VME basic zone writing manual

Ken Perry

VME basic zone writing manual

by Ken Perry

Table of Contents

List of Tables

Chapter 1. Introduction

1.1. Giving credit where credit is due!

The, basic zone writing manual, you are reading now, didn't just come flying out of this authors head. In fact it is a conglomeration of several old texts which are no longer used. I had first thought about quoting and giving credit to each person who had ever modified or created portions of the first documentation but because of the nature of the way they were built it was impossible to know who did what. I therefore have decided to create a list here of everyone who has ever worked on or had a hand in the development of the basic zone writing documentation and references. I will also give a short summary for each. I would also like to thank those listed below for their contribution to the development of one of the best mud servers on the internet today.

Original DIKU coding Team

- Hans Henrik Staerfeldt
- Sebastian Hammer
- Michael Seifert
- Katja Nyboe
- Tom Madsen
- Lars Balker Rasmussen

You should all know these people they are not only the original DIKU mud developers but a few of them designed and developed the new VME server. One or more of the previous mentioned people were the authors of the following texts that have been swallowed up by this book.

- `abilities.txt`
- `guild.txt`
- `monster.txt`
- `objects.txt`
- `rooms.txt`
- `vmc.txt`

Andrew Cowan

One of the original administrators of Valhalla mud and now the administrator of the mud connector. Andrew created the first zone tutorial, which was later included into the `vmc.txt` to clear up some things missing in the old `vmc.txt`. Again my thanks go out to Andrew for his contributions not only to this document but to the growth of the DIL programming language.

Ryan Holliday

Made major updates to the `tutorial.txt` created by Andrew Cowan, which became version two of the tutorial.

Peter Ryskin

Wrote the original explanation of how to define exits.

John Clare

Made major updates to the `tutorial.txt` created by Andrew Cowan and updated by Ryan Holliday, which became version three of the tutorial.

Marc Bellemare

An editor and an author Marc combined and revised the `tutorial` and the `vmc.txt` into one document making it less Valhalla specific and more for any mud running the VME server.

Jennifer Garuba

Created the first document on how to create shop keepers. She also was one of the first builders to have the insite and to make it clear that a manual like this was needed.

Brian Spanton

Converted the original `vmc.txt` into HTMl, while at the same time fixing many inconsistencies and typos in it.

Kathy Perry

Wrote the original `compiler howto` to make it easier for new builders to compile their zone. She was also the first builder to open my eyes to the fact we needed a true manual on how to build.

Mark Pringle

Main editor, which spell checked and looked over my major grammer mistakes. I am sure he didn't catch them all but at least the ugly ones were caught.

Morgan Shafer

Wrote the guild definition primer which explained teachers.

1.2. Who should read this book?

This book was designed to be read by anyone who is thinking of writing areas for a VME server. If you have wrote for other mud servers you will still need to at least skim every chapter because the VME is like no other mud engine and has some very interesting differences.

1.3. What does this book cover?

The topics covered in this book are everything to do with writing an area for the VME server. While we do cover using DIL functions to make your monsters, Rooms, and objects better and smarter we do not cover how to write the DIL functions covered in the DIL manual. The following are topics covered by this book.

- compiling and debugging
- Macros using the CPP
- Overall Zone structure
- Writing rooms
- Writing Objects
- Writing Monsters
- Doing the resets

Chapter 2. General compiler information

In order to get your zone onto a valhalla Mud Engine (VME) server you must convert your zone from readable english text to binary form the server can understand. The way you do this is with a compiler. No don't freak out you don't have to be a skilled programmer to use a compiler. The only thing you have to do is format your rooms, objects, and Non-player characters (NPC) in a form which the compiler can understand. The great thing about the VME is you can do all your zone writing in your favorite editor with out having to log on to code. For those of you who have coded for other mud servers and are used to coding online this may be a new experience to you but you will find you can plan out better and more well designed areas offline than you can on line.

> **Note:** In the future the VME coding team is thinking of adding an online coding module for those mud administrators that can not live with out it. If you are one of these make sure you write <whistler@valhalla.com> and express your desires so you can be counted.

This chapter will mainly cover the Valhalla Mud Compiler (VMC), how it works, and the Valhalla Mud pre processor (VMC -p) works. We will also throw in some debugging hints but debugging will be covered more as you begin creating parts of your areas in the following chapters.

2.1. The compiler

VMC is the Valhalla Mud Engine Compiler. for VME servers. A compiler takes a source file or better described as your areas input file and converts it to a binary file the server can then load and use online. In the VME we call areas you build 'zones', therefore the source file for a zone has the extension 'zon'. In order to make this more clear we will start with our first example.

Lets say you were making a Zone of dragons. You may want to call the file something resembling its contents like, `dragon.zon`. Notice we have appended the '.zon' extension. The compiler requires all zones to end in '.zon' in order for it to know this is a zone source file.

Now lets say we have completed writing our first zone and want to compile it. The command is simply: **VMC> dragon.zon** If the zone compiles correctly it will indicate success by printing a message to the screen and outputting two files both with the same root name as the original zone source file but with different extensions. In this case there would be the following:

`dragon.data`

The file holding the binary version of the zone

`dragon.reset`

The file containing the reset information for the zone.

If the zone doesn't compile correctly and you have errors it will print a list of the errors and the location where they can be found so you can fix them. The debugging process will be explained more as you learn how to create rooms, monsters, and objects.

2.2. The VMC pre-processor

The VMC Pre-Processor (VMC -p) can be thought of as a powerful search and replace tool used by the compiler before it converts the zone to its binary form. This tool gives you the builder the ability to add comments, create short hand expressions for repeated items, include other files in your zone, and even to do some minor calculations when necessary.

> **Note:** If you have coded in C or c++ before the Pre Processor the VMC uses is no different and you can skip this section.

2.2.1. Commenting your zone

The practice of adding comments to your zone is a good thing to get into so the administrators and other builders can help you with your zone and know what you were trying to do if there are problems. Comments aren't as important when writing the zone as they will be when you start writing your own special DIL functions but it is important to know how comments work and that you can use them if you need to. A comment is a block of text the compiler will never see and is there only for you and who ever reads the file. In order to make it so the compiler will not see the block of text you must surround it by a set of symbols that tell the CPP to strip it out before passing the zone on to the compiler. These symbols are the '/*' and the '*/' symbols or the '//' symbols together in front of a single line.

In order to best explain how comments work we will give you a some what strange example. First we will start by showing you a very basic line you will see time and time again in rooms.

```
title "this is a title"
```

This is a title it will show up in everything from rooms, to objects and even NPCs. Now lets see what a commented line would look like.

```
//I am going to make a title now
title /* I put the keyword
first*/ "this is a title/*then the title*/
```

This of course is very ugly but the point is not to be pretty it is to show you both the first way and the second way will look exactly the same to the compiler because all comments are removed before the compiler ever gets it. A better use of a comment in a zone however would be something like this:

```
/*
```

```
The following ten rooms are the vineyards,
there are 97 rooms in the zone.
*/

//Zone first created 1994
```

You will find comments will make coding large zones much easier because you can add text meant just for the builders eyes.

> **Note:** You will have to decide if you want a multi-line comment or a single line comment and use the '//' or the '/**/' respectively. The rule of thumb is if the comment is longer than 1 line it is easier to put the '/**/' around the comment than to comment each individual line.

2.2.2. Macros and what they can do for you

When making a zone you will find there are things you use more than once. In fact you may find things you want others to use or things you want to use in multiple zones. Its true you could block and copy and stick them everywhere. in fact that is what I did when I first started building. I soon found my zone file was extremely large and hard to upkeep. With a few minor changes and a lot of deleting I used short hand or better known in the world of coding as macros to make my zone readable.

Lets say you had some flags you were going to set in fifty rooms and you knew they would all be the same. You could type the following line 50 times.

```
flags {UNIT_FL_NO_WEATHER, UNIT_FL_CAN_BURY}
```

With the macros however you could make this much easier by just doing the following at the beginning of your zone.

```
#define DIRTFLOOR flags {UNIT_FL_NO_WEATHER, UNIT_FL_CAN_BURY}
```

Then where ever you want the flags you just type DIRTFLOOR. You are probably thinking, yeah big deal I can do that with block and copy. True but there is another benefit to this. Lets say later you wanted to also make these 50 rooms no teleport. All you would have to change is the define like this:

```
#define DIRTFLOOR flags {UNIT_FL_NO_WEATHER,UNIT_FL_CAN_BURY,UNIT_FL_NO_TELEPORT}
```

Now when you recompile all 50 rooms are changed and you didn't even have to do a search and replace.

You can also make macros that take arguments. The ability to take arguments is where macros take a leap and a bound out in front of your favorite editor to allow you to do things you can not do easily with

search and replace. Lets say you have an exit descr you want to use in 50 swamp rooms because heck everything looks the same in a swamp when you look one direction to the next.

```
east to swamp1 descr
"You see the swamp stretch out for miles";
```

This could be made into a macro like:

```
#define sexit(direction, place) direction to place descr \
"You see the swamp stretch out for miles.";
```

Then all you need to use it is:

```
SEXIT(east,swamp1)
SEXIT(north,swamp2)
SEXIT(south,swamp3)
```

> **Note:** There is no space between 'SEXIT' and '(' that is important because the CPP sees 'SEXIT(' and 'SEXIT (' as two different things. It is also important to notice all defines must start at the beginning of the line and be either one line long or have a '\' telling the Pre Processor that it should continue with the next line as if it was this line.

You can also combine macros together so you have a set of macros like:

```
#define DIRTFLOOR flags {UNIT_FL_NO_WEATHER,UNIT_FL_CAN_BURY,UNIT_FL_NO_TELEPORT}
#define DIRTSECT  movement SECT_INSIDE \
DIRTFLOOR
```

You may have noticed I capitalize all macros. This is not a must but it is suggested so you can easily tell what is a macro and what is not.

2.2.3. Including other files in your zone

Another function of the VMC Pre Processor, '#include', allows you to include other files in your zone file. The VME comes with some basic include files you can use the macros out of and use as examples on how to make your own include files. These files are the `composed.h`,, `vme.h`, `values.h`, `base.h`, `liquid.h`, and `wmacros.h`. Including `composed.h` will include all the rest of the include files into your zone because it has include statements that use all the others.

> **Note:** You will want to include the files at the beginning of your zone file because all defines you use must be defined before you use them.

2.2.4. Doing minor calculations

You can also do minor calculations in a macro. Lets say you wanted to make it so the higher level an NPC was the heavier he was and the taller he was. This would be simple with a macro.

```
#define MLEVEL(lvl) \
level lvl \
height lvl+72 \
weight lvl*9
```

This macro would increase the height and weight depending on what level you made the NPC pretty simple. There is much more a macro can do for you but the Pre Processor and all its uses go far beyond the scope of this manual. If you are really interested in all the neat things it can do type the following command at the '$' prompt on your Linux box. **man cpp** The C-Pre Processor is what the VMC Pre Processor is based on and most if not all functions of the CPP work in the VMC.

Chapter 3. Zone source file

In this chapter we will define all the sections of a zone file and go in-depth on the zone info section. Once complete with this chapter you should be able to create an empty yet compilable zone.

A zone source file is split up into 6 sections. A zone-declaration section, a mobile (NPC) section, an object section, a room section, a reset section, and the DIL section. The zone section is the only section that has to be in the file, and they may appear in any order.

Each section is preceded by a section header. These are the six possible headers:

- %zone
- %rooms
- %mobiles
- %objects
- %reset
- %DIL

The first four sections may be considered lists of definitions. The reset section can be considered a program in a simple programming language. And the DIL section is a bit special - it includes the zone templates (DIL functions that can be used from any zone, on anything, as opposed to "specialized" DIL functions placed inside a unit's definitions). After all sections you are using are defined you must tell the compiler you are done the special symbol '%end' must be placed at the end of the zone for this reason.

3.1. Definition types

When creating your zone there are six main building blocks. We call these definition types. Each type represents some kind of data you want the compiler to be able to recognize. These data definitions take the basic form:

```
field value
```

Where field is the name of a data field, and value is some value. Values are of one of 6 types:

integer

A whole number or if you are in practice of using Hex you can use the C style hex numbers in either upper or lower case (i.e 0X0f3 0x0f3)

string

Text enclosed in Double Quotes. The string can span more than one line as it would in a description.

```
title "The dark dragon altar"
descr
"There are many things you can see and there are many things that
can't be seen but this is still a description none the less."
```

stringlist

A set of strings, it can be a single string or multiple depending on your needs. These are used in names, extras, creators, and special keywords all to be defined later in their respective places. These are defined in the following manor.

```
<fieldname>    {"string1","string2","string3", ...}
```

intlist

A list of numbers which can be used with an extra. This type works like the stringlist but doesn't need the quotes.

```
extra {"mynumberlist"} {1,2,3,4,5,6,7,...}
"This is a number list attached to an extra"
```

flags

Like the Intlist the flag is defined with a list of numbers. The list of numbers is not taken literally however it is combined to create one number by binary oring the number list together. If that confuses you don't worry, it takes some getting used to. These types are used for Manipulation, flags, and positions.

```
flags {2,8}
manipulate {8}
```

In the previous example the 'flags' value after this zone compiles would be 10 because binary oring the two flags together is a lot like adding. The two numbers probably make no sense so most flags you use will have defines if I used the defines found in `vme.h` the previous example would look like this:

```
flags {UNIT_FL_INVISIBILE,UNIT_FL_BURIED}
manipulate {WEAR_BODY}
```

We will cover this more in-depth later but it was necessary to give a good overview so you understand this field type enough to recognize what it is when you see it.

symbol

A label you reference from other parts in your zones. Every unit (room,object,room) and even the zone itself has a unique label that can be referenced. It is important to make symbol names that are clear so the Administrators of the mud know what each item is when using the online administration commands.

```
dark_sword /*good symbol*/
rm_5892 /*Bad symbol*/
```

When loading items online the zone symbol and the item symbol are combined to create a reference to the item. For example if our zone name was 'dragon' and our item was 'dark_sword' the symbolic name for this item would be 'dark_sword@dragon'. Using symbols will be covered more in the DIL manual and in the administration manuals for loading objects online. For now it is enough to understand symbols must follow the following rules when being defined.

- The first letter of the symbol must be a letter of the alphabet or a '_' character
- Characters following the first can be numbers, alphabet letters, and '_' characters
- The name can be no longer than 15 characters
- No reserved keywords can be used as a name Appendix B

Note: the end tag that ends all unit definitions is also considered a symbol it is just a symbol that must be included with

There are two other field types that can not be defined as a regular field type. These are the function reference and the Structure. The function reference can be either a reference to a DIL function or a special function called from the base code.

Note: Special functions are being replaced with DIL for better performance and should only be used when no DIL functions exist to replace them

The Structure field types are a combination of other field types to make a special field type for the unit being defined. A good example of this is a 'exit' for a room. The exit has everything from flag, string, stringlist, and even description fields. The exit field will be defined much more in-depth in the chapter on rooms but it is important to know some fields are considered Structure fields because they can have many values. The only two Structure fields are the exit and extra fields which will both be defined more later because they can be used differently depending on what you are using them for.

3.2. Zone information section

The zone information section is the only section that must exist in the source file of your area. With out this section the compiler is unable to create the zone because frankly it doesn't know what to call it. It is also the easiest of the sections to learn because there is only a few possible fields. The Zone-section defines the global parameters for the current zone. It is usually wise to place this section in the top of the source file to make it easy to find the zone information when editing the file.

Table 3-1. Zone section field descriptions

Field	Type	Description
creators	Stringlist	This field is where you place the creators of the zone. With this field filled out the Administrators and builders can easily find out who the zone was written by and be able to contact them if there are problems.
lifespan	Number	This defines the interval between resets for this zone, in minutes. Default is 60 if this field is left out of the information section.
notes	String	This is a plain text description of the zone for administrators and builders. It is often a good idea to include your e-mail address in the notes so you can be reached easily by the administrators.
reset	Number	This combined with 'lifespan' defines if the zone will be reset. This field gives the condition that must be met to reset the zones you should use the defines in the `vme.h`, RESET_NOT, RESET_IFEMPTY, and RESET_ANYHOW. Default is RESET_ANYHOW, which means, the zone will be reset even if players are present within it.

Field	Type	Description
title	String	This is the title of the zone, for example Dragons Nest, Dark station, and creators hide out. It is used mainly for the areas command so players can get a list of all the areas in the game. It can however be accessed by the 'zoneptr' variable type in DIL. If you have a zone that spans across multiple source files you only need to define the title once. If you put the title in all source files it will show up multiple times in the area listing. You would also leave this blank if the zone should not be on the areas list like an administration zone.
weather	Integer	This field sets the humidity level of the zone. If for example you want a hot desert like zone you would want to set this to its highest value. The range of this field is 1000 to -1000. This is an optional field and will not be covered else where because it is simple to use.
%zone	Symbol	This entry defines the name of the zone. Default is the preceding component of the current filename, minus the trailing ".zon". Note, the symbol should be added after the %zone tag, which should always be put, even if you do not add a symbol after it.

The only field that must exist when you go to create the zone information section is the '%zone'. Leaving the '%zone' field out will cause an error when you try to compile it. We suggest you not only put the '%zone' field but you also add a symbol or as I call it a zone name. The following are three legal examples of a Zone information header. You be the judge of which is more informative.

```
/*very bad*/
%zone

/*bad but better than nothing*/
%zone bug_planet
```

```
/*The way it should be done!*/
%zone dragonst
lifespan 20
reset RESET_ANYHOW
creators {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."
```

If you felt like it you could add a '%end' to the proceeding examples and compile them. They would create an empty zone so wouldn't be very exciting but at least its possible. We will not go into any compiling until we have at least one unit type to compile because it is pretty useless to do. the next chapters will define the basic unit building blocks you will use for rooms, objects, and NPCs and start you off on compiling.

Chapter 4. Unit building blocks

When creating your zone you will find some basic structures are used in all three unit types rooms, objects, and NPCs. In this chapter we will define the main building blocks of all units along with some helpful hints

No matter which unit type you are dealing with there is a simple basic structure that lets the compiler know not only what unit type it is dealing with but where one unit begins and where it ends. The way the compiler tells what unit type it is dealing with is by the section header '%rooms', '%objects', and '%mobiles'. All rooms must be defined under the '%rooms' header and likewise objects and NPCS under their respective headers. Each unit starts with a symbolic name called the unit symbol and ends with the keyword end. The following would be a legal definition of any unit type:

```
symbol_name
end
```

If you define a unit like this when it loads it will be blank, while this is not extremely useful it is good to know you can leave out any field you don't feel you need.

Unit building blocks

symbol field

In the last chapter we defined the different field types and the rules you must follow when defining a symbol. It is important enough to relist these rules here so you do not run into problems when creating your units.

- The first letter of the symbol must be a letter of the alphabet or a '_' character
- Characters following the first can be numbers, alphabet letters, and '_' characters
- The symbol can be no longer than 15 characters
- No reserved keywords can be used as a symbolAppendix B

> **Note:** It is also important to know currently it is hard to deal with units having capital letters in them, this may be fixed in the future but it is good practice to use all lower case characters in the symbols.

Another thing you should think about when defining your symbol names is to be clear about what the unit is. When you list units on the VME server with **wstat** the symbolic names are shown if you were to see the list: `i6853 b2419 l1854` You would have no idea what those three items were unless you personally built them recently, therefore it is a much better coding practice to name things what they are like: `long_sword healer_hut dog`

title field

The title field is probably the easiest field on all units it is what is shown on the first line of a room when you enter and it is the name shown when you get an object or attack a NPC. There is only two important things to look at when defining titles one is punctuation and the other is capitalization. Room titles need to be capitalized and so do proper names but the first letter of an object title or a NPC title do not normally need to be capitalized. this is best explained by some examples.

```
title "The dragons in."/*good*/
title "a big bull dog."/*bad has a period at the end*/
title "Bill the destroyer"/*good*/
title "A long dagger"/*bad capital 'a'*/
```

Now to show why some of those are good we will demonstrate by some sample output in the game. `prompt: l The Dragons Inn You are standing in a moldy inn. prompt: get A long dagger You get A long dagger. prompt kick dog You kick a bull dog. in the head.` Notice the 'A' and the extra period do not really look right where they end up appearing in the game. These may be minor nit picky details but if you do it right the first time you won't have to deal with the english major that just happens to be playing on your mud.

names field

The 'names' field defines the names everything in the game can use to interact with your unit. For rooms the names are used as teleport or goto points for characters and NPCs or they are sometimes used for special DIL functions on objects to trigger in certain rooms. On NPCs and objects names can be used for anything from poking a player to giving a player an object. The names field is very flexible it has even been used to store what a container is holding in order to have quick access to the names in the container. it

When making rooms it is not necessary to put room names. In fact it is a good way of making sure players can't teleport to certain rooms because if the room doesn't have a name the person can't teleport to it. Objects and NPCs must have names because if you leave names off of them you can not pick them up or kill them depending on which you are dealing with. It is also good practice to use all combinations of the object or NPCs title so a player will know exactly what to type to use the item for example:

```
bad_obj
title a small dog"
names{"small dog","small","dog"}
end
```

It is up to you as a builder if you want to use names like 'small' in your names list since you would not 'get small' in real life it may not have to be added to the names list. It is important however to define your extras from big to small because of how the VME command interpreter handles names of things. For example if you had the following names:

```
small_item
title "a small item"
```

```
names {"small","item","small item"}
```

When you try to **give small item Bill**. The interpreter would try to give small to item and that would not be what you wanted to do. Don't worry the compiler will catch this and give you an error something like: `Fatal name ordering on line ###.`

descr field

The description building block is used in many places. You will find description fields on extras, exits, rooms, NPCs, objects, and as you have already seen the help and the notes field of the zone information section are also description fields.

Depending on what you are working on description fields can mean totally different things to the person looking in the room. A description field on a room can be the inside or outside of the rooms description. A description on extras can be an NPCs descr or an extra description on the room like if you looked at a 'rug' or a 'chair'. On an exit the descr field describes what you see when you look in that direction from the room you are in. The important thing right now is no matter where you use them they all work the same.

description fields like the title field have a tag or unlike the title field can have a set of tags before them like in extras or exits but like titles they are just a string surrounded by quotes. You can make multiple line descriptions if the description is on a NPC you may not want to since it is the description shown when you walk into the room. The following would be some examples of room descriptions.

```
descr
"this is how you would define a room descr and as you can see it can
be much longer than a line if you like."

extra {"basic extra"}
"This is a description field on an extra."

extra{"more advanced","extra"}{1,2,3,4,5,6}
"This is still the description field.  Like the room description or
any description field for that matter this can be longer than a
line."

east to bathroom descr
"You see one big toilet!";
```

extra fields

The extra field is the work horse for the builder. It is the building block that most brings your zone to life. You can use it to describe body parts or items in a room. When you use DIL you will use extras to store information like quest information or special information you need later. Extras also

store the main description on NPCs since the descr field on NPCs is really the long title shown in the room which will be explained later in Chapter 6.

The extra Is the first structure definition you have run into so it may take some playing with to figure it out. The extra has actually 4 fields three of them must be there in some form. The first is the identifier that tells the compiler that this is an extra, the second is a stringlist, the third is an Optional Intlist, and the final one is the extra description field. If the extra had all the fields it would look like this:

```
extra {"small chair","chair"} {1,2,3,4,5} "Its a chair."
```

If the previous wasn't an example we would have left out the Intlist because it is not necessary to put the Intlist on a chair unless you want it there for some special DIL you are creating. the Intlist is the only field that can be totally left out but you can leave the string list blank when you want to make the extra the default description when the object, room, or NPC is being looked at, like this:

```
extra {}
"This would be what you see when you look at any of the names in the
names list."
```

We will go more in-depth into why you want to do this in the following chapters.

You are now ready to put these building blocks to work. You have only to learn how to put these fields to work for each of the three unit types and you will be off and running.

Chapter 5. The room section

The previous chapter gave you the basic building blocks that will be used all through creating rooms, NPCs, and objects. If you jumped straight to this chapter without reading about the general building blocks you might want to return to Chapter 4 first. This chapter will deal with all the fields of rooms what they are and how to use them but with out the previous chapter you may get lost.

In order to get started building rooms you should first be aware of the room fields you can use. The Table 5-1 shows a full listing of all the room fields and their types as defined in Chapter 4.

Table 5-1. Room fields and types listing

Field	Type		Field	Type
symbolic name	Symbol		manipulate	Integ
names	Stringlist		alignment	Integ
title	String		flags	Integ
descr	String		weight	Integ
outside_descr	String		capacity	Integ
extra	Structure		light	Integ
minv	Integer		exit	Struc
key	string		movement	Integ
spell	Integer		end	Symb
dilbegin or dilcopy	Function pointer			

As you can see there is not a whole lot of fields you have to learn in order to make a room. In fact as you will see shortly some of these fields are not even used on rooms. In Section 5.1 we will expand your knowledge from just knowing what the field types are to how to set them as well.

5.1. Description of room fields

symbolic name

The rules of the symbols has been explained in Chapter 4, if you didn't read them yet you may want to review. The important thing to realize with the room symbol is it is always good practice to give the room a symbol resembling the title so administrators and builders can use the **goto** and the **wstat** to easily goto the room in question.

title

The room title field should start with a capital and depending on your preference the compiler will not complain if you add punctuation at the end. The following are good examples of a room title.

```
title "The Post Office"
```

```
title "The deep dark jungle floor:"
title "The Dragon Station control room"
```

It is really up to you weather you want to use punctuation or not, it is more administrator personal opinion than anything.

names

The names field on the rooms are not that important and only should be used if the builder wishes the room to be accessed by the players by a teleport command. If the room has no names no one will be able to teleport to it. On some muds there will be no teleport spell so the only use for this field will be for DIL functions the administrator creates. If a builder wants the room to be accessible by teleport then the names should match the title since that is what the player will try to teleport to. A few good examples of names on a room would look as follows.

```
title "The Post Office"
Names {"post office","office"}

title "the thrown room"
names {"thrown room","thrown"}
```

descr

The description field is what the player sees when walking into the room or when looking with no arguments.

outside_descr

This field is what is shown to a char if the room is loaded inside another room. For example if you had a room linked inside another room and called a barrel this would be the description that lets the character know it is a barrel. An example would be like:

```
outside_descr "a big old barrel is laying here on its side."
```

This allows a builder to make a room that looks like an object inside another room.

movement

The movement field defines the endurance cost to a character when moving through this room. In the future these fields will be adjustable by the use of a define file. Currently all movement fields are constants and are defined in the `vme.h` The following is the movement sector types and their values.

Table 5-2. Sector movement values

Symbol	Name	Endurance Cost
SECT_INSIDE	inside	1

Symbol	Name	Endurance Cost
SECT_CITY	city	1
SECT_FIELD	field	2
SECT_FOREST	forest	3
SECT_HILLS	hills	4
SECT_MOUNTAIN	mountain	6
SECT_DESERT	desert	8
SECT_SWAMP	swamp	8
SECT_WATER_SWIM	water-swim	4
SECT_WATER_SAIL	water-sail	50
SECT_UNDER_WATER	under-water	8
SECT_SNOW	snow	8
SECT_SLUSH	slush	6
SECT_ICE	ice	10

The movement is simply defined by placing the 'movement' keyword first followed by the type of sector you desire. For example a few movement fields would look as follows:

```
movement SECT_FOREST
movement SECT_HILLS
```

Note: Only one movement is needed for a room if you put more than one the last one added will be the one used.

flags

This field on a room is used to set special attributes in order to make the room private or no-teleportable and many others. The following is the list of possible already defined flags. Extras can also be used to create special room flags.

Table 5-3. Room unit flag affects

Flag	Description
UNIT_FL_PRIVATE	When this flag is set on a room it marks it as a private room. Commands that honor the private flag will not let more than 2 players into this room. Commands like **goto** and direction commands are a few commands that do honor this flag.
UNIT_FL_INVISIBLE	Makes unit invisible
UNIT_FL_NO_BURY	Makes a hard floor so items can't be buried.
UNIT_FL_BURIED	Makes unit buried when loaded

Flag	Description
UNIT_FL_NO_TELEPORT	makes unit so no one can teleport to it
UNIT_FL_NO_MOB	Makes it so no mobile will enter the unit
UNIT_FL_NO_WEATHER	keeps weather and natural light out of unit
UNIT_FL_INDOORS	Makes unit inside and doesn't affect weather
UNIT_FL_TRANS	Makes unit transparent
UNIT_FL_NO_SAVE	Makes it so you can't save with unit
UNIT_FL_SACRED	Makes unit a double gain unit
UNIT_FL_MAGIC	Marks a unit to be magic

extra

Extras are the work horse of the VME. Extras are used in everything from DIL to just normal extra descriptions on rooms. The first job for an extra was to hold extra description information on a room. For example if you had a computer room and you described it might look something like this:

```
descr
"This small room houses the computer power of the VME development team.
All four walls are lined with various pieces of computer equipment old pizza
boxes and plenty of empty soda cans."
```

The problem is as a player if you saw this description you might want to know what kind of pizza we eat or maybe you would want to see what kind of soda we drink. Or heaven forbid you might want to know what kinds of computer equipment is scattered about the room. In the VME servers we do this by adding extra descriptions to the room. In this case the builder of the zone may do something like this:

```
descr
"This small room houses the computer power of the VME development team.
All four walls are lined with various pieces of computer equipment old pizza
boxes and plenty of empty soda cans."

extra {"soda cans", "cans", "soda", "can"}
"These cans are all Canadian blue.  Maybe the Valhalla team hates American
beer.  Strange all of them look to have strange indentations."

extra {"strange indentations", "strange indentation","indentation"}
"They are human bite marks.  Is this what happens when code doesn't work right?"

extra {"pizza boxes","pizza","boxes","box"}
"Dominos could make a fortune from all these boxes and probably already have
from the VME team.  you notice all the boxes are empty at least they finish what
they start."

extra {"computer pieces","computer parts", "equipment","hardware", "pieces", "parts"}
```

```
"I bet you thought you would see what we have running.  Yeah right you might come
over and rob us if we told you that.  All you see is an old XT."

extra {"xt"}
"Its a hunk of junk really!"
```

There is a lot to notice in the previous examples. First we will start with extras when defined on rooms, NPC, and objects must be in length order for the names. There are a few reasons for this but lets just say the most important reason is we wanted it this way. If you don't put them in order the VMC will give you a fatal error and will not compile your zone.

The next thing you should notice is we have used an extra to describe something in another extra. We actually did this twice once for the beer cans and once for the computer parts. That way you can actually give quest information but make the person really have to explore your rooms descriptions to find it.

