The Dirt Game System

A manual for the aspiring world builder

Copyright (C)1993 The Free Software Foundation, Inc 675 Mass Ave, Cambridge, MA 02139, USA

written by Valentin Popescu (vpopesc@opus.calstatela.edu)

additions/changes/modifications by Vitastjern (Anna.Eklund@ludd.luth.se)

This document is for the Dirt 3.0 game system, modified for Northern Lights.

A zone as used within this document, is a text file written in a special format (a language, if you will). It contains definitions for objects, mobiles, and/or rooms used in the game.

This is intended mostly as a reference manual. Don't be afraid to experiment, and always work on backups.

Zones

"...and in the beginning there was nothing."

What exactly is a zone?

A zone, as we said before, is nothing more than a text file. All you need to create one is an editor that writes plain text files, a little spare time, and a lot of imagination.

Assuming you have finished your zone, the following steps should be taken to create your zone:

1. add a '.zone' ending to your zone's filename. For example, if you created a new zone called 'askani', rename it to 'askani.zone'.

2. copy your zone to the ../data/ZONE directory

3. edit the ../data/ZONE/files and add your new zone's (askani.zone) name there. Each line in the file is one entry. The first word on a line is the zone name as it will be seen in the game, the second one is the file name (askani.zone). If you

simply put in askani.zone without worrying about the first name, the game will simply name the zone "askani".

4. rebuild the world. At this point you will be notified of any errors in the file. Major errors will be reported immediately. Minor errors will be placed in a file in the data dir, called mkdata.log. Correct those errors and rebuild the world.

Tips for creating and maintaining your zone

When creating your zone, it is helpful to divide it in three parts, namely objects, mobiles, and locations. When the world is build, a special program will scan the zone file, and look for one of three keywords:

```
%mobiles    %objects   %locations
```

What these keywords do, is tell the program what type of data follow. For example, the outline of a zone file may look like this:

%mobiles

[definitions for your objects]

%objects

[definitions for your mobiles]

%locations

[definitions for your locations (rooms)]

Note: The order of the above fields is important, since the C preprocessor that translates the data to machine readable form will else fail.

Also, you must add the following three lines at the very beginning of a zone file:

```
#include "undef.h"
#include "cflags.h"
```

They must start at the very begining of the line. They read the definitions of the flags you will be learning about in subsequent pages, and also prevent some headaches you may have later with cpp interaction.

Advanced features

Remember, zones are run through the C preprocessor when the world is rebuilt. Therefore, you can use C preprocessor features, such as comments using the /* and */ delimiters, macros, etc. If you don’t exactly know what that means, don’t worry about it.

Or.. well. just remember.. to put a comment inside a zone file, enclose it between /* and */'s. For example:

```
/* this is a comment */
```

This will have absolutely no effect on the zone. Nothing after a /* will be processed until a */ is found.

If you use #defines, try to use capital letters for your macro name. This is a convention, and also a way to prevent headaches when one of your macro names appears inside a decription.

Creating Mobiles

"...or things that go bump in the night."

What is a mobile?

A mobile (or NPC, or whatever you want to call it) is whatever moves on the game and doesn't appear in the 'who' listing.

To start off, remember to preface your mobile data with the %mobiles keyword. This has to be done at least once in the file.

See the Zones section for more info on this.

A typical mobile can be defined by the following fields. Note that many of these are not required.

```
%mobiles

Name             = Askani
Pname            = "Lady Askani"
Location         = waitingroom
SFlags           { Female }
MFlags           { CanFireball CanFrost }
PFlags           { NoZap }
Strength         = 150
Damage           = 20
Armor            = 0
Aggression       = 50
Speed            = 5
Description      = "Askani the Lady in Waiting is here."
Examine          = "
You see a beautiful lady in waiting before you."
```

```
End             = Askani
```

Some explanations are in order here.

Notice the fields *Name* and *End*. They are the most important fields in the description. The Name field gives the mobile a name, and also signals that what follows should be assigned to that mobile, until the End field is encountered. The Name and End field should equal the same thing.

