for internal use.  Do not set.
4096  m  HOUSE_CRASH   Reserved for internal use.  Do not set.
8192  n  ATRIUM        Reserved for internal use.  Do not set.
16384 o  OLC           Reserved for internal use.  Do not set.
32768 p  BFS_MARK      Reserved for internal use.  Do not set.
```

Sector Type

A single number (*not* a bitvector) defining the type of terrain in the room. Note that this value is not the number of movement points needed but just a number to identify the sector type (the movement loss is controlled by the array `movement_loss[]` in the file `constants.c`). The Sector Type can be one of the following:

0	INSIDE	Indoors (small number of move points needed).
1	CITY	The streets of a city.
2	FIELD	An open field.
3	FOREST	A dense forest.
4	HILLS	Low foothills.
5	MOUNTAIN	Steep mountain regions.
6	WATER_SWIM	Water (swimmable).
7	WATER_NOSWIM	Unswimmable water - boat required for passage.
8	UNDERWATER	Underwater.
9	FLYING	Wheee!

Direction Fields and Extra Descriptions

This section defines the room's exits, if any, as well as any extra descriptions such as signs or strange objects that might be in the room. This section can be empty if the room has no exits and no extra descriptions. Otherwise, it can have any number of `D` (Direction Field) and `E` (Extra Description) sections, in any order. After all exits and extra descriptions have been listed, the end of the room is signaled with the letter `S`. The Direction Fields and Extra Descriptions are described in more detail in the following sections.

3.2 The Direction Field

The general format of a direction field is:

```
D<direction number>
<general description>
~
<keyword list>~
<door flag> <key number> <room linked>
```

Direction Number

The compass direction that this Direction Field describes. It must be one of the following numbers:

0	North
1	East
2	South
3	West
4	Up
5	Down

General Description

The description shown to the player when she types "look <direction>." This should not be confused with the room description itself. Unlike the room description which is automatically viewed when a player walks into a room, the General Description of an exit is only seen when a player looks in the direction of the exit (e.g., "look north").

Keyword List

A list of acceptable terms that can be used to manipulate the door with commands such as "open," "close," "lock," "unlock," etc. The list should be separated by spaces, e.g.:

```
door oak big~
```

Door Flag

Can take one of three values (0, 1 or 2):

0

An unrestricted exit that has no door, or a special door cannot be opened or closed with the "open" and "close" commands. The latter is useful for secret doors, trap doors, or other doors that are opened and closed by something other than the normal commands, like a special procedure assigned to the room or an object in the room.

1

Normal doors that can be opened, closed, locked, unlocked, and picked.

2

Pickproof doors: if locked, can be opened only with the key.

The initial state of all doors is open, but doors can be opened, closed, and locked automatically when zones reset (see the zone file documentation for details).

Key Number

The virtual number of the key required to lock and unlock the door in the direction given. A value of -1 means that there is no keyhole; i.e., no key will open this door. If the Door Flag for this door is 0, the Key Number is ignored.

Room Linked

The virtual number of the room to which this exit leads. If this number is -1 (NOWHERE), the exit will not actually lead anywhere; useful if you'd like the exit to show up on "exits," or if you'd like to add a description for "look <direction>" without actually adding an exit in that direction.

3.3 Room Extra Descriptions

Extra descriptions are used to make rooms more interesting, and make them more interactive. Extra descriptions are accessed by players when they type "look at <thing>," where <thing> is any word you choose. For example, you might write a room description which includes the tantalizing sentence, "The wall looks strange here." Using extra descriptions, players could then see additional detail by typing "look at wall." There can be an unlimited number of Extra Descriptions in each room.

The format of an extra description is simple:

```
E
<keyword list>~
<description text>
~
```

Keyword List

A space-separated list of keywords which will access the description in this `E` section.

Description Text

The text that will be displayed when a player types "look <keyword>," where <keyword> is one of the keywords specified in the Keyword List of this `E` section.

3.4 World File Example

Here's a sample entry from a CircleMUD world file:

```
#18629
The Red Room~
   It takes you a moment to realize that the red glow here is coming
from a round portal on the floor.  It looks almost as if someone had
painted a picture of a dirt running through a field on the floor of
this room.  Oddly enough, it is so realistic you can feel the wind in
the field coming out of the picture.
~
186 ad 0
D0
You see a big room up there.
~
~
0 -1 18620
D1
You see a small room.
~
oak door~
1 18000 18630
E
portal floor~
It looks as if you could go down into it... but you can't be sure of where
you will end up, or if you can get back.
~
S
```

This room is virtual number 18629, called "The Red Room." It is dark and indoors, with an "INDOORS" sector type. It has an exit north and east. The north exit leads to room 18620; if a player types "look north" it will say "You see a big room up there." The exit east is a normal, pickable door that leads to room 18630 and which takes key number 18000. There is one extra description for "portal" and "floor."

4 Mobile (Monster) Files

4.1 The Format of a Mobile

The format of a mobile is:

```
#<virtual number>
<alias list>~
<short description>~
<long description>
~
<detailed description>
~
<action bitvector> <affection bitvector> <alignment> <type flag>
{type-specific information; see below for details}
```

The format of mobiles varies depending on the Type Flag. See below for documentation of the formats of the various types.

Virtual Number

This number is critical; it is the identity of the mobile within the game. It is the number that will be used to reference the mobile from zone files and is the number used to "load" mobiles from within the game. The virtual numbers must appear in increasing order in the mob file.

Alias List

The list of keywords, separated by spaces, that can be used by players to identify the mobile. The mobile can only be identified using the keywords that appear in its alias list; it cannot be identified by a word that appears only in its name. Great care should be taken to ensure that the spellings of names and aliases match. Fill words such as "the," "a," and "an" should not appear in the Alias List.

Short Description

The description of the mobile used by the MUD when the mobile takes some action. For example, a short description of "The Beastly Fido" would result in messages such as "The Beastly Fido leaves south." and "The Beastly Fido hits you hard." The Short Description should never end with a punctuation mark because it will be inserted into the middle of sentences such as those above.

Long Description

The description displayed when a mobile is in its default position; for example, "The Beastly Fido is here, searching through garbage for food." When the mobile is in a position other than its default position, such as sleeping or incapacitated, the short description is used instead; for example, "The Beastly Fido is lying here, incapacitated." Unlike the Short Description, the Long Description should end with appropriate punctuation.

Detailed Description

The description displayed for a mobile when a player looks at the mobile by typing "look at <mobile>."

Action Bitvector

A bitvector (see section 2.4) with the following values:

1	a	SPEC	This flag must be set on mobiles which have special procedures written in C. In addition to setting this bit, the specproc must be assigned in spec_assign.c, and the specproc itself must (of course) must be written. See the section on Special Procedures in the file coding.doc for more information.
2	b	SENTINEL	Mobiles wander around randomly by default; this bit should be set for mobiles which are to remain stationary.
4	c	SCAVENGER	The mob should pick up valuables it finds on the ground. More expensive items will be taken first.
8	d	ISNPC	Reserved for internal use. Do not set.
16	e	AWARE	Set for mobs which cannot be backstabbed. Replaces the ACT_NICE_THIEF bit from Diku Gamma.
32	f	AGGRESSIVE	Mob will hit all players in the room it can see. See also the WIMPY bit.
64	g	STAY_ZONE	Mob will not wander out of its own zone -- good for keeping your mobs as only part of your own area.
128	h	WIMPY	Mob will flee when being attacked if it has less than 20% of its hit points. If the WIMPY bit is set in conjunction with any of the forms of the AGGRESSIVE bit, the mob will only attack mobs that are unconscious (sleeping or incapacitated).
256	i	AGGR_EVIL	Mob will attack players that are evil-aligned.
512	j	AGGR_GOOD	Mob will attack players that are good-aligned.
1024	k	AGGR_NEUTRAL	Mob will attack players that are neutrally aligned.
2048	l	MEMORY	Mob will remember the players that initiate attacks on it, and initiate an attack on that player if it ever runs into him again.
4096	m	HELPER	The mob will attack any player it sees in the room that is fighting with a mobile in the room.

			Useful for groups of mobiles that travel together; i.e. three snakes in a pit, to force players to fight all three simultaneously instead of picking off one at a time.
8192	n	NOCHARM	Mob cannot be charmed.
16384	o	NOSUMMON	Mob cannot be summoned.
32768	p	NOSLEEP	Sleep spell cannot be cast on mob.
65536	q	NOBASH	Large mobs such as trees that cannot be bashed.
131072	r	NOBLIND	Mob cannot be blinded.

Affection Bitvector

A bitvector (see section 2.4) with the following values:

1	a	BLIND	Mob is blind.
2	b	INVISIBLE	Mob is invisible.
4	c	DETECT_ALIGN	Mob is sensitive to the alignment of others.
8	d	DETECT_INVIS	Mob can see invisible characters and objects.
16	e	DETECT_MAGIC	Mob is sensitive to magical presence.
32	f	SENSE_LIFE	Mob can sense hidden life.
64	g	WATERWALK	Mob can traverse unswimmable water sectors.
128	h	SANCTUARY	Mob is protected by sanctuary (half damage).
256	i	GROUP	Reserved for internal use. Do not set.
512	j	CURSE	Mob is cursed.
1024	k	INFRAVISION	Mob can see in dark.
2048	l	POISON	Reserved for internal use. Do not set.
4096	m	PROTECT_EVIL	Mob is protected from evil characters.
8192	n	PROTECT_GOOD	Mob is protected from good characters.
16384	o	SLEEP	Reserved for internal use. Do not set.
32768	p	NOTRACK	Mob cannot be tracked.
65536	q	UNUSED16	Unused (room for future expansion).
131072	r	UNUSED17	Unused (room for future expansion).
262144	s	SNEAK	Mob can move quietly (room not informed).
524288	t	HIDE	Mob is hidden (only visible with sense life).
1048576	u	UNUSED20	Unused (room for future expansion).
2097152	v	CHARM	Reserved for internal use. Do not set.

Alignment

A number from -1000 to 1000 representing the mob's initial alignment.

```
-1000...-350   Evil
 -349...349    Neutral
  350...1000   Good
```

Type Flag

This flag is a single letter which indicates what type of mobile is currently being defined, and controls what information CircleMUD expects to find next (i.e., in the file from the current point to the end of the current mobile).

Standard CircleMUD 3.0 supports two types of mobiles: S (for Simple), and E (for Enhanced). Type C (Complex) mobiles was part of the original DikuMUD Gamma and part of CircleMUD until version 3.0, but are no longer supported by CircleMUD v3.0 and above.

Check with your local implementor to see if there are any additional types supported on your particular MUD.

4.2 Type S Mobiles

For type S mobs, the type-specific information should be in the following format:

```
<level> <thac0> <armor class> <max hit points> <bare hand damage>
<gold> <experience points>
<load position> <default position> <sex>
```

Level

The level of the monster, from 1 to 30.

THAC0

"To Hit Armor Class 0" – a measure of the ability of the monster to penetrate armor and cause damage, ranging from 0 to 20. Lower numbers mean the monster is more likely to penetrate armor. The formal definition of THAC0 is the minimum roll required on a 20-sided die required to hit an opponent of equivalent Armor Class 0.

Armor Class

The ability of the monster to avoid damage. Range is from -10 to 10, with lower values indicating better armor. Roughly, the scale is:

```
AC  10    Naked person
AC   0    Very heavily armored person (full plate mail)
AC -10    Armored Battle Tank (hopefully impossible for players)
```

Note on THAC0 and Armor Class (AC): When an attacker is trying to hit a victim, the attacker's THAC0 and the victim's AC, plus a random roll of the dice, determines whether or not the attacker can hit the victim. (If a hit occurs, a different formula determines how much damage is done.) An attacker with a low THAC0 is theoretically just as likely to hit a victim with a low AC as an attacker with a high THAC0 is to hit a victim with a high AC. Lower attacker THAC0's and higher victim AC's favor the attacker; higher attacker THAC0's and lower victim AC's favor the victim.

Max Hit Points

The maximum number of hit points the mobile is given, which must be given in the form "xdy+z" where x, y, and z are integers. For example, `4d6+10` would mean sum 4 rolls of a 6 sided die and add 10 to the result. Each individual instance of a mob will have the same max number of hit points from the time it's born to the time it dies; the dice will only be rolled once when a particular instance of the mob is created. In other words, a particular copy of a mob will always have the same number of max hit points during its life, but different copies of the same mob may have different numbers of max hit points.

Note that all three numbers, the "d" and the "+" must always appear, even if some of the numbers are 0. For example, if you want every copy of a mob to always have exactly 100 hit points, write `0d0+100`.

Bare Hand Damage (BHD)

The amount of damage the mob can do per round when not armed with a weapon. Also specified as "xdy+z" and subject to the same formatting rules as Max Hit Points. However, unlike Max Hit Points, the dice are rolled once per round of violence; the BHD of a mob will vary from round to round, within the limits you set.

For BHD, xdy specifies the dice rolls and z is the strength bonus added both to BHD and weapon-inflicted damage. For example, a monster with a BHD of 1d4+10 will do between 11 and 14 hitpoints each round without a weapon. If the monster picks up and wields a tiny stick which gives 1d2 damage, then the monster will do 1d2 + 10 points of damage per round with the stick.

Gold

The number of gold coins the mobile is born with.

Experience

The number of experience points the mobile is born with.

Load Position

The position the mobile is in when born, which should be one of the following numbers:

```
0  POSITION_DEAD       Reserved for internal use.  Do not set.
1  POSITION_MORTALLYW  Reserved for internal use.  Do not set.
2  POSITION_INCAP      Reserved for internal use.  Do not set.
3  POSITION_STUNNED    Reserved for internal use.  Do not set.
4  POSITION_SLEEPING   The monster is sleeping.
5  POSITION_RESTING    The monster is resting.
6  POSITION_SITTING    The monster is sitting.
7  POSITION_FIGHTING   Reserved for internal use.  Do not set.
8  POSITION_STANDING   The monster is standing.
```

Default Position

The position to which monsters will return after a fight, which should be one of the same numbers as given above for Load Position. In addition, the Default Position defines when the mob's long description is displayed (see "Long Description" above).

Sex

One of the following:

```
0    Neutral (it/its)
1    Male (he/his)
2    Female (she/her)
```

4.3 Type S Mobile Example

```
#3062
fido dog~
the beastly fido~
A beastly fido is mucking through the garbage looking for food here.
~
The fido is a small dog that has a foul smell and pieces of rotted meat
hanging around his teeth.
~
afghq p -200 S
0 20 10 1d6+4 1d4+0
0 25
8 8 1
```

This is mobile vnum 3062. The Fido's action bitvector indicates that it has a special procedure (bit a), is aggressive (bit f), stays in its own zone (bit g), is wimpy (bit h), and cannot be bashed (bit q). Also, the Fido cannot be tracked (affection bit p), and has an initial alignment of -200.

After the `S` flag we see that the Fido is level 0, has a THAC0 of 20, an Armor Class of 10, 1d6+4 hit points (a random value from 5 to 10), and will do 1d4 hit points of bare hand damage per round. The Fido has 0 gold and 25 experience points, has a load position and default position of STANDING, and is male.

4.4 Type E Mobiles

Type E mobiles are specific to Circle 3.0 and are designed to provide an easy way for MUD implementors to extend the mobile format to fit their own needs. A type E mobile is an extension of type S mobiles; a type E mobile is a type S mobile with extra data at the end. After the last line normally found in type S mobs (the one ending with the mob's sex), type E mobiles end with a section called the Enhanced section. This section consists of zero or more enhanced mobile specifications (or *E-specs*), one per line. Each E-spec consists of a keyword followed by a colon (":") and a value. The valid keywords are listed below. The literal letter `E` must then come after all E-specs to signal the end of the mob.

The format of an E mobile is as follows:

```
<level> <thac0> <armor class> <max hit points> <bare hand damage>
<gold> <experience points>
<load position> <default position> <sex>
{E-spec list}
E
```

4.5 Type E Mobile Example

Let's say that you wanted to create an enhanced Fido like the one in the previous example, but one that has a bare-hand attack type of 4 so that the Fido bites players instead of hitting them. Let's say you also wanted to give this Fido the a strength of 18. You might write:

```
#3062
fido dog~
the beastly fido~
A beastly fido is mucking through the garbage looking for food here.
~
The fido is a small dog that has a foul smell and pieces of rotted meat
hanging around his teeth.
~
afghq p -200 E
0 20 10 1d6+4 1d4+0
0 25
8 8 1
BareHandAttack: 4
Str: 18
E
```

In the above example, the two E-specs used were BareHandAttack and Str. Any number of the E-specs can be used in an Enhanced section and they may appear in any order. The format is simple: the E-spec keyword, followed by a colon, followed by a value. Note that unlike type S mobiles, type E mobiles require a terminator at the end of the record (the letter `E`).

4.6 E-Spec Keywords Valid in CircleMUD 3.00

The only keywords supported under Circle 3.00 are BareHandAttack, Str, StrAdd, Int, Wis, Dex, Con, and Cha. However, the E-Specs have been designed such that new ones are quite easy to add; check with your local implementor to see if your particular MUD has added any additional E-Specs. Circle 3.10's Enhanced section will have considerably more features available such as the ability to individually set mobs' skill proficiencies.

BareHandAttack

This controls the description of violence given during battles, in messages such as "The Beastly fido bites you very hard." BareHandAttack should be one of the following numbers:

```
0   hit/hits
1   sting/stings
2   whip/whips
3   slash/slashes
4   bite/bites
5   bludgeon/bludgeons
6   crush/crushes
7   pound/pounds
8   claw/claws
9   maul/mauls
10  thrash/thrashes
11  pierce/pierces
12  blast/blasts
13  punch/punches
14  stab/stabs
```

Messages given when attackers miss or kill their victims are taken from the file `lib/misc/messages`. The attack type number for weapons is 300 plus the number listed in the table above, so to modify the message given to players when they are mauled, attack type number 309 in `lib/misc/messages` should be changed. Note that adding new attack types requires code changes and *cannot* be accomplished simply by adding new messages to `lib/misc/messages` (see the *CircleMUD Coding Manual* `<http://www.circlemud.org/~jelson/circle/cdp/coding>` for more information).

Str, StrAdd, Int, Wis, Dex, Con, Cha

The mobile's Strength, Strength Add, Intelligence, Wisdom, Dexterity, Constitution and Charisma, respectively. These values should be between 3 and 18.

5 Object Files

5.1 The Format of an Object