The previous example is what we consider normal extras in a room. There are also extras that hold information for DIL functions. These special extras can have extra fields and they can be hidden to the players eyes. Here are some examples of special extras.

```
extra {"$rockcount"}
"5"

extra {"$playerkill"}
"0"

extra {"$coke","$milk","$water"}{1,5,10}
"Drinks and amounts"
```

These extras all have the '$' sign appended to the front of the names in order to tell the look command the player shouldn't be able to look at the extra. If you have not already seen DIL coding you may not understand why you would want extras players can't see. The DIL language can manipulate these extras by reading and writing them in order to change the way a command or another function works. For example the last DIL could be used for a shopkeeper to tell how many of each type of drink he has. Notice the drink extra also has something you haven't seen yet, an added integer list after the namelist. all extras can have these but only extras being used with DIL functions really need them.

Some of these special functions are supported already in the code and the ones that affect the rooms are as follows. In the following $1n is the activator and $2n is the unit in question.

There is only one special extra already supported for rooms and that would be the '$get'. As we have previously mentioned the extras that start with a dollar sign are not seen by the players. This one however is shown to the player when the person types get on the other names in the extras list. This easier to describe in an example than in words so the following would be a good example.:

```
extra {"$get", "statues", "statue"}
"You attempt to pick up a statue but quickly discover your feeble
attempts will never work."

{"$get", "red roses", "roses"}
"You bend down to pick a rose, but then decide to leave the beautiful
flower to itself."
```

With this one special extra we have made it so you don't need to make millions of items so the person can act upon them. You can just make the acts as if the items were in the room.

Exits

Every room has ten possible exits; North, East, South, West, Northeast, Southeast, Southwest, Northwest, Up and Down. To enable mobile use of these commands, you must specify these exits as outlined below:

```
exit <direction> [to <destination>] [open {<infoflags>}]
[key <keyname>] [keyword {<keywords>}] descr <description> ;
```

exit <directions>

Is the direction the exit leads, ie. one of north, south .. up, down.

to <destinations>

The symbolic reference to the room, you want this exit to lead to. If you reference a room within another zone, post pend the name with @<zone name>

```
to myotherroom

to hisotherroom@hiszone
```

open <info flags>

These flags describe the state of the door. The following is the list of possible door flags.

EX_OPEN_CLOSE

Set this if you can open and close this exit, be it a door, gate or otherwise.

EX_CLOSED

Set this if you want the exit to be closed at boot time.

EX_LOCKED

Set this if you want the exit to be clocked at boot time.

> **Note:** An interesting aspect is, if you do not specify a key, you can only unlock this door with the 'pick' skill, 'unlock' spell or from DIL with UnSet();

EX_PICK_PROOF

Using this flag renders the 'pick' skill and 'unlock' spell un useable on the lock of this exit.

EX_HIDDEN

If this bit is set, the exit is hidden until the mobile has successfully searched for it, using the 'search'-command.

key <keyname>

The symbolic name of a key object used for unlocking this exit.

```
key mykey@myzone
```

keyword { <stringlist> }

This stringlist holds all the names of the exit, you specify to manipulate the exit. If the exit is hidden exit, these are the keywords the mobile can search for.

```
keyword {"wooden door","door"}

keyword {"hidden door","door","hatch","floor"}
```

descr <description>

This string is the description of what you see if you look in the direction of the exit.

;

Every exit statement needs to be terminated with a semi-colon.

General notes: Even though you do not need an exit in all directions, you can use it to place descriptions of the direction.

```
exit north descr "An unsurmountable mountain blocks your way.";
```

minv

This field is rarely used on rooms. It could however be used to make a room invisible inside another room. Or it could be used to store numbered values on a room. The reason this field is on a room is it is part of the base object which all objects are derived from. If the room is going to be inside another room and you don't want it visible the following would make it invisible to all players below the level of 20.

```
minv 20
```

key

This field is not used normally on a room. It is a string that can be used for anything you desire. The reason it exists on rooms is it is a part of the base object all unitptrs (unit pointers like, rooms, objects, and NPCs) are derived from.

manipulate

This field is not used normally on a room. It is an integer that can be used for anything you desire. The reason it exists on rooms is it is a part of the base object all unitptrs are derived from.

alignment

This field is not used normally on a room. It is an integer that can be used for anything you desire. The reason it exists on rooms is it is a part of the base object all unitptrs are derived from.

weight

This field is not used normally on a room. It is an integer that can be used for anything you desire. The reason it exists on rooms is it is a part of the base object all unitptrs are derived from.

capacity

This field is not used normally on a room. It is an integer that can be used for anything you desire. The reason it exists on rooms is it is a part of the base object all unitptrs are derived from.

light

This field sets the light on a room. Normally this is not done directly, instead it is set using macros defined in `wmacros.h`.

Table 5-4. Light defines

Define	Light Value	Affect
ALWAYS_LIGHT	1	Room is always light no matter time of day
IN_ALWAYS_DARK	-1	When an inside room is always dark - both inside and outside
OUT_DARK_NON_NOON	-1	Always a dark room, except when it is high noon
OUT_ALWAYS_DARK	-2	Always a Dark room, no matter the time of day

To set natural light that changes depending on the type of day nothing is needed to be put in the light field the compiler will default to '0'. If you for some reason want to set the light to default lighting you can do so but you don't need to. You will also notice there are two macros that set the light to the exact same value. This is for compatibility with older code base and if you wish to combine these two macros or only use one it would not change the way the mud works.

This is probably one of the simplest fields you will have to deal with in the rooms. In order to set it all that is needed is to place the macro or the light and value on a line in the room and your all done.

```
//To Set always light with macro
ALWAYS_LIGHT

//To set Always light with out macro
light 1
```

You can decide which is easiest for you.

link

spell

dilbegin or dilcopy

The DIL functions are what give VME servers the edge over all other muds. We will only give some examples here and leave it up to the DIL manual to teach you how to create your own functions that will make your rooms, NPC, and objects more than special.

There are only currently three room functions that come standard with a VME in the `function.zon`. There are much more in the zones released with the VME but you will have to hunt for those. The three that come standard are Safe room, Death room, and forced move. The safe room makes it impossible for players to kill each other, the death room is a function that lets you make things like rock slides and quick sand, and the forced move lets you make an easy river.

Since these are just DIL's written by builders for the Valhalla mud all you have to do is use the dilcopy keyword in the room with the function name you want to use and the arguments the function requires. The following is what you would find in the `function.zon` for death room.

```
/* Death room DIL
 *tick is in 4th's of seconds
 * Damage is damage done per tick
  *act_s is string shown to damaged player.
 */
dilbegin death_room(tick: integer, damage: integer, act_s: string);

var ext: extraptr;
    u  : unitptr;
    i  : integer;

code
{

if (tick < 12) tick := 12;

heartbeat := tick;

if (damage < 0)
    damage := -damage;

if ("$death room for mobs" in self.extra)
    i := UNIT_ST_PC|UNIT_ST_NPC;
```

```
else
    i := UNIT_ST_PC;

while (TRUE)
{
wait (SFB_TICK, TRUE);

foreach (i, u)
    {
    if (u.level >= IMMORTAL_LEVEL)
        continue;

    if (("$no death room" in u.extra) and (u.type == UNIT_ST_NPC))
        continue; // Don't allow pcs to get this flag

    if (act_s != "")
        act ("&[hit_me]"+act_s, A_ALWAYS, u, null, null, TO_CHAR);
    else
        act ("&[hit_me]You bleed from your wounds.",
            A_ALWAYS, u, null, null, TO_CHAR);

    u.hp := u.hp - damage;
    position_update (u);
    }

}

}

dilend
```

If this DIL function scares you don't worry you don't have to understand or adjust it you just have to use it. In this function it requires a time, damage, and act. So you could use this in a room definition like this:

```
dilcopy death@function (60,25,"Flames shoot from the floor burning
your rear.");
```

This says to copy the DIL from zone function with the arguments 60 seconds, damage 25% and act as shown. Pretty simple eh?

All released DIL room functions are described in Section 5.4. Then we put some to work so you can see how to use them in Section 5.5

5.2. Building your first room

Now you are ready! With all you have learned about room fields you are now ready to build your first room. I personally like dragons and I like space so I have chosen to make a dragon station. We will first do a simple room and build on to it in the next sections. In this section we will walk you through creating a basic room and why we choose what we do where we do.

When making rooms you create the zone source file first as shown in Chapter 3. If you only have rooms you do not need the %reset, %objects, and %mobiles. For the examples in this chapter we will use the zone we created in Chapter 3 and add the %room tag where we will put all the rooms. At the end of the chapter we will have the entire zone so you can see it all together.

The first part of all rooms is the symbolic name it is good to always pick a name that will match the title so you can use the administrator command **goto** to easily get to the room. The reason is when you use the command **wstat** it will only show you a list of the rooms by symbolic name for example if you type **wstat zone dragon room** You will get the following:

```
List of rooms in zone Dragon:
chamber portal office
```

If you didn't make it clear what the rooms were by the symbolic name it might look like this:

```
List of rooms in zone Dragon:
st_rm1 st_rm2 st_rm3
```

While this might be great when you first start imagine trying to remember what all one hundred rooms are.

The first room we will create will be a simple chamber with nothing special. We can build on to it later if we need to.

```
chamber
end
```

Pretty easy so far. Now lets add some life to it. The room will need a title and a description. The title should be a one line description of a room sort of like if you were walking someone around your house and telling them what each room was like this is 'My houses big bathroom' or maybe this is 'The computer room'. The description should be something you would tell an interior decorator you were talking to on the phone and asking for advice. He would want to know everything about the room you can see so he could give you good advice.

```
chamber
title "The middle chamber of the station"
descr
"This chamber seems to have the entire station rotating around it.
```

```
Small human size ornate chairs with dragon designs scrawled on the
arms and back are arranged in a triangle like setting with one large
chair at the front.  This must be where all station meetings are held.
large pictures cover the walls depicting dragons in all kinds of
situations.  Small passages lead of to the west and the east.
."
end
```

It is a matter of taste if you want the descriptions of the exits in your description or not but I like them so I put them. Now if you were reading this description to someone the person might ask you what is on the pictures or maybe even what exactly do the chairs look like so we better take care of that by adding some extras to our little room.

```
extra {"chairs","chair"}
"The chairs are made of some metal you don't recognize and every inch is covered
with some kind of dragon."

extra  {"dragon picture","picture"}
"Thousands of dragons dot the skies of this rather life like picture.  In the
center you see something move.  It looks to be a little green dragon."

extra{"green dragon","dragon","green"}
"An intelligent looking dragon is sitting perched on a large chair watching you."
```

Normally we could put a movement type for the amount of endurance lost when a person is moving through the area but we will leave it blank and go with the default which is SECT_CITY since with the fake gravity that is the closest we could come with the sector types availible. The last thing we need to finish our room is the two exits leading to the other rooms.

```
west to portal descr "You see a small room.";

east to office descr "You see what looks to be an office.";
```

Thats it that is all there is to making a room. In the next couple of sections we are going to add some more rooms with more advanced features and give some debugging hints for compiling your rooms but if you understand everything so far your going to have no problem. Lets take a look at our entire finished first room

```
chamber
title "The middle chamber of the station"
descr
"This chamber seems to have the entire station rotating around it.
Small human size ornate chairs with dragon designs scrawled on the
arms and back are arranged in a triangle like setting with one large
chair at the front.  This must be where all station meetings are held.
large pictures cover the walls depicting dragons in all kinds of
situations.  Small passages lead of to the west and the east.
."

extra {"chairs","chair"}
```

```
"The chairs are made of some metal you don't recognize and every inch is covered
with some kind of dragon."

extra  {"dragon picture","picture"}
"Thousands of dragons dot the skies of this rather life like picture.  In the
center you see something move.  It looks to be a little green dragon."

extra{"green dragon","dragon","green"}
"An intelligent looking dragon is sitting perched on a large chair watching you."
west to portal descr "You see a small room.";

east to office descr "You see what looks to be an office.";
end
```

5.3. Compiling and debugging your first room

It is time we put the zone header information together with your first zone and compile it into a format the VME server can use. This is done by using the VMC compiler. Depending on if you are doing this on your own Linux server or if you are building for a VME already set up you will have to use the compiler access method they have defined. No matter if you are compiling by email, ftp, or at the command line with VMC the error messages will all be the same. Since I have no idea how your particular set up is designed I will explain the errors that the compiler will return and you will have to ask your system administrator how to access the compiler. The rest of this section is written as if you have your own VME running on your own Linux box using the VMC at the command line.

When you are working on your first zone it is always a good idea to start with one or two rooms and compile them instead of writing all the rooms and then trying to compile. The reason is the more rooms you have the more confused you can make the compiler if you have a lot of errors and you may not be able to figure out where your first mistake was easily. In our case we only have our first room and the header information for the zone so lets put it together now and try and compile it.

```
#include composed.h>
%zone dragonst
lifespan 20
reset RESET_ANYHOw
creators {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."

%rooms
```

```
chamber
title "The middle chamber of the station
descr
"This chamber seems to have the entire station rotating around it.
Small human size ornate chairs with dragon designs scrawled on the
arms and back are arranged in a triangle like setting with one large
chair at the front.  This must be where all station meetings are held.
large pictures cover the walls depicting dragons in all kinds of
situations.  Small passages lead of to the west and the east.
."

extra {"chair","chairs"}
"The chairs are made of some metal you don't recognize and every inch is
covered with some kind of dragon."

extra  {"dragon picture","picture"}
"Thousands of dragons dot the skies of this rather life like picture.  In the
center you see something move.  It looks to be a little green dragon."

extra{"green dragon","dragon","green"}
"An intelligent looking dragon is sitting perched on a large chair watching you."
west to portal descr "You see a small room.";

east to office descr "You see what looks to be an office.";
end

%end
```

We added the %room tag to our zone header stuck our room in and now its ready to be compiled and put into the VME server for you to be able to look at it in the game. If you downloaded our example zones for this document you can compile this zone along with us and fix the errors as we do for practice. The filename is `debug_rm.zon`. Just so you know the errors in this zone are intentional so please don't write me an email telling me there are errors in it.

The command to compile the zone is **VMC debug_rm.zon**. Here is what we get when we first try and compile the zone.

```
VMC v2.0 Copyright (C) 2001 by Valhalla [Apr 28 2001]
Compiling 'debug_rm.zon'
<debug_rm.zon> @ 2: Bad include argument
<debug_rm.zon> @ 48: Token too long
Fatal error compiling in preprocessor stage in file 'debug_rm.zon'.
```

Don't worry if this looks scary, it really is much easier to read than it looks like. The first thing you need to realize about compiling is always fix one error and compile again because it might fix two or three errors after with one fix. The reason is once a compiler hits something it doesn't understand it gets confused with the rest of the file. It is sort of like if you thought the word 'water' meant 'fire' and you

tried to read a book it would get confusing really fast. So you have to correct the definition of 'water' to understand the rest of the book.

Lets take the first error with this in mind. The first error shows up on line three of the error file it says:

```
<debug_rm.zon> @ 2: Bad include argument
```

This line is not really cryptic reading it in a more english form would sound like: In file 'debug_rm.zon' you have an error at line 2, the argument to the include statement is not correct. Not all errors will be this clear but the compiler does its best to get you close to the error. Now if you look at line two in `debug_rm.zon`, you will find, we forgot to put in the '<' symbol. If you fix the line to look like:

```
#include <composed.h>
```

Then recompile you will have fixed your first error and get a whole new set to play with. The following is the errors we got after fixing line two:

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_rm.zon'
<debug_rm.zon> @ 47: EOF in string
debug_rm.zon: 4: parse error
   Token: 'RESET_ANYHOw'
debug_rm.zon: 21: parse error
   Token: 'This'
debug_rm.zon: 26: parse error
   Token: 'and'
debug_rm.zon: 26: parse error
   Token: '.'
Grave errors in file 'debug_rm.zon'.
```

Now this looks to be a much more interesting error file than the previous one. Remember we mentioned you should always fix the first error first so the compiler doesn't get confused. In this error file the first line is not the first error we need to fix. We have to do some logical reasoning here. The first error the compiler came across was the one on line 4 that shows up around line 4 of the error file. The lines before it are letting you know somewhere else in the file there is a missing quote. If we clean up the first error however we might be able to find this missing quote much easier. So lets do that lets start by looking at line 4 which is saying the compiler doesn't understand what the token 'RESET_ANYHOw' is. This makes sense the token should be 'RESET_ANYHOW' and the compiler is case sensitive. So all we need to do to fix this one is capitalize the 'w' and the error should be cleared up lets try that and recompile and see what the errors look like. With that line fixed the following is the errors we get.

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_rm.zon'
<debug_rm.zon> @ 47: EOF in string
debug_rm.zon: 21: parse error
   Token: 'This'
debug_rm.zon: 26: parse error
   Token: 'and'
```

```
debug_rm.zon: 26: pars
e error
   Token: '.'
Grave errors in file 'debug_rm.zon'.
```

Again we must figure out which error message we should deal with first. As before we need to deal with the lowest number error. The error we need to fix first then is the one on line '21'. If you go to line '21' you will notice the line looks fine. When you run into an error like this where the error is not exactly on the line scroll up to the field before the one in question and you should find the problem. In this case we forgot a '"' on the 'title' line and confused the compiler because it thought the ending quote was the one after the 'descr' field. Therefore the compiler didn't understand the 'This' as a field. This is one of the harder errors to find but once you get used to it will come naturally and the compiler does try to get you close. Now if we add the '"' we are missing and recompile the following is the output we get.

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_rm.zon'
VMC Done.
```

Notice there are no errors and it says 'VMC done', this means you have now successfully compiled the zone. I want you to look at the last error file and the fact that we only changed a quote to go from it to no errors. This is why you always deal with one error at a time. Sometimes fixing one error can fix a lot of the weird errors that make no sense. In fact I have seen one quote cause as much as 50 errors so if you jump around trying to fix errors that look like they make sense you may end up making more work for yourself.

Now that you have a compiled zone you should check and make sure all the files are there. When you compile a zone you will end up with three extra files. the files will have the same filename as your zone with a new extension in this case you should have the following.

```
debug_rm.data
debug_rm.err
debug_rm.reset
debug_rm.zon
```

If you have all of these you are all set to go. If not then there is something seriously wrong and you may want to write the VME staff for help. To get your new zone in the mud all that is needed is to make sure your zone is in the zonelist in the VME etc directory and copy these files into your zone directory. Then reboot the mud. You should be able to log on your builder character and goto your zone by typing, **goto chamber@dragonst**.

There you go you have now compiled your first zone. Its not much to look at but with what you already know you could make a full zone of very basic rooms. The next few sections will teach you some of the more interesting things you can do when making your rooms.

5.4. DIL functions for rooms

The DIL language is the language a builder can use to make his own special functions on rooms, NPCs, objects, PCs, and much more. This manual is for basic zone writing and therefore will not go into how to write your own DIL functions. The VME however is released with many functions for you as an Administrator and your builders to use to make special rooms, NPCs, and objects. The following is a list of all room functions released with the VME 2.0 server.

Death room

This function is a simple function that allows you to create a room to do damage to a player. The death_room can kill them slowly like a fire cave would or it can kill them quickly as if you stuck them in a microwave it is all up to how you set the arguments. It also lets you see the acts the players see so this function can be used on any number of death style rooms. There is no need to understand how the function works just how to use it so with that in mind the following is the functions header.

```
//In function.zon
dilbegin death_room(tick: integer, damage: integer, act_s: string);
```

As with any function all you have to do is 'dilcopy' the function onto your room with the correct zone name and arguments and it will do the rest. In this DIL you have three arguments to pass The first is the 'tick' or time which in this DIL is broken down into 'ticks' which are 4 ticks per second. Thus if you wanted to get something to do damage every minute you would put '60*4' in that spot. The next is the amount of damage you want done per your time. If for example you want '60' hit +points damage done each round you just put '60' as that argument. Finally is the act shown to the character as a string in quotes. So a finished death room on your room would look like this.

```
dilcopy death_room@function(4*60,60,
"Flames shoot up from the floor burning your butt.");
```

Climb

This special DIL is used for the climb skill and should be set on stationary objects (stationary mast, robe, tree, wall, etc). The 'difficulty' is the skill-amount required to climb. A skill of 100 would be a 50% chance for the expert thief / climber. The 'damage' is how much damage is given if you fail to climb the object. When you fail, you "fall" to the 'destination', so you can make gravity work correctly. The destination can be the same room in which you started but it doesn't have to be. The 'direction' is the direction in which you want the person to have to climb enclosed in quotes.

With all this in mind the following is the DIL definition and an example use of it.

```
//DIL definition
dilbegin climb(destination:string, difficulty:integer,
                damage:integer,direction:integer);

//Example use of Climb
dilcopy climb@function("deck@ship", 17, 20, "up");
```

We should note here, if you wanted the person to have to climb back down you will need to put a climb DIL on the room you are climbing too that has the "down" directory as an argument. This DIL also allows you to link two rooms not linked by normal directions but we suggest you should think before doing this because it may not make sense to not have a link between the two places.

Force move

This function allows you to move a player or NPC from a room to another room with out having the player or NPC type or do anything.

The following is the definition of the force move DIL

```
dilbegin force_move (tick: integer, strings: string, random: integer);
```

The 'tick' parameter is how fast you want the force move to be triggered. The 'tick' is in 1/4 second increments so to get a one second wait you would place a four. The second parameter is two strings the first being the symbolic name of the room you are forcing the character to. The second string is the act you want shown to the player or NPC when it is moved. The final parameter is either a one or a zero. The one stands for true and it would make the timer trigger randomly fifty percent of the time, while a zero would make it not random.

The following is what the force move would look like if you wanted it to trigger every 30 seconds and give acts of a river.

```
dilcopy force_move@function(4*30,{"river2@riverzon","You float down the river."},0);
```

Safe room

This function creates a safe room where combat is not allowed. The following is the definition and an example of how to use it.

```
//Safe room DIL definition
dilbegin safe_room ();
```

```
//Example use of Safe room
dilcopy safe_room@function ();
```

5.5. A more complex set of rooms

In the last section you learned to make basic rooms. In this section we will build on what you already know to allow you to make much more fancy rooms. IN this section we will give a much better view of the exits and what can be done with them including doors, hidden doors and rooms inside other rooms. We will also show some examples of the room DIL functions being used that were described in the previous section. Finally we will pull it all together in a completed zone for you to compile and play with.

5.5.1. Exits with doors

When we first defined exits we included the 'keyword' and 'open' fields on a door. In this section we will give an example of two rooms linked together with a door. There is no new information you have not already encountered so we will start with an example.

```
hallway
title "Module tunnel"
descr "The hallway is about 50 meters long and around 100 meters from
side to side and top to bottom...."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to chamber descr
"The hallway opens up into a chamber.";

east to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

end

office
title "The station office"
descr
"Large paintings fill the walls of this part of the station...."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}
```

```
west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};
end
```

One important thing you should notice is, whatever you put as a keyword, along with the direction, is what a person must use to open the door. To make sure the door closes at reset time you will have to add the doors reset to the '%reset' section. The door resets will be explained in Chapter 8. Notice also in this example we have a direction both in the room you are going to and the room you came from. This means you need a 'west' direction for every 'east' direction leading to it. If you do not put both you will end up with a one way direction.

5.5.2. Locked exits

Now that you have making a door down, you may find that it is not safe to leave your doors unlocked. Well the VME is ready for you. You have already seen the 'keyword' and 'open' sections and what you can set in them. Now lets use the 'EX_LOCKED field with them and introduce a new macro to allow you to set the difficulty to unlock the lock with out a key.

First lets look at the macro that allows you to set a difficulty on a lock. If you set the lock with out this macro it will default to 0 and thus be easy to pick.

```
#define DOOR_LOCK_DEF(north_lock, east_lock, south_lock, west_lock,\
up_lock, down_lock, northeast_lock, northwest_lock, southeast_lock,\
southwest_lock)
```

When using this macro you only set the value of the exit you want to add the difficulty to, you can leave the rest of the exits '0'. Only one of these macros are needed on a room no matter how many exits because each of the exits are included.. If you have an exit to the north that is locked and you want it to have a difficulty of 50% the following would be the line you would add to your room

```
DOOR_LOCK_DEF(50, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Now lets add the macro and the locked flag to the exits and create the room.

```
hallway
title "Module tunnel"
descr "The hallway is about 50 meters long and around 100 meters from
side to side and top to bottom...."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}
```

```
west to chamber descr
"The hallway opens up into a chamber.";

DOOR_LOCK_DEF(0, 50, 0, 0, 0, 0, 0, 0, 0, 0)
east to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

end

office
title "The station office"
descr
"Large paintings fill the walls of this part of the station...."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

DOOR_LOCK_DEF(0, 0, 0, 50, 0, 0, 0, 0, 0, 0)
west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};
end
```

The only thing you may be wondering about in this example is the 'key' field. I have picked 'nokey' as my value of the key. There is no key in this zone so all this does is create a key hole. If you leave the 'key' field out totally the only way you can open the lock is by a magical spell. It is also important that you read about resets of door locks in Chapter 8.

5.5.3. Hidden exits

Locking the doors may not be enough. In fact sometimes you may not want to lock the door but you might want to hide it. You can do both or either with the following macro added to your exit.

```
#define SECRET_DOOR_DIFFICULTY(DIR, SKILL) \
extra{SECRET_DOOR} {DIR, SKILL}\
""
```

So if you wanted a door or just a passage to the north hidden you would add this before or after your exits.

```
SECRET_DOOR_DIFFICULTY(NORTH, 50)
```

Now lets put it all together and link two rooms together with a hidden door.

```
office
title "The station office"
descr
"Large paintings fill the walls of this part of the station..."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

SECRET_DOOR_DIFFICULTY(SOUTH, 50)
south to portal_room descr
"You see what looks to be a portal room."
keyword {"air lock door","air lock","staff","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED,EX_HIDDEN};

end

portal_room
title "Green field room"
descr
"Like the other rooms on the station this one is large enough for
dragons to comfortably fit in.  The strange thing about this room though
is it is totally empty except for a green field right in the center.
there is a door that leads to another room to the north."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

north  to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

end
```

5.5.4. Rooms inside of rooms

Now that you have normal exits down its time to take a look at something a bit different. Lets say you wanted to put a barrel in a room that is also a room that has exits to other rooms. Or maybe in my case I want to put a transporter pad in the room that is also a room so you can exit back into the room you came

from. In the case of the teleporter I could use an object but as you will find it is much easier to deal with a room for this than an object.

To put a room in a room it is much different than using the normal exit fields. The only thing needed is the 'in' keyword and the room you are linking the current room into. There is no need for a semi-colon. The following is what the line would look like.

```
in <room that room is in>
```

Not too hard. The following are two rooms one in the other.

```
portal_room
title "Green field room"
descr
"Like the other rooms on the station this one is large enough for
dragons to comfortably fit in.  The strange thing about this room though
is it is totally empty except for a green field right in the center.
there is a door that leads to another room to the north."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"green field","field"}
"The field looks to be a green fog shifting and churning as you watch.
if you are nuts you could probably enter it."

north  to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

//A link to the portal is also here
end

room_port
names{"green field", "field"}
title "Green field"
descr
"Green Mist swirls about you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

in portal_room
end
```

Note: After adding a room in a room you should note the room in the description or put an 'outside_descr' on the room inside it. In our example we have chosen to add the description into the room instead of using the 'outside_descr'. Also doors and locks work the same way as before you can even hide this exit.