The *Pname* field is the name which will be seen by players. It may be more than one word, but in that case enclose it in quotes. If the Name field is the same as the name you wish the players to see, you don't have to use this field.

Location is the room in which the mobile starts off. This is a text string, which will be explained better in the section describing location definitions.

The *Strength* is the mobile's strength. An averagely strong mobile will be ca 120 strength, whereas a tough one would be around 300.

The *Damage* is how much damage a mobile does on a successful hit. Of course, that is altered based on a number of factors, but think of it as the base damage the mobile will usually do.

The *Armor* field acts as invisible armor. A mobile with an armor field of 10 will be like wearing a shield of 10AC. You can use this for mobiles that in their description are described as wearing armor.

Speed is how fast the mobile moves around. If you want your mobile not to move around at all, set this to 0.

Aggresion is what is the probability that mobile will attack a mortal in the room, on every turn. A turn only takes a few seconds.

Description is what you see when you enter the room that mobile is in.

Examine is what you see when you examine the mobile. (Note: Unfortunately this field currently causes the mud to crash, so any special examine texts will have to be added as a separate file just like the players' descriptions.)

The remaining fields, *SFlags*, *MFlags*, and *PFlags* are a collection of flag fields. Note that you should not use the = sign on those fields, but rather enclose the needed flags in curly brackets, { and }'s.

The SFlags

The only sflag you should think about using is the Female sflag. The rest should not be used in a zone file. If you want your mobile to be male, just delete this field altogether.

The PFlags

The PFlags you may use here are:

NoExorcise
: This mobile may not be exorcised from the game (cannot be kicked off) except by the very highest powers.

NoZap
: This mobile cannot be zapped except by the very highest powers.

NoMagic
: You cannot use aggresive magic spells on this mobile.

NoSummon
: Lower powers cannot summon this mobile.

NoHassle
: This mobile cannot be attacked.

NoSteal
: You can't steal from this mobile.

NoAlias
: This mobile cannot be aliased.

Or of course, you can leave this field out altogether.

The MFlags

This is the main set of flags for a mobile. You can think of these as MobileFlags.

CanFireball
: This mobile may use fireballs during fights.

CanMissile
: This mobile may use missile spells during fights.

CanShock
: This mobile may use shock spells during fights.

CanFrost
: This mobile may use frost spells during fights.

CanCurse

This mobile might curse you during fights.

NegFrost
: This mobile will get **stronger** if you use the frost spell against it.

NegFireball
: This mobile will get **stronger** if you use the fireball spell against it.

NegMissile
: This mobile will get **stronger** if you use the missile spell against it.

NegShock
: This mobile will get **stronger** if you use the shock spell against it.

NoHeat
: This mobile will not attack you if you are carrying a source of light. Think of the yeti in the blizzard pass.

Thief
: This mobile may steal stuff you are carrying.

StealWorn
: This mobile may steal stuff you are wearing.

StealWeapon
: This mobile may steal the weapon you are wielding.

PitIt
: This mobile may pit stuff it picks up. See PickStuff.

DrainLev
: You may lose your current level if you fight this mobile. This is a more serious version of the DrainScr flag.

DrainScr
: You may lose points if you attack this mobile. Think of the Wraith near Shazareth.

BarDir
: The flags BarNorth, BarSouth, BarWest, BarEast, BarDown, and BarUp will make the mobile hold back mortals trying to go that direction. Think of the figure in the library.

Qfood
: This mobile will quit the game if it is given a food item.

Blind

This mobile will attempt to blind you if you attack it.

Deaf
: This mobile will attempt to deafen you if you attack it.

Dumb
: This mobile will attempt to mute you if you attack it.

Cripple
: This mobile will attempt to cripple you if you attack it.

NoGrab
: This mobile will not let you pick up stuff in the same room as it is.

GrabHostile
: Same as NoGrab, but the mobile will also get nasty and attack you if you try to pick up something.

PickStuff
: This mobile will pick up stuff it finds while wandering around. It will also wield the best weapon it can find and wear all armor it picks up.