```
#<virtual number>
<alias list>~
<short description>~
<long description>~
<action description>~
<type flag> <extra (effects) bitvector> <wear bitvector>
<value 0> <value 1> <value 2> <value 3>
<weight> <cost> <rent per day>
{Zero or more Extra Descriptions and/or Affect Fields}
```

There can be an unlimited number of Extra Descriptions and up to 6 Affect Fields.

Virtual Number

This number is critical; it is the identity of the object within the game. It is the number that will be used to reference the object from zone files and is the number used to "load" objects from within the game. The virtual numbers must appear in increasing order in the object file.

Alias List

The list of keywords, separated by spaces, that can be used by players to identify the object. The object can only be identified using the keywords that appear in its alias list; it cannot be identified by a word that appears only in its name. Great care should be taken to ensure that the spellings of names and aliases match. Fill words such as "the," "a," and "an" should not appear in the Alias List.

Short Description

The description of the object used by the MUD when the object is used. For example, a short description of "a long, green stick" would result in messages such as "The Beastly Fido picks up the long, green stick." The Short Description should never end with a punctuation mark because it will be inserted into the middle of sentences.

Long Description

The description displayed when the object is seen lying on the ground, for example, "A furled umbrella is lying here." Unlike the Short Description, the Long Description should end with appropriate punctuation.

Action Description

Action Descriptions are primarily used for magical objects (staves, wands, scrolls, and potions) to specify what message displayed to the room when the magical item is used. The Action Description should be given in the act format specified in act.doc. If no Action Description is present, a default message will be used:

Staves: Rasmussen taps <object> three times on the ground.

Wands: Rasmussen points <object> at Onivel.

Scrolls: Rasmussen recites <object>.

Potions: Rasmussen quaffs <object>.

For objects which are readable papers, the Action Description contains the text on the paper.

Type Flag

A number which specifies what type of object is being defined; also controls the meanings of value0 through value4. The Type Flag must be one of the following numbers:

```
 1   LIGHT         Item is a light source.
 2   SCROLL        Item is a magical scroll.
 3   WAND          Item is a magical wand.
 4   STAFF         Item is a magical staff.
 5   WEAPON        Item is a weapon.
 6   FIREWEAPON    Currently not implemented.  Do not use.
 7   MISSILE       Currently not implemented.  Do not use.
 8   TREASURE      Item is treasure other than gold coins (e.g. gems)
 9   ARMOR         Item is armor.
10   POTION        Item is a magical potion.
11   WORN          Currently not implemented.  Do not use.
12   OTHER         Miscellaneous object with no special properties.
13   TRASH         Trash -- junked by cleaners, not bought by shopkeepers.
14   TRAP          Currently not implemented.  Do not use.
15   CONTAINER     Item is a container.
16   NOTE          Item is a note (can be written on).
17   DRINKCON      Item is a drink container.
18   KEY           Item is a key.
19   FOOD          Item is food.
20   MONEY         Item is money (gold coins).
21   PEN           Item is a pen.
22   BOAT          Item is a boat; allows you to traverse SECT_WATER_NOSWIM.
23   FOUNTAIN      Item is a fountain.
```

Extra (Effects) Bitvector

A bitvector (see section 2.4), to define the "special effects" of the object. Flags that are marked as "cosmetic" merely add an interesting message to the object when it is examined, but has no substantive effect otherwise. The flags have the following values:

1	a	GLOW	Item is glowing (cosmetic).
2	b	HUM	Item is humming (cosmetic).
4	c	NORENT	Item cannot be rented.
8	d	NODONATE	Item cannot be donated.
16	e	NOINVIS	Item cannot be made invisible.
32	f	INVISIBLE	Item is invisible.
64	g	MAGIC	Item has a magical aura and can´t be enchanted.
128	h	NODROP	Item is cursed and cannot be dropped.
256	i	BLESS	Item is blessed (cosmetic).
512	j	ANTI_GOOD	Item can´t be used by good-aligned characters.
1024	k	ANTI_EVIL	Item can´t be used by evil-aligned characters.
2048	l	ANTI_NEUTRAL	Item can´t be used by neutral align characters.
4096	m	ANTI_MAGIC_USER	Item can´t be used by the Mage class.
8192	n	ANTI_CLERIC	Item can´t be used by the Cleric class.
16384	o	ANTI_THIEF	Item can´t be used by the Thief class.
32768	p	ANTI_WARRIOR	Item can´t be used by the Warrior class.
65536	q	NOSELL	Shopkeepers will not buy or sell the item.

Wear Bitvector

A bitvector (see section 2.4) with the following values:

1	a	WEAR_TAKE	Item can be taken (picked up off the ground).
2	b	WEAR_FINGER	Item can be worn on the fingers.
4	c	WEAR_NECK	Item can be worn around the neck.
8	d	WEAR_BODY	Item can be worn on the body.
16	e	WEAR_HEAD	Item can be worn on the head.
32	f	WEAR_LEGS	Item can be worn on the legs.
64	g	WEAR_FEET	Item can be worn on the feet.
128	h	WEAR_HANDS	Item can be worn on the hands.
256	i	WEAR_ARMS	Item can be worn on the arms.
512	j	WEAR_SHIELD	Item can be used as a shield.
1024	k	WEAR_ABOUT	Item can be worn about the body.
2048	l	WEAR_WAIST	Item can be worn around the waist.
4096	m	WEAR_WRIST	Item can be worn around the wrist.
8192	n	WEAR_WIELD	Item can be wielded; e.g. weapons.
16384	o	WEAR_HOLD	Item can be held (the ``hold´´ command).

Note that the TAKE bit controls whether or not an item can be picked up using the "get" command, whereas the HOLD bit controls if the object can be worn using the "hold" command.

Value 0, Value 1, Value 2, Value 3

These values are very central. They define the object's abilities based on the Type Flag. See the subsection "Object Value Definitions" below for a detailed description of the four Value fields.

Weight

The weight of the object. The weight controls how many strength points a character must have to take the object, and is used to calculate when bags and other containers become full.

Cost

The value of the object in gold coins; used by shopkeepers.

Rent Per Day

The cost per day to rent the object in the Reception.

5.2 Object Value Definitions

The meaning of an object's four values (value 0 through value 3) vary depending on the Type Flag of the object.

In the table below, "unused" means that the server ignores the value; it can be set to any number (but 0 is always a safe bet). "unimplemented" indicates a Type Flag currently not recognized by the server.

An index of spell numbers for use with magical objects can be found in the Appendix A.

LIGHT (Type Flag 1)

```
value 0: unused
value 1: unused
value 2: Capacity of light in hours.
           0: Burned out light.
          -1: Eternal light source.
value 3: unused
```

SCROLL (Type Flag 2)