5.5.5. A room using force move.

Sometimes you will want to help players along their path. This could be for a river that flows strongly enough to force a players raft down stream or maybe for a room of quick sand that sucks the player into another room. In these situations we need to use the force move DIL, explained in the previous section. Here we have built two rooms linked only by a forced move.

```
portal_room
title "Green field room"
descr
"Like the other rooms on the station this one is large enough for
dragons to comfortably fit in.  The strange thing about this room though
is it is totally empty except for a green field right in the center.
there is a door that leads to another room to the north."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"green field","field"}
"The field looks to be a green fog shifting and churning as you watch.
if you are nuts you could probably enter it."

north  to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

//A link to the portal is also here

end
ship_port
names{"green field", "field"}
title "Green field"
descr
"Green Mist swirls about you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

in ship
```

```
dilcopy force_move@function(
//Time to activation
4,
//room and act
"portal_room@dragonst!You feel your body dissolving for lack of a better
description.&nYou appear on the deck of a ship.",
//True or False for randomizing or not
FALSE);


end
```

5.5.6. A death room

As a final touch to my little example zone I want to create a room that will kill a player instantly. I will use the DIL function Death room and the room would simply look as follows.

```
deathspace
title"Open space"
descr
"You see the ship and the station far off in the distance and you are in Space!"

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

dilcopy death_room@function (
//how often is damage done 4 would be 1 second
4,
//damage
400,
//act for the damage.
"You realize   to late that was the trash disposal transporter and you feel your lungs expl

end
```

5.6. Putting the rooms together

Using all you have learned so far and putting it all together into one zone. You end up with a very interesting space station with some secret rooms and traps.. The full zone all together looks like this.

```
#include <composed.h>
%zone dragonst
```

```
lifespan 20
reset RESET_ANYHOW
creators {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."

%rooms

chamber
title "The middle chamber of the station"
descr
"This chamber seems to have the entire station rotating around it.  It is
unbelievably large the ceiling seems to be a good 200 meeters high and
the room is perfectly cubic. Small human size ornate chairs with dragon
designs scrawled on the arms and back are arranged in a triangle like
setting with one large chair at the front.  This must be where all
station meetings are held. large pictures cover the walls depicting
dragons in all kinds of situations.  large passages lead of to the west
and the east.."

extra {"chair","chairs"}
"The chairs are made of some metal you don't recognize and every inch is covered
with some kind of dragon."

extra  {"dragon picture","picture"}
"Thousands of dragons dot the skies of this rather life like picture.  In the
center you see something move.  It looks to be a little green dragon."

extra{"green dragon","dragon","green"}
"An intellegence looking dragon is sitting perched on a large chair watching you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}


west to disposal_room descr
"You see a small room.";

east to hallway descr
"You see what looks to be a hallway.";

end

hallway
title "Module tunnel"
```

```
descr "The hallway is about 50 meters long and around 100 meters from
side to side and top to bottom.  The hallway seems to be dust free.  The
walls and the floors seem to be made out of the same sterile
metal-plastic that all space agencies uses.  There are large plate glass
windows that open up into space.  The hallway is filled with a dim light
that seems to come from everywhere yet no where all at once.  You notice
a glimmer of bright light coming from the windows.  To the east you see
an air lock and to the west the hallway opens up into a larger room."

extra {"windows","window"}
"Your eyes are drawn to a large ship lit up with running lights sitting
about 1 kilometer from the station."

extra{"floor","walls","wall"}
"Well what can be said it looks to be in perfect condition.  what else would you want to kn

extra {"large ship" ,"ship"}
"The ship looks really big and is shaped like a dragon.  The scales
sparkle and seem to be multiple colors."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to chamber descr
"The hallway opens up into a chamber.";

east to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

end

office
title "The station office"
descr
"Large paintings fill the walls of this part of the station.  The room
is as large as the other rooms big enough for Dragons to lounge while
still having a desk in one corner small enough for a humanoid.  The
floor along the north wall is lined with some kind of fabric and seems very soft to
walk on, it may be some kind of dragon lounge judging by how large an
area it covers.  There is a passage to the west."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"paintings","painting"}
"The paintings are of many dragons and riders in all kinds of tasks from
combat to look out.  All the figures seem to be staring at a staff
being held by a depiction of a wizard on the south wall."
```

```
extra {"wizard","staff"}
"The wizard has his hand stretched out and it seems there is a place
you can almost grab the staff. Maybe if you searched the staff you would
find it."

extra {"desk"}
"Its a desk alright but there doesn't seem to be any drawers and it
seems totally empty."

extra{"fabric"}
"Wussshhhhh you bound across the comfortable floor wasn't that fun."

west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

SECRET_DOOR_DIFFICULTY(SOUTH, 50)
south to portal_room descr
"You see what looks to be a portal room."
keyword {"air lock door","air lock","staff","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED,EX_HIDDEN};

end

 portal_room
 title "Green field room"
 descr
 "Like the other rooms on the station this one is large enough for
 dragons to comfortably fit in.  The strange thing about this room though
 is it is totally empty except for a green field right in the center.
 there is a door that leads to another room to the north."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

 extra {"green field","field"}
 "The field looks to be a green fog shifting and churning as you watch.
 if you are nuts you could probably enter it."

 north  to office descr
 "You see what looks to be an office."
 keyword {"air lock door","air lock","door"}
 key nokey
 open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

 //A link to the portal is also here from room_port
 end

 ship_port
 names{"green field", "field"}
```

```
 title "Green field"
 descr
 "Green Mist swirls about you."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

 in ship

 dilcopy force_move@function(
 //Time to activation
 4,
 //room and act
 "portal_room@dragonst!You feel your body dissolving for lack of a better
 description.&nYou appear on the deck of a ship.",
 //True or False for randomizing or not
 FALSE);


 end

room_port
names{"green field", "field"}
title "Green field"
descr
"Green Mist swirls about you."
movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

in portal_room

dilcopy force_move@function(
//Time to activation
4,
//room and act
"ship@dragonst!You feel your body dissolving for lack of a better
description.&nYou appear on the deck of a ship.",
//True or False for randomizing or not
FALSE);


end

disposal_room
title "Red field room"
descr
"Like the other rooms on the station this one is large enough for
dragons to comfortably fit in.  The strange thing about this room though
is it is totally empty except for a red field right in the center.
there is a door that leads to another room to the east."
```

```
movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"red field","field"}
"The field looks to be a red fog shifting and churning as you watch.
if you are nuts you could probably enter it."

east to chamber descr
"You see the main chamber.";

 //A link to the portal is also here from dis_port
end

dis_port
names {"red field","field"}
title "Red field"
descr
"Red Mist swirls about you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}
dilcopy force_move@function(
//how fast to force move in seconds
4,
//room to force move to and act
"deathspace@dragonst!You feel your body dissolving for lack of a better description.",
//true or false random move or not
0);
in disposal_room

end

ship
title "War dragon"
descr
"Blue light softly glows from con duets that line the walls of this ship.
The floors beside the east and west wall have what looks to be soft
fabric covering.  The south wall has small controls that seem to be made
for humanoids with two small chairs that look to be pilot seats.  view
portals are about 50 meters up the side of the ship on the west and east
wall and some kind of electronic screen covers the south wall.  The ship
seems to be a one room ship but there is a green field by the north
wall."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"view port"}
"Sorry your not 50 meters tall maybe it is made for a dragon?"
```

```
extra {"view screen","screen"}
"It seems to be the pilots view screen but you can't seem to see a way
to turn it on."

extra {"controls","control"}
"The controls are in some weird language and your afraid if you start
pushing buttons you might rocket in to the station or worse slam into
a planet."

extra {"soft fabric","fabric"}
"It looks to be a dragon lounge area."

 //A link to the portal is also here from ship_port
end

deathspace
title"Open space"
descr
"You see the ship and the station far off in the distance and you are in Space!"

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

dilcopy death_room@function (
//how often is damage done 4 would be 1 second
4,
//damage
400,
//act for the damage.
"You realize   to late that was the trash disposal transporter and you feel your lungs expl



end




%end
```

5.7. Suggested room exercises

1. Create a door between the Disposal room and the main chamber of the station. Make the new door pick-proof and magic-proof.
2. Create another hallway to the south of the main chamber that leads to a ship attached by an airlock. You will need a door before the hallway and before the ship so no one gets sucked out in space. You will need two more rooms for this one for the hallway and one for the ship.

3. Using the Dragon station zone as a guide create your own zone with eight rooms. Don’t worry to much about descriptions and extras. Link all eight of the rooms to each of the other rooms. This means each room should have seven exits. If you were to map this it would look like a cube marked with ’X’ on the sides.
4. Make a 3 room cliff that uses the climb DIL function to climb from the bottom to top.

Chapter 6. The NPC section

Now that you have rooms down it is time to start filling your area with some life. The NPC is the Non-player Character or mobile. These are the things players will hunt and interact with

In order to get started building NPCs you should first be aware of the NPC fields you can use. The Table 6-1 shows a full listing of all the NPC fields and their types as defined in Chapter 4.

Table 6-1. NPC fields and types

Field	Type		Field	Type
Symbolic name	Symbol		level	Integ
names	Stringlist		height	Integ
title	String		race	Integ
descr	String		attack	Integ
inside_descr	String		armour	Integ
extra	Structure		speed	Integ
manipulate	Integer		position	Integ
flags	Integer		default	Integ
weight	Integer		ability	two I
capacity	Integer		weapon	two I
dilbegin or dilcopy	Function pointer		spell	two I
defensive	Integer		romflags	Integ
offensive	Integer		light	Integ
mana	Integer		alignment	Integ
hit	Integer		minv	Integ
money	Integer		key	Strin
exp	Integer		open	Integ
sex	Integer		end tag	Symb

Many of the same fields you found in rooms, as you can see from Table 6-1, can also be found in NPCs. The fields do not always have exactly the same use when coding rooms, NPCs, and objects but they are normally set in the same manor. It is very important that you read and understand the differences of each field as they pertains to rooms and or NPCs.

6.1. Description of NPC fields

symbolic name

The rules of the symbols has been explained in Chapter 4, if you didn't read them yet you may

want to review. The important thing to realize with the NPC symbol is it is always good practice to give the NPC a symbol that resembles the title so that administrators and builders can use the **load** and the **wstat** to easily locate, examine, and load the NPC in question.

title

The NPC title is what is shown if the NPC is being attacked or talking. It is also what is shown if the NPC can be picked up. there should be no punctuation in the NPC title because of how it is used in the VME server. If you add punctuation or forget to capitalize something that the VMC thinks you should it will give you a warning when you compile.

```
title "a small dog"
title "Hansen"
title "a black dragon"
title "Drako"
title "an elephant"
```

descr

The description field is what the player sees when walking into the room or when looking with no arguments. The description on a NPC will only show up when the NPC is in the standing position. All other positions will show the title and the position they are in.

```
descr
"a small fluffy dog is chasing its tail here."
descr
"Hansen is standing here sorting the mail."
```

names

The NPC name field is much more important then the room name field. It is what is used when players are hunting and killing and even for administrators who are trying to use their administrator commands on a NPC. The names should match the title and the descr fields and have all normal combinations of each.

```
title "a baby black dragon"
descr "a tiny baby black dragon is here playing."
names {"tiny baby black dragon", "tiny black dragon",
"baby black dragon", "black dragon", "tiny dragon",
"baby dragon","dragon"}

title "Hansen"
descr "Hansen the post man is standing here sorting mail."
names{"postman","hansen"}
```

The idea of course is to make any combination that a player may type to try and act upon your NPC. You would not want to describe and title your NPC with an entirely different theme than you created its names with because a player would not know what it is called.

inside_descr

The inside description is what a player sees if it is inside the NPC. This could be used to show the player its stomach or if on a mount it could be used to show the back of the horse.

```
inside_descr
"The lining of this stomach looks indestructible.  Looks like you are in
for a long digestion cycle."
```

extra

The extra's on the NPC can be used to do many things. It can be used to store information for DIL programs or it can be used to show a part of the NPC like the room extras show a part of the room. There is also a special extra, the NPCs description when you look at it with the look <NPC> command.

Lets show the NPC description extra first. If you use an extra with no names list it will become the NPCs description when you look at any of the names on the NPC.

```
extra {}
"The green furry hamster seems to be glowing and it doesn't seem very
happy."
```

You can also use extras to show parts of the NPC.

```
extra {"hamster head","head"}
"This human like head is covered with a lot of green fur and it looks
really  upset."
```

You can also use the extras to give more detailed and vivid descriptions when the NPC is acted upon.

Table 6-2. NPC special action extras

Extra	Description
$get_s	A message shown to activator when getting a NPC.
$get_o	A message shown to others when getting a NPC.
$drop_s	a message shown to activator when dropping a NPC.

Extra	Description
$drop_o	A message shown to others when dropping an NPC.
$enter_s	A message shown to activator When mounting
$enter_o	A message shown to others when mounting.
$exit_s	A message shown to others when dismounting
$exit_o	a message shown to others when dismounting

In the following example, $1n is the activator and $2n is the unit in question. Assume you are defining a familiar.

```
extra {"$get_s"}
"You pick up the $2n it is very warm and cuddles right up to you."

extra {"$get_o"}
"$1n picks up the $2n and you see them cuddle together."
```

Finally you can use extras to store information for DIL programs. We will not cover this because it is a topic covered in-depth in the DIL documentation.

manipulate

The manipulate filed only has two values for NPCs The two values are 'MANIPULATE_TAKE' and 'MANIPULATE_ENTER'. The 'MANIPULATE_TAKE' makes it possible for a NPC to be picked up this would be good for something like a familiar. The 'MANIPULATE_ENTER' is used for things like mounts when making a mount you will also have to set the capacity so a fat player can jump on. The following is how you set the manipulate flag.

```
//Make a  NPC takable.
manipulate {MANIPULATE_TAKE}

//Make a NPC takable and able to be entered
manipulate {MANIPULATE_TAKE|MANIPULATE_ENTER}
```

flags

This field on a NPC is used to set special attributes in order to make the NPC able to be buried or not or no-teleportable and many others. The NPC flag list uses the UNIT_FL_* variables that both the objects and the rooms also use, therefore while you can set some flags on an NPC it may not have any affect unless you as a builder or administrator adds the functionality. You can also add

extras on an NPC that can be used as a special flag which you will learn as you learn to use DIL. The following is a full list of all unit flags and how they affect NPC, if they do.

Table 6-3. NPC unit flag affects

Flag	Description
UNIT_FL_PRIVATE	Currently has no affect on a NPC.
UNIT_FL_INVISIBLE	Makes unit invisible
UNIT_FL_NO_BURY	Makes it so you can create NPC that can be taken like familiars or pets that can not be buried. This flag is not needed on every NPC because the bury command will not allow you to bury an NPC outside of your inventory.
UNIT_FL_BURIED	Makes unit buried when loaded
UNIT_FL_NO_TELEPORT	Makes it so you can not summon the NPC with this flag and the NPC with this flag can not teleport. You can still teleport to NPC with this flag the current way teleport is written. Remember all spells are in DIL and you can modify them in `spells.zon`
UNIT_FL_NO_MOB	Currently has no affect on a NPC.
UNIT_FL_NO_WEATHER	Currently has no affect on a NPC.
UNIT_FL_INDOORS	Currently has no affect on NPC.
UNIT_FL_TRANS	Makes unit transparent If the Unit is transparent you will be able to see any other NPCs that it is carrying. For example if a NPC was carrying a familiar you would see that as you walked into the room. It also is used in mounts so the PC can see outside its mount. If the flag is not set on its mount the player will not see what is in the room.
UNIT_FL_NO_SAVE	Makes it so a PC can't save with unit
UNIT_FL_SACRED	Currently has no affect on a NPC.
UNIT_FL_MAGIC	Currently has no affect on a NPC.

If you wanted to make a NPC that a player can carry around but can not save you would set the manipulate and flags as follows.

```
manipulate {MANIPULATE_TAKE}
flags {UNIT_FL_NO_SAVE}
```

romflags

Like flags these are just integer values that are used to change how the NPC interacts with its environment.

Table 6-4. Room flags for NPCs

Flag	Description
CHAR_PROTECTED	Set this flag if the character is protected by the law-system.
CHAR_LEGAL_TARGET	This flag is used by the law system and should not be set unless you are re-writing your law system.
CHAR_OUTLAW	This flag is used by the law system and should not be set unless you are re-writing your law system.
CHAR_GROUP	This is used by the follow and group commands and should not be set unless you are re-writing your, movement, status, and combat systems.
CHAR_BLIND	Set this if the character is blinded.
CHAR_HIDE	Set flag if character is hidden.
CHAR_MUTE	Set flag if character is mute.
CHAR_SNEAK	Set flag if character is in sneaking mode.
CHAR_DETECT_ALIGN	No actual effect on NPCs.
CHAR_DETECT_INVISIBLE	Set flag if character can see invisible units.
CHAR_DETECT_MAGIC	No actual effect on NPCs.
CHAR_DETECT_POISON	No actual effect on NPCs.
CHAR_DETECT_UNDEAD	No actual effect on NPCs.
CHAR_DETECT_CURSE	No actual effect on NPCs.
CHAR_DETECT_LIFE	No actual effect on NPCs.
CHAR_WIMPY	Set flag if character if wimpy. Wimpy characters flee when they are low on hit points, and they gain less experience when killing others. If a character is both wimpy and aggressive (NPC_AGGRESSIVE) it will only attack sleeping players.
CHAR_SELF_DEFENCE	This is an internal combat flag set this only if you create your own combat system.

These flags are set in the same way as other flags in rooms NPCs and objects. The following are a few examples.

```
//wimpy and hidden
romflags {CHAR_HIDDEN, CHAR_WIMPY}

//NPC can see invisible
romflags {CHAR_DETECT_INVISIBLE}
```

weight

The weight is the weight of the NPC in pounds. In the future we may adjust this to allow you to make things lighter for example you could set it in ounces or grams. Right now however all we have is pounds so we have some pretty heavy feathers out there.

To use this you just enter the 'weight' keyword and then the value.

```
/80 lbs.
weight 80
```

Note: The weight affects the NPCs natural attack damage.

capacity

This field along with the NPCs strength and dexterity decides how much a NPC can carry. If you set the capacity to 300 lbs. the NPC will only be able to carry that much depending if it has the strength and dexterity to carry that much. This of course doesn't affect DIL programs that link the objects directly into the NPC. To set the capacity you just put the keyword and the amount in your NPC.

```
capacity  300
```

height

The height field is the size of the NPC in centimeters. This determines the size of the equipment the NPC is wearing. You will learn more about size and height in the object section but for now just understand this makes your NPC the right or wrong size. To set the 'height' you just put the 'height' keyword followed by the number of centimeters.

```
//6 feet tall since 1 inch equals 2.54
height 183
```

dilbegin or dilcopy

The DIL functions are what give VME servers the edge over all other muds. We will only give some examples here and leave it up to the DIL manual to teach you how to create your own functions that will make your rooms, NPC, and objects more than special.

There are several NPC functions that come standard with the VME 2.0. The following is a list of the functions.

- Mercenary
- obey
- evaluate
- guard direction
- shop keeper
- combat magic
- fido
- zone wander
- global wander
- team work
- rescue
- agressive

These are the only NPC functions currently documented in the VME 2.0 release but if you go through the zones that are released with the VME you are sure to find many more. Hopefully with the descriptions in Section 6.5. You will be able to use the functions listed here and figure out ones that are not.

Since these are just DIL's written by builders for the Valhalla mud all you have to do is use the dilcopy keyword in the NPC with the function name you want to use and the arguments that function require. The following is what you would find in the `function.zon` for evaluate.

```
/* Evaluate DIL. amt is the cost of evaluation in iron pieces.
  Note: not to be confused with evaluate@commands, which
  is a command.
*/
dilbegin aware evaluate (amt: integer);
```

```
var u1 : unitptr;
    arg: string;
    buf: string;
    cur: integer;
    craft: integer;
    category: integer;

    pc : unitptr;
    pcn: string;

    arm_text  : stringlist;
    shi_text  : stringlist;
    craft_text: stringlist;
code
{

craft_text:=  {"horrible","very bad","bad","worse than average","average",
  "a little better than average","better than average","good","very good",
  "supreme"};
arm_text:= {"clothes", "leather", "hard leather", "chain", "plate"};
shi_text:= {"small", "medium", "large"};

heartbeat:= PULSE_SEC*3;

:start:
arg:= "";
u1:= null;

wait (SFB_CMD, command("evaluate"));

if (visible(pc, self) == FALSE)
    goto start; // Pc is just trying to evaluate using the command

block;

pc:= activator;
if (pc.type == UNIT_ST_PC) pcn := pc.name;
else pcn := pc.title;

arg:= argument;

if (visible(self, pc) == FALSE)
    {
    exec ("say I don't do business with people I can't see.", self);
    goto start;
    }

if (arg == "")
    {
    exec ("say Which item do you wish to evaluate, "+pcn+"?", self);
    goto start;
    }
```

```
u1:= findunit (pc, arg, FIND_UNIT_IN_ME, null);

if (not u1)
    {
    exec ("say You do not have such an item, "+pcn+".", self);
    goto start;
    }

if ((u1.type != UNIT_ST_OBJ) or ( (u1.objecttype != ITEM_WEAPON) and
    (u1.objecttype != ITEM_ARMOR) and (u1.objecttype != ITEM_SHIELD) ))
    {
    exec ("say The "+u1.name+" is neither a sword, shield nor armor!", self);
    goto start;
    }

// Currency, skip for now

if (not transfermoney (pc, null, amt * IRON_MULT))
    {
    exec ("say The cost is merely "+moneystring(amt*IRON_MULT, TRUE)+
        ", get them first.", self);
    goto start;
    }

category:= u1.value[0];
craft:= u1.value[1] / 5 + 4; // / 5 + 4 is to get corresponding craft_text val

if (craft < 0) craft := 0;  if (craft > 9) craft := 9;

// Change the following to use skill_text(craft) instead of itoa(craft)
if (u1.objecttype == ITEM_WEAPON)
    buf := "say The "+u1.name+" is a "+weapon_name(category)+" of "+
        craft_text.[craft]+" craftmanship and material.";

if (u1.objecttype == ITEM_ARMOR)
    buf := "say The "+u1.name+" is made of "+arm_text.[category]+" and is of "+
        craft_text.[craft]+" craftmanship and material.";

if (u1.objecttype == ITEM_SHIELD)
    buf := "say The "+u1.name+" is a "+shi_text.[category]+" shield of "+
        craft_text.[craft]+" craftmanship and material.";

exec (buf, self);

goto start;

}

dilend
```

If this DIL function scares you don't worry you don't have to understand it or adjust it you only have to use it. In fact this is a really easy DIL to use. It only has one argument which is 'amt', the integer value of money you want the evaluator to charge when a person evaluates their stuff. So to use this function it would look like this on an NPC.

```
dilcopy evaluate@function (5*GOLD_PIECE);
```

this tells the evaluate DIL to charge 5 gold pieces each time a player evaluates something. For more information on the money see the money field.

All released DIL NPC functions are described in Section 6.5. Then we put some to work so you can see how to use them in Section 6.6

defensive

This field sets the NPC natural defense. The defense is a natural bonus the NPC gets when being attacked. If this value is set high enough the NPC becomes almost indestructible. You should use the macro to set the NPC natural attack and defense in Section 6.2.2.

offensive

This field sets the NPC natural offense. The offense is a natural bonus the NPC gets when being attacked. If this value is set high enough the NPC can do some serious damage when attacking. You should use the macro to set the NPC natural attack and defense in Section 6.2.2.

mana

This sets the NPC max mana points. Using this field you can create special NPCs that have more or less mana points than the VME server would normally give when a NPC is loaded.

this field is simple all you have to do to set it is put the 'mana' keyword followed by the amount of mana points you want the NPC to have as its max. The following definition would make an NPC with only 100 mana points no matter what level.

```
mana 100
```

hit

This sets the NPC max hit points. Using this field you can create special NPCs that have more or less hit points than the VME server would normally give when a NPC is loaded.

this field is simple all you have to do to set it is put the 'hit' keyword followed by the amount of hit points you want the NPC to have as its max. The following definition would make an NPC with only 100 hit points no matter what level.

```
hit 100
```

money

The money field is how you give your NPC money to have while going along its marry way through your world. The money field is an integer that tells the VME how much money the NPC is carrying. It would however be hard to calculate the amount you want on an NPC with out the macros we have provided. For example to put 5 gold on an NPC you would have to use the following on your NPC.

```
money 25600
```

I of course am not sure this will make 5 gold pieces since I did the math in my head and with all this righting I am doing my math mind doesn't seem to be working right. So to make life easier for you and me we have added some macros to help that are rather self explanatory.

```
IRON_PIECE
COPPER_PIECE
SILVER_PIECE
GOLD_PIECE
PLATINUM_PIECE
```

Now if we wanted to make a NPC carrying five gold it would be as simple as this:

```
money 5*GOLD_PIECE
```

the macro method also gains you the ability to tell the VME what amount of each coin you want on the NPC. If you set it using a single integer the compiler would pick how many of each coin. This of course is not what is desired in fact you want to be able to set your money however you like. So setting more than one coin is as simple as adding a comma between the first and second coin.

```
money 5*GOLD_PIECE, 20*IRON_PIECE
```

exp

By default a monster gives 100% of the experience it is worth. This amount is calculated according to the level of the NPC verses the level of the person fighting it. Sometimes the amount of experience is not right since the NPC is really hard to kill for example a dragon with breath weapon and heal at the same level as a merchant with a dagger. These should of course give different experience. The 'exp' field is designed to do just that. The possible range for the 'exp' field is -500% to 500%. If you put the NPC at a negative experience value it will take experience away when it is killed. If you want the default of 100% you do not even need to place this field in your NPC.

```
//add 50% to the experience gained
exp 150

/subtract 150% from the experience gained
exp -50
```

sex

Gender, one of these:

```
SEX_NEUTRAL
SEX_MALE
SEX_FEMALE
```

the values are pretty obvious which is which gender so all we will show here is how to set it.

```
//Setting a male NPC
sex SEX_MALE
```

level

When creating a NPC it must be between level 0 and 199. The level of the NPC decides how many skill points and ability points the NPC has. It, along with the 'exp' percentage, determines the amount of experience gained when the NPC is killed. To set the level of the NPC you use the 'level' keyword and then follow it by the level you are setting.

```
//set a NPC to level 50
level 50
```

race

The 'race' keyword is what you use to set the characters race. Races in VME are defined by using an integer that lets the spells and skills act differently for each specific type. The VME comes standard with many races defined in the `values.h`. For ease in access we have provided them in Appendix C. For now you can look at the short list below taken from the over all race list.

```
#define RACE_RAT            1102
#define RACE_HORSE          1103
#define RACE_BADGER         1104
#define RACE_SKUNK          1105
#define RACE_BOAR           1106
#define RACE_MOUSE          1107
#define RACE_MONKEY         1108
#define RACE_PORCUPINE      1110
#define RACE_ELEPHANT       1112
#define RACE_CAMEL          1113
#define RACE_FERRET         1114
```

If for example you wanted to make a monkey you could simply put the 'race' keyword and follow it by the monkey define like this:

```
race RACE_MONKEY
```

If the race your looking for doesn't exist in the `values.h` list, you can either add one buy picking a number not already used and creating your own define in the `values.h` or by adding the define to your zone. Defines added to a single zone will not be accessible if another builder wants to use it. You could also just set it using a number. The following two methods would act the same.

```
//add define to values.h and use in your zone
#define RACE_SPACE_TROLL  5059
race RACE_SPACE_TROLL

//Just plug in a number
race 5059
```

If you don't use the macros things can get confusing really fast with the amount of races there are.

attack

This field sets the NPCs natural attack type. Do not use this field directly instead use the macro described in Section 6.2.1

armour

This field sets the NPCs natural armour type. Do not use this field directly instead use the macro described in Section 6.2.1

speed

Speed determines the NPCs speed. The range is one to twelve. You should not set this when compiling an NPC since the result is really undefined. If you have a special NPC that you want to try to make move faster and hit faster then you could try setting this. This field is mainly added so DIL can adjust speed. The lower the number the faster the NPC speed.

```
//fastest speed.
speed 0

//slowest speed
speed 12
```

position

This field sets the position that the NPC will be in when it is first loaded. The following positions are recognized by the compiler.

```
POSITION_DEAD
POSITION_MORTALLYW
POSITION_INCAP
POSITION_STUNNED
POSITION_SLEEPING
POSITION_RESTING
POSITION_SITTING
POSITION_FIGHTING
POSITION_STANDING
```

Some of these positions make no sense you would not load a NPC into a fight or you would not load an NPC that is already dead. The positions are availible for DIL and you will need to read the DIL manuals to find out what you would want those for. For now the following are enough.

```
  POSITION_SLEEPING
POSITION_RESTING
POSITION_SITTING
```

```
POSITION_STANDING
```

The position combined with the default position determines what will be shown when a player looks in the room. If the position of the NPC matches its default position the NPC description will be shown. If it doesn't match the NPCs title and position will be shown. The default value for both 'position' and 'default' is 'POSITION_STANDING'

To set the position like with other fields you type the 'position' keyword first and follow it by the position you are setting.

```
position POSITION_SITTING
```

default

The default position along with the position determines what is shown when a NPC is in each position. If the position and default positions match the 'descr' field is shown. If they do not match the NPC title is shown along with the current position information. If 'default' is not set it defaults to 'POSITION_STANDING'. The following are possible default positions.

```
    POSITION_DEAD
POSITION_MORTALLYW
POSITION_INCAP
POSITION_STUNNED
POSITION_SLEEPING
POSITION_RESTING
POSITION_SITTING
POSITION_FIGHTING
POSITION_STANDING
```

Setting the default field is exactly like setting the 'position' field you place the 'default' keyword first and then the position you want to be default like this:

```
default POSITION_RESTING
```

ability

this field is used to set each of the NPCs abilities. It should not be used directly but instead set through the macro described in Section 6.2.3

weapon

this field is used to set each of the NPCs weapon proficiencies. It should not be used directly but instead set through the macro described in Section 6.2.4

spell

this field is used to set each of the NPCs spells. It should not be used directly but instead set through the macro described in Section 6.2.4

light

The light field on NPC is not normally set. If however you have a strange creature like a 'light bug' you can set a light value on a NPC. The default light is set to 0 which means it neither adds or subtracts from the rooms light. To set the light value on a NPC you just put the 'light' keyword first and then the value you want to add to the current light.

```
//add one to light in room
light 1

//default
light 0

//take one away
light -1
```

alignment

This field is a value between -1000 and +1000, where -1000 is ultimate evil, 0 is neutral and +1000 is ultimate good. Good is per definition any value from +1000..+350, neutral is any value from +349..-349 and evil is any value from -350..-1000. Any value in between can also be used.

```
// Quite evil, maybe a Ghoul
alignment -750

// Barely evil.
alignment -350

//barely good
```

```
alignment 350
```

minv

This field is the administrator invisible level of the NPC it is set on. This means that if you set the 'minv' to two hundred it will make it so the NPC can not be seen by anyone below the administrator level of two hundred. This is good for hiding ghosts that only come visible when they attack. In order for the 'minv' to be removed an administrator or a DIL function must change it.

```
minv 239
```

key

Currently this field is not used in the VME 2.0 release. It was added so in the future you wanted to add keys to a NPC for some weird reason like a living trunk then you can. In order to set the key you first place the 'key' keyword and then add the symbolic name of the key.

```
//if the key is in your zone
key mynpckey

//if the key is in some other zone
key someoneelses@keyzone
```

open

this field is not used yet in the VME 2.0 release. The field was added so you could make a NPC that can be opened, closed, locked, and everything else that a room or an object can have set on it. For now we will not document this but if you are interested in how you could use it study the open fields on objects or rooms.

6.2. NPC macros

6.2.1. The attack and armour macro

The natural attack and armour fields allow you to set the NPC to do damage like a certain type of weapons and to defend like a certain type of armour respectively. Lets say you had a metal cougar it would have an attack type of claw and an armour type of plate while a normal dog would have an armour type of leather and an attack type of bite. The 'NATURAL_DEF' macro is what allows you to set these fields. This macro is defined in `wmacros.h` and looks like this.

```
#define NATURAL_DEF(weapon_category, armour_category) \
   armour armour_category \
   attack weapon_category
```

The word natural can sometimes be a little confusing since you can set any of the weapons types you like on the NPC. It doesn't exactly make sense to have a dog that attacks as if it uses a long sword but if you wish it you can do it. The following is a short list of just the natural weapon types but you can find a full list in Appendix D or in the `values.h` of your mud just in case you have added some weapon types.

```
#define WPN_FIST         34
#define WPN_KICK         35
#define WPN_BITE         36
#define WPN_STING        37
#define WPN_CLAW         38
#define WPN_CRUSH        39
```

Again you don't have to use leather for dogs as we have already mentioned with our metal cat idea you could make a cloth dragon if you really want but its up to you to keep some sanity on your VME. The following is the list of armour types that can be set. You will see that the list is exactly the same as the list you will find later when making armour.

```
#define ARM_CLOTHES  0  /*Same as a Human in jeans and a T-shirt*/
#define ARM_LEATHER  1  /* A soft flexible leather base armour   */
#define ARM_HLEATHER 2  /* A hard un flexible leather base armour */
#define ARM_CHAIN    3  /* A flexible armour composed of interlocking rings */
#define ARM_PLATE    4  /* An un flexible plate armour. */
```

Now that you have the defines to work with we will return to our metal cat and normal dog. The definitions for them would look something like this.

```
//Metal Cat
NATURAL_DEF(WPN_CLAW, ARM_PLATE)

//normal dog
NATURAL_DEF(WPN_BITE, ARM_LEATHER)
```

You should know that the weight of the monster determines the maximum amount of damage it can give when using a natural attack. The weight is categorized as follows:

Table 6-5. Weight size chart

LBS	Size
0 - 5	Tiny
6 - 40	Small
41 - 160	Medium
161 - 500	Large
500 and up	Huge

By default monsters are medium. So make sure you take this into account when you are creating your NPC.