NoSteal
: You may not steal from this mobile.

NoCorpse
: This mobile will leave no corpse if killed.

FearHoly
: This mobile will not attack if you are carrying a holy item, such as the cross for instance. Think of the Wraith.

NoCorpse
: This mobile will leave no corpse on the ground when killed.

Creating Objects

"A rose by any other name..."

What is an object?

An object on mud can be many things. It can be as simple as a banana, or as complex as an invisible mist blocking an exit. This chapter will deal with basic objects. Refer to the "doors" chapter for an analysis of linked objects, such as doors.

A standard object may contain the following fields:

```
Name            = little_knife
AltName         = dagger
PName           = "knife"
State           = 1
MaxState        = 1
Armor           = 0
Damage          = 5
BValue          = 30
Size            = 5
Weight          = 4
Counter         = 0
UnlockText      = "txt1"
OpenText        = "txt2"
CloseText       = "txt3"
LockText        = "txt4"
Location        = IN_ROOM:armory@arcadia
Desc[0]         = "
A small knife has been dropped here."
Oflags          { Weapon GetFlips }
Desc[1]         = "
A small knife rests on the table."
Examine         = "
It  is  a  small  knife  covered  with  blood."
End             = little_knife
```

There are a few other fields that have not been included above, namely the *Linked and Key* fields. (Note that the Key field can be used for any lockable object, not only doors.) These fields will be discussed in the doors chapter. Also many of these fields are not required.

Once again, the field defined by *Name* is the object's label. At the end of the description, note the *End* field, which has the same value as Name.

The *AltName* field is an alternative name for the object. It may help mortals to pick up the object. For example, a branch may be alternatively called a stick, or a knife may be called a dagger.

The *Pname* field is how players will refer to the object, although AltName will still be valid. This will be the primary name. If the name defined in Name is the one you wish mortals to use, you need not bother with this field.

Armor is the armor class of the object, if it is wearable. It must have the oflags Wearable and Armor set, for this field to have any effect.

Damage is the damage a weapon may make. It must have the Weapon oflag set for this to have any effect.

BValue is the base value of the object. The actual value is adjusted according to the number of players on the game (the more players, the higher the value.)

Size and *Weight* are self explanatory.

The *Counter* entry is mainly used to code special cases. It might however get used somewhen in the future as a way to compute how long burning objects will last.

MaxState and *State* are special flags. Each object has a state asociated with it. Maxstate specifies the maximum state, whereas State specifies the initial state. According to the state of the object, a different description may be seen. For example, if the state of the object is 0, the test defined as Desc[0] will be seen upon entering the room. If state is 1, then Desc[1].. etc. See also the discussion on the GetFlips and PushToggle oflags, at the end of this chapter.

The *UnlockText, OpenText, CloseText and LockText* are used if you wish to specify a special message when you open/close/unlock/lock an object or a door. There is no need to specify all these strings. Any string you don't specify will default to the value NULL and result in a standard message. If the OpenText value is not NULL, while the CloseText value is NULL, but the object does not have the oflag Openable, then players will be able to open the object, but it will not be able to be closed again. The same may be applied to the oflag Lockable. Put generally, setting values other than NULL to the text values will override the flags. This will allow the creation of objects that can be open/closed/locked/unlocked only once.

An example on the usage of LockText

```
Name      = gate
Pname     = "gate"
Altname   = "padlock"
Location  = IN_ROOM:Somewhere
Oflags    { Openable NoGet }
State     = 1
MaxState  = 2
Linked    = gate2
Desc[0]   = "An iron gate sits here wide open."
Desc[1]   = "
A closed iron gate has a open padlock waiting to be locked"
Desc[2]   = "
The gate is locked tight by an old looking padlock."
LockText  = "You close the gate and snap the padlock closed."
End       = gate

Name      = gate2
Pname     = "gate"
Altname   = "padlock"
Location  = IN_ROOM:Somewhere_else
Oflags    { Openable NoGet }
State     = 1
MaxState  = 2
Linked    = gate
```

```
Desc[0]    = "An iron gate sits here wide open."
Desc[1]    = "
A closed iron gate has a open padlock waiting to be locked"
Desc[2]    = "
The gate is locked tight by an old looking padlock."
LockText   = "You close the gate and snap the padlock closed."
End        = gate2
```

The gate can be opened and closed as much as the players like, once it has been locked though, thats it, locked till the next reset, or a wizard+ opens it of course. If the filed *Key = key@church* to each of the gates then only someone with the key from the church would be able to lock the gate... It would still remain locked after that though.