```
value 0: Level at which scroll´s spells are cast.
value 1: Spell number 1
value 2: Spell number 2
value 3: Spell number 3
     If less than 3 spells are desired, set unused values to -1.
```

WAND (Type Flag 3)

```
value 0: Level at which wand´s spell is cast.
value 1: Charge capacity of wand (>= 1)
value 2: Current number of charges remaining
value 3: Spell number
```

STAFF (Type Flag 4)

```
value 0: Level at which staff´s spell is cast.
value 1: Charge capacity of staff (>= 1)
value 2: Current number of charges remaining
value 3: Spell number
```

WEAPON (Type Flag 5)

```
value 0: unused
value 1: Number of damage dice
value 2: Size of damage dice
value 3: Weapon type for damage messages, one of:
          0    hit/hits
          1    sting/stings
          2    whip/whips
          3    slash/slashes
          4    bite/bites
          5    bludgeon/bludgeons
          6    crush/crushes
          7    pound/pounds
          8    claw/claws
          9    maul/mauls
          10   thrash/thrashes
          11   pierce/pierces
          12   blast/blasts
          13   punch/punches
          14   stab/stabs
```

FIREWEAPON (Type Flag 6)

unimplemented (do not use)

MISSILE (Type Flag 7)

unimplemented (do not use)

TREASURE (Type Flag 8)

```
value 0: unused
value 1: unused
value 2: unused
value 3: unused
```

ARMOR (Type Flag 9)

```
value 0: AC-apply of the armor.  Note that the effective change
          to AC is this value times a multiplier based on where
          the armor is worn.  Values >0 enhance the armor class;
          values <0 damage the armor class (cursed armor, for
          example).
value 1: unused
value 2: unused
value 3: unused
```

POTION (Type Flag 10)

```
value 0: Level at which the potion´s spells are cast.
value 1: Spell number 1
value 2: Spell number 2
value 3: Spell number 3
     If less than 3 spells are desired, set unused values to -1.
```

WORN (Type Flag 11)

unimplemented (do not use)

OTHER (Type Flag 12)

```
value 0: unused
value 1: unused
value 2: unused
value 3: unused
```

TRASH (Type Flag 13)

```
value 0: unused
value 1: unused
value 2: unused
value 3: unused
```

TRAP (Type Flag 14)

unimplemented (do not use)

CONTAINER (Type Flag 15)

```
value 0: Capacity (max containable weight) of container
value 1: Container flag bitvector (MUST be a numeric bitvector)
     1   CLOSEABLE      Container can be closed and locked.
     2   PICKPROOF      Lock on container can´t be picked.
     4   CLOSED         Container is closed when created.
     8   LOCKED         Container is locked when created.
value 2: The vnum of the key object that opens this container.
          -1 if it has no key.
value 3: Reserved for internal use -- always set as 0.
```

NOTE (Type Flag 16)

```
value 0: Language of writing (unimplemented).
value 1: unused
value 2: unused
value 3: unused
```

DRINKCON (Type Flag 17)

See Appendix B.

KEY (Type Flag 18)

```
value 0: unused
value 1: unused
value 2: unused
value 3: unused
```

FOOD (Type Flag 19)

```
value 0: The number of hours of hunger satisfied by this food.
value 1: unused
value 2: unused
value 3: Non-zero if the food is poisoned, 0 otherwise.
```

MONEY (Type Flag 20)

```
value 0: The number of gold coins in the pile.
value 1: unused
value 2: unused
value 3: unused
```

PEN (Type Flag 21)

```
value 0: unused
value 1: unused
value 2: unused
value 3: unused
```

BOAT (Type Flag 22)

```
value 0: unused
value 1: unused
value 2: unused
value 3: unused
```

FOUNTAIN (Type Flag 23)

See Appendix B.

5.3 Object Extra Descriptions

Object Extra Descriptions allow players to examine certain aspects of objects defined by the world builder, just like Room Extra Descriptions. There can be an unlimited number of Extra Descriptions per object. The format is exactly the same as for rooms:

```
E
<keyword list>~
<description text>
~
```

Keyword List

A space-separated list of keywords which will access the description in this E section.

Description Text

The text that will be displayed when a player types "look <keyword>," where <keyword> is one of the keywords specified in the Keyword List of this E section.

5.4 Object Affect Fields

Object Affect Fields give objects magical properties. They affect characters when the object is worn, not when picked up.

The format of an Object Affect Field is:

```
A
<location> <value>
```

Location

The aspect of the character affected by the object. It must be one of the following numbers:

```
0   NONE           No effect (typically not used).
1   STR            Apply to strength.
2   DEX            Apply to dexterity.
3   INT            Apply to intelligence.
4   WIS            Apply to wisdom.
5   CON            Apply to constitution.
6   CHA            Apply to charisma.
7   CLASS          Unimplemented.  Do not use.
8   LEVEL          Unimplemented.  Do not use.
9   AGE            Apply to character´s MUD age, in MUD-years.
10  CHAR_WEIGHT    Apply to weight.
11  CHAR_HEIGHT    Apply to height.
12  MANA           Apply to MAX mana points.
13  HIT            Apply to MAX hit points.
14  MOVE           Apply to MAX movement points.
15  GOLD           Unimplemented.  Do not use.
16  EXP            Unimplemented.  Do not use.
17  AC             Apply to armor class (AC).
18  HITROLL        Apply to hitroll.
19  DAMROLL        Apply to damage roll bonus.
20  SAVING_PARA    Apply to save throw: paralyze
21  SAVING_ROD     Apply to save throw: rods
22  SAVING_PETRI   Apply to save throw: petrif
23  SAVING_BREATH  Apply to save throw: breath
24  SAVING_SPELL   Apply to save throw: spells
```

Value

The number used to modify the Location.

For example, an A field which reads:

```
A
12 50
```

will add 50 to the maximum mana of the character.

5.5 Object File Example

```
#901
shield minotaur~
a dark minotaur shield~
A dark minotaur shield has been left here.~
~
9 dgh 513
12 0 0 0
15 5000 1350
E
shield minotaur~
A strong, sturdy shield.  It brings to mind legends of a shield that
provided protection from poisonous gases.
~
A
23 -4
A
4 2
```

This object is virtual number 901, is a Type 9 object (armor), cannot be donated, has a magical aura, and cannot be dropped. It can be picked up and worn as a shield. It has an AC-apply of 12, weighs 15 pounds, is valued at 5000 coins and costs 1350 coins per day to rent. Its Affect fields indicate that this object affects spell saving throws by -4 and increases Wisdom by 2.

6 Zone Files

Zone files are the files that control how areas are configured and how they reset. They integrate the mobiles, objects, and rooms to create an inhabited world.

A zone file contains certain initial information (specified below), followed by a series of reset commands. Each time a zone is reset, the server executes all the commands in order from beginning to end. All zones are reset when the server first boots, and periodically reset again while the game is running.

6.1 The Format of a Zone File