6.2.2. The defense and offense bonus macro

There comes a time when you may want to make your NPC super naturally powerful. It is for those times that the offense and defense fields are available for you to set. Normally they default to 0 but you can set them from 0 to 5000. The higher you set the offense number the harder you will hit people you a re in combat with. The higher you set the defense the harder it will be for people to hit your NPC. The following macro allows you to set both the offense and defense.

```
#define ATTACK_DEFENSE(attack, defense) \
offensive attack \
defensive defense
```

Using this macro is rather easy you just put the value you want for each and your all done

```
//a really hard hitting hard to kill NPC
ATTACK_DEFENSE( 1000, 1000)
```

6.2.3. The NPc abilities macro

All abilities are in the range [1..200]. Players usually have a maximum of 150, modified by magic... 200 is considered divine. When creating a monster you can not directly specify the size of the abilities, instead you specify a percentage distribution of points. The amount of points are then distributed by the computer according to the specified level. The 'MSET_ABILITY' macro is available for this purpose, and is defined as:

```
#define MSET_ABILITY(str,dex,con,hpp,bra,cha,mag,div)   \
  ability[ABIL_STR]  str   \
  ability[ABIL_DEX]  dex   \
```

```
ability[ABIL_CON]  con   \
ability[ABIL_HP]   hpp   \
ability[ABIL_BRA]  bra   \
ability[ABIL_MAG]  mag   \
ability[ABIL_DIV]  div   \
ability[ABIL_CHA]  cha
```

Note: Note the sum of the ability values must be 100%. This is thus an example of an ability distribution:

```
MSET_ABILITY(25,15,10,15,10,5,10,0)  /* Sum is 100% */
```

The amount of points distributed depends directly upon the level of the monster and the percentage. If the percentage is too high and the level is also set High some ability points may be lost since a NPC gets all abilities over 255 cut off. For example a level 199 monster with an ability percentage a bit above 20% will make an ability above the 255 points maximum. In the current combat system in the VME 2.0 it is not necessary to spend points on both 'mag' and 'div' on the NPC since only one or the other is ever used depending on which is higher.

6.2.4. The NPc weapon and spell macros

NPCs know about weapons and spells but not at the same detailed level as the player. For NPCs the spell and weapon group are used. Thus the Axe hammer category defines all defence and all attack for all kinds of axes and hammers, whereas the player would have to train individually in each axe and hammer type. The same is true for spells. Thus if a monster has 25 points in the weapon sword category it will fight (and defend) with all sword-like weapons at skill 25. When you define weapon and spell skills (monsters have no skill skills) you also define these as percentages, and the program automatically distributes the points. Use the pre-defined macros:

```
#define MSET_WEAPON(axe_ham, sword, club_mace, pole, unarmed, special)  \
weapon[WPN_AXE_HAM]    axe_ham   \
weapon[WPN_SWORD]      sword      \
weapon[WPN_CLUB_MACE]  club_mace  \
weapon[WPN_POLEARM]    pole \
weapon[WPN_UNARMED]    unarmed    \
weapon[WPN_SPECIAL]    special
```

Table 6-6. MSET_WEAPON arguments

Argument	Description
axe_ham	any hammer or axe
sword	any sword like weapon, including dagger and rapier, etc.
club_mace	any club or mace like weapon, flails, morning star, etc.
polearm	any spear or pole like weapon: spear, trident, sickle, scythe etc.
unarmed	Is any bite, claw, sting or other natural attack.
special	any very peculiar weapon, currently only whip.

```
#define MSET_SPELL(div, pro, det, sum, cre, min, hea, col, cel, int, ext)  \
spell[SPL_DIVINE]      div  \
spell[SPL_PROTECTION]  pro  \
spell[SPL_DETECTION]   det  \
spell[SPL_SUMMONING]   sum  \
spell[SPL_CREATION]    cre  \
spell[SPL_MIND]        min  \
spell[SPL_HEAT]        hea  \
spell[SPL_COLD]        col  \
spell[SPL_CELL]        cel  \
spell[SPL_INTERNAL]    int  \
spell[SPL_EXTERNAL]    ext
```

Table 6-7. MSET_SPELL arguments

Argument	Description
div	Covers all divine sphere spell.
pro	Covers all protection sphere spells.
det	Covers all detection sphere spells.
sum	Covers all summoning spells.
Cre	Covers all creation spells.
min	Covers all mind spells.
hea	Covers all heat spells (fireball, etc.)
col	Covers all cold spells (frostball, etc.)
cel	Covers all cell (electricity) spells (lightning bolt, etc.)
int	Covers all internal (poison) spells (toxicate, etc.)
ext	Covers all external (acid) spells (acid ball etc).

Note: If your not sure what your weapon or spell is categorized as you can look in the `weapons.def` or the `spells.def` for that you are using for your VME server.

The sum of all spell and weapon skills must be 100%. For example, the following would be a legal setting of weapons and spells.

```
//  75%  Total,  Club/Mace  is primary
      MSET_WEAPON(10,10,20,5,15,5)

//  25%  Total,  Fire  is primary
      MSET_SPELL(8,0,0,3,0,3,2,3,3,3,3)
```

Remember that the groups define both attack and defence. Thus if you make an Orc which has 0% in the flail group it can only use its dexterity to defend itself. Likewise with spell groups. For this reason the groups are both "resistance" as well as attack groups.

6.2.5. Using the composed.h

The file composed.h contains many standard monsters. It is a good idea to study these definitions, as they form the basis of many different monsters. Note that the definitions by no means are perfect, but we are hoping to make a more or less complete monster compendium. If you create certain (general) monsters, please design it as a macro so it can be incorporated in the file. The more monsters created by using these macros the easier it will be for your builders to create NPCs. If you think you have a really all inclusive Composed.h and want to share it with the rest of the VME servers running out there on the internet. Feel free to submit it to the VME staff and we will put it in the contribution directories on our release site.

Note: For more information on how to use the composed.h when building your NPC see Section 6.3.

6.3. Building your first NPC

Now that you have built your first zone with rooms its time to populate it with Non playing characters for your players to hunt, kill, and or interact with. In the last couple of sections you have looked through the fields. In this section we are going to make a nice easy NPC and then show how to use the `composed.h` to make your NPC building easier. As I have previously stated in the section on room building I like dragons so the first NPC your going to learn to build is a big bad ugly dragon. Don't worry if you hate dragons or you just want to build a normal world you will learn plenty from my dragon example to be able to adjust it to whatever you want.

When making NPCs you create the zone source file first as shown in Chapter 3. If you only have NPCS you do not need the %reset, %objects, and %rooms fields. For the examples in this chapter we will use the zone we created in Chapter 5 and add the %mobiles section where we will put all the NPC

definitions. At the end of this chapter, in Section 6.7, we will bring it all together with the rooms we have already defined and our NPCs.

The first part of all NPC definitions is the symbolic name it is good to always pick a name that will match the name of the NPC so it will be easy to load the NPC. The reason the symbolic and name should match is when you use the command **wstat** it will only show you a list of the NPCs by symbolic name for example if you type **wstat zone dragon mobiles** You will get the following:

```
List of mobiles in zone Dragon:
dragon clerk trashman
```

If you didn't make it clear what the NPC was by the symbolic name it might look like this:

```
List of mobiles in zone Dragon:
npc1 npc2 npc3
```

While this might be great when you first start imagine trying to remember each NPC if you have over 30 of them.

Now lets get started with our dragon. As with the rooms all that is required to make a NPC is the symbolic and end fields. That of course will make a NPC with all defaults.

```
bldragon
end
```

Thats it for that dragon right? Nope not quite that makes a NPC with all defaults. That means this will probably be a very wimpy human with a symbolic name of dragon and no description, in short it will be a blank when you load it. Now lets start putting the Dragon together.

The first three things we need are the dragons title, description and names. The description should be what you see when you do a 'look' in the room. The title should be what you see when the NPC is talking or fighting. Finally the names should cover everything in the title and description fields so if you wanted to kill them you can easily know what to type.

```
bldragon

title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}
...
end
```

The names, title and description shouldn't be to hard so I don't think its necessary to go into any more description on the subject. Lets move on. Now we have to take care of what a player sees when he or she looks at a NPC. to make the main description of an NPC you place an extra on the NPC with no names in

the list. The blank extra is a special extra that will be shown every time you look at anything in the names list of the NPC. So a description of a NPC would look something like this.

```
extra {}
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."
```

Now that you have a main description for the NPC you need to make any smaller descriptions that you want the player to be able to look at. In this case the dragon eyes and claw would be good things to describe and maybe a few other things just for humor.

```
extra {"eyes","eye"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"

extra {"bat wings","wings"}
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."
```

Now that we have the NPC all described we should start setting things like race, gender, level, height, weight, defense, offense, alignment, abilities, and finally skills and spells. First we will pick a race. Normally the list of races are in your `values.h`. We have the list in Appendix C and I have searched for dragon and found the race I want it is as follows.

```
race RACE_DRAGON_BLACK
```

If you don't think there is any difference in dragons you can set them all to one race like you could define a race called 'RACE_DRAGON' or something like that.

Now we should chose the gender of our dragon. Make sure your descriptions match what you pick. It would be pretty weird to have a female dragon and have the descriptions say 'he, him, or his'

```
    sex SEX_MALE
```

Now lets set the height and weight. Remember you set the height in centimeters and the weight in pounds. In the future the VME will standardize to one or the other but for now we have to play the conversion game.

```
//20 feet  (1 inch = 2.54 cm
height 625

//566 KG (1 lb. = .45359 kg)
  weight 1250
```

Now that we have the size of the dragon we can pick a level. My dragon is only 20 feet tall and 1250 pounds so I think maybe he is a bit young and will make his level around 70.

```
  level 70
```

The dragon may not wear armour but by default a dragon should have an armour equal to someone wearing plate mail. Not only that but my dragon doesn't like to use his mouth as his natural attack and he sure can't punch with those big claws so we need to set his attack type to claw.

```
  NATURAL_DEF(WPN_CLAW, ARM_PLATE)
```

Not many black dragons are good so I will pick almost totally evil, since totally evil is -1000 I will set it as follows.

```
  alignment -900
```

Finally we get to the hard part. Many people have trouble setting the NPC abilities, weapons, and spells. There really is nothing to it if you have read Section 6.2.3 and Section 6.2.4. The only thing you have to remember is the numbers you are putting are not the actual amount but a percentage of the skill points you want the game to use on your NPC when having your NPC practice as it loads and all amounts of abilities must add up to 100% as well as all weapons and spells must add up to 100%. For the dragon the following would be what the abilities, weapons, and spells would look like.

```
//big strong and magical most ability points on str and mag
MSET_ABILITY(20,12,12,12,12,12,20,0)

//Set natural attack highest of weapons because he fights with claws
MSET_WEAPON(10,10,10,5,30,5)

//black dragons have natural defense from acid set it highest
MSET_SPELL(0,0,0,0,0,0,0,0,0,30)
```

That is all you need to make a basic dragon. True this dragon will not do anything but claw you to death if you attack it but in Section 6.6 we will give the dragon some more abilities with some special DIL functions. For now lets take a look at our finished product.

```
bldragon

title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}

extra {}
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."

extra {"eyes","eye"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"

extra {"bat wings","wings"}
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."

 race RACE_DRAGON_BLACK
 sex SEX_MALE
 height 625
 weight 1250
 level 70
   NATURAL_DEF(WPN_CLAW, ARM_PLATE)
 alignment -900
MSET_ABILITY(20,12,12,12,12,12,20,0)
MSET_WEAPON(10,10,10,5,30,5)
MSET_SPELL(0,0,0,0,0,0,0,0,0,30)

end
```

As you can see there is a bit more to building an NPC than a room but its really not that much harder. Try changing some stuff and make sure to practice debugging your errors the more compiling you do the better you will get.

6.4. Compiling and debugging your first NPC

As we have previously mentioned in Section 5.3 it is always a good idea to build one or two things and then compile to make finding errors easy. In this case we have one NPC to compile and rather than having all the rooms get in my way while compiling it I have removed them and only have the '%mobiles' section. The following is what the zone looks like when it has only one NPC in it.

```
#include <composed.h>
%zone dragonst
lifespan 20
reset RESET_ANYHOW
creators {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."

%mobiles

bldragon

title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}

extra
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."

extra {"eye","eyes"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"

extra {"bat wings","wings"}
```

```
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."

 race RACE_DRAGON_BLACK
 sex SEX_MALE
 height 625
 weight 1250
 level 70
 NATURAL_DEF (WPN_CLAW, ARM_PLATE)
 alignment -900
MSET_ABILITY(21,12,12,12,12,12,20,0)
MSET_WEAPON(10,10,10,5,30,5)
MSET_SPELL(0,0,0,0,0,0,1,0,0,0,30)

end

%end
```

I removed the '%rooms' section added a '%mobiles' section and stuck the dragon in and now its ready to be compiled and put into the VME server for you to be able to look at it in the game. If you downloaded our example zones for this document you can compile this zone along with us and fix the errors as we do for practice. The filename is `debug_npc.zon`. Just so you know the errors in this zone are intentional so please don't write me an email telling me that there are errors in it.

The command to compile the zone is **VMC debug_npc.zon**. Here is what we get when we first try and compile the zone.

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_npc.zon'
debug_npc.zon: 32: parse error
   Token: 'extra'
debug_npc.zon: 55: parse error
   Token: 'alignment'
Compilation aborted.
```

This error file doesn't look any harder than the last one we dealt with when compiling our first room. The problem is when we go to line '32' and look for an error we don't find one. This normally means that the error was hard for the compiler to figure out. The best way to deal with an error like this is to start at the line it gives you and go up and look for an error. When we do this we notice that the extra right above the line that the error is on is missing '{}' so we will add them back in. Most of the time you want to do one error and recompile but sometimes you can shorten the process for example in this error file the word 'alignment has been spelled wrong so we can fix that before we recompile so go to line '56' and fix that. Now with those two errors fixed we can recompile and this is what we get:

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_npc.zon'
FATAL: Abilities in 'bldragon' sums up to 101,and not 100.
FATAL: Spells&weapons in 'bldragon' sums up to 101, and not 100.
WARNING: Fatal errors in zone.
```

As we have said before you have to make sure that abilities add up to 100 percent this error is telling us that my math sucks and that I have added 1 extra percent to the abilities. Not only that but again if we look at both errors I have also put 1 extra on weapons and spells. So we can fix both of these at once. Notice it doesn't give a line number but that is not a problem because you can search for 'MSET_ABIL' and it will take you right to the problems. After I subtract one from the abilities and one from either the spells or weapons the following is the error file I get.

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_npc.zon'
VMC Done.
```

Notice there are no errors and it says 'VMC done', this means that you have now successfully compiled the zone. The main thing I want to point out is that you can sometimes fix more than one error at a time but be carefull when doing this if you try to fix some errors before fixing the first you will be trying to fix things that are not broken. The safest way to compile stuff is still fix one error at a time.

Now that you have a compiled zone you should check and make sure that all the files are there. When you compile a zone you will end up with three extra files. the files will have the same filename as your zone with a new extension in this case you should have the following.

```
debug_npc.data
debug_npc.err
debug_npc.reset
debug_rm.zon
```

If you have all of these you are all set to go. If not then there is something seriously wrong and you may want to write the VME staff for help.

To get your new zone in the mud all that is needed is to make sure your zone is in the zonelist in the VME etc directory and copy these files into your zone directory. Then reboot the mud. You should be able to log on your builder character and load your NPC by typing **load bldragon@dragonst** and you can list your zones NPCs by typing **wstat zone dragonst mobiles**.

There you go you have now compiled your first NPC. As you can see with as little as you have learned so far you can already make a variety of monsters and NPCs of any kind. The next section will cover the DIL functions you can use with a NPC and then we will get right into some more complex examples.

6.5. DIL functions for NPCs

The DIL language is the language a builder can use to make his own special functions on rooms, NPCs, objects, PCs, and much more. This manual is for basic zone writing and therefore will not go into how to write your own DIL functions. The VME however is released with many functions for you as an Administrator and your builders to use to make special rooms, NPCs, and objects. The following is a list of all NPC functions released with the VME 2.0 server.

Mercenary

This function allows you to make an NPC hireable by players in the game. You simply dilcopy this unto a mob and the player can then type 'contract <character name> and the mob will hunt that char for a fee. This function takes no special arguments. the following is the function definition found in `function.zon`:

```
dilbegin mercenary_hire();
```

To use it you simply dilcopy it to your NPC as follows:

```
dilcopy mercenary_hire@function();
```

Obey

This function when dilcopied on the NPC will make it obey there master. And example mobile of this is the familiar. Upon completing a small quest a Mage receives a familiar that will obey that player and that player only. The player can simply type **Tell <familiar name> cast heal <player name>** and it will carry out the command. This function takes no arguments. The following is the definition found in `function.zon`:

```
dilbegin obey();
```

To use this function simply dilcopy it to the NPC you want to be a mercenary like so:

```
dilcopy obey@function();
```

Evaluate

This function when placed on an NPC allows it to evaluate items for a fee. The function definition looks as follows:

```
dilbegin evaluate (amt:integer);
```

The function has one argument 'amt' that lets you set the cost of the evaluation of items. If you wanted to set the value to four gold it would work just like when setting the money field and look as follows:

```
dilcopy evaluate@function(4*GOLD_PIECE);
```

Guard Direction

This is an enhanced version of the Guard Way Function. It will allow both certain players to enter as well as certain mobs. An optional stop DIL can be supplied if you wish to do something special. It takes two arguments, the activator and the direction. The following is the function definition:

```
dilbegin guard_dir(direction : string, excludepc : stringlist,
                   excludenpc : stringlist, stopdil : string);
```

This function is dilcopied onto the mob in the room that the mob is initially loaded. Thus is the mob is summoned or commanded away it will not block the directions until it is back to where it was first created. This DIL takes four arguments. The first is the direction to block. The second arguments is for those PC's you wish to allow to pass in that direction without being stopped. The next for the NPCs you wish to allow to pass. The last is the 'act you wish the blocking mob to display to the PC's that are blocked from proceeding in the selected direction. The third and forth arguments may be 'null', this will pass the defaults to the dilcopy. The third argument is a stringlist that tells which players or NPCs are excluded from the guard. The forth is a DIL you can pass in to do something special to people who are stopped. We will not show how to use the forth argument because it takes more DIL knowledge than this manual covers. The following would be a valid dilcopy for this function:

```
dilcopy guard_dir@function("south", {"papi",
{"rejji"}, null, null);
```

Combat magic

This function when placed on a mobile allows it to use any of the combat spells. The function definition is as follows: It also allows the NPC to cast heal during combat on itself.

```
dilbegin combat_mag(atk_spl : string, def_spl : string,
                    def_pct:integer, spd: integer);
```

Table 6-8. Combat magic arguments

argument	Type	description
atk_spl	string	Attack spell ie "fireball" or "" for none
def_spl	string	Defense Spell ie "heal" or "" for none
def_pct	integer	At what % of hitpoints defense spell will be cast
spd	integer	speed at which mob will uses its attack magic, 1 for all at once (every round) to 5 for every 5 rounds. I suggest 2.

Defense spells take priority when the hit points fall below the % specified, after (if) the hits have been restored above that number attack magic will resume. If def_spl is used, function automatically makes sure that it retains enough mana for at least one healing, ie it will attack 4 times if it don't need a healing. The following would be an example of what you would put on your NPC:

```
dilcopy combat_mag@function ("harm", "heal", 25, 2);
```

Fido

This function turns the NPC into a corpse eating mobile. The following is the functions definition:

```
dilbegin fido(txt1:string,txt2:string);
```

Table 6-9. Fido arguments

Argument	Type	description
txt1	string	The text shown when mob finds and eats corpses, default: 'XXX savagely devours a corpse.' will be shown if txt1 is set to "". If txt1 is set to "stop", the mob will NOT devour corpses (convenient if you want your dogs to only eat food leftovers but not corpses).

Argument	Type	description
txt2	string	The text shown when mob finds and eats ITEM_FOOD, default: 'XXX hungrily devours YYY.' will be shown if txt2 is set to "". If txt2 is set to "stop", the mob will NOT devour ITEM_FOOD (convenient if you want to make a corpse-eating ghoul, who'd choke on normal food, etc).

Note: In both cases $1n is the mob itself, $2n is the title of the item devoured.

An example of the 'fido' function on an NPC would look as follows:

```
dilcopy fido@function("$1n slowly devours $2n, crunching the bones.",
                      "$1n grabs $2n and hungrily munches it.");
```

Wander zone

This function allows a mob to wander around more than one zone, but not all zones. You specify what zones the mob can wander. Has optional intelligence which allows the opening and closing of doors. The following is the function definition:

```
dilbegin wander_zones(zones : string, spd : integer, doors : integer,
                      lckd_doors : integer);
```

Table 6-10. Zone wander arguments

argument	Type	description
zones	string	A string of zone names separated by spaces.
spd	integer	The speed (in seconds) at which the mob wanders. Minimum = 5 secs (for process time).
doors	integer	Can open/close doors (0 = false, 1 = true)

argument	Type	description
lckd_doors	integer	Can open/closed locked doors (0=false, 1=true)

The options are not too hard so we will show how to use it and leave it at that

```
dilcopy wander_zones@function ("halfzon haon_dor", 5, 1);
```

Note: @loadzone option which is used by inputting @loadzone somewhere in the zones string will replace it by the zone the mob is loaded in like this:

```
dilcopy wander_zones@function ("halfzon haon_dor @loadzone", 5, 1);
```

Global wander

This is similar to wander_zones because it's just a modified version of it, which requires no zones argument since it moves in all zones. The following is the function definition:

```
dilbegin global_wander(spd : integer, doors : integer,
                       lckd_doors :integer);
```

Table 6-11. Global wander arguments

argument	Type	description
spd	integer	The speed (in seconds) at which the mob wanders. Minimum = 5 seconds (for process time).
doors	integer	Can open/close doors (0 = false, 1 = true)
lckd_doors	integer	Can open/closed locked doors (0=false, 1=true)

The options are not to hard so we will show how to use it and leave it at that

```
dilcopy global_wander@function (60, 1, 1);
```

Team work

This function when placed on a mob, located in the same room as another mob, will allow the mobile to assist the other in combat. The following is the function definition:

```
dilbegin aware teamwork(lst: string);
```

This function takes one argument which is a string of the mobiles it is to protect. So if we wanted a NPC to protect both Jesper and Enver the dilcopy would look as follows:

```
dilcopy teamwork@function("jesper/enver");
```

Note: For this to work both mobs must be in the same room or following each other.

Rescue

This function when placed on a mob, located in the same room as another mob, will allow the mobile to rescue the other mob if it is attacked. The function is as follows:

```
dilbegin aware rescue(lst: string);
```

This function takes one argument, a string of those mobiles you wish this NPC to assist should they be attacked.

Note: For this to work both NPCs must be in the same room or following each other.

Again we will use our test subjects Jesper and Enver and if we wanted an NPC to protect them and come to their aid the following would be the dilcopy:

```
dilcopy teamwork@function("jesper/enver");
```

Agressive

This function makes a Mob hostile to person(s) in the room with it, under certain conditions which are provided as arguments. In short, the all-singing, all-dancing aggression DIL. The following is this functions definition:

```
dilbegin aggressive (sx : integer, rce : integer, opp : integer,
                     levl : integer, sanc : integer, tme : integer,
                     tar : integer, align : string, attack :
                     stringlist);
```

Table 6-12. Agressive arguments

argument	Type	description
sx	integer	NOTE: not the sex values in values.h. This decides the sex of your mob's victim. 0 - Sex doesn't matter, 1 - Attack opposite sex to self (if not neutral!), 2 - Attack SEX_MALE, 3 - Attack SEX_FEMALE, 4 - Attack SEX_NEUTRAL.
rce	integer	Any of the PC races from 0 to 14. A value of -1 means we don't care about the victim's race.
opp	integer	0 - Non race specific (same as rce := -1) 1 - Attack the specified rce, 2 - Attack any pc race _but_ the specified rce.
levl	integer	Allow level specific aggression. A value of 30 would make the mob hostile to all pcs level 30 and above. A value of -30 (note the -) would make the mob hostile to all pcs level 30 or below. A value of 0 means level doesn't matter.

argument	Type	description
sanc	integer	Does this mob obey the sanc/soothe rules? (ie, if someone has cast sanctuary on themselves, will this mob recognize it, and not attack, or attack anyway). 0 - Doesn't obey sanc or soothe 1 - Obeys only sanc 2 - Obeys only soothe 3 - Obeys both sanc and soothe (SOOTHE is a new spell for ranger's guild)
tme	integer	Time in ticks to wait before attacking (is automatically put to RANTIME, ie, time variance of time-time/2 to time+time/2). Values accepted are from 0 to 400 (that's 0 - 100 seconds. Can be specified using PULSE_SEC).
tar	integer	This is a special value which determines which of the eligible victims we pick. -2 - Last eligible victim to into the room. -1 - Weakest eligible victim in room. 0 - Random eligible victim. +1 - Strongest eligible victim in room. +2 - First eligible victim into the room.
align	string	The desired alignment of the victim. "ANY" - We don't care about the alignment. "GOOD" - Attack only good alignment. "EVIL" - Attack only evil alignment. "NEUTRAL " - Attack only neutral alignment. "OPPOSITE" - Attack opposite alignment to self (provided self isn't neutral). "SALIGN" - Attack same alignment as self. "DALIGN" - Attack any alignment different to self.

argument	Type	description
attack	stringlist	This is a 2 string stringlist. These are the messages sent to the people in the room except the victim, and the victim itself, in that order. If the second (victim) string is "", the first string will be shown to the victim, as if they were anyone else in the room. You can leave both blank if you wish. $1n is the mob name (self), $3n is the victim's name. NOTE: the $ values only apply if you supply BOTH string 1 and 2.

The argument descriptions pretty much explain everything there is to know about the agressive function and what it was so we will finish off by giving an example of a dilcopy that will create a NPC that fits the following description

Let's say our mob is a level 40 Goblin who doesn't like dwarves. He's very particular in that he doesn't like evil female dwarves who are level 20 and above. He does recognize the sanctuary spell, but he doesn't recognize soothe, and he'll wait 10 seconds on average before he attacks. The 2 messages sent are: "$1n savagely attacks $3n with his big axe!" and "$1n attacks you!"

Here's what the function call would look like:

```
dilcopy aggressive (3, 2, 1, 20, 2, PULSE_SEC*10, 0, "EVIL",
                    {"$1n savagely attacks $3n with his big axe!",
                     "$1n attacks you!"});
```

In this example, 3 (attack females), 2 (attack dwarves), 1 (just dwarves), 20 (Level 20+ victims), 2 (obey only soothe), PULSE_SEC*10 (wait around 10 seconds before attacking), EVIL (attack only evil), and the strings in the stringlist are displayed to the victim and the room, in that order.

Janitors

This function turns the NPC into a janitor that will pick up items left lying around and will also tell new players. those under level 20, where there corpse is if they come across it in there wanderings. You must also supply the wander_zone DIL for this to work correctly. The definition for the janitor function is as follows:

```
dilbegin janitors(rate: integer);
```

The Janitor function only takes one argument and that determines how fast the janitor will pick up stuff in seconds.

To make a Janitor that will pick up stuff every thirty seconds the following would be the dilcopy:

```
dilcopy janitor@function(30);
```

Shop Keeper

This is one of the more complex dilcopies and considered by some to be confusing. Below I will step you through the creation of the mobile, and the application of this DIL using defines to hopefully simplify the use of this function. The function is defined as:

```
dilbegin aware shopkeeper(prod: stringlist, custom_acts : stringlist,
                          opentimes : stringlist, itemtype: string,
       sellprofit : integer,buyprofit: integer,
       maxcash : integer,closedil : string,
       dilparams : string );
```

Step 1

First, figure out what you want your shopkeeper to sell. He can sell pretty much any item from any zone, as long as you know its symbolic name. For Hogan (our example shopkeeper), he is a bartender, so he sells things you would commonly find in a pub, tavern, or bar. His list of items (their symbolic names at which zone) are:

```
grain_alcohol@gobtown1
pretzels@gobtown1
beer_nuts@gobtown1
rum_coke@gobtown1
tuborg@udgaard
```

Now, since I use defines, I make a define for Hogan's products. I called it TAVERN_PROD (for tavern's products). And use the following setup and just stick the names of your items in where his items are. Each item will have a setup like this:

```
 "tuborg@udgaard 15 20"
```

A symbolic name followed by two numbers, with the entire thing in quotes. Where :

- The "tuborg@udgaard" is your item's symbolic name

- The 15 (our first number in the string) is the number of that item that will load into your shop daily
- The 20 (our second number in the string) is the limit available of that item in the shop ever

So this shop would sell tuborgs.. and every mudday, 15 tuborgs will load into our shop to replenish our stock.. and the maximum tuborgs we can have at any given time is 20.

Next we place all this info into the define we created above, which would look like this:

```
#define TAVERN_PROD \
{"grain_alcohol@gobtown1 15 20", \
 "pretzels@gobtown1 10 15", \
 "beer_nuts@gobtown1 15 20", \
 "rum_coke@gobtown1 10 15", \
 "tuborg@udgaard 15 20"}
```

The above is my define for Hogan's products that he sells. Each one is in quotes and they have commas separating them. If you put each item on its own separate line, you need to put a space and then a back slash after the comma which separates the items, like it shows above. The entire define is enclosed in curly brackets, {}. Now we are finished with the items he sells.

Step 2

Now you must decide what kind of dialogue you would like your shopkeeper to say. There are ten separate responses you can make for your shopkeeper. They do go in a specific order and I will list what each do.

Response 1

The first response will be used if the shopkeeper doesn't have the particular item you're trying to buy in stock, or if he doesn't sell that item at all.

Response 2

The second response is used when the player is trying to sell the shopkeeper something which the player does not have in his inventory. Items worn by the player can not be sold until removed from there worn position.

Response 3

The third response is used when the player is trying to sell the shopkeeper something that is not one of those of his trade types. (This will be more clear when we get to the trade type part below. Such as a player trying to sell our bartender a canoe. =)

Response 4

The fourth response is used when the player tries to buy something from the shopkeeper, but doesn't have enough money to buy it.

Response 5

The fifth response is used when the sale was successful and the player buys something from the shopkeeper.

Response 6

The sixth response is again used when the sale was successful, but this time it's for when the player successfully sells something to the shopkeeper.

Response 7

The seventh response is used when the shopkeeper doesn't have enough of the item to sell as the player requests. Such as someone trying to buy 10 tuborgs, but our bartender only has 9.

Response 8

The eighth response is Used when the shop is closed and a player tries to buy something from or sell something to the shopkeeper.

Response 9

The ninth response is used when the player tries to sell an item to the shopkeeper that the shopkeeper doesn't trade that item.

Response 10

The tenth (and last!) response is for when the shopkeeper has run out of money and can't afford to buy what the player is trying to sell him.

Now. again, I made a define called TAVERN_MSG (for tavern messages) and I placed all my responses in it. Don't forget, they do go in that specific order! And below is our setup with responses in it.

Note: In the responses below, $1n refers to the shopkeeper, $2n refers to the item being sold, bought, etc.. and $3n refers to the person dealing with the shopkeeper.