The *Desc[x]* fields are the object descriptions upon entering a room. The current description will be chosen according to the current state of the object. See the previous paragraph, and also the GetFlips and PushToggle flags. Also refer to the appendix, the section on entering string fields. Sometimes it is desirable that the objects have no description. That is fine, just leave the field out. Or you may specify just one field. Or three. The maximum number of states is four, ranging from 0 to 3.

The location of an object is a bit more complex than it appears. An object can be in one of six states: carried by someone, worn by someone, wielded by someone, both wielded and worn by someone, in a container, or in a room. The way you can specify this is by the use of six different flags. Example:

```
Location        = CARRIED_BY:puff
Location        = IN_ROOM:forest
Location        = WORN_BY:seamas
Location        = WIELDED_BY:cosimo
Location        = BOTH_BY:asmodeus
Location        = IN_CONTAINER:sack
```

In first example, an object is carried by mobile Puff. In second example, the object is simply in the room labelled "forest". In the third example, it is carried by the mobile Seamas. The fourth example is for a weapon, wielded by the mobile Cosimo. The fifth example is an object that is both wielded and worn by the mobile Asmodeus. In the last example, the object is in the sack.

The *Oflags* field is a collection of object flags, enclosed in curly brackets, {'s and }'s. Note there is no = sign on this field.

Object Flags and their meanings

The following object flags are acceptable:

Destroyed
: This oflag really means 'invisible to mortals'. Indeed, mortals will not be able to see it.

NoGet

This object cannot be taken using the "take" or "get" commands. It should be use for objects that should not be taken, like doors furniture, etc.

Openable

This object can be opened. Note that opening an object toggle its state. The opened state is state 0, and Desc[0] should reflect this. Desc[1] reflects on the object's closed state. The OpenText and CloseText fields can be specified to give the object an customized text for the open and close commands.

Lockable

This object can be locked. Note that it should also be openable. A locked object requires a key to be opened. The locked state is state 2, Desc[2] should reflect this. The LockText and UnlockText fields can be specified to give the object an customized text for the lock and unlock commands. Use the Key field if you wish to make the object openable by one key only.

Pushable

This object is pushable. The state will change to 0 (once) when it is pushed.

PushToggle

This object will toggle status back and forth when pushed.

Food

This object is edible, and will improve stamina (hitpoints) when eaten.

Armor

This object is armor. The Armor field will be considered during fights. The object must also be set wearable.

Wearable

This object can be worn by a player. That doesn't mean it is armor.

Lightable

The object can be lit, to provide light.

Extinguish

The object can be extinguished.

Key

The object can be used to unlock doors.

GetFlips

This flag will cause the status of the object to be set to 0, when a successful get/take is performed. The reason is as follows.

At times, you may have to have a room, with a table, and on the table a pen described as "A pen is on the table." However, if someone takes the pen, and then drops it in a room with no table, the description will still read "A pen is on the table." Getflips to the rescue!

By setting GetFlips on, and State to 1, you can set Desc[1] to the original description. When someone takes the object, it will be toggled to state 0, so if he drops it again, Desc[0] will be seen.

Or in other words:

```
MaxState = 1
State    = 1
Oflags   { GetFlips }
Desc[0]  = "A pen has been dropped here."
Desc[1]  = "A pen is on the table."
```

Lit
: The object is providing light. This does not imply it is extinguishable, or re–lightable.

Container
: The object is a container. You can put stuff in it, and take stuff out. The amount of objects that can be put in the container is depending of its size.

Weapon
: The object is a weapon. The Weapon field will be taken in consideration during fights.