```
#<virtual number>
<zone name>~
<top room number> <lifespan> <reset mode>
{zero or more zone commands}
S
```

Lines starting with `*` are considered comments and ignored. Zone commands themselves may also be followed by a comment delimited by a single `*`. The zone's commands must then be terminated by the literal letter `S`.

Virtual Number

An arbitrary number used to identify the zone. Zone numbers are traditionally the room numbers of the zone divided by 100; for example, Midgaard, which consists of rooms 3000 through 3099, is zone 30.

Zone Name

A label given to the zone so that it can be identified in system logs.

Top Room Number

The highest numbered room belonging to this zone. A room with virtual number V belongs to zone N if TopRoom(zone N-1) $<$ V $<=$ TopRoom(zone N) for all N $>$ 0. Rooms belong to zone 0 if their number is between 0 and the top of zone 0.

Lifespan

The number of real-time minutes between zone resets for this zone. When the age of the zone (measured in minutes since the last time that zone has been reset) reaches the zone's lifespan, the zone is queued for reset. The zone is then reset when it reaches the front of the queue, and the conditions of the Reset Mode (see below) are satisfied.

Reset Mode

Can take one of three values (0, 1, or 2):

0

Never reset the zone. In this case, the age of the zone is never updated, and it will never be queued for reset. Thus, the value of the Lifespan is effectively ignored.

1

Reset the zone only after it reaches its Lifespan *and* after the zone becomes deserted, i.e. as soon as there are no players located within the zone (checked once every minute). This can make a zone more "fair" because it will keep the hard mobs from reappearing in the zone until everyone leaves, but on a busy MUD it can prevent a zone from ever being reset since the zone may never stay empty for more than one minute.

2

Reset the zone as soon as it reaches its Lifespan, regardless of who or what is in it. This is the most commonly used Reset Mode.

6.2 Zone Commands

Each command consists of a letter, identifying the command-type, followed by three or four arguments. The first argument, common to all the commands, is called the "if-flag." If the if-flag for a command is 1, that command is only executed if the command immediately before it was executed as well. If the if-flag is 0, the command is always executed. If-flags are useful for things like equipping mobiles–you don't want to try to equip a mobile that has not been loaded.

Commands that load mobiles and objects also include a "max existing" argument. This specifies the maximum number of copies of the mobile or object that are allowed to exist in the entire world at once. If the number currently existing is greater than or equal to the "max existing" limit, the command is not executed.

The valid zone-reset commands are M, O, G, E, P, D, and R.

M: load a mobile

Format: M <if-flag> <mob vnum> <max existing> <room vnum>

Mob vnum is the vnum of the mob to be loaded. Room vnum is the vnum of the room in which the mob should be placed. The mob will be loaded into the room.

O: load an object

Format: O <if-flag> <obj vnum> <max existing> <room vnum>

Obj vnum is the vnum of the obj to be loaded. Room vnum is the vnum of the room in which the obj should be placed. The object will be loaded and left lying on the ground.

G: give object to mobile

Format: G <if-flag> <obj vnum> <max existing>

Obj vnum is the vnum of the obj to be given. The object will be loaded and placed in the inventory of the last mobile loaded with an "M" command.

This command will usually be used with an if-flag of 1, since attempting to give an object to a non-existing mobile will result in an error.

E: equip mobile with object

Format: E <if-flag> <obj vnum> <max existing> <equipment position>

Obj vnum is the vnum of the obj to be equipped. The object will be loaded and added to the equipment list of the last mobile loaded with an "M" command. Equipment Position should be one of the following:

```
0   Used as light
1   Worn on right finger
2   Worn on left finger
3   First object worn around neck
4   Second object worn around neck
5   Worn on body
6   Worn on head
7   Worn on legs
8   Worn on feet
9   Worn on hands
10  Worn on arms
11  Worn as shield
12  Worn about body
13  Worn around waist
14  Worn around right wrist
```

```
15   Worn around left wrist
16   Wielded as a weapon
17   Held
```

This command will usually be used with an if-flag of 1, since attempting to give an object to a non-existing mobile will result in an error.

P: put object in object

Format: P <if-flag> <obj vnum 1> <max existing> <obj vnum 2>

An object with Obj Vnum 1 will be loaded, and placed inside of the copy of Obj Vnum 2 most recently loaded.

This command will usually be used with an if-flag of 1, since attempting to put an object inside of a non-existing object will result in an error.

D: set the state of a door

Format: D <if-flag> <room vnum> <exit num> <state>

Room vnum is the virtual number of the room with the door to be set. Exit num being one of:

```
0    North
1    East
2    South
3    West
4    Up
5    Down
```

State being one of:

```
0    Open
1    Closed
2    Closed and locked
```

Care should be taken to set both sides of a door correctly. Closing the north exit of one room does not automatically close the south exit of the room on the other side of the door.

R: remove object from room

Format: R <if-flag> <room vnum> <obj vnum>

If an object with vnum Obj Vnum exists in the room with vnum Room Vnum, it will be removed from the room and purged.

6.3 Zone File Example

A sample zone file annotated with comments follows.