```
#define TAVERN_MSG \
{"$1n says, 'This is a tavern, I don't sell such an item as that!'", \
 "$1n says, '$3n, you don't even have that!'", \
 "$1n says, 'I don't trade with things such as $2n! Just buy a beer!'", \
 "$1n says, '$3n, you can't afford $2n.'", \
 "$1n says, 'Thank you, $3n, here are %s for $2n.'", \
 "$1n says, 'Thank you, $3n.'", \
 "$1n says, 'I don't have that many $2ns in stock.'", \
 "$1n says, 'I'm on break, come back later.'", \
```

```
"$1n says, 'I haven't got a use for $2n.'", \
"$1n says, 'I'd like to buy it, but I can't afford it, sorry.'"} \
```

Just like with the items, the responses are in quotes, and are separated with commas. Since I put each on their own separate lines (for neatness sake), I put a space and a back slash, \, after each. The entire define is enclosed in curly braces,{}. And then you're done with your shopkeeper's messages! Don't worry, it gets simpler from here.

Step 3

Now we get to decide when we want our shop to be open. We get to use military time, so 1 refers to 1am, 6 refers to 6am, 12 refers to 12pm (noon), 18 refers to 6pm, and both 0 and 24 refer to 12am (midnight).. etc.. =) And my define for this I named TAVERN_OPEN_TIMES (for when the tavern is open) and I used the following setup:

```
#define TAVERN_OPEN_TIMES {"12","18"}
```

You just have to make your define and stick your times in quotes and enclose them in curly braces. So Hogan will be opened from 1am ("1") until 11pm ("23"). He will close after 11pm and reopen again at 1am. You may also add multiple open and close times as per the following example:

```
#define TAVERN_OPEN_TIMES {"1","12","16","20"}
```

Step 4

For this step you get to choose your shopkeeper's trade types. And what you pick all depends on what kinds of things you want your shopkeeper to sell and buy. You can find the item types in `values.h`, but I'll also list them below.

Table 6-13. Sale types

Value	Type	Value	Type
1	ITEM_LIGHT	14	ITEM_TRAP
2	ITEM_SCROLL	15	ITEM_CONTAINER
3	ITEM_WAND	16	ITEM_NOTE
4	ITEM_STAFF	17	ITEM_DRINKCON
5	ITEM_WEAPON	18	ITEM_KEY
6	ITEM_FIREWEAPON	19	ITEM_FOOD
7	ITEM_MISSILE	20	ITEM_MONEY
8	ITEM_TREASURE	21	ITEM_PEN
9	ITEM_ARMOR	22	ITEM_BOAT

Value	Type	Value	Type
10	ITEM_POTION	23	ITEM_SPELL
11	ITEM_WORN	24	ITEM_BOOK
12	ITEM_OTHER	25	ITEM_SHIELD
13	ITEM_TRASH		

We are making a bartender, so he should sell food and drinks. So I selected ITEM_FOOD, and it's corresponding number is 19. So I made my define TAVERN_ITEM_TYPE (for our tavern's type of items) using the following setup:

```
#define TAVERN_ITEM_TYPE "19"
```

Now, if we had a magic shop, and wanted to sell potions and scrolls, I would make a define called for example MAGIC_ITEM_TYPE and make it like this:

```
#define MAGIC_ITEM_TYPE "2 10"
```

The 2 was our number corresponding to the scrolls, and the 10 corresponded to the potions. That is how you make your shopkeeper sell more than one item type, just stick the corresponding numbers inside quotes with a space separating them.

Step 5

Now you get to decide how much money your shopkeeper gets to have to buy things with. One platinum piece is equal to 40960. So if I wanted him to have two platinum pieces to spend on buying things from players, I would make a define called TAVERN_MAX_CASH (for the maximum amount of money our shopkeeper gets) and use the following setup:

```
#define TAVERN_MAX_CASH 81920
```

I got 81920 by multiplying the 40960 (one platinum) by 2

Step 6

The sixth thing we get to do is to decide how much profit does our shopkeeper get when he sells his items. Items should have a cost to them already tagged on them, so their whatever their cost is, that is equal to 100%. Now, if your shopkeeper wants to take an extra 10% on the items he's selling so he can support his family, then you would make his selling profit 110. So I made a define called TAVERN_SELL_PROFIT (for his profit when he sells things) and gave him 110% selling profit, and used the following setup:

```
#define TAVERN_SELL_PROFIT 110
```

Step 7

This last define is almost exactly like our selling profit, but now, we're doing our buying profit... when a player sells an item to the shopkeeper, he doesn't want to pay full price because it must be used since a player has it. Now the item's cost is 100%, and maybe the shopkeeper only wants to buy items for half its cost. So I would make his buying profit 50, and use the following setup.

```
#define TAVERN_BUY_PROFIT 50
```

All of the previous defines you can place anywhere in the zone. I like to place mine before the %DIL section just to keep them somewhere where I know they all are. You can place each one on the mob itself, but I think that looks cluttered so's why I put them where I put them. And you're going to have a dilcopy placed on your mob also, which I'll explain in step 8.

Step 8

Okay one last step. Now you get to place the DIL on your shopkeeper mob using all of those cute little defines you just made. The syntax for the DIL is:

```
dilcopy shopkeeper@function(products, responses, opentimes,
                            tradetypes, sellprofit, buyprofit,
                            maxcash, closedil, dilparams);
```

Take your defines from above the defines that you made are called:

```
TAVERN_PROD, TAVERN_MSG, TAVERN_OPEN_TIMES, TAVERN_ITEM_TYPE,
TAVERN_SELL_PROFIT, TAVERN_BUY_PROFIT, and TAVERN_MAX_CASH.
```

Now all you have to do is place them in the syntax where they belong like in the below example.

```
dilcopy shopkeeper@function(TAVERN_PROD, TAVERN_MSG, TAVERN_OPEN_TIMES,
                            TAVERN_ITEM_TYPE, TAVERN_SELL_PROFIT,
                            TAVERN_BUY_PROFIT, TAVERN_MAX_CASH, "", "");
```

For the last two fields, I just put "", "" because I don't make my own unique DIL for those. By just putting and empty set of quotes for each, it makes it go to the default. And make sure you don't forget that semicolon at the end either.

You are pretty much done now. All you have to do is place your defines either up before %DIL or somewhere in %mob if you'd like, and place the dilcopy onto your mob.

Just to make sure I've completely beaten a very dead horse, this is what it would look like in the end.

```
%zone sample_zon

#define TAVERN_PROD \
{"grain_alcohol@gobtown1 15 20", \
 "pretzels@gobtown1 10 15", \
 "beer_nuts@gobtown1 15 20", \
 "rum_coke@gobtown1 10 15", \
 "tuborg@udgaard 15 20"}

#define TAVERN_MSG \
{"$1n says, 'This is a tavern, I don't sell such an item as that!'", \
 "$1n says, '$3n, you don't even have that!'", \
 "$1n says, 'I don't trade with things such as $2n! Just buy a beer!'", \
 "$1n says, '$3n, you can't afford $2n.'", \
 "$1n says, 'Thank you, $3n, here are %s for $2n.'", \
 "$1n says, 'Thank you, $3n.'", \
 "$1n says, 'I don't have that many $2ns in stock.'", \
 "$1n says, 'I'm on break, come back later.'", \
 "$1n says, 'I haven't got a use for $2n.'", \
 "$1n says, 'I'd like to buy it, but I can't afford it, sorry.'"} \

#define TAVERN_OPEN_TIMES {"1","23"}
#define TAVERN_ITEM_TYPE "19"
#define TAVERN_MAX_CASH 81920
#define TAVERN_SELL_PROFIT 110
#define TAVERN_BUY_PROFIT 50


%mobiles

bartender
names {"hogan","goblin","bartender"}
title "Hogan"
descr "Hogan stands behind the bar waiting to take your order."
extra {}
"He looks back at you, patiently waiting for your order."
M_AVG_GOBLIN(76,SEX_MALE)
alignment -1000
exp 100
money 2 SILVER_PIECE, 2 COPPER_PIECE

dilcopy shopkeeper@function(TAVERN_PROD, TAVERN_MSG, TAVERN_OPEN_TIMES,
                            TAVERN_ITEM_TYPE,TAVERN_SELL_PROFIT,
                            TAVERN_BUY_PROFIT, TAVERN_MAX_CASH, "", "");
end
```

6.6. A more complex set of NPCs

In the last sections you learned all the fields and how to make a basic NPC. In this section we will use the information from the last sections to create some more unique NPCs for our dragon station zone There is not a lot of new information here we will be using the DIL functions from the previous section and adding some flags to the NPCs to make them act different.

6.6.1. Magic casting NPC

The basic Dragon we made in the Section 6.3 looks like a real dragon but it is a bit boring when you fight a dragon and don't get toasted by an acid or fire spell or two. in Section 6.5 the 'combat_magic' function was described. This is the function you use to make all NPCs cast magic while in combat. With that in mind lets take a look at how we can make our dragon a bit more interesting g.

Well lets see a black dragon is supposed to have the ability to either cast fire breath or acid breath depending on who you talk to. Once you learn DIL you could even make fire breath be a skill not a spell but for now we will stick with what we know. I like the acts of the acid breath spell so we will use that as the spell of choice. The NPC as we defined it before hasn't changed so the following would be the entire dragon with the combat magic function.

```
bldragon

title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}

extra {}
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."

extra {"eye","eyes"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"
```

```
extra {"bat wings","wings"}
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."

 race RACE_DRAGON_BLACK
 sex SEX_MALE
 height 625
 weight 1250
 level 70
 NATURAL_DEF (WPN_CLAW, ARM_PLATE)
 alignment -900
MSET_ABILITY(20,12,12,12,12,12,20,0)
MSET_WEAPON(10,10,10,5,30,5)
MSET_SPELL(0,0,0,0,0,0,0,0,0,30)

//Combat Magic added.
   dilcopy combat_mag@function("acid breath", "", 25, 2);

end
```

That is all there is to it. If you are wondering where I got the 'acid breath' part of the DIL copy all the spells names are in the `spells.def` or you can just pick a spell by what you would type if you wanted to cast it yourself in the game. Now if this still looks like a lot to do to make a dragon the dragon above could have been created with exactly the same information by using the `composed.h` and the 'M_DRAGON_BLACK_OLD' macro. If we use the macro the dragon would look like this

```
bldragon

title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}

extra {}
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."

extra {"eye","eyes"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"
```

```
extra {"bat wings","wings"}
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."

M_DRAGON_BLACK_OLD(SEX_MALE)

end
```

As you can see with the second way the dragon was a lot easier to make because we let the macro set all the other values we just described the NPC and set the macro.

6.6.2. A wandering janitor

Our space station is nice but we wouldn't want a lot of people coming to visit if we didn't have a janitor to walk around cleaning up. Dragons most likely wouldn't be low life enough to be Janitors so I think we will leave that up to a hobgoblin. the following would be the definition of a hobgoblin wandering Janitor.

```
janitor
names {"ugly janitor", "janitor", "hobgoblin"}
title "an ugly janitor"
descr "an ugly janitor is walking around, cleaning up."

extra{}
"This ugly green thing looks more goblin than hobgoblin but he seems
intent on cleaning everything around him."

race RACE_HOBGOBLIN
level 6
sex SEX_MALE
height   130       /* cm            */
weight   120       /* Pounds        */
// he is sort of good for cleaning so much
alignment 900

NATURAL_DEF(WPN_FIST, ARM_LEATHER)
MSET_ABILITY(10,10,10,23,15,22,10,0)   \
MSET_WEAPON(10,10,10,10,10,10)    /*  Average in everything, any wpn */
MSET_SPELL(4,2,2,2,2,2,2,6,6,6,6)  /* Resistances       */


//give him some money
money 5 IRON_PIECE

dilcopy janitors@function(15);

// only want him cleaning the station
dilcopy wander_zones@function("dragonst", 20, 1, 1);

end
```

Like with the dragon if we wanted to create this NPC easier we would use the Hobgoblin defines in `composed.h` so we don't have to fill out all the information. If we did this it would simply look as follows:

```
janitor
names {"ugly janitor", "janitor", "hobgoblin"}
title "an ugly janitor"
descr "an ugly janitor is walking around, cleaning up."

extra{}
"This ugly green thing looks more goblin than hobgoblin but he seems intent
on cleaning everything around him."

M_AVG_HOBGOBLIN(6, SEX_MALE)

// he is sort of good for cleaning so much
alignment 900

//give him some money
money 5 IRON_PIECE

dilcopy janitors@function(15);

// only want him cleaning the station
dilcopy wander_zones@function("dragonst", 20, 1, 1);

end
```

As you can see you can combine the DIL functions you learned in Section 6.5 to make your NPC do more than one thing.

6.6.3. Creating a teacher

Setting up teachers on valhalla is harder and more strict formed than most things in the game. The reason is the way you set them up is to use an old form of functions called special. In the future guilds will be in DIL like everything else but for now the teachers that are released with the VME are base code. Even though the teachers are base code they still allow for you to adjust many things. The truth is you don't have to use our teachers you could code your own in DIL but many find this easier.

Guild teacher definitions are actually less complex than people think they are. There really are no big, mysterious secrets. The key is simply to understand the numbers and how the balance is achieved. (From here on in, GTD will refer to "guild teacher definition", and "entity" will be used to refer to the ability/spell/skill/weapon being offered by the GTD.)

> **Note:** Using TAB characters in GTDs is a very bad idea! It will have extremely adverse effects, including causing the mud to crash.

We need to tart first and explain that to use the teacher special functions you need to use a field not described anywhere else in the NPC descriptions. The field is called a 'special'. The reason the 'special' field was not included in the field descriptions is because it is being removed from the VME as soon as banks and teachers have been replaced by DIL there will be no more special functions. For now however we need to use them for the for mentioned purposes. The format for a special function is as follows:

```
special <Function define> <function arguments>
```

For teachers the function definition is 'SFUN_TEACH_INIT' and the arguments range from who can join, what the acts of the teacher are, and what the teacher teaches.

In the first part of a SFUN_TEACH_INIT definition, there are several pieces of important information which allow the mud to know what you are trying to provide to players. The entire set of arguments are enclosed in double quotes. So a teacher with nothing at all in it would look like this:

```
special SFUN_TEACH_INIT "<arguments go here>"
```

After the opening double quote ("), the first thing necessary is the text formatting code '&l', which tells the compiler to ignore the standard text formatting protocols for the rest of the string or until the '&f' code is given.

Immediately following the '&l' code, with no spaces at all, the type of entity the function will teach is defined. It is one of the following types:

- abilities
- spells
- skills
- weapons

The first two lines of the SFUN_TEACH_INIT looks like this:

```
special SFUN_TEACH_INIT
"&labilities;0;
```

> **Note:** The '0;' is the end of field marker that will be used through out the teacher fields

The next set of arguments to the teacher function is the acts the teacher will use when a person is trying to train. There are seven acts total and they are as follows:

Teacher acts

Line 1

This line is displayed when the player makes a mistake in typing the name of an entity, whether or not it is actually offered at the teacher.

Line 2

This line is similar to the first, but is usually evident of attempting to practice an entity not offered at this particular teacher.

Line 3

This is the act displayed when the player has insufficient funds to practice the desired entity.

Line 4

This is displayed when the player has not got enough ability or skill practice points to learn the desired entity.

Line 5

Simply put, the player has exceeded the teacher's expertise in the offered entity and must go elsewhere to learn more, if at all possible.

Line 6

This line is displayed when the player is either wearing magical, stat-modifying equipment or is affected by spells/skills which modify stats.

Line 7

This line is what is shown when your armour affects you using special functions in base code. This is rare and may never be seen. In the future when the teachers are made in DIL this act may not be necessary.

Now we can add the acts to our example it would look as follows:

```
special SFUN_TEACH_INIT
"&labilities;0;
 $1n tells you, 'I have never heard of such an ability.'; $1n
tells you, 'I do not know how to teach this ability.'; $1n tells you,
'You haven't got %s for me.'; $1n tells you, 'You haven't got %d ability
points.'; $1n tells you, 'I can not teach you any more.'; $1n tells you,
'You must be unaffected by magic, else I can't teach you.'; $1n tells
you, 'Remove all equipment, please.';
```

Note: $1n is the substitution variable for the teacher's title (see the DIL document for more details).

%s is the amount of money the teacher requires to teach the entity in question.

%d is the amount of ability or skill practice points required to practice the given entity.

The standard GTD termination character, a semicolon (;) ends each act

about the actual GTDs, or the lines that actually define what the teachers teach and everything to do about how expensive and how high they teach it. A GTD is a single line, composed of several fields. The data contained in these fields determines how a particular entity is practiced.

Here is an example of a GTD:

```
  0;  100; scan                          ;   9;  9000;   7;          0;
```

Note that it has a lot of white space in it. It is really not necessary, as the field terminators are the semicolons. This means that you could just as easily write:

```
0;100;scan;9;9000;7;0;
```

When making teachers most people prefer to put the white space in to make it easier to read and find errors. A larger example would look as follows:

```
  0;  100; scan                          ;   9;  9000;   7;          0;
  0;   90; consider                      ;   4;  4000;   5; 10;      0;
  0;  100; appraisal                     ;   9;  4000;   5;          0;
  0;  100; fleeing                       ;   9;  9000;   6; 12;      0;
```

Each part of the GTD separated by a semicolon is called a field. The following is the fields and their definitions.

Field 1

> Field 1 is the guild level. It is simply the level at which one is allowed to practice the particular entity in the guild. A level of 0 indicates that one can practice it from the beginning.

Field 2

> This is the maximum percentage one is allowed to practice the particular entity. It can be anywhere from 1 to 200. For abilities, it should be in increments of 2 (or divisible by 2), and for spells, skills, or weapons, in increments of 5 (or divisible by 5). This means that the effective minimum depends on whether the entity is an ability or a spell, skill or weapon.
>
> Therefore, the minimums for abilities is 2, and the minimum for spells, skills or weapons is 5. Anything less will be automatically practiced to the effective minimum. This also means that putting an odd number for abilities will result in the player practicing an additional point, and for spells, skills or weapons, practicing up to the nearest 5.

For example: Percentage maximum for an ability is set for 95%. Player practices the ability, which then becomes 96%. Player then decides to practice a weapon skill. It is set to 77% maximum. After practicing, the player has 80% in the weapon. It is better to simply make the ability 96% and the weapon 80% maximum instead.

Field 3

Name of Entity

Field 4

This is the practice cost minimum in Old gold pieces. Anything less than 10 here will simply be translated as 1 iron piece. Anything other than a number divisible by 10 is rounded up to the nearest iron piece.

Field 5

This is the practice cost maximum in old gold pieces. It is generally defined as being 1000 times what Field 4 is defined as:

The reason for this is because there is a scale related to the current practice percentage of the entity. The practice minimum is what the entity costs when it is at 0%, and the practice maximum is what it costs to practice it to maximum percentage.

It is not necessary to multiply the minimum by 1000 to define the maximum. It can be any number, as long as the maximum exceeds the minimum. Failure to do so will result in the mud crashing. It is unknown what will happen if the fields have 0 entered in them. I warn that it could be rather unpredictable.

Setting the maximum too high will only serve to dissuade players from practicing and indeed even playing. This is not to say you cannot make a truly powerful entity very expensive. Just make sure that the entity is worth the cost.

Field 6

This is the only necessary practice points field. It allows one to use their skill or ability practice points to increase their character's powers. This number is generally determined by the desire of the builder to make the entity easier or harder to practice, based upon the logical consideration of how this class would learn/use the entity.

For instance, a Fighter class would find practicing "fist" to be exceedingly easy, so they may only need 4 points to practice this skill, but on the other hand, a Mage class would not have such an experience so it may cost them 15 points to practice.

Don't go overboard, making this field an outrageous number will mean that no one is likely to practice the entity, mainly because of lack of points. However, like in the cost fields, it can be made

expensive to reflect the power of the entity.

Field 7 - X

You can repeat field 6 as many times as you want the player to be able to practice in one level so you can set a point cost each time. This is so each time they practice at a level you can make it more expensive.

An example of this would look as follows:

```
  0;   90; consider                      ;   4;  4000;   5; 10;      0;
```

This example has consider being practiced at guild level 0 up to 90%, costing 4 old gold pieces minimum, 4000 old gold pieces maximum, 5 practice points for the first practice per level and 10 for the second.

For 15 points a level, the player can practice consider twice. It is actually quite nice to be able to do this, and generally easier entities take advantage of this more often.

Each additional field is followed by a semicolon

Field X

This is the termination field for the GTD. It tells the compiler that the GTD is finished. In fact, if you set a practice point field to 0, it would consider that Field X, and end the GTD.

Every GTD line must have an end of field marker!

When we refer to old gold pieces they represent the VME money system as follows:

Table 6-14. Old to new money conversions.

Old gold piece	New money
10	1 Iron Piece (IP)
80	1 copper piece (CP)
640	1 silver piece (SP)
5120	1 gold piece (GP)
40960	1 platinum piece (PP)

Of course, to finish up the entire SFUN_TEACH_INIT, you must include a closing double quotation

mark (''). Failure to do so will cause no end of problems for you.

A finished teacher of ability function would look like this:

```
special SFUN_TEACH_INIT
"&labilities;0;
$1n tells you, 'I have never heard of such an ability.';
$1n tells you, 'I do not know how to teach this ability.';
$1n tells you, 'You haven't got %s for me.';
$1n tells you, 'You haven't got %d ability points.';
$1n tells you, 'I can not teach you any more';
$1n tells you, 'You must be unaffected by magic, otherwise I can't teach
you.';
$1n tells you, 'Remove all equipment, please.';

  0;  100; Strength                    ;   4;  4000;   8;       0;
  0;   90; Dexterity                   ;  14; 14000;  12;       0;
  0;   90; Constitution                ;  14; 14000;   9;       0;
  0;  100; Hitpoints                   ;   4;  4000;  11;       0;
  2;   60; Brain                       ;  23; 23000;  14;       0;
  4;   80; Charisma                    ;  18; 18000;  14;       0;
"
```

There are two other easy to use specials that you will want to use with your teacher. they are 'SFUN_GUILD_BASIS' and 'SFUN_MEMBERS_ONLY'.

SFUN_GUILD_BASIS

This initializes the teacher and the only argument is the guild name

```
special SFUN_GUILD_BASIS "Udgaard Fighter"
```

SFUN_MEMBERS_ONLY

This simply blocks anyone who is not in the guild from practicing This means the argument is the guild name. and the act separated by a '#', an example would look as follows:

```
special SFUN_MEMBERS_ONLY "Udgaard fighter#$1n says, 'Buggar ye off, $3n.'"
```

Finally we will show you what a full teacher would look like with the entire specials and the NPC definition.

```
jones

names {"blacksmith", "smith", "Jones"}
title "Jones the blacksmith"
descr "The venerable Jones Blacksmith is here."
```

```
extra {}
"The smith is old but his arms still retain the strength of his youth.
He looks as if he has retired, but he can still put you through the
drills of physical training."
flags {UNIT_FL_NO_TELEPORT}
romflags {CHAR_PROTECTED}
//Define from composed.h
M_HUMAN_WARRIOR_AXE(80, SEX_MALE)

//negative exp to discourage killing teacher
exp -100

special SFUN_GUILD_BASIS GUILD_UDG_FIGHTER
special SFUN_MEMBERS_ONLY GUILD_UDG_FIGHTER+"#$1n says, 'Buggar ye off, $3n.'"

special SFUN_TEACH_INIT
"&labilities;0;
$1n tells you, 'I have never heard of such an ability.';
$1n tells you, 'I do not know how to teach this ability.';
$1n tells you, 'You haven't got %s for me.';
$1n tells you, 'You haven't got %d ability points.';
$1n tells you, 'I can not teach you any more';
$1n tells you, 'You must be unaffected by magic, otherwise I can't teach
you.';
$1n tells you, 'Remove all equipment, please.';

  0;  100; Strength                    ;   4;  4000;   8;       0;
  0;   90; Dexterity                   ;  14; 14000;  12;       0;
  0;   90; Constitution                ;  14; 14000;   9;       0;
  0;  100; Hitpoints                   ;   4;  4000;  11;       0;
  2;   60; Brain                       ;  23; 23000;  14;       0;
  4;   80; Charisma                    ;  18; 18000;  14;       0;
"
```

6.6.4. Guild master functions

When you make a guild you have to make a guild master for it. This NPC will let people join and leave the guild and it also gives titles. To create a guild master you need to use three 'special' functions. The functions you need are 'SFUN_GUILD_BASIS', 'SFUN_GUILD_MASTER', and 'SFUN_GUILD_TITLES'.

SFUN_GUILD_BASIS

This initializes the master and the only argument is the guild name

```
special SFUN_GUILD_BASIS "Udgaard Fighter"
```

SFUN_GUILD_MASTER

The guild master function takes 6 arguments. Like the teacher function the arguments must be surrounded by double quotes. The strings in the arguments must be surrounded by tilde marks (~). The following is a description of the arguments and what they do.

Line1:

The first argument on is what the guild name is. As in the teacher you need to pre-pend the '&l' symbol so the VME doesn't mess with the formatting of the string when the guild master is compiled.

```
&lGuild = ~Udgaard fighter~
```

Line 2

This argument is what quest needs to be done before the character can enter the guild.

```
Guild Enter Quest = ~Fighter Proven~
```

Line 3

This argument is the amount it costs to enter the guild in old gold pieces.

```
Guild Enter Cost = 640
```

Line 4

this argument is the quest the player must do before leaving the guild. If the player has not completed this quest the guild master will not let the player leave.

```
Guild Leave Quest = ~Wimp proven~
```

Line 5

This argument is how much old gold pieces it will cost to quit the guild. If the player doesn't have enough money the guild master will not let the player join.

```
Guild Leave Cost = 3200
```

Line 6

This argument is what guild the guild master will not accept players from. For example the following will make it so no Thief can be in the fighter guild.

```
Guild Exclude Quest = ~Udgaard Thief Quitter~
```

Note: When we refer to old gold pieces they represent the VME money system as shown in Table 6-14

The following is what a full 'SFUN_GUILD_MASTER' function would look like

```
special SFUN_GUILD_MASTER
"&lGuild               = ~Udgaard fighter~
Guild Enter Quest    = ~Fighter Proven~
Guild Enter Cost     = 640
Guild Leave Quest    = ~Wimp proven~
Guild Leave Cost     = 3200
Guild Exclude Quest  = ~Udgaard Fighter Quitter~"
```

SFUN_GUILD_TITLES

This function allows the player to request a title at the guild master. A title will be given every 5 levels up to level 100 so you have 20 titles you can set. The title function takes one title for male chars and one title for female chars so you have to set 40 titles.

The arguments of this function are easy. The first thing you do is the normal '&l' to let the VME know not to mess with this string. Then you put the guild name. Finally you follow it by the list of 40 titles. The following is the fighter guilds title list

```
special SFUN_GUILD_TITLES
"&lUdgaard fighter
the %s Swordpupil
the %s Swordpupil
the %s Recruit
the %s Recruit
the %s Sentry
the %s Sentress
the %s Fighter
the %s Fighter
the %s Soldier
the %s Soldier
```

```
    the %s Warrior
    the %s Warrior
    the %s Veteran
    the %s Veteran
    the %s Swordsman
    the %s Swordswoman
    the %s Fencer
    the %s Fenceress
    the %s Combatant
    the %s Combatess
    the %s Hero
    the %s Heroine
    the %s Myrmidon
    the %s Myrmidon
    the %s Swashbuckler
    the %s Swashbuckleress
    the %s Mercenary
    the %s Mercenaress
    the %s Swordmaster
    the %s Swordmistress
    the %s Lieutenant
    the %s Lieutenant
    the %s Champion
    the %s Lady Champion
    the %s Dragoon
    the %s Lady Dragoon
    the %s Cavalier
    the %s Cavalier
    the %s Knight
    the %s Lady Knight"
```

Put all three of these functions on your NPC and your all set you have a guild master.

6.6.5. NPC banker

The banker function is the easiest 'special' function there is to use. The following placed on an NPC will make a banker:

```
special SFUN_BANK
```

As you see its very simple, so we will just show you a completed banker and leave it at that.

```
bob

names {"Bob"}
title "Bob"
descr "Bob the Banker is here, sitting behind the counter."
```

```
extra {}
"He has a very serious look on his face."

// define from composed.h
M_SHOP_KEEPER(4, SEX_MALE, RACE_HUMAN)

//discourage people from killing banker
exp -500

flags {UNIT_FL_NO_TELEPORT}

special SFUN_BANK
end
```

6.7. Dragon station with rooms and NPCs

Now we will add the NPCs we have built to the zone from the previous chapter. This is still not complete while it does compile and you can log into your zone, you still have to load your NPCs and there are no objects. These will be added as you progress through this manual. The following is the source file so far.

```
#include <composed.h>
%zone dragonst
lifespan 20
reset RESET_ANYHOW
creators {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."