Boat
: The object is a boat. It will allow mortals to travel over OnWater rooms.

Broken
: The meaning of this flag is a bit obscure, and left to the interpretation of the programmer. For example, if the flags Boat and Broken are set, the boat would sink at sea.

Unlocatable
: This object cannot be located using the where/locate spells.

NoSummon
: This object cannot be summoned.

Booze
: This object will make you drunk if you drink/eat it. It must have the Food flag set.

Holy
: The object will keep mobiles with FearHoly flag from attacking.

Cursed
: This object is cursed. (Will have some bad affects.)

Blessed
: This object is blessed. Some of the objects effects might be improved.

Magical
: Just a flag to show players that the object might have some interesting properties.

Creating Rooms

"Where no man has gone before!"

What is a room?

A room is what you find yourself in every time you move on a mud. It is composed of four main parts:

1. Room number and exits
2. Room flags
3. Room title
4. Room description

Room entries may look as follows:

```
%locations

waitingroom n:bedroom e:outside w:hallway;
lflags {}
The Waiting Room^
 You stand  inside  a small room. Nothing fancy here, just
ladies in waiting. Maybe you should leave?
^

outside n:forest2 e:forest1 s:forest@arcadia w:waitingroom;
lflags {Outdoors NoSummon}
In the garden^
 You stand in a beautiful garden.
^
```

You just saw how two rooms are defined. For each room, the **very first** field is the label of the room. In the case of the first room it is labelled "waitingroom", and in the case of the second one, "outside". You may use any name you wish here, but you may not have two different rooms with the same label inside the same zone. This will never be seen by players, it is just for programmers.

The next fields on the same line are exits. **e:outside** simply means that there should be an exit leading to the room labelled "outside" in the current zone. Valid exits are **n**, **e**, **s**, **w**, **u**, **d**. To make it easier for the programmer, please keep them in the order mentioned above.

When making an exit to another room, you may want to make sure that room has an exit leading back to this original room, else the player will get stuck there without an exit.

Direct your attention now to the second room, exit leading south. It states **s:forest@arcadia**.

What this means is that the exit to the south will lead to the room labelled forest, but in a **different** zone, namely arcadia.

There is a special kind of exits, which depend on objects. That is the case with doors. However, refer to the section on doors for a detailed explanation of handling those.

To finish up this very first line of a room definition, the trailing semicolon (**;**) must be included. It signals the end of that particular field.

The second line is a set of lflags. If you don't want any special lflags for this room, you must still specify that line, but put nothing in between the curly brackets. The various flags are described at the end of this chapter.

The third field is the title of the room. Notice the trailing **^** sign. That sign must exist there, to mark the end of the title. It will not be printed to players.

The remaining lines of the room are the room description itself. As with the title, end it with a **^** on an empty line. It is standard practice that room descriptions start with a 4−letter indentation, but that is up to you.

Lflags and their meaning

Valid lflags are as follows:

Death
: This is a very special room, which mortals dread very much. Any mortals entering this room will die. They will not lose points, but their inventory will be trapped here, and they will be kicked off the game.

CantSummon
: People cannot be summoned away from this room.

NoSummon
: People cannot be summoned to this room.

NoQuit

Players cannot quit from this room.

NoSnoop

Players may not be snooped in this room.

NoMobiles

Mobiles will not enter this room. This flag is useful to stop mobiles from wandering in or out certain places.

NoMagic

Mortals may not cast magic from inside this room.

Peaceful

There will be no fighting permitted inside this room. This is a safe haven for mortals.

Soundproof

This room is soundproof. Tells, shouts, etc., will not be heard inside the room.

OnePerson

This room is only big enough for one person. Mortals will not be able to enter it if there is already a person inside.

Party

Anyone is allowed to use the Emote command here.

Private

This room is only big enough for two persons. A third mortal trying to enter the room will be told the room is full.

OnWater

This room is on−water. A boat will be required for mortals.

Underwater

This room is under water. Mortals will need some magic object to survive here.