```
#30                          * This is zone number 30
Northern Midgaard Main City~ * The name of the zone
3099 15 2                    * Top of zone is room #3099; it resets every
```

```
*                             * 15 minutes; resets regardless of people
*
* Mobile
M 0 3010 1 3062         * Load the Postmaster to room 3062
* Shopkeepers
M 0 3003 1 3011         Load the Weaponsmith into room 3011
* Now, give the weaponsmith items (to be placed in his inventory)
* max 100 of each of these objects can exist at a time in the world
G 1 3020 100                    Dagger
G 1 3021 100                    Small Sword
G 1 3022 100                    Long Sword
G 1 3023 100                    Wooden Club
G 1 3024 100                    Warhammer
G 1 3025 100                    Flail
* and lastly, give him a long sword to wield
E 1 3022 100 16                 Long Sword
* Load Boards
O 0 3099 2 3000         Mortal Bulletin Board in room 3000
O 1 3096 5 3003         Social Bulletin Board in room 3003
O 1 3096 5 3018         Social Bulletin Board in room 3018
O 1 3096 5 3022         Social Bulletin Board in room 3022
O 1 3096 5 3028         Social Bulletin Board in room 3028
* "S" must appear after all commands for a particular zone
S
```

7 Shop Files

CircleMUD v3.0 now has a new shop file format. Since the old format is still supported, both formats will be documented. If you'd like to convert shop files in the old format to that of the new format, compile and run the utility shopconv. 3.0 shops must have a special marker (described below) to tell the server that the file is in the new format.

7.1 CircleMUD v3.0 Shop Format

The overall format of a v3.0 Shop File is:

```
CircleMUD v3.0 Shop File~
<Shop 1>
<Shop 2>
.
.
.
<Shop n>
$~
```

3.0 shop files start with the literal line "`CircleMUD v3.0 Shop File~`", followed by any number of shop definitions, and terminated by $~. The format of a shop definition is:

```
#<Shop Number>~
<Item Vnum 1>
<Item Vnum 2>
<Item Vnum 3>
  .
  .
  .
<Item Vnum n>
-1
<Profit when selling>
<Profit when buying>
<Buy Type 1>  [Buy Namelist 1]
<Buy Type 2>  [Buy Namelist 1]
<Buy Type 3>  [Buy Namelist 1]
  .
  .
  .
<Buy Type n>  [Buy Namelist n]
-1
<Message when item to buy does not exist>~
<Message when item to sell does not exist>~
<Message when shop does not buy offered item>~
<Message when shop can´t afford item>~
<Message when player can´t afford item>~
<Message when successfully buying an item>~
<Message when successfully selling an item>~
<Temper>
<Shop Bitvector>
<Shop Keeper Mobile Number>
<With Who Bitvector>
<Shop Room 1>
<Shop Room 2>
<Shop Room 3>
   .
   .
   .
<Shop Room n>
-1
<Time when open start 1>
<Time when open end 1>
<Time when open start 2>
<Time when open end 2>
```

Shop Number

A unique number for the shop (used only for display purposes).

Item Vnum 1...Item Vnum n

An arbitrarily long list of the virtual numbers of objects that the shop produces (i.e., items which will always be available, no matter how many are bought). The list must be terminated with -1.

Profit When Selling

The price of an object when a shopkeeper sells it is the object's value times Profit When Selling. This is a floating point value. It should be >= 1.0.

Profit When Buying

The amount of money a shopkeeper will offer when buying an object is the object's value times Profit When Buying. This is a floating point value. It should be <= 1.0.

Buy Types and Buy Namelists

These lines control what types of items that the shop will buy. There can be an arbitrarily long list of buy types terminated by -1. The first argument, called "Buy Type" is the Type Flag of items the shop will buy (see "Type Flag" under "Format of an Object" in this document). Numerical and English forms are both valid (5 or WEAPON, 9 or ARMOR, etc.).

The second (optional) argument is called a Buy Namelist and allows you to provide optional keywords to define specific keywords that must be present on the objects for the shopkeeper to buy or sell it. For further details on these expressions, see the section "Item Name Lists" below.

Message when item to buy does not exist

The message given to a player if he tries to buy an item that the shopkeeper does not have in his inventory.

Message when item to sell does not exist

The message given to a player if he tries to sell an item that the player does not have in his inventory.

Message when shop does not buy offered item

The message given to a player if he tries to sell an item that the shopkeeper does not want to buy (controlled by the Buy Types and Buy Namelists.)

Message when shop can't afford item

The message given to a player if he tries to sell an item to a shop, but the shopkeeper does not have enough money to buy it.

Message when player can't afford item

The message given to a player if he tries to buy an item from a shop but doesn't have enough money.

Message when successfully buying an item

The message given to a player when he successfully buys an item from a shop. The expression `%d` can be used in place of the cost of the item (e.g., `That'll cost you %d coins, thanks for your business!`

Message when successfully selling an item

The message given to a player when he successfully sells an item to a shop. The expression `%d` can be used in place of the cost of the item as above.

Temper

When player can't afford an item, the shopkeeper tells them they can't afford the item and then can perform an additional action (-1, 0, or 1):

-1

No action other than the message.

0

The shopkeeper pukes on the player.

1

The shopkeeper smokes his joint.

Further actions can be added by your local coder (e.g., attacking a player, stealing his money, etc.)

Shop Bitvector

A bitvector (see section 2.4) with the following values:

1	a	WILL_START_FIGHT	Players can to try to kill shopkeeper.
2	b	WILL_BANK_MONEY	Shopkeeper will put money over 15000 coins in the bank.

A brief note: Shopkeepers should be hard (if even possible) to kill. The benefits players can receive from killing them is enough to unbalance most non monty-haul campaigns.

Shop Keeper Mobile Number

Virtual number of the shopkeeper mobile.

With Who Bitvector

A bitvector (see section 2.4) used to designate certain alignments or classes that the shop will not trade with, with the following values:

1	a	NOGOOD	Don´t trade with positively-aligned players.
2	b	NOEVIL	Don´t trade with evilly-aligned players.
4	c	NONEUTRAL	Don´t trade with neutrally-aligned players.
8	d	NOMAGIC_USER	Don´t trade with the Mage class.
16	e	NOCLERIC	Don´t trade with the Cleric class.
32	f	NOTHIEF	Don´t trade with the Thief class.
64	g	NOWARRIOR	Don´t trade with the Warrior class.

Shop Room 1...Shop Room n

The virtual numbers the mobile must be in for the shop to be effective. (So trans'ed shopkeepers can't sell in the desert). The list can be arbitrarily long but must be terminated by a -1.

Times when open

The times (in MUD-hours) between which the shop is open. Two sets of Open/Close pairs are allowed so that the shop can be open twice a day (for example, once in the morning and once at night). To have a shop which is always open, these four values should be

```
0
28
0
0
```

7.2 Item Name Lists for 3.0 Shops

Name lists are formed by boolean expressions. The following operators are available:

```
´,^ = Not      *, \& = And     +, | = Or
```

The precedence is Parenthesis, Not, And, Or. Take the following line for an example:

```
WEAPON [sword & long|short | warhammer | ^golden & bow] & magic
```

This shop will buy the following items of type WEAPON:

1. sword long magic
2. short magic (the first & is done before the first |)
3. warhammer magic
4. ^golden bow magic

Note that the ^ in front of golden affects ONLY golden, and nothing else in the listing. Basically, the above expression could be written in English as:

```
[(sword and long) or short or warhammer or (not golden and bow)] and magic
```

If you want the shop to only buy "short magic" only if they were also swords, you could change the expression to:

```
WEAPON [sword \& (long|short) | warhammer | ^golden \& bow] \& magic
                ^-Changes--^
```

You can also include object extra flags (listed in the section "Format of an Object" above). The previous example used "magic" as a keyword that had to be on the object. If we wanted to make it so that the MAGIC flag had to be set on the item, we would change "magic" to "MAGIC." Similar changes could be made to add other flags such as "HUM" or "GLOW." It should be noted that these expressions are case sensitive and that all keywords should appear in lower-case, while the flag names should be in all caps.

7.3 The DikuMUD Gamma and CircleMUD 2.20 Shop Format

This format is obsolete but is presented because it is still supported by CircleMUD 3.0. In most cases, it is easier to simply use the "shopconv" utility shipped with Circle to convert older shop files to the new format.

```
#num~
     Shop Number (Used only for display purposes)

num1
num2
num3
```

```
num4
num5
     Virtual numbers of the objects that the shop produces.  -1´s should be
inserted in unused slots.

Profit when selling
     The object value is multiplied by this value when sold.  This is a
floating point value.  Must be >= 1.0

Profit when buying
     The object value is multiplied by this value when bought.  This is a
floating point value.  Must be <= 1.0

num1
num2
num3
num4
num5
     These five numbers are the item-types traded with by the shop (i.e.
valid Type Flags of objects that the shopkeeper will buy).

Message When Item to buy is non existing~
Message When item trying to sell is non existing~
Message When wrong item-type sold~
Message when shop can´t afford item~
Message when player can´t afford item~
Message when buying an item~
     Price is represented by %d.
Message when selling an item~
     Price is represented by %d.

Temper
     When player can´t afford an item, the shopkeeper tells them they can´t
afford the item and then:
     0 - The shopkeeper pukes on the player.
     1 - The shopkeeper smokes his joint.
     other - No action besides message above.

Shop Bitvector
     A bitvector (see section "Using Bitvectors" above) with the following values:

1    a    WILL_START_FIGHT    Players can to try to kill this shopkeeper.
2    b    WILL_BANK_MONEY     Shopkeeper will put money over 15000 coins
                              in the bank.
```

```
     A brief note:  Shopkeepers should be hard (if even possible) to kill.
The benefits players can receive from killing them is enough to unbalance
most non monty-haul campaigns.

Shop Keeper Mobile Number
     Virtual number of the shopkeeper mobile.

With Who Bitvector
     A bitvector (see section "Using Bitvectors" above) used to designate certain
alignments or classes that the shop will not trade with, with the following
values:

1     a   NOGOOD         Keeper won't trade with positively-aligned players.
2     b   NOEVIL         Keeper won't trade with evilly-aligned players.
4     c   NONEUTRAL      Keeper won't trade with neutrally-aligned players.
8     d   NOMAGIC_USER   Keeper won't trade with the Mage class.
16    e   NOCLERIC       Keeper won't trade with the Cleric class.
32    f   NOTHIEF        Keeper won't trade with the Thief class.
64    g   NOWARRIOR      Keeper won't trade with the Warrior class.

Shop Room Number
     The virtual number the mobile must be in for the shop to be effective.
(So trans'ed shopkeepers can't sell in the desert).

Time when open start 1
Time when open end 1
     The hours between which the shop is open.

Time when open start 2
Time when open end 2
     The hours between which the shop is open.
```

A Spell Numbers

These spell numbers are the spells shipped by default with CircleMUD 3.0. Note that most implementors add new spells so this list may not be complete for your particular MUD. Check with your implementor for details.

animate dead	45
armor	1
bless	3
blindness	4
burning hands	5

call lightning	6
charm	7
chill touch	8
clone	9
color spray	10
control weather	11
create food	12
create water	13
cure blind	14
cure critic	15
cure light	16
curse	17
detect align	18
detect invis	19
detect magic	20
detect poison	21
dispel evil	22
dispel good	46
earthquake	23
enchant weapon	24
energy drain	25
fireball	26
group armor	47
group heal	48
group recall	49
harm	27
heal	28
identify	201
infravision	50
invisible	29
lightning bolt	30
locate object	31
magic missile	32
poison	33
prot from evil	34
remove curse	35
remove poison	43
sanctuary	36
sense life	44
shocking grasp	37
sleep	38
strength	39
summon	40
teleport	2
ventriloquate	41
waterwalk	51
word of recall	42

B Item Values for Drink Containers

DRINKCON (Type Flag 17) and FOUNTAIN (Type Flag 23)

```
value 0: Capacity of container
value 1: Current quantity in container
value 2: see below
value 3: Non-zero if the drink poisoned, 0 otherwise.
```

Value 2 is a number which defines the type of liquid in the drink container, from the following table:

Type	nr.	Effect of Liquid On:		
		Drunkness	Fullness	Thirst
LIQ_WATER	0	0	1	10
LIQ_BEER	1	3	2	5
LIQ_WINE	2	5	2	5
LIQ_ALE	3	2	2	5
LIQ_DARKALE	4	1	2	5
LIQ_WHISKY	5	6	1	4
LIQ_LEMONADE	6	0	1	8
LIQ_FIREBRT	7	10	0	0
LIQ_LOCALSPC	8	3	3	3
LIQ_SLIME	9	0	4	-8
LIQ_MILK	10	0	3	6
LIQ_TEA	11	0	1	6
LIQ_COFFE	12	0	1	6
LIQ_BLOOD	13	0	2	-1
LIQ_SALTWATER	14	0	1	-2
LIQ_CLEARWATER	15	0	0	13

The above values for drunkness, fullness, and thirst are in the units of one hour of effect per four units of liquid drunk. For example, imagine that Quifael drinks an entire bottle (say 7 units) of saltwater. According to the table above, saltwater has a drunkness value of 0, fullness value of 1 and thirst value of -2. Therefore:

His Drunkness is not changed ((7/4)*0) His Fullness increases by ((7/4)*1) hours His Thirst increases by ((7/4)*-2) hours, thus making him more thirsty.

A player's drunkness, fullness, and thirst can range from 0 to 24. 24 is the maximum; 0 means the person is completely sober, hungry, or thirsty respectively. For immortals, these values are typically -1.