%rooms

chamber
title "The middle chamber of the station"
descr
"This chamber seems to have the entire station rotating around it.  It is
unbelievably large the ceiling seems to be a good 200 meeters high and
the room is perfectly cubic. Small human size ornate chairs with dragon
designs scrawled on the arms and back are arranged in a triangle like
setting with one large chair at the front.  This must be where all
station meetings are held. large pictures cover the walls depicting
dragons in all kinds of situations.  large passages lead of to the west
and the east.."
```

```
extra {"chair","chairs"}
"The chairs are made of some metal you don't recognize and every inch is covered
with some kind of dragon."

extra  {"dragon picture","picture"}
"Thousands of dragons dot the skies of this rather life like picture.  In the
center you see something move.  It looks to be a little green dragon."

extra{"green dragon","dragon","green"}
"An intellegence looking dragon is sitting perched on a large chair watching you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}


west to disposal_room descr
"You see a small room.";

east to hallway descr
"You see what looks to be a hallway.";

end

hallway
title "Module tunnel"
descr "The hallway is about 50 meters long and around 100 meters from
side to side and top to bottom.  The hallway seems to be dust free.  The
walls and the floors seem to be made out of the same sterile
metal-plastic that all space agencies uses.  There are large plate glass
windows that open up into space.  The hallway is filled with a dim light
that seems to come from everywhere yet no where all at once.  You notice
a glimmer of bright light coming from the windows.  To the east you see
an air lock and to the west the hallway opens up into a larger room."

extra {"windows","window"}
"Your eyes are drawn to a large ship lit up with running lights sitting
about 1 kilometer from the station."

extra{"floor","walls","wall"}
"Well what can be said it looks to be in perfect condition.  What else would
you want to know?"

extra {"large ship" ,"ship"}
"The ship looks really big and is shaped like a dragon.  The scales
sparkle and seem to be multiple colors."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to chamber descr
```

```
"The hallway opens up into a chamber.";

east to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

end

office
title "The station office"
descr
"Large paintings fill the walls of this part of the station.  The room
is as large as the other rooms big enough for Dragons to lounge while
still having a desk in one corner small enough for a humanoid.  The
floor along the north wall is lined with some kind of fabric and seems
very soft to walk on, it may be some kind of dragon lounge judging by
how large an area it covers.  There is a passage to the west."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"paintings","painting"}
"The paintings are of many dragons and riders in all kinds of tasks from
combat to look out.  All the figures seem to be staring at a staff
being held by a depiction of a wizard on the south wall."

extra {"wizard","staff"}
"The wizard has his hand stretched out and it seems there is a place
you can almost grab the staff. Maybe if you searched the staff you would
find it."

extra {"desk"}
"Its a desk alright but there doesn't seem to be any drawers and it
seems totally empty."

extra{"fabric"}
"Wussshhhhh you bound across the comfortable floor wasn't that fun."

west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

SECRET_DOOR_DIFFICULTY(SOUTH, 50)
south to portal_room descr
"You see what looks to be a portal room."
keyword {"air lock door","air lock","staff","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED,EX_HIDDEN};

end
```

```
 portal_room
 title "Green field room"
 descr
 "Like the other rooms on the station this one is large enough for
 dragons to comfortably fit in.  The strange thing about this room though
 is it is totally empty except for a green field right in the center.
 there is a door that leads to another room to the north."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

 extra {"green field","field"}
 "The field looks to be a green fog shifting and churning as you watch.
 if you are nuts you could probably enter it."

 north  to office descr
 "You see what looks to be an office."
 keyword {"air lock door","air lock","door"}
 key nokey
 open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

 //A link to the portal is also here from room_port
 end

 ship_port
 names{"green field", "field"}
 title "Green field"
 descr
 "Green Mist swirls about you."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

 in ship

 dilcopy force_move@function(
 //Time to activation
 4,
 //room and act
 "portal_room@dragonst!You feel your body dissolving for lack of a better
 description.&nYou appear on the deck of a ship.",
 //True or False for randomizing or not
 FALSE);


 end

room_port
names{"green field", "field"}
title "Green field"
```

```
descr
"Green Mist swirls about you."
movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

in portal_room

dilcopy force_move@function(
//Time to activation
4,
//room and act
"ship@dragonst!You feel your body dissolving for lack of a better
description.&nYou appear on the deck of a ship.",
//True or False for randomizing or not
FALSE);


end

disposal_room
title "Red field room"
descr
"Like the other rooms on the station this one is large enough for
dragons to comfortably fit in.  The strange thing about this room though
is it is totally empty except for a red field right in the center.
there is a door that leads to another room to the east."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"red field","field"}
"The field looks to be a red fog shifting and churning as you watch.
if you are nuts you could probably enter it."

east to chamber descr
"You see the main chamber.";

 //A link to the portal is also here from dis_port
end

dis_port
names {"red field","field"}
title "Red field"
descr
"Red Mist swirls about you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}
dilcopy force_move@function(
//how fast to force move in seconds
```

```
4,
//room to force move to and act
"deathspace@dragonst!You feel your body dissolving for lack of a better description.",
//true or false random move or not
0);
in disposal_room

end

ship
title "War dragon"
descr
"Blue light softly glows from con duets that line the walls of this ship.
The floors beside the east and west wall have what looks to be soft
fabric covering.  The south wall has small controls that seem to be made
for humanoids with two small chairs that look to be pilot seats.  view
portals are about 50 meters up the side of the ship on the west and east
wall and some kind of electronic screen covers the south wall.  The ship
seems to be a one room ship but there is a green field by the north
wall."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"view port"}
"Sorry your not 50 meters tall maybe it is made for a dragon?"

extra {"view screen","screen"}
"It seems to be the pilots view screen but you can't seem to see a way
to turn it on."

extra {"controls","control"}
"The controls are in some weird language and your afraid if you start
pushing buttons you might rocket in to the station or worse slam into
a planet."

extra {"soft fabric","fabric"}
"It looks to be a dragon lounge area."

 //A link to the portal is also here from ship_port
end

deathspace
title"Open space"
descr
"You see the ship and the station far off in the distance and you are in Space!"

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

dilcopy death_room@function (
```

```
//how often is damage done 4 would be 1 second
4,
//damage
400,
//act for the damage.
"You realize to late that was the trash disposal transporter and you feel your
lungs explode.");


end

%mobiles

bldragon

title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}

extra {}
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."

extra {"eye","eyes"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"

extra {"bat wings","wings"}
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."

M_DRAGON_BLACK_OLD(SEX_MALE)

end

janitor
names {"ugly janitor", "janitor", "hobgoblin"}
title "an ugly janitor"
descr "an ugly janitor is walking around, cleaning up."
```

```
extra{}
"This ugly green thing looks more goblin than hobgoblin but he seems intent
on cleaning everything around him."

M_AVG_HOBGOBLIN(6, SEX_MALE)

// he is sort of good for cleaning so much
alignment 900

//give him some money
money 5 IRON_PIECE

dilcopy janitors@function(15);

// only want him cleaning the station
dilcopy wander_zones@function("dragonst", 20, 1, 1);

end

bob

names {"Bob"}
title "Bob"
descr "Bob the Banker is here, sitting behind the counter."
extra {}
"He has a very serious look on his face."

// define from composed.h
M_SHOP_KEEPER(4, SEX_MALE, RACE_HUMAN)

//discourage people from killing banker
exp -500

flags {UNIT_FL_NO_TELEPORT}

special SFUN_BANK
end









%end
```

6.8. Suggested NPC exercises

1. Using the DIL function for team work found in Section 6.5 create a guard that will help all other guards.
2. Using the DIL function for rescue found in Section 6.5 add to your guard from exercise one and make it so will now rescue as well as help.
3. Using the shop keeper function from Section 6.5; make a shop keeper that sells two types of food. The shop keeper should make 5 of them a day and it should only buy items of the food type. For all other arguments be creative.
4. Using the shop keeper you created from exercise three, turn your shop keeper into a global wondering sales person.
5. Using the DIL function for agressive found in Section 6.5 create a Dwarf agressive to any Orc that walks into the room.

Chapter 7. The objects section

The previous chapters would be enough for you to create an entire game of nudists with no technology and no items of any kind. This of corse would be a very boring game of naked people fighting with no weapons. don't worry the VME has a solution to this you can build objects to dress up the NPCs and to fill the rooms with cluttered junk.

In order to get started building objects you should first be aware of the object fields you can use. The Table 7-1 shows a full listing of all the object fields and their types as defined in Chapter 4.

Table 7-1. Object fields and types

Field	Type		Field	Type
symbolic name	Symbol		affect	affec
names	Stringlist		dilbegin or dilcopy	funct
title	String		key	Strin
descr	String		open	Integ
inside_descr	String		manipulate	Integ
extra	Structure		spell	Integ
minv	Integer		value	Integ
alignment	Integer		cost	Integ
flags	Integer		rent	Integ
weight	Integer		type	Integ
capacity	Integer		end tag	Symi
light	Integer			

Many of the same fields you found in rooms and NPCs, as you can see from Table 7-1, can also be found in objects. The fields do not always have exactly the same use when coding rooms, NPCs, and objects but they are normally set in the same manor. It is very important that you read and understand the differences of each field as they pertains to rooms, objects, and or NPCs.

7.1. Description of object fields

symbolic name

The rules of the symbols has been explained in Chapter 4, if you didn't read them yet you may want to review. The important thing to realize with the object symbol is it is always good practice to give the object a symbol that resembles the title and description so administrators and builders can use the **load** and the **wstat** to easily locate, examine, and load the object in question.

title

The object title is what is shown if the object is being picked up, dropped, given to someone, when you do the inventory command, , or being used in combat. there should be no punctuation in the object title because of how it is used in the VME server. If you add punctuation or forget to capitalize something that the VMC thinks you should it will give you a warning when you compile. The following are good examples of an object title.

```
title "a big rock"
title "the flame tongue"
title "a lap top"
title "a garbage bag"
title "an oval hover car"
```

descr

The description field is what the player sees when walking into the room or when looking with no arguments. It is good practice to make this no longer than one line not counting the 'descr' tag.

Some examples of the object description field would be as follows:

```
descr
"a green bloody sword is laying here."

descr
"A massive wooden round table sits here."

descr
"a funny looking hammer is laying here."
```

names

The object names are as important as the NPC names. They are what you act on when picking the object up, dropping it, throwing it, just about anything you do to objects use these name fields. On drink containers you add the liquid name at the end, so people can drink the liquid. You always need to make sure you put every possible name that the player may use to examine or take your item. The rule of thumb is if it is in the title or description it should be in the names list. conversely if it is not

in the title or description it shouldn't be in the names list because the players will not use it if they don't know about it.

The following is some examples of good 'names' fields with respect to their 'title' and 'descr'.

```
title "a big rock"
descr "a big rock is here blocking the road."
names {"big rock","rock"}

title "an old twisted staff"
descr "An old twisted staff has been discarded here."
names{"old twisted staff","twisted staff","old staff","staff"}
```

The idea of course is to make any combination that a player may type to try and act upon your object. You would not want to describe and title your object with an entirely different theme than you created its names with because a player would not know what it is called.

inside_descr

The inside description is what a player sees if it is inside the object. This is used for things like Coffins or boxes or boats that a player can climb inside. The inside description is defined the same way the normal description is but you can make it as many lines as you want like you would with a room description.

```
                                        inside_descr
"You are inside a black coffin with a red velvet padding - scary!"

                                        inside_descr
"You are inside the pink time machine.  a small control panel is on the
floor and seems to be operated by stepping on it."
```

extra

The extra's on the object like the NPC, can be used to do many things. It can be used to store information for DIL programs or it can be used to show a part of the object like the room extras show a part of the room. They can even be used to create new acts when a person picks the item up, drops, or enters it.There is also a special extra that is the object's description when you look at it with the look <object> command.

Lets go over the object description extra first. If you use an extra with no names list it will become the object's description when you look at any of the names on it.

```
extra {}
"Its just a rock nothing special about it."

extra {}
"The ice cube is about 40 meters perfectly cubed.  It seems to be
melting slightly but waiting for it to finish would be sort of like
waiting for the ice age to end."
```

You can also use extras to show parts of the object.

```
extra {"crack"}
"There is a big crack in the side of the ice cube.  Maybe if you mess
with the crack you will be able to open it or something."

extra {"bed post","post"}
"Its a big gold bed post don't you wish you could get this sucker off it
would make you a rich adventurer indeed."
```

You can also use the extras to give more detailed and vivid descriptions when the object is acted upon.

Table 7-2. Object special action extras

Extra	Description
$wear_s	A message shown to activator when wearing (+wield/grab/hold) an item.
$wear_o	A message shown to others when wearing an item.
$rem_s	A message shown to activator when removing worn stuff.
$rem_o	A message shown to others when removing an item.
$get_s	A message shown to activator when getting an item.
$get_o	A message shown to others when getting an item.
$drop_s	A message shown to activator when dropping an item.
$drop_o	A message shown to other when dropping an object.
$enter_s	A message shown to activator when entering an item.
$enter_o	A message shown to other when entering an item.
$exit_s	A message shown to activator when leaving an item.

Extra	Description
$exit_o	A message shown to other when leaving an item.

In the following example of an ice cube, 1n is the activator and $2n is the unit in question.

```
extra {"$get_s"}
"You pick up the $2N, it is very cold and begins to melt in your hands."

extra {"$get_o"}
"$1n picks up the $2N, you notice that a drop of water hits the ground as
it begins to melt in $1s hand."
```

Finally you can use extras to store information for DIL programs. We will not cover this because it is a topic covered in-depth in the DIL documentation.

manipulate

This field is what defines the things that can be done to the object. For example a piece of armour should be able to be taken and worn, while a fountain should be able to be entered but not taken unless its some magical portable fountain. There are two sets of manipulate flags even though you can use them together. We separate them because the first two are flags that tell you if you can take or enter something while the rest of the manipulate flags are for worn positions.

First the two flags for taking and entering are:

Table 7-3. Take and enter flags

Manipulate	Description
MANIPULATE_TAKE	Set this flag if the unit can be taken (picked up/moved about).
MANIPULATE_ENTER	Set this flag if it is possible to enter a unit, ie set it in a coffin if you want players to be able to enter the coffin.

These flags are set to indicate on what body positions a particular object can be worn:

- MANIPULATE_WEAR_FINGER
- MANIPULATE_WEAR_NECK
- MANIPULATE_WEAR_BODY
- MANIPULATE_WEAR_HEAD
- MANIPULATE_WEAR_LEGS

- MANIPULATE_WEAR_FEET
- MANIPULATE_WEAR_HANDS
- MANIPULATE_WEAR_ARMS
- MANIPULATE_WEAR_SHIELD
- MANIPULATE_WEAR_ABOUT
- MANIPULATE_WEAR_WAIST
- MANIPULATE_WEAR_WRIST
- MANIPULATE_WIELD
- MANIPULATE_HOLD
- MANIPULATE_WEAR_EAR
- MANIPULATE_WEAR_BACK
- MANIPULATE_WEAR_CHEST
- MANIPULATE_WEAR_ANKLE

Currently you can only set one of the worn positions flags on an item at a time. You can set both enter and take on an item with a position or just one or the other. Some legal examples of combinations are as follows:

```
//An earring
manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_EAR}

//A backpack
manipulate {MANIPULATE_TAKE, MANIPULATE_ENTER, MANIPULATE_WEAR_BACK}

//strange true but its legal an earring pack
manipulate {MANIPULATE_TAKE, MANIPULATE_ENTER, MANIPULATE_WEAR_EAR}
```

flags

This field on an object is used to set special attributes in order to make the object able to be buried or not or no-teleportable and many others. The object flag list uses the UNIT_FL_* variables that both the NPCs and the rooms also use, therefore while you can set some flags on an object it may not have any affect unless you as a builder or administrator adds the functionality. You can also add extras on an object that can be used as a special flag which you will learn as you learn to use DIL. The following is a full list of all unit flags and how they affect objects, if they do.

Table 7-4. Object unit flag affects

Flag	Description
UNIT_FL_PRIVATE	Currently has no affect on a NPC.

Flag	Description
UNIT_FL_INVISIBLE	Makes unit invisible
UNIT_FL_NO_BURY	Makes it so you can create objects that can not be buried for example a weapon that for some reason shouldn't be buried.
UNIT_FL_BURIED	Makes unit buried when loaded
UNIT_FL_NO_TELEPORT	Makes it so you can not teleport into this object. This flag only works on containers.
UNIT_FL_NO_MOB	Currently has no affect on an object.
UNIT_FL_NO_WEATHER	Currently has no affect on a NPC.
UNIT_FL_INDOORS	Currently has no affect on an object.
UNIT_FL_TRANS	Makes unit transparent If the Unit is transparent you will be able to see any NPCs that it is carrying. For example if a canoe was carrying a familiar you would see that as you walked into the room. If this flag is not set and you are in a canoe you will not see outside the canoe and no one will see in.
UNIT_FL_NO_SAVE	Makes it so a PC can't save with unit
UNIT_FL_SACRED	Currently has no affect on an object.
UNIT_FL_MAGIC	This flag is used by spells to tell if the object is magic.

If you wanted to make an object that a player can carry around but can not save you would set the manipulate and flags as follows.

```
manipulate {MANIPULATE_TAKE}
flags {UNIT_FL_NO_SAVE}
```

type

This field is what you use to set the objects type. The type field is used when spells are cast or commands are executed on the object. You can add your own item types but they will not change the actions of base code commands. The following is the list of item types and what they mean when you set them. Some are not supported with the current code but you can add support for them if you like by making DIL commands, which is covered in another manual.

Table 7-5. Item types

Type	Description

Type	Description
ITEM_LIGHT	Items of this type can be lighted and extinguished.
ITEM_SCROLL	Items of this type can be read as a magical scroll.
ITEM_WAND	Items of this type can be used with the **use** command.
ITEM_STAFF	Items of this type can be used with the **tap** command as a magical staff
ITEM_WEAPON	Items of this type are used as weapons.
ITEM_FIREWEAPON	Currently not supported but could be used to classify a special type of weapon.
ITEM_MISSILE	Currently not supported but could be used to classify a special type of weapon.
ITEM_TREASURE	Items of this type are of some great value to sell but nothing else like a Gem or a block of gold.
ITEM_ARMOR	Items of this type can be worn or used as armour.
ITEM_POTION	Items of this type can be used with the **quaff** as a position.
ITEM_WORN	Items of this type can be worn but not normally used for armour it is more for clothing.
ITEM_OTHER	This item type is for items that don't fit any other type. Now that you can make your own commands with the VME 2.0 you should just make your own item type instead of using this value.
ITEM_TRASH	Items of this type are usually junk or broken equipment.
ITEM_TRAP	Not currently supported but could be used to make a trap command by creating a trap item
ITEM_CONTAINER	Items that can be used as containers.
ITEM_NOTE	Items of this type can be used to write on like paper or slates.
ITEM_DRINKCON	Items of this type can carry liquids.
ITEM_KEY	Items of this type can be used as a key.
ITEM_FOOD	Items of this type can be eaten
ITEM_MONEY	Items of this type can be spent as currency
ITEM_PEN	No longer supported but could be used to force people to have a writing instrument before writing a message.
ITEM_BOAT	Items of this type can be used as a water craft

Type	Description
ITEM_SPELL	Not currently supported but it could be used to make a page in a spell book
ITEM_BOOK	Not currently supported but could be used to make regular and spell books.
ITEM_SHIELD	Items of this type can be used as a shield.
ITEM_SKIN	Not currently supported in the release but could be used to make the skin command and create skins of animals
ITEM_BOARD	Items of this type are used for public communications in the form of boards that can be read from and written to.

Unlike flags and manipulate fields only one item type can be set on an object at a time. The format for the 'type' field is simply the keyword followed by the value as follows:

```
type ITEM_BOARD
```

weight

The weight is the weight of the object in pounds. In the future we may adjust this to allow you to make things lighter for example you could set it in ounces or grams. Right now however all we have is pounds so we have some pretty heavy feathers out there.

To use this you just enter the 'weight' keyword and then the value.

```
/80 lbs.
weight 80
```

capacity

This field sets the size of a container object. If the object does not have the manipulate enter flag set then this field doesn't have to be set. The capacity is currently by pounds since the weight of objects is set in pounds. In the future we may take into account size and weight but right now it goes only by weight. The following line of code would set an item to carry 600 pounds of stuff.

```
capacity 600
```

key

The key field sets the key name of the key that will open the item. This field should be set to the symbolic name of the key that opens the item it is on. If the item is in the same zone as the key then you do not need to put the zone extension on the key name. The following are the three possible examples of using the key field.

```
//if object and key are in same zone.
key brasskey

//if key and object are in same zone
key brasskey@zonename

//if key and object are not in same zone
key brasskey@otherzonename
```

Notice you can put the zone name on it if the key is in the same zone but if the key is not in the same zone you must put the zone name on it.

cost

This is the field you set to add a cost to your object. If you leave this field out it will default to no cost and will not be able to be sold at stores. The system for setting cost on an item is the same as setting money on a NPC. As with a NPC we could set it using a single number but it would not be easy to understand. For example 5 gold pieces would be something like:

```
money 25600
```

I am no more sure this will make five gold pieces than I was when I used this same example with the money field in NPC. The problem is I just did the math in my head so its not very accurate. It is much easier to use the defined money types to set exactly what you want as follows:

```
IRON_PIECE
COPPER_PIECE
SILVER_PIECE
GOLD_PIECE
PLATINUM_PIECE
```

Now if we wanted to make an object costing five gold it would be as simple as this:

```
money 5*GOLD_PIECE
```

the define method also gains you the ability to tell the VME what amount of each coin you want on the NPC. If you set it using a single integer the compiler would pick how many of each coin. This of course is not what is desired in fact you want to be able to set your cost however you like. So setting more than one coin is as simple as adding a comma between the first and second coin.

```
money 5*GOLD_PIECE, 20*IRON_PIECE
```

rent

This field tells how much it costs you to keep an item while your offline. The rent is not always taken if the VME server is set up to not take any rent then it will not matter if you set this or not. Also the VME can be set up to take a percentage of this field so it may not take the exact amount you et. If the VME server is set up to take 100% of the rent then what you set will be taken. To set this field you do the same as you do with the cost field.

```
money 5*GOLD_PIECE, 20*IRON_PIECE
```

minv

This field is the administrator invisible level of the object it is set on. This means that if you set the 'minv' to two hundred it will make it so the object can not be seen by anyone below the administrator level of two hundred. This is good for hiding objects that you need for administrators but you don't want players to see. In order for the 'minv' to be removed an administrator or a DIL function must change it.

```
minv 239
```

alignment

The object alignment is not currently used. It is an integer value that can be set on an object to be used with any DIL functions. In the future it will be what determines if a good or evil person can wield an item. The value is set by placing the 'alignment' keyword first followed by the alignment desired from -1000 to +1000.

```
alignment -250
```

open

The open field is used if you want to give your object the ability to be opened, closed, and or locked. If you add the open flags you need to also add a key field which has already been explained. The following are all the possible open flags and what they are used for.

EX_OPEN_CLOSE

Set this if you can open and close this object.

EX_CLOSED

Set this if you want the object to be closed when loaded.

EX_LOCKED

Set this if you want the object to be locked when loaded.

> **Note:** An interesting aspect is that if you do not specify a key, you can only unlock this door with the 'pick' skill, 'unlock' spell or from DIL with UnSet();

EX_PICK_PROOF

Using this flag renders the 'pick' skill and 'unlock' spell un useable on the lock of this object.

EX_INSIDE_OPEN

Usable on container objects only, this enables the mobile to 'open' and 'lock' from the inside.

The simplest use of this field is to make an object that opens and closes. A coffin for example would have its flags set as follows:

```
open {EX_OPEN_CLOSE}
```

If you wanted to set an object that is locked and closed and having a brass key that can open it, when it is loaded. It would look as follows.

```
open {EX_OPEN_CLOSE, EX_CLOSED, EX_LOCKED}
key brass_key
```

You would have to define the key in the object section as well and the symbolic name for that key would be 'brass_key'

spell

The spell field is the power of the objects defense against spells. You can set it from zero which is just not setting the field all the way to 200% which means a person who has 100% in a spell will fail almost all the time. To set this field it would look as follows:

```
//Spell resistance at 150%
spell 150
```

value

The object values are used for just about any special item from armour to drink containers. They should not be set directly unless you have a reason to do so, like a special DIL command that checks a value on an item. You also have to be carefull not to over write what a value is already used for example value one is already used on weapons and armours for craftsman ship that will be explained later in Section 7.2.

If you find you need to set the values there are a total of five of them and they can be set to any integer value as follows:

```
value[0]   5
value[1] 16
value[2] -2
value[3] -10
value[4] 12
```

affect

The affect field should not be set directly, instead you should use the macros defined in Section 7.2.

dilbegin or dilcopy

As has been mentioned in previous sections the DIL functions are what give VME servers the edge over all other muds. We will only give some examples here and leave it up to the DIL manual to teach you how to create your own functions that will make your rooms, NPC, and objects more than special.

There are several object functions that come standard with the VME 2.0. The following is a list of those functions.

- Guild restrict
- Anti-guild restrict
- Quest restrict
- Quests restrict
- Alignment restrict
- Level restrict
- Virtual level restrict
- Race restrict
- Ability restrict
- Skill restrict
- Spell restrict
- Weapon restrict
- Gender restrict
- Player restrict
- boards
- tuborg/dilbegin

These are the only object functions currently documented in the VME 2.0 release but if you go through the zones that are released with the VME you are sure to find many more. Hopefully with the descriptions in Section 7.5. You will be able to use the functions listed here and figure out ones that are not.

Since these are just DIL's written by builders for the Valhalla mud all you have to do is use the dilcopy keyword in the NPC with the function name you want to use and the arguments that function requires. The following is what you would find in the `function.zon` for tuborgs.

```
dilbegin tuborg(s:string);
external
 sub_drink_info@commands(d:unitptr);
var
  u : unitptr;
code
{
  :start:
  wait(SFB_CMD, ( (command("drink")) or
                  (command("sip")) or
           (command("taste")) ) );
 u := activator;
 secure (u,start);
if (findunit (activator,argument,FIND_UNIT_INVEN|FIND_UNIT_SURRO,null)!=self)
```

```
 goto start;
 if ( command("sip") or command("taste") )
 {
   block;
   act("$1n tastes $2n enjoying every drop.", A_HIDEINV, u, self, null,
        TO_ROOM);
    act("The taste of the $2N is nothing less than divine.", A_HIDEINV, u, self,
      null, TO_CHAR);
   goto start;
   }

  if ( u.thirst >20 )
  {
    block;
   act("Your not thirsty.", A_HIDEINV, u, null, null, TO_CHAR);
     goto start;
  }
block;
   act ("You drink $2n and it makes you feel more energetic!", A_HIDEINV, u, self,
       null, TO_CHAR);
   act ("$1n drinks $2n and looks more energetic!", A_HIDEINV, u, self,
       null, TO_ROOM);

  u.thirst := u.thirst + 10;
  u.full := u.full + 10;
  if (u.thirst > 24)
  {
    u.thirst := 24;
  }

  if (u.full > 24)
  {
    u.full := 24;
  }
  u.endurance := u.endurance+50;
  if (u.endurance > u.max_endurance)
  {
    u.endurance := u.max_endurance;
  }
  sub_drink_info@commands(self);
  quit;
}
dilend
```

If this DIL function scares you don't worry you don't have to understand it or adjust it you only have to use it. In fact this is a really easy DIL to use. The argument on the tuborg function is not used yet so all you have to do is pass in a blank string or any string for that matter. So if you wanted to make a tuborg in the game you would just add this to your drink container.

```
dilcopy tuborg@function ("");
```

All of the above listed DIL object functions are described in Section 7.5. Then we put some to work so you can see how to use them in Section 7.6

7.2. Object macros

To make the creation of some objects easier we have provided a set of Macros. The macros range from general armour and weapons macros to macros that help you create special affects on all items. We will first cover what craftsmanship and magical modifiers are in Section 7.2.1 and Section 7.2.2 respectively. After which we will show the use of craftsmanship and magical modifiers in Section 7.2.3 and Section 7.2.4.

7.2.1. Weapon and armour craftsmanship

The craftsmanship is a way of expressing the overall quality of a piece of armour or weapon. The quality on the VME servers currently means the amount of hit points given to an item. The craftsmanship ranges from 25 to -25 and the hit points range from 125 to 6000. The craftsmanship can be looked at as how tough or good the armour or weapon is. The following table should help you in deciding how tough your armour or weapon should be.

Table 7-6. Approximate hit points verses craftsmanship

Craftsmanship	Hit points
25	6000
20	5000
15	4000
10	3000
5	2000
0	1000
-5	875
-10	650
-15	425
-20	300
-25	125

It is suggested the higher the craftsmanship the higher the cost of the weapon should be. This is not a must but it goes with out saying the quality of an item should be represented in the cost of it. Of corse there is the time you would want to sell your players poor quality items at a high cost just to make them think they are getting something cool.

7.2.2. Magical modifier

The magical modifier can be said to modify damage done to an opponent. In a combat the damage is calculated and then the magical bonuses on armour or weapons is added in. This is best explained by an example.

Lets say that you were about to give 25 hit points of damage to a person. Your sword has a plus 25% in magical bonus. The bonus is added to your damage to make it a total of 50 hit points of damage. The player you are hitting however has a +25% magical bonus on his armour that you are about to hit him on. That will reduce the damage back to its 25% hit points originally done. This is just a nice way to add a bit of damage for a special weapon.

The magical modifier ranges from 25 to -25. It affects both the damage being given to a player and the damage being given to a weapon or a piece of armour.

It is suggested that you modify the costs of your objects to fit the amount of magical bonus along with adding the magical flag tot he objects flag list so an identify spell can pick up that there is magic about the object. This is not a must but your players will love you for it.

7.2.3. Setting weapon fields

To create a weapon you only need three pieces of information. The weapons craftsmanship and magical modifiers defined in Section 7.2.1 and Section 7.2.2 and the weapon type. You have seen the weapon types before when defining a NPCs natural attack type in Section 6.2.1. The full list of weapon types that are released with the VME 2.0 can be found in Appendix D. With craftsmanship, magical modifier and the weapon type all you need to do is pick from one of the following macros and insert your numbers.

```
#define WEAPONSZ_DEF(weapon_category, craftsmanship, magic_bonus, hgt)  \
  WEAPON_DEF(weapon_category, craftsmanship, magic_bonus)\
  height hgt

#define SHIELD_DEF(shield_type, craftsmanship, magic_bonus)  \
  manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_SHIELD} \
  type ITEM_SHIELD            \
  value[0] shield_type        \
  value[1] craftsmanship      \
  value[2] magic_bonus
```

As you can see the first macro uses the second macro so the only difference between them is the first one sets the height field. Using the first macro will force your weapon to be a certain size when loaded. While not setting the height field by using the second macro would let the VME server set the size of the weapon by what NPC it is loaded on.

A flail (two handed) of non-pure iron (-3%), a little better than average craftsmanship (5%) and no magic bonuses would have:

```
WEAPON_DEF(WPN_FLAIL, +2, 0)
```

A rusty (-5%) mean sacrificial dagger by a skilled smithy (+5%) and magically enchanted might be:

```
flags {UNIT_FL_MAGIC}
WEAPON_DEF(WPN_DAGGER, 0, +5)
```

An old shaky wooden stick made for a 400 cm tall person could be:

```
WEAPONSZ_DEF(WPN_CLUB, -5, 0,400)
```

A wooden bastard sword would have considerable less craftsmanship than listed since wood prevents the slashing effect, also it would be non-sense to apply better than average craftsmanship in this case.