Outdoors

This room is outdoors. Weather messages will be seen and time of day and night will show. Do not combine this flag with TempExtreme or Seasons.

TempExtreme

There will be daytime changes in this room. Hot during the day, cold during the night. A fitting flag for a desert. Do not combine this flag with Outdoors or Seasons.

Seasons

This room is subject to day/night messages, and the effects thereof. Do not combine this flag with Outdoors or TempExtreme.

NoSleep

It is too dangerous to fall asleep in this room.

NegRegen

Mortals will actually lose strength if sitting around in this room.

NoRegen

Mortals will not regain any strength while in this room.

Cold

This room is a very cold room. If it is outdoors, any rain will turn to snow.

Hot

This room is a very hot room. Mortals will need protection.

Maze

This room is a part of a maze. Players with the Goto pflag will not see the full names of the exits.

Creating Doors

"Break on through to the other side..." – The Doors

What's so special about doors?

Well, nothing is special about doors, except that you need two of them. First, let's consider two rooms, labeled 'outside' and 'inside'.

```
outside s:^outside_door;
lflags {Outdoors}
Outside^
    You are outside.
^

inside n:^inside_door;
lflags {}
Inside^
    You are inside.
^
```

Notice how the two room exits, **s:** and **n:** respectivelly, point to something starting with a **^** sign? That marks an object. What this means, is that the room exit depends on the state of the object named there.

Let’s now define the two objects, inside_door and outside_door:

```
Name     = inside_door
PName    = door
Linked   = outside_door
State    = 1
MaxState= 2
Key      = key@church
Location= IN_ROOM:inside
oflags  { Openable NoGet Lockable }
Desc[0] = "The door is open."
Desc[1] = "The door is closed."
Desc[2] = "The door is locked."
End      = inside_door


Name     = outside_door
PName    = door
Linked   = inside_door
State    = 1
MaxState= 2
Key      = key@church
Location= IN_ROOM:outside
oflags  { Openable NoGet Lockable }
Desc[0] = "The door is open."
Desc[1] = "The door is closed."
Desc[2] = "The door is locked."
End      = outside_door
```

This is all it takes to make a door. Note the Linked fields on the object descriptions. They point to each others. So, in theory, what happens is that if someone tries to go through an exit that points to an object, the game will check the status of the object. If the object is status 0, it will check the Linked field, and allow the player to move into the room where the Linked object is.

Note that you do NOT have to always specify a *Key* field. As a matter of fact, most of the time, you should not specify this field at all. When the key field is empty, any key will be able to lock and unlock this door.

So, to recap. To make a door, you must do the following:

1. Create two objects, one for each side of the door. The objects must have a Linked field. On the Linked field, put each other’s name. Note that the Openable/Lockable flags depend on your applications. For some applications, you may not need those flags. NoGet flag is highly desirable, tho! If you wish the door to be unlocked/locked with a special key (it does not need to have the "key" oflag), add a Key field to the doors. You can also decide to have special texts for when the doors gets opened/closed/locked/unlocked. This is explained

in the Creating Objects section.

2. Edit the description of the room you are leaving from and set the respective exit to ^object_name, where object_name is the label of the first object you created. Then edit the descriptions of the destination room, and put an exit back to the first room, pointing to the other object.

Some Important Notes

"Mind your step!"

Referencing one zone from another

You may reference any object contained in one zone, from another zone, using the @ sign. For example, you would like to refer to the object stick in room blizzard, you would specify it as **stick@blizzard**. Some examples:

```
Location        = forest@arcadia
Location        = CARRIED_BY:dragon@cave
```

etc...

Entering text strings

Certain fields accept variable length strings. However, you must first acquaint yourself with the legal (and illegal) ways to type in a string.

Legal ways:

```
Field1  = "This is legal."

Field2  = "
This is also perfectly legal."

Field3  = "
And this too. There is no problem with typing in a few lines
of description. Just make sure you don't forget the ending
quote."
```

Illegal ways:

```
Field1 =
"This is not legal. The beginning quote must be on the same
field as the equal sign."
```

Last edited April 19, 1994.
aber@ludd.luth.se