```
WEAPON_DEF(WPN_BROAD_SWORD, -15, 0)
```

7.2.4. Setting armour fields

When designing armour it is no more difficult then when designing weapons. There is five main armour types. The types don't define the material type for example if you wanted to create a wooden pair of armour that protected like plate armour you could do this by defining the armour type as plate and then adding the material as defined in Section 7.2.6. The five armour types are as follows:

- Clothes
- Leather
- Hard leather
- Chain
- Plate

The armours macros are almost the same as the weapons macro it looks as follows.

```
#define ARMOUR_DEF(atype, craftsmanship, magic_bonus) \
   manipulate {MANIPULATE_TAKE} \
   type ITEM_ARMOR              \
```

```
   value[0] atype
      value[1] craftsmanship       \
   value[2] magic_bonus

#define ARMOURSZ_DEF(atype, craftsmanship, magic_bonus, hgt)\
   ARMOUR_DEF(atype,craftsmanship, magic_bonus) \
   height hgt
```

The craftsmanship and magical modifier fields have already been explained so the only thing new that you need to pass into these macros is the 'atype' which stands for armour type. As we have mentioned there are five different armour types. The following are the defines for each:

- ARMOUR_CLOTHES
- ARMOUR_LEATHER
- ARMOUR_HLEATHER
- ARMOUR_CHAIN
- ARMOUR_PLATE

The armour type defines how different weapons and spells are defended against for example plate would be better against acid maybe and worse against electricity. You as a VME administrator will have to decide which armours are better at what by changing your `weapons.def` and `spells.def`

This explains the entire armour define but there is some more to it. The rest will be covered in Section 7.6.6 For now an example use of the armour define would be as follows:

```
flags {UNIT_FL_MAGIC}
ARMOR_DEF(ARM_PLATE,+15,+5)
```

7.2.5. Setting shield fields

We have tried to keep the interface of making armours, weapons, and shields the same. If you have already looked through the defines for weapons and armours you will find that there is very little difference here. The following is the define for the macro that sets the shield values.

```
#define SHIELD_DEF(shield_type, craftsmanship, magic_bonus)  \
   manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_SHIELD} \
   type ITEM_SHIELD            \
   value[0] shield_type        \
   value[1] craftsmanship      \
   value[2] magic_bonus

#define SHIELDSZ_DEF(shield_type, craftsmanship, magic_bonus, hgt)  \
   SHIELD_DEF(shield_type, craftsmanship, magic_bonus)\
```

```
height hgt
```

You have already seen the craftsmanship and magical modifiers in Section 7.2.1 and Section 7.2.2, so the only thing different here is the shield type. There are three shield types available in the current combat system and they are categorized by size. the three sizes are small, medium, and large. to set the type you use the defines from `vme.h` which define the following:

- SHIELD_SMALL
- SHIELD_MEDIUM
- SHIELD_LARGE

the larger the shield the better chance of blocking an attack. You may want to remember to add weight as you add size to your shield so players are weighted down and can not carry the best of everything but that is up to the administrator of the VME server.

A small magical wooden shield could be assigned:

```
flags {UNIT_FL_MAGIC}
SHIELD_DEF(SHIELD_SMALL, 0, +5)
```

7.2.6. Setting material types

Currently Material types are not used greatly in spells or skills but in the future we hope to add more functionality for materials. For example in the future if you are hit by an acid spell we want your armour to be damaged depending on the material it is. The material doesn't have any affect on damage given or taken it is just a way you can check in DIL what the weapon is made out of. The following is the list you would find in `wmacros.h` of in the VME 2.0 release.

```
#define MATERIAL_WOOD(DESCR)    extra {"$material", "$mat_wood"} DESCR
#define MATERIAL_METAL(DESCR)   extra {"$material", "$mat_metal"} DESCR
#define MATERIAL_STONE(DESCR)   extra {"$material", "$mat_stone"} DESCR
#define MATERIAL_CLOTH(DESCR)   extra {"$material", "$mat_cloth"} DESCR
#define MATERIAL_LEATHER(DESCR) extra {"$material", "$mat_leather"} DESCR
#define MATERIAL_SKIN(DESCR) extra {"$material", "$mat_skin"} DESCR
#define MATERIAL_ORGANIC(DESCR) extra {"$material", "$mat_organic"} DESCR
#define MATERIAL_GLASS(DESCR)   extra {"$material", "$mat_glass"} DESCR
#define MATERIAL_FIRE(DESCR)    extra {"$material", "$mat_fire"} DESCR
#define MATERIAL_WATER(DESCR)   extra {"$material", "$mat_water"} DESCR
#define MATERIAL_EARTH(DESCR)   extra {"$material", "$mat_earth"} DESCR
#define MATERIAL_MAGIC(DESCR)   extra {"$material", "$mat_magic"} DESCR
```

Therefore if you had a wooden staff you could add the following to your weapon so spells would know what it was made out of.

```
MATERIAL_WOOD("a hard oak")
```

7.2.7. Drink container macros

There are two different kinds of macros for drink containers. The one you use depends on the need at the time. The harder macro is made so you can create a drink of any kind. If however you want normal drinks like water, beer, or even lemonade there are more simple macros already defined for you to use in `liquid.h`. The following is a couple of the macros from the `liquid.h` for a full listing see Appendix E.

```
#define LIQ_WATER(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("clear", WEIGHT,CAPACITY,INSIDE,10,1,0,POISON)
#define LIQ_BEER(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("brown", WEIGHT,CAPACITY,INSIDE,5,2,3,POISON)
#define LIQ_WINE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("red", WEIGHT,CAPACITY,INSIDE,5,2,5,POISON)
#define LIQ_COFFEE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("black", WEIGHT, CAPACITY, INSIDE, 6, 1, 0,POISON)
```

To use these macros the arguments are pretty simple.

weight

The first argument just says how heavy your drink container is when empty. Like a barrel might be 15 pounds and a glass might be 1 pound. You may be thinking there is no way the glasses in your kitchen are one pound. The truth is if we had less than a pound then we would set the glass to less but currently all units are measured in pounds so the least we can make it is a pound. In the future of the VME we will be converting to smaller measurements like grams or ounces.

capacity

The second argument is what your container can carry weight wise. So if your barrel is 15 pounds and your barrel can carry 35 pounds of a liquid then the total weight when full would be 50 pounds, if my math is working today. To make a container with infinity liquid like a fountain you just set capacity to '-1'.

> **Note:** Making capacities ridiculously large can cause weight bugs. If your going to allow ridiculous amounts you might just want to give them the infinite amount or really work on your drink and pour functions

inside

The third argument is how much liquid by weight your container has inside. This value should not be greater than the capacity but if you mess up it will be fixed when the player tries to drink from it. The total weight of the drink container and the liquid it contains will be the weight added to this value.

poison

the forth argument is the amount of poison in your drink. There is no limit on the amount of poison but understand that a value of ten would be a very high poison value.

So if you wanted to make a simple small glass of water you would use the water macro and it would look like this:

```
LIQ_WATER(1,2,2,0)
```

You are probably wondering what it takes to fill a player from empty. The players thirst ranges from -24 which is dieing of thirst all the way to positive 24 which is full. So if you have a barrel that can hold 24 capacity one full barrel can take a person from zero thirst to full.

Now lets say you want to make something more exotic. All the normal drinks are made or at least a great number of them are in the `liquid.h` but what if you had a race from outer space that drank nothing but silicone oil. This obviously is not covered in our liquid file so you would have to make one yourself or use the more complex macro.

```
#define LIQ_DEF(color, wgt, max_cap, inside,thirst,full,drunk,poison) \
  type ITEM_DRINKCON           \
  weight (wgt)+(inside) \
  capacity max_cap \
  value[0] inside        \
  value[3] poison \
  extra {"$drink_color"}color \
  extra {"$drink_thirst"} #thirst \
  extra {"$drink_full"} #full \
  extra {"$drink_drunk"} #drunk
```

As you can see this define has much more information you need to pass it but it really is not that hard. The following are the arguments and what they do.

color

the first argument is the color of the liquid. this color will be shown when you look at the liquid in the container.

wgt

The second argument is the weight of the container as in the last macros. It is what the container would weigh empty.

max_cap

The third argument is the maximum capacity of the container. If the value is set to 15 and the container is fulled it will contain 15 pounds of liquid which adds to the base weight to get the total weight of the container. If you want a container to have unlimited contents then you set the capacity to '-1' and the weight will be that of the 'wgt' field.

inside

The forth argument is the amount of liquid the container starts with. This amount should not be greater than the 'max_cap' field, but if it is it will be corrected when the player takes a drink or acts on the container.

thirst

The fifth field is how much thirst this gives per pound of liquid consumed. For example if you have a glass of water and it only has a capacity of 1 with 1 inside. this value will be added once to the players thirst field.

this can be a bit confusing so lets first explain that the thirst field can be anything from 0 to 10 or even greater but we suggest only 10 max. With that in mind know that we set water at 10 because it is one of the best thirst quenchers known to man. Therefore a glass with 1 capacity and 1 quantity inside will give a player +10 to his thirst so if the player was down to zero thirst value one drink will give them 10. remember that a players thirst ranges from -24 to +24 so with three drinks of water a person could fill his thirst need entirely.

With that in mind when setting this field you have to think what kind of thirst quencher is my drink. If for example it is vodka it would have little to know thirst quenching power so you would set this field to 0 or 1.

full

The sixth field like the thirst field sets how the drink will affect the chars fullness. The chars fullness field is normally set when a player is eating but as you know drinking some drinks will also give you the feeling of being full. One drink like this would be milk. The chars fullness field ranges from -24 to 24 like the thirst field and the argument you are setting on this field should range from 0 to 10 unless you have a pro teen drink that fills them like food. Milk might have a fullness of something like 5.

drunk

the seventh field like thirst and full deals with the Drunkenness of a character. A character can range from 0 (not drunk) all the way up to 24 (smashed). The drunk field on this macro sets how much drunk is added for each quantity of the liquid is consumed. Therefore something like vodka should

have a value of 10 while something like water should be down at 0 unless you have some weird race that gets drunk from water.

poison

The eighth and final field is again like the last macro we looked at it sets the poison factor of a liquid. The value ranges from 0 (no poison) to whatever you want but understand that 10 is an extreme poison factor and a player drinking this will most likely die quickly.

So with the definitions of each arguments in mind lets return to the example of making a silicone oil based liquid. We will first show what it would look like using the hard macro then what the new easy macro that you could create would look like.

```
 LIQ_DEF("blue",5,10,10,5,1,0,0)
```

Now if you want to make this a liquid your going to use a lot you would define your own easier macro like this.

```
#define LIQ_SILICONE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("blue", WEIGHT,CAPACITY,INSIDE,5,1,0,POISON)
```

That covers the use of the macros but for more information on the drink containers see Section 7.6.3.

7.2.8. The food macros

The food macro is much easier than that of the drink macro so if you have drink containers down you will have no problem making food to go with your beverage. The players fullness value ranges from -24 to 24 as we have already learned when making drink containers and in the current food system on the VME server we do not allow thirst to be modified by food. Therefore the only thing you have to set is the amount of fullness and the poison factor if there is any. The following is what the define for food looks like.

```
#define FOOD_DEF(food_amount, poison_factor)  \
   type ITEM_FOOD         \
   value[0] food_amount   \
   value[3] poison_factor
```

Therefore if you wanted to make sure that only one of your foods that you were creating would fill a player entirely in one bite you would set it like this:

```
FOOD_DEF(50,0)
```

It is recommended that you only set the value between 1 and 10 so that players have to eat a bit as if they were eating in the real world.

7.2.9. Light object macro

The light macro is very simply to use it only has two values duration and brightness of light. The macro is defined as follows:

```
#define LIGHT_DEF(hours, how_bright)   \
   type ITEM_LIGHT     \
   value[0] hours      \
   value[1] how_bright
```

The first argument is the duration in mud hours which is about 5 minutes per mud hour. The second argument is how bright the object is 0 would be stupid cause it would give off no light but you never know maybe you want to do something like that. One, two, and three would be small torch, large torch, and lantern respectively. You could set a brightness greater than 3 but you should be carefull not to over light your characters or you may cause light bugs.

7.2.10. Container macro

The container macro is a simple macro that just sets two fields. The only information you have to give it is the capacity of the container. Remember that capacity of an item is in weight not size. Therefore if some ones corpse weighs 230 pounds you will need a container that has a capacity of 230 to fit the corpse in it. The following is the macros definition as found in `wmacros.h`:

```
#define CONTAINER_DEF(max_capacity)  \
   type ITEM_CONTAINER               \
   capacity max_capacity
```

If you wanted to create a coffin that could carry any normal human corpse you could set it something like this:

```
container_def(300)
```

7.2.11. Money macro

Money is one of the simplest objects you can make on the VME server. With this macro all you need is the symbolic before the macro and the end keyword after the macro and you have 1 piece of money or a whole pile. The macro is defined in the `wmacros.h` and looks exactly as follows:

```
#define MONEY(coin_type, coins) \
```

```
  type ITEM_MONEY \
  manipulate MANIPULATE_TAKE \
  title coin_type \
  value[0] coins
```

The arguments are simple the first argument is the type of money the five possible values are:

- IRON_PIECE
- COPPER_PIECE
- SILVER_PIECE
- GOLD_PIECE
- PLATINUM_PIECE

The second argument is the amount of coins. If you set it to zero then it will still make exactly 1 of the coins. The following would be what one platinum piece would be like in a zone file.

```
platinum_piece

MONEY(PLATINUM_PIECE, 0)
/* Rest of values are inserted at runtime */

end
```

Now if you want to make a whole pile of money it would look like this:

```
platinum_pile

MONEY(PLATINUM_PIECE, 80)
/* Rest of values are inserted at runtime */
extra {}
"Holy cow thats a stash."

end
```

7.2.12. Cursed objects macro

Sometimes when making special objects you want to make an item that a person can wear but can't remove. With the cursed object macro this is a simple thing. The cursed object macro is defined in `wmacros.h` and looks as follows:

```
#define CURSED_OBJECT \
affect \
```

```
   id ID_CURSE \
   duration -1 \
   firstf TIF_NONE \
   tickf  TIF_NONE \
   lastf  TIF_NONE \
   applyf APF_MOD_OBJ_FLAGS \
   data[0] OBJ_NO_UNEQUIP;
```

to use this macro it is simply a matter of putting the define in your object like this:

```
CURSED_OBJECT
```

When you set this macro on an object it adds an affect that can only be removed by the 'set' command or by the 'remove curse spell'.

7.2.13. Potion, wand, and staff macros

The macros for potions, scrolls, wands, and staffs are almost the same. In fact there is only two differences. The first is the potions and scrolls can cast three spells while wands can only cast two, The second is wands and staffs have multiple charges possible while scrolls and potions only can be used once. The following are the macros for all four as found in `wmacros.h`.

```
#define POTION_DEF(power,spell1,spell2,spell3)  \
  manipulate {MANIPULATE_TAKE, MANIPULATE_HOLD} \
  flags {UNIT_FL_MAGIC} \
  spell power      \
  type ITEM_POTION \
  value[0] power   \
  value[1] spell1  \
  value[2] spell2  \
  value[3] spell3

#define SCROLL_DEF(power,spell1,spell2,spell3)  \
  manipulate {MANIPULATE_TAKE, MANIPULATE_HOLD} \
  flags {UNIT_FL_MAGIC} \
  spell power      \
  type ITEM_SCROLL \
  value[0] power   \
  value[1] spell1  \
  value[2] spell2  \
  value[3] spell3

#define WAND_DEF(power,charge,spell1,spell2)  \
  manipulate {MANIPULATE_TAKE, MANIPULATE_HOLD} \
  flags {UNIT_FL_MAGIC} \
  spell power      \
  type ITEM_WAND \
```

```
  value[0] power   \
  value[1] charge  \
  value[2] spell1  \
  value[3] spell2  \
  value[4] charge  /* The max charge */

#define STAFF_DEF(power,charge,spell1,spell2)  \
  manipulate {MANIPULATE_TAKE, MANIPULATE_HOLD} \
  flags {UNIT_FL_MAGIC} \
  spell power      \
  type ITEM_STAFF  \
  value[0] power   \
  value[1] charge  \
  value[2] spell1  \
  value[3] spell2  \
  value[4] charge  /* The max charge */
```

The arguments are as follows for the macros.

power

The first argument on potions, scrolls, wands, and staff is the power the spell will be cast at. You can have the power set in the range 1-200. The spell power works the same as a player training in the spell. The higher the number the more powerfull the cast.

charge

The second argument on the staffs and wands is how many charges it has.

spell#

On potions and scrolls you can set up to 2 spells you do not have to set them both the one you don't want set to 0. On staffs and wands you can set three spells. Again if you want only one or two you can leave the one you do not want set to 0.

7.2.14. Magical transfer macros

There are times when you want to give a player a bonus in a ability, weapon, skill, and or weapon. There is even times when you want to adjust a characters speed or add a flag to a player when they wear an item. The following macros are what you would use to do any of those when a person uses an item.

- CHAR_FLAG_TRANSFER(_MFLAGS)
- SKILL_TRANSFER(skill, amount)
- WEAPON_TRANSFER(weapon, amount)
- SPELL_TRANSFER(spell, amount)
- STR_TRANSFER(amount)

- DEX_TRANSFER(amount)
- CON_TRANSFER(amount)
- CHA_TRANSFER(amount)
- BRA_TRANSFER(amount)
- MAG_TRANSFER(amount)
- DIV_TRANSFER(amount)
- HIT_TRANSFER(amount)
- SPEED_TRANSFER(newspeed)
- SLOW_TRANSFER(amount)

Note: For the full definitions of the transfer macros see Appendix F or the header file `wmacros.h`.

The transfer macros can be broken down into three groups those which transfer percentage, flags, and speed. The skill, weapons, spells, and ability macros transfer the amount of percentage you put. If you give a negative percentage it will take that much away from the player or NPC in that catagory. The character flag transfer actually adds the flag to the player while the player is using the item. The speed transfer macros add or subtract the amount of speed you give them. The range for speed is from zero to twelve with twelve being the slowest.

7.3. Building your first object

Now that you have learned how to make rooms and NPCs its time to make the objects for your little world. In the last couple of sections you have looked through the fields. In this section we are going to make a nice easy object. There is really not that much new from what you have learned with rooms and NPCs so this should be a real quick section. As always we will start with something I like which as you remember is dragons. So the first object we will make is a dragon head. I didn't say I liked them alive now did I? Anyway this will be a nice simple object that your player can pick up and carry around.

When making objects you create the zone source file first as shown in Chapter 3. If you only have objects you do not need the %reset, %mobiles, and %rooms fields. For the examples in this chapter we will use the zone we created in Chapter 5 and add the %objects section where we will put all the object definitions. At the end of this chapter, in Section 7.7, we will bring it all together with the rooms and NPCs we have defined already.

The first part of all object definitions is the symbolic name it is good to always pick a name that will match the name of the object so it will be easy to load the object. The reason the symbolic and name should match is when you use the command **wstat** it will only show you a list of the objects by symbolic name for example if you type **wstat zone dragon objects** You will get the following:

```
List of objects in zone Dragon:
claw info_board dragon_head
```

If you didn't make it clear what the object was by the symbolic name it might look like this:

```
List of objects in zone Dragon:
obj1 a_obj2 o3
```

While this might be great when you first start imagine trying to remember each object if you have over 30 of them.

Now lets get started with our dragon head. As with the rooms and npcs all that is required to make an object is the symbolic and end fields. That of course will make a NPC with all defaults.

```
dragon_head
end
```

Thats it for that dragon head right? Nope not quite, like before with NPCs, that makes an object with all defaults. That means this will probably be a very blank spot on the screen with no names and no way your players can interact with it. Now lets start putting the Dragon heads other more interesting fields on.

Like with rooms and NPCs, the first three things we need are the dragon heads title, description and names. The description should be what you see when you do a 'look' in the room. The title should be what you see when the object is in your inventory or you are whacking someone over the head with it. Since we are not making a weapon though the title is what will be shown when you are picking up or dropping the object. Finally the names should cover everything in the title and description fields so if your player wants to pick the object up or wear it will be easy to figure out what the names are.

```
dragon_head

title "a gold dragon head"

descr "A large golden dragon head is laying here looking sad."

names {"large golden dragon head","large gold dragon head",
       "golden dragon head","large dragon head","gold dragon head",
       "dragon head","large head", "sad head","head"}
...
end
```

The names, title and description shouldn't be to hard so I don't think its necessary to go into any more description on the subject. Lets move on. Now we have to take care of what a player sees when he or she looks at an object. to make the main description of an NPC you place an extra on the NPC with no names in the list. The blank extra is a special extra that will be shown every time you look at anything in the names list of the object. So a description of an object would look something like this.

```
extra {}
"The head is large and beautiful, at least as beautiful as a dead
```

```
dragon head can be.  There is an extreme look of sorrow on the dragons
face and it seems to be for much more than its own death."
```

Now that you have a main description for the object you need to make any smaller descriptions that you want the player to be able to look at. In this case it may be good to give some secret information if the player looks at the face of the head directly.

```
extra {"gold dragon head face","dragon head face","head face","face"}
"Looking into the dragons face your eyes are drawn to the eyes of the
dead dragon.  Could there be something there?"

extra {"eyes","eye"}
"A world of blue skies and no storms is visible through the eyes and it
seems to be moving as if you were watching the world from space."
```

Now that we have the object all described we only need to give the object the manipulate flags it needs, weight, height, and maybe some extras that will make some cool acts when a player picks it up or drops it.

First thing to do though is pick the manipulate flags you want on the object. This is not a weapon or armour so all the player really needs to be able to do with it is pick it up and maybe hold it if you want and I do. The flags would then be as follows:

```
manipulate {MANIPULATE_TAKE,MANIPULATE_HOLD}
```

If you were feeling a little weird you could even make the person be able to wear the dragon head on his head but that would just be strange. of course its always good to know you have options.

Now lets set the height and weight. Remember you set the height in centimeters and the weight in pounds. In the future the VME will standardize to one or the other but for now we have to play the conversion game.

```
//20 feet  (1 inch = 2.54 cm
height 33

//566 KG (1 lb. = .45359 kg)
  weight 50
```

The final touch to our little dragon head is some cute acts when the player picks it up or drops it. If you remember from the extra fields in Section 7.1, there are some special extras that are made just for this purpose.

```
extra {"$get_s"}
```

```
"You suddenly feel very sad for a world that you don't even know."

extra {"$get_o"}
"A strange look of sadness crosses $1ns face."

extra {"$drop_s"}
"You feel much happier but you remember a feeling of great sorrow."

extra {"drop_o"}
"$1n seems to cheer up a bit."
```

There are other things we could add to this item but I want to keep this first object simple. The finished head would then look like this:

```
dragon_head

title "a gold dragon head"

descr "A large golden dragon head is laying here looking sad."

names {"large golden dragon head","large gold dragon head",
       "golden dragon head","large dragon head","gold dragon head",
       "dragon head","large head", "sad head","head"}

    extra {}
"The head is large and beautiful, at least as beautiful as a dead
dragon head can be.  There is an extreme look of sorrow on the dragons
face and it seems to be for much more than its own death."


extra {"gold dragon head face","dragon head face","head face","face"}
"Looking into the dragons face your eyes are drawn to the eyes of the
dead dragon.  Could there be something there?"

extra {"eyes","eye"}
"A world of blue skies and no storms is visible through the eyes and it
seems to be moving as if you were watching the world from space."

manipulate {MANIPULATE_TAKE,MANIPULATE_HOLD}

height 33
  weight 50

extra {"$get_s"}
"You suddenly feel very sad for a world that you don't even know."

extra {"$get_o"}
"A strange look of sadness crosses $1ns face."

extra {"$drop_s"}
"You feel much happier but you remember a feeling of great sorrow."
```

```
extra {"drop_o"}
"$1n seems to cheer up a bit."

end
```

Thats all there is to making regular items. The rest is just adding functionality to what you already know. We will get much deeper into what you can do with items in Section 7.6 but first we will go over a debugging example and then all the special DIL functions made for objects.

7.4. Compiling and Debugging your first object

As we have previously mentioned in Section 5.3 and Section 6.4 it is always a good idea to build one or two things and then compile to make finding errors easy. In this case we have one object to compile and rather than having all the rooms and NPCS get in my way while compiling it I have removed them and only have the '%objects' section. The following is what the zone looks like when it has only one object in it.

```
#include <composed.h>
%zone dragonst
lifespan 20
reset RESET_ANYHOW
creator {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."

%objects

dragon_head

title "a gold dragon head"

descr "A large golden dragon head is laying here looking sad."

names {"large golden dragon head","large gold dragon head,
       "golden dragon head","large dragon head","gold dragon head",
       "dragon head","large head", "sad head","head"}

    extra {}
"The head is large and beautiful, at least as beautiful as a dead
dragon head can be.  There is an extreme look of sorrow on the dragons
face and it seems to be for much more than its own death."
```

```
extra {"gold dragon head face","dragon head face","head face","face"}
"Looking into the dragons face your eyes are drawn to the eyes of the
dead dragon.  Could there be something there?"

extra {"eyes","eye"}
"A world of blue skies and no storms is visible through the eyes and it
seems to be moving as if you were watching the world from space."

manipulate MANIPULATE_TAKE,MANIPULATE_HOLD

height 33
  weight 50

extra {"$get_s"}
"You suddenly feel very sad for a world that you don't even know."

extra {"$get_o"}
"A strange look of sadness crosses $1ns face."

extra {"$drop_s"}
"You feel much happier but you remember a feeling of great sorrow."

extra {"drop_o"}
"$1n seems to cheer up a bit."

end

%end
```

I removed the '%rooms' and '%mobiles' sections added a '%objects' section and stuck the dragon head in and now its ready to be compiled and put into the VME server for you to be able to look at it in the game. If you downloaded our example zones for this document you can compile this zone along with us and fix the errors as we do for practice. The filename is `debug_obj.zon`. Just so you know the errors in this zone are intentional so please don't write me an email telling me that there are errors in it.

The command to compile the zone is **VMC debug_obj.zon**. Here is what we get when we first try and compile the zone.

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_obj.zon'
<debug_obj.zon> @ 65: EOF in string
debug_obj.zon: 5: parse error
   Token: '{'
debug_obj.zon: 25: parse error
   Token: 'golden'
Grave errors in file 'debug_obj.zon'.
```

This error file doesn't look any harder than the last one we dealt with when compiling our first room or NPC. We can not stress enough always fix the smallest numbered error first. In this case the lowest

numbered error shows up in line five of the zone. The error says something is wrong with the '{' but looking at the line it is obvious the compiler got confused because I forgot 's' at the end of 'creators'. If we fix line five and recompile this is what we get:

```
 VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_obj.zon'
<debug_obj.zon> @ 65: EOF in string
debug_obj.zon: 25: parse error
   Token: 'golden'
Grave errors in file 'debug_obj.zon'.
```

Now we have come to another one of those weird errors. If you look at line 25 you will find that the line looks like it is correct. As we have said before when you find an error like this it most likely means that you are missing a quote or a '{}'. The only way to find the problem is start at the quote or '{}' before the word in the error and go backwards through the file till you find a missing one. Lucky for us the missing one is in the very next string. If you add a double quote just before the ending comma on line 24 and recompile you will get the following:

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_obj.zon'
debug_obj.zon: 42: parse error
   Token: ','
Compilation aborted.
```

This error is a little tricky. It seems to be pointing at the ',' as the problem. If you look at the line though and remember what you need for a manipulate field you will notice that the surrounding '{}' are missing. The reason the compiler is pointing at the comma is because it doesn't understand what to do with the comma with out the '{}' grouping symbols. Fixing these and recompiling results in the following message from the compiler.

```
VMC v2.0 Copyright (C) 2001 by Valhalla [May  9 2001]
Compiling 'debug_obj.zon'
VMC Done.
```

Notice there are no errors and it says 'VMC done', this means that you have now successfully compiled the zone. This is the last debugging walk through in the manual. If you still have a lot of trouble figuring out errors don't stress compiling is an art the more you do it the easier it will get. We suggest you take the zones we have provided in our release and create errors so you can get used to the messages you will see when you are making your own zones. Never be afraid to ask for help from someone else sometimes a bug is so simple you will over look it and sometimes it just takes a second person a single glance to find it. Another trick to finding errors if you have been looking for more than 5 minutes take a break and come back in 10 minutes sometimes that short relaxing time will help you find the problem.

You have now compiled your first object. The steps are the same to get it into the game as it was for the rooms and NPCs. We will not go over them again except to say copy your files that the compiler made over into the zone directory of your mud and reboot. From there log on and you should be able to **wstat**

and **load** your object by using its full symbolic name. It would be a good idea to try and get this zone into your server and lay with the object a bit so when you get to Section 7.6 you will be ready for anything.

7.5. DIL functions for objects

The DIL language is the language a builder can use to make his own special functions on rooms, NPCs, objects, PCs, and much more. This manual is for basic zone writing and therefore will not go into how to write your own DIL functions. The VME however is released with many functions for you as an Administrator and your builders to use to make special rooms, NPCs, and objects. The following sections contain a full list of all object functions released with the VME 2.0 server.

7.5.1. Restriction functions

The desire to have different equipment comes from every players desire to be different. The restrict functions were designed to help make this a reality by only allowing certain groups of players to wear some items. The restrict functions can be used alone or together to make a greater restricted item. for example you could make an item, restricted to only females or you could make an item, restricted to females that had a strength greater than 20 and who have done a certain quest.

All restrict functions have a name that describes what the restriction function is for and four other arguments. the 2nd, 3rd, and 4th argument is the exact same for all restrict functions only the name of the restrict and the first argument changes. The format for the restrict functions is as follows.

```
dilcopy <function name> (arg 1, <max damage>,
    <percentage>,<Optional DIL>);
```

We will skip the function name and the first argument and get back to them later.

max damage and percentage

> The second and third arguments set the damage done when the wrong player wears an object. The reason we are explaining them together is they can work together or separately depending on how you set them. The second argument is the max damage the third argument is the percentage damage.
>
> When both arguments are set to 0 no damage will be done when the item is illegally worn. When the second argument is set to a number like 100 and the third argument is set to 0, exactly 100 damage will be done to the player no matter how many hit points he/she has. So by setting the second argument to a number and setting the third to 0 you could possibly kill your victim since it will remove the amount specified no matter how much the player has.
>
> If you do not want to possibly kill your victim the Third argument should be used. If you set the second argument to 0 and the third argument to a number it will do a percent of damage to the player. for example if The third argument was set to 25 it would do 25 % damage to a player.

You can also use the second and third argument together if you want to set a max amount of damage without possibly killing them. for example if you set the second argument to 100 and third to 25. The item will do 25% damage up to 100 hit points of damage. This all might be a bit confusing so let me show you a few examples and tell you what they would do.

second= 0 third = 25

This would do 25% damage to a player. second =100 third = 0 this would do 100 damage to a player no matter his amount of hit points.

second = 25 third = 25

This would do 25 % damage to a player up to 25 hit points. So if a player had 150 hit points the max that could be removed is 25 hit points.

optional DIL

All restrict DIL functions have a default set of acts. If you want to make your own set of acts you have to create your own DIL that does nothing more than act. If you don't understand how this works. You may want to look in the DIL manual about passing DIL functions as arguments.

Now we should get back to the first argument and function name since they are what control the restricts. The function name has been chosen so you can easily tell what the restrict was created for while the first argument changes depending on which restrict you use. The following are what each function restricts name is and what the function name and first argument are.

Guild Restrict

This function restricts an object to players in a certain guild or guilds. Anyone not in the guild or guilds, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin guild_restrict
(guilds:stringlist,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is a stringlist. This means you can restrict this item to more than one guild. All guilds in the string list will be able to wear this object. If we wanted to make an item that only Paladins and sorcerers could wear it would look like this.

```
dilcopy guild_restrict@function ({"Midgaard Paladin",
                          "Midgaard Sorcerer"},0,25,"");
```

Anti-guild Restrict

This function restricts an object to players not in a certain guild or guilds. Anyone not in the guild or guilds listed, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin anti_guild_restrict
(guilds:stringlist,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is a stringlist. This means you can restrict this item from more than one guild. All guilds in the string list will not be able to wear this object. If we wanted to make an item that only Paladins and sorcerers could not wear it would look like this.

```
dilcopy anti_guild_restrict@function ({"Midgaard Paladin",
                                "Midgaard Sorcerer"},0,25,"");
```

Quest Restrict

This function restricts an object to players who have done a certain quest. Any player who has not done the quest, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin quest_restrict
(qst:string,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is a string. The quest restrict is only made to restrict the by one quest at a time. If you want the item to have multiple quests restrictions you just add another dilcopy to it. The following Would be an object restricted to one quest.

```
dilcopy quest_restrict@function ("Eagles quest complete",0,25,"");
```

Quests Restrict

This function restricts an object to players who have a certain quest or quests. Anyone not having all quests in the list of quests, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin quests_restrict
(qsts:stringlist,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is a stringlist. This means you can restrict this item to more than one quest. Players that have not done every quest will not be able to use the object. If we wanted to make an item that only players that have finished both the 'Eagles quest complete' and the 'Feather fall quest complete' could wear. It would look like this.

```
dilcopy quests_restrict@function ({"Eagles quest complete",
                           "Feather fall quest complete"},0,25,"");
```

Alignment Restrict

This function restricts an object to players with a certain alignment range. Anyone not in the alignment range, will have the damage done as set in the 3nd and 4th arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory.

We said at the beginning of this section that all restricts have only 4 arguments. This was a bit of a lie since this restrict has five arguments. The reason we didn't count this one is because the first and second argument in the alignment restrict function are used together and thus are only really one argument. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin ali_restrict
(max:integer,min:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first and second arguments are two integers. the first is the max alignment that can wear or use this object and the second is the minimum alignment. So if we wanted to restrict an item to only good players it would look like this.

```
dilcopy ali_restrict@function (1000,350,0,25,"");
```

Level restrict

This function restricts an object to players above or equal to a certain level. Any player not at least the level or higher, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. This restrict works in the old level system from one to fifty which means if the level restrict is set higher than fifty no player will be able to wear or use this object. This is good if you have objects that only your administrators should be able to use. If you want to restrict an object to

greater than level fifty for players you need to use the 'vlevel' restrict. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin level_restrict
(lvl:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is an integer. The integer for the level restrict can range from 1 to 255. Thus if we wanted to make an object that only administrator could wear or use it would look like this

```
dilcopy level_restrict@function (51, 0,25,"");
```

Virtual Level Restrict

This function restricts an object to players above or equal to a certain level. Any player not at least the level or higher, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. This restrict works in the new level system from one to infinity. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin vlevel_restrict
(lvl:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is an integer. The integer for the level restrict can range from 1 to infinite. Thus if we wanted to make an object that only players that have reached the level of 5000 could wear or use, it would look like this.

```
dilcopy vlevel_restrict@function (5000, 0,25,"");
```

Race restrict

This function restricts an object from players of a certain race. Any player not of the selected race, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin race_restrict
(rc:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is an integer. The integer being passed in should be the race you want to restrict the object from. You can pass in the defines as listed in Appendix C or if

you have added races you will find the list of races in `values.h`. If we wanted to restrict an object from humans the following is what it would look like.

```
dilcopy race_restrict@function (RACE_HUMAN,0,25,"");
```

Gender restrict

This function restricts an object to players of a certain gender. Anyone not of the gender, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin sex_restrict
(sx:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is an integer. The integer you should pass in is one of the defines from `vme.h`. The gender defines are as follows.

```
#define SEX_NEUTRAL   0
#define SEX_MALE      1
#define SEX_FEMALE    2
```

If we wanted to make an item that could only be worn by a female player, it would look like this.

```
dilcopy sex_restrict (SEX_FEMALE,0,25,"");
```

Player restrict

This function restricts an object to players who have a specific name. Any player of the wrong name, will have the damage done as set in the 2nd and 3rd arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin ply_restrict
(pname:string,damage:integer,percent:integer,action:string);
```

As the definition indicates the first argument is a string. The quest restrict is only made to restrict the quest to one person at a time. While this DIL was designed to be put on when a player receives a quest item and thus was made to be dilcopied in a DIL you can still put it on when you first create the object if you are making special items for administrators or players that you know in advance. If you want more information about copying a DIL from with in another DIL you will have to read the

DIL manual If however you want to restrict this to a single player at compile time of your zone it would look something like this.

```
dilcopy ply_restrict@function ("Whistler",0,25,"");
```

Ability restrict

This function restricts an object from a player with less than a certain amount of a certain ability. Any player not having the correct amount of a certain ability, will have the damage done as set in the 3nd and 4th arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory.

We said at the beginning of this section that all restricts have only 4 arguments. This was a bit of a lie since this restrict has five arguments. The reason we didn't count this one is because the first and second argument in the ability restrict function are used together and thus are only really one argument. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin abi_restrict
(abi:integer,min_abi:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first and second arguments are two integers. the first is the ability type and the second is the amount of that ability the player needs to have or greater to wear or use the item. the ability types can be found in `vme.h` and are listed here for convenience.

```
#define ABIL_MAG              0
#define ABIL_DIV              1
#define ABIL_STR              2
#define ABIL_DEX              3
#define ABIL_CON              4
#define ABIL_CHA              5
#define ABIL_BRA              6
#define ABIL_HP               7
```

If you wanted to restrict an object to people having more than 50% strength it would look like this:

```
dilcopy abi_restrict@function (ABIL_STR,50,0,25,"");
```

If you want to restrict an object to more than one ability you only need to add another restrict to the item. For example if you wanted to restrict it to people having greater than or equal to 50% divine and 30% brain. The item would have these two lines.

```
dilcopy abi_restrict@function (ABIL_DIV,50,0,25,"");
dilcopy abi_restrict@function (ABIL_BRA,30,0,25,"");
```

Skill restrict

This function restricts an object from a player with less than a certain amount of a certain skill. Any player not having the correct amount of a certain skill, will have the damage done as set in the 3nd and 4th arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory.

We said at the beginning of this section that all restricts have only 4 arguments. This was a bit of a lie since this restrict has five arguments. The reason we didn't count this one is because the first and second argument in the skill restrict function are used together and thus are only really one argument. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin ski_restrict
(ski:integer,min_ski:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first and second arguments are two integers. the first is the skill and the second is the amount of that skill the player needs to have or greater to wear or use the item. the skills can be found in Appendix G and `values.h`. We have also included the first five skills here for convenience in explaining how the function works.

```
#define SKI_TURN_UNDEAD        0
#define SKI_SCROLL_USE         1
#define SKI_WAND_USE           2
#define SKI_CONSIDER           3
#define SKI_DIAGNOSTICS        4
```

If you wanted to restrict an object to people having more than 50% 'turn undead' it would look like this:

```
dilcopy ski_restrict@function (ASKI_TURN_UNDEAD,50,0,25,"");
```

If you want to restrict an object to more than one skill you only need to add another restrict to the item. For example if you wanted to restrict it to people having greater than or equal to 50% in 'turn undead' and 30% in 'scroll use'. The item would have these two lines.

```
dilcopy ski_restrict@function (SKI_TURN_UNDEAD,50,0,25,"");
dilcopy ski_restrict@function (SKI_SCROLL_USE,30,0,25,"");
```

Spell restrict

This function restricts an object from a player with less than a certain amount of a certain spell. Any player not having the correct amount of a certain spell, will have the damage done as set in the 3nd and 4th arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory.

We said at the beginning of this section that all restricts have only 4 arguments. This was a bit of a lie since this restrict has five arguments. The reason we didn't count this one is because the first and second argument in the spell restrict function are used together and thus are only really one argument. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin sp_restrict
(spl:integer,min_sp:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first and second arguments are two integers. the first is the spell and the second is the amount of that spell the player needs to have or greater to wear or use the item. the spells can be found in Appendix H and `values.h`. We have also included the first five spells here for convenience in explaining how the function works.

```
#define SPL_LOCK              52
#define SPL_UNLOCK            53
#define SPL_DROWSE            54
#define SPL_SLOW              55
#define SPL_DUST_DEVIL        56
```

If you wanted to restrict an object to people having more than 50% 'lock' spell, it would look like this:

```
dilcopy sp_restrict@function (ASPL_LOCK,50,0,25,"");
```

If you want to restrict an object to more than one spell you only need to add another restrict to the item. For example if you wanted to restrict it to people having greater than or equal to 50% in 'lock' and 30% in 'unlock' spells. The item would have these two lines.

```
dilcopy sp_restrict@function (SPL_LOCK,50,0,25,"");
dilcopy SPL_restrict@function (SPL_LOCK,30,0,25,"");
```

Weapon restrict

This function restricts an object from a player with less than a certain amount of a certain weapon. Any player not having the correct amount of a certain weapon, will have the damage done as set in

the 3nd and 4th arguments unless none is set. Even if no damage is set the object will be removed off of the players equipment and placed in their inventory.

We said at the beginning of this section that all restricts have only 4 arguments. This was a bit of a lie since this restrict has five arguments. The reason we didn't count this one is because the first and second argument in the weapon restrict function are used together and thus are only really one argument. The following is the definition of the DIL as found in `function.zon`.

```
dilbegin weap_restrict
(wpn:integer,min_wpn:integer,damage:integer,percent:integer,action:string);
```

As the definition indicates the first and second arguments are two integers. the first is the weapon and the second is the amount of that weapon the player needs to have or greater to wear or use the item. the weapons can be found in Appendix D and `values.h`. We have also included the first five weapons here for convenience in explaining how the function works.

```
#define WPN_BATTLE_AXE    7  /* Two Handed */
#define WPN_HAND_AXE      8
#define WPN_WAR_MATTOCK   9  /* Two Handed */
#define WPN_WAR_HAMMER   10
#define WPN_GREAT_SWORD  11  /* Two Handed */
```

If you wanted to restrict an object to people having more than 50% 'battle axe', it would look like this:

```
dilcopy weap_restrict@function (WPN_BATTLE_AXE,50,0,25,"");
```

If you want to restrict an object to more than one weapon you only need to add another restrict to the item. For example if you wanted to restrict it to people having greater than or equal to 50% in 'hand axe' and 30% in 'battle axe' spells. The item would have these two lines.

```
dilcopy weap_restrict@function (WPN_HAND_AXE,50,0,25,"");
dilcopy weap_restrict@function (WPN_BATTLE_AXE,30,0,25,"");
```

7.5.2. Tuborg function

What game would be complete with out the Denmark water! With that in mind the VME 2.0 has a tuborg function that makes a drink give endurance and health when drank. The function is defined in `function.zon` as follows:

```
dilbegin tuborg (s:string);
```

As the definition indicates the tuborg function only has one argument. The real surprise is that the argument is not used yet in the DIL so no matter what you set it to it doesn't matter. In the future this argument is going to allow different kinds of tuborgs to be made but for now its just a place holder and all that is needed is a set of double quotes.

To create a tuborg you just add the following line to your drink container.

```
dilcopy tuborg@function ("");
```

7.5.3. Message board

Every game needs a way for Administrators and players to exchange ideas. The message boards have been designed for this purpose. The boards function can be easy to use or more difficult depending on what all you want them to do. The board function is defined in `boards.zon` as follows.

```
dilbegin board
(idxfile:string,l_res:string,r_res:string,p_res:string,bmax:integer);
```

This looks pretty hard I know but to make a normal board we have made it as simple as possible while allowing for the boards to be used in almost any situation. After you make your first board it is pretty much block and copy and change the first argument. The arguments are as follows:

idxfile

> The first argument is the board index filename. It tells the board DIL what name to store the board under so if you create more boards with the same name they will all be pointing to the same messages. You can put any legal symbolic name in this string and it will work with no problems.

l_res

> the second argument is a DIL you pass in that does any checks to see if the player looking at the board is allowed to. This requires some knowledge in DIL but we have given some example DIL functions in the `boards.zon`.
>
> ```
> //used to restrict players access to a board
> dilbegin string admin_res (u:unitptr,v:unitptr);
>
> //used to restrict non-admin from removing posts
> dilbegin string rem_res (u:unitptr, v:unitptr);
> ```
>
> So with the 'admin_res' you could do something like this:
>
> ```
> dilcopy board@boards ("wizbrd","admin_res@boards"...);
> ```

Putting the 'admin_res' function in the second argument would make it so only administrators could look at the board. If you put an empty string or two double quotes as the argument it will let anyone look at the board.

r_res

the third argument is a DIL you pass in that does any checks to see if the player trying to remove a post at the board is allowed to. This requires some knowledge in DIL but we have given some example DIL functions in the `boards.zon`

```
//used to restrict players access to a board
dilbegin string admin_res (u:unitptr,v:unitptr);

//used to restrict non-admin from removing posts
dilbegin string rem_res (u:unitptr, v:unitptr);
```

So with the 'rem_res' you could do something like this:

```
dilcopy board@boards ("citizen","","rem_res@boards",...);
```

With the 'rem_res' in the third argument only administrators can now remove from this board but anyone can look at it because of the empty string in the second argument. Putting an empty string in the third argument will make it so anyone can remove from this board.

p_res

the forth argument is a DIL you pass in that does any checks to see if the player trying to post at the board is allowed to. This requires some knowledge in DIL but we have given some example DIL functions in the `boards.zon`.

```
//used to restrict players access to a board
dilbegin string admin_res (u:unitptr,v:unitptr);

//used to restrict non-admin from removing posts
dilbegin string rem_res (u:unitptr, v:unitptr);
```

As you can see we haven't made a post restriction DIL because as of yet we haven't found a need for one. If you have a need for one just look over the two restrict DIL functions we have already mentioned and you will find it is really easy to make. We want to allow anyone to post so our dilcopy looks like this:

```
dilcopy board@boards ("citizen","","rem_res@boards","",...);
```

With the 'rem_res' in the third argument only administrators can now remove from this board but anyone can post to it because of the empty string in the forth argument. The empty string again in the second argument also allows everyone to look at the board.

max

The fifth argument is simply the amount of posts that you want to allow to be posted before the board is full.

To make a free for all board where everyone can post, remove posts, look at what posts are on the board, and have a max of 50 posts it would simply be as follows:

```
dilcopy board@boards("citizen","","","",50);
```

When making a board for players to post concerns to the administrators and only have the administrators be able to remove them, wile still allowing everyone to read them it would look like this.

```
dilcopy board@boards("citizen","","rem_res@boards","",100);
```

7.6. More complex objects

In the last sections you learned all the fields and how to make a basic object. In this section we will use the information from the last sections to create some more unique objects for our dragon station zone There is not a lot of new information here we will be using the DIL functions, fields, and flags to make objects we have only mentioned before.

7.6.1. Making a communication board

In Section 7.5.3 you learned all there you need to know about the boards DIL to create a board. In this small section we are going to show you the rest of a board and what a finished one looks like.

As with all objects the first step is to fully describe and name your board. We will stick with the space station theme since our goal is to have a complete example zone for you. The boards symbolic, names, title, description and extra turned out like this.

```
info_board

title "a merchant information board"
descr "A merchant information Board is mounted on a wall here."

names {"merchant information board","information board","merchant
board","board"}

extra {} "A large flashy black steal board."
```

Just incase the VME server we have has a spell that can damage inanimate objects we will give this board a material type.

```
MATERIAL_METAL("A very fine quality black steel")
```

Now for the special stuff for the board. We need to give the board a type and copy the board DIL to it.

```
type ITEM_BOARD
dilcopy board@boards("info","","rem_res@boards","",100);
```

There you go nothing to it you have just created your first board. Now lets bring it all together and tag on an end symbol and we are all finished.

```
 info_board
title "a merchant information board"
descr "A merchant information Board is mounted on a wall here."
names {"merchant information board","information board","merchant
board","board"}

extra {}
"A large flashy black steal board."

MATERIAL_METAL("A very fine quality black steel")
type ITEM_BOARD
dilcopy board@boards("info","","rem_res@boards","",100);

end
```

7.6.2. Making a container

I thought it would be cool to have a small weapons locker on the space station not to mention event hough we went over the container macro in Section 7.2.10, we didn't cover everything you need in fact the macro eaves a few things out because you may or may not want to set them.

As with all objects we start right off by describing the item. There is nothing new here so we will just show it to you and go on.

```
wpn_locker

title "a weapons locker"
names {"weapons locker","weapon locker","locker"}

descr "a small weapons locker hangs on the wall here."

extra {}
```

```
"It is an ordinary weapons locker that looks like it holds any illegal
weapons that are taken on the station."
```

Now we need to put in all the information that makes this item a container that can't be taken but it can be opened and it is locked.

```
manipulate {MANIPULATE_ENTER}
CONTAINER_DEF(500)
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
key black_key
```

Notice we didn't make the item 'MANIPULATE_TAKE' because we don't want people to be able to walk off with our weapons locker. One final touch and we are all finished with the weapons locker. It is always nice to put a material type on your items so a spell or a skill can tell if you can do anything with them. So with our material added in the full locker would look like this.

```
wpn_locker

title "a weapons locker"
names {"weapons locker","weapon locker","locker"}
descr "a small weapons locker hangs on the wall here."

extra {}
"It is an ordinary weapons locker that looks like it holds any illegal
weapons that are taken on the station."

MATERIAL_METAL("A very fine quality steel")

manipulate {MANIPULATE_ENTER}
CONTAINER_DEF(500)
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
key black_key

end
```

7.6.3. Creating drinks

In Section 7.2.7, we covered how to set the size and weight of the container and its content but now we need to talk about some other special things about a drink container verses other objects.

The drink container is one of the few objects that has rules on how you set the title, description and names fields. The title and description fields should ot have anything to do with the liquid inside the container. This means if you have a barrel full of water you do not put the word water in the title or the description. In our case we have a bag full of wine so we do not put wine in either the title or description. The reason for this is if the player drinks the bag empty and then fills it with water and we had put wine

in the title or description it would still be there but the bag would now be full of water.. Our symbolic, title, and description would then look like this.

```
liq_ration

title "a red bag"
descr "A red bag has been gently placed here."
```

The names on the other hand MUST have the drink name as the last name in the name list. The reason it must be the last one is when a player drinks or pours out the liquid the name is removed. If a player then refills the container the name of the new liquid is added to the last name. The bag we are making is full of wine so our names list would look like this.

```
names {"red bag", "bag", "wine"}
```

Now we add the liquid define for wine

```
LIQ_WINE(1,2,2,0)
```

Finally we add the material type for the bag, the cost to buy the container, an extra players can look at, and finally an identify extra so if a player casts either the identify or improved identify spell on the bag they will se it. with all that added The finished drink container looks like this.

```
liq_ration
names {"red bag", "bag", "wine"}
title "a red bag"
descr "A red bag has been gently placed here."

extra {}
"A small label reads Tassel Grove's finest.  Year 321"


MATERIAL_ORGANIC("a soft plastic")
manipulate {MANIPULATE_TAKE}
LIQ_WINE(1,2,2,0)
cost 2 IRON_PIECE

extra {"$identify"}
"Its the special wine from Tassel grove a small halfling village on the
planet Valhalla.  It seems like a great vintage wine."

end
```

7.6.4. Creating food

The food is very simple to make its just a regular item with the macros you learned in Section 7.2.8. In fact making food is so simple we almost left it out. I am only adding this to show how simple it is to

change a regular item like the dragon head we showed you in Section 7.3 into a food item. Now only a sick person would make a dragon head into food but if you wanted to you just add the 'FOOD_DEF(...)' and your all set. here is a basic food that you might find laying around a space station.

```
beef_stick

title "a tough leathery stick"
descr "A tough leathery looking stick is laying here."
names {"tough leathery stick","tough leather stick","leathery stick",
"leather stick","tough stick","stick"}

extra {}
"This has the word BEEF burnt into it."

manipulate {MANIPULATE_TAKE}
FOOD_DEF(5,0)
weight 1
cost 1 COPPER_PIECE
MATERIAL_ORGANIC("tough beef")

end
```

7.6.5. Making a weapon

Whats a game with out some kind of weapon to chop or bash things into little pieces. We examined how to set the weapon fields in Section 7.2.3, the object transfers in Section 7.2.14, and the restriction DIL functions in Section 7.5.1. Now we will pull all we have learned together and make a pretty nifty little stiletto.

The first part as with all our example objects is to set up the symbolic, names, title, description, object extra, and material type. this is no different from any other object so here is what we ended up with

```
 w_stiletto
 title "a stiletto"
  names {"stiletto", "dagger"}
 descr
 "A deadly looking stiletto has been left here."

extra{}
"This looks like a thieves dream come true. "

MATERIAL_METAL("A very fine quality steel")
```

Now lets add the defines and DIL functions that make this a special weapon along with the manipulate flags that makes it able to be wielded. We will give it a bonus in magic and good craftsmanship along with a plus in backstab for all those assassins on the game.

```
manipulate {MANIPULATE_TAKE, MANIPULATE_WIELD}
```

```
WEAPON_DEF(WPN_DAGGER, 1, 2)
SKILL_TRANSFER(SKI_BACKSTAB, 2)
dilcopy abi_restrict@function (ABIL_DEX,10,0,25,"");
```

to finish it off we will give the weapon a cost, rent, and finally two identifies for the two identify spells. Now that we have it all defined we pull it together and it looks like this.

```
 w_stiletto
 title "a stiletto"
 names {"deadly looking stiletto","deadly stiletto","stiletto", "dagger"}
 descr
 "A deadly looking stiletto has been left here."

extra{}
"This looks like a thieves dream come true. "

MATERIAL_METAL("A very fine quality steel")

manipulate {MANIPULATE_TAKE, MANIPULATE_WIELD}
WEAPON_DEF(WPN_DAGGER, 1, 2)
SKILL_TRANSFER(SKI_BACKSTAB, 2)
dilcopy abi_restrict@function (ABIL_DEX,10,0,25,"");
weight 2
cost 2 GOLD_PIECE
rent 1 COPPER_PIECE

extra {"$identify"}
"The stiletto looks magical in nature and seems really sharp.  You can
tell if you wield it you would be able to backstab someone really easy."

extra{"$improved identify"}
"The stiletto gives you a magical bonus of +1 and has a quality of +2.
It also raises your back stabbing skill  by 2%.  You have to have at
least 10% in dex before you can wield this magical weapon."

end
```

7.6.6. Making armour

In Section 7.2.4 we explained how to set the armour fields now we will finish off by adding some more important information about armour in general.

The most important thing to realize is that not all wear positions on the VME server are armour positions. In fact only seven of the wear positions count as armour the rest are non-armour positions which we will cover next in Section 7.6.7. The following are the armour positions and their defines.

Table 7-7. Armour positions

Position	Define
head	MANIPULATE_WEAR_HEAD
hands	MANIPULATE_WEAR_HANDS
arms	MANIPULATE_WEAR_ARMS
body	MANIPULATE_WEAR_BODY
legs	MANIPULATE_WEAR_LEGS
feet	MANIPULATE_WEAR_FEET
cloak	MANIPULATE_WEAR_ABOUT

Note: There is one more field that works as armour, 'MANIPULATE_WEAR_SHIELD' but since that uses another define it is not shown here. We will leave that for an exercise for you to do later.

First we do the same as we have for every other item, pick the symbolic, title, description, extra description, and material type for the plate.

```
pol_plate
names {"polished breast plate","polished plate","breast plate","plate"}
title "a polished breast plate"
descr "A polished breast plate has been left here."
MATERIAL_METAL("A high luster silver colored metal")
```

Now we pick the armour type in this case I want it to be made like plate mail and I want it to have a magical bonus and a high craftsmanship. Obviously since this is a plate we will pick the body position.

```
manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_BODY}
ARMOUR_DEF(ARM_PLATE,5,9)
```

All that is left is to add the cost, rent, the identify extras, and I felt like putting a 40% strength restriction on the armour. With all that added together we finish up with the following piece of armour.

```
pol_plate
names {"polished breast plate","polished plate","breast plate","plate"}
title "a polished breast plate"
descr "A polished breast plate has been left here."

extra{}
"This is one shiny plate it seems to be made out of one perfect piece of
metal.  There doesn't even seem to be any markings of owner ship."

MATERIAL_METAL("A high luster silver colored metal")

manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_BODY}
ARMOUR_DEF(ARM_PLATE,5,9)

dilcopy abi_restrict@function (ABIL_STR,40,0,25,"");
```

```
cost 2 GOLD_PIECE
rent 3 COPPER_PIECE weight 25

extra{"$identify"}
"This is a high quality plate with a magical feeling."

extra{"$improved identify"}
"The plate has a magical bonus to your defence of a +5 and a quality of
+9.  You need 40% in strength to be able to wear it."
end
```

7.6.7. Making non-armour worn objects

In the previous section we defined armour that actually protects the char in combat. Here we will learn how to make the clothing and jewelery that may not do anything directly to combat but it can give your characters bonuses that help in combat in the long run. We will start by listing all the non-armour worn positions and their manipulate defines and then we will give a simple ring object.

Table 7-8. Non-armour positions

Position	define
ear	MANIPULATE_WEAR_EAR
neck	MANIPULATE_WEAR_NECK
wrist	MANIPULATE_WEAR_WRIST
finger	MANIPULATE_WEAR_FINGER
chest	MANIPULATE_WEAR_CHEST
back	MANIPULATE_WEAR_BACK
waist	MANIPULATE_WEAR_WAIST
ankle	MANIPULATE_WEAR_ANKLE

> **Note:** When giving ability, skill, weapon, or spell bonuses make sure you realize that positions like ear, neck, wrist, and ankle can all have two on a player. This means any bonuses you give can be doubled if the player gets two of them

I don't want to beat a dead horse so since I have already explained armour in Section 7.6.6 the only difference here is there is no 'ARMOUR_DEF' everything else is the same. The following was one of the first items my wife made as a new builder and I have always liked it. I know, I am a lush but this way I don't have to write an example.

```
  maskwa

names {"maskwa platinum ring", "platinum ring","maskwa ring","maskwa","ring"}
title "a Maskwa ring"
```

```
descr "A Maskwa platinum ring is laying here."
MATERIAL_METAL("Platinum, and other precious metals")
extra {}
"The ring has a large bear head.  Could this be the legendary head of
Maskwa?  Any thing formed with its head on it is said to strengthen the
wearer."
type ITEM_WORN
manipulate {MANIPULATE_TAKE, MANIPULATE_HOLD, MANIPULATE_WEAR_FINGER}
cost 100 COPPER_PIECE
rent 50 IRON_PIECE
weight 1
STR_TRANSFER(+1)
end
```

Note: One last thing I forgot to mention. The item type also changes but then that is not hard to understand since this is not armour it should be some other thing. In the case of non-armour worn items the item type is 'ITEM_WORN'.

7.7. Dragon station with rooms, NPCs, and objects

Now we will add the objects we have built to the zone from the previous chapter. This is still not complete while it does compile and you can log into your zone, you still have to load your NPCs and objects but they do not load themselves and they are not dressed with their armour. This will be taken care of in Chapter 8 and you will finally have a finished zone. The following is the source file so far.

```
#include <composed.h>
%zone dragonst
lifespan 20
reset RESET_ANYHOW
creators {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."

%rooms

chamber
title "The middle chamber of the station"
descr
"This chamber seems to have the entire station rotating around it.  It is
```

```
unbelievably large the ceiling seems to be a good 200 meeters high and
the room is perfectly cubic. Small human size ornate chairs with dragon
designs scrawled on the arms and back are arranged in a triangle like
setting with one large chair at the front.  This must be where all
station meetings are held. large pictures cover the walls depicting
dragons in all kinds of situations.  large passages lead of to the west
and the east.."

extra {"chair","chairs"}
"The chairs are made of some metal you don't recognize and every inch is covered
with some kind of dragon."

extra  {"dragon picture","picture"}
"Thousands of dragons dot the skies of this rather life like picture.  In the
center you see something move.  It looks to be a little green dragon."

extra{"green dragon","dragon","green"}
"An intellegence looking dragon is sitting perched on a large chair watching you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}


west to disposal_room descr
"You see a small room.";

east to hallway descr
"You see what looks to be a hallway.";

end

hallway
title "Module tunnel"
descr "The hallway is about 50 meters long and around 100 meters from
side to side and top to bottom.  The hallway seems to be dust free.  The
walls and the floors seem to be made out of the same sterile
metal-plastic that all space agencies uses.  There are large plate glass
windows that open up into space.  The hallway is filled with a dim light
that seems to come from everywhere yet no where all at once.  You notice
a glimmer of bright light coming from the windows.  To the east you see
an air lock and to the west the hallway opens up into a larger room."

extra {"windows","window"}
"Your eyes are drawn to a large ship lit up with running lights sitting
about 1 kilometer from the station."

extra{"floor","walls","wall"}
"Well what can be said it looks to be in perfect condition.  what else would
you want to know?"

extra {"large ship" ,"ship"}
"The ship looks really big and is shaped like a dragon.  The scales
```

```
sparkle and seem to be multiple colors."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to chamber descr
"The hallway opens up into a chamber.";

east to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

end

office
title "The station office"
descr
"Large paintings fill the walls of this part of the station.  The room
is as large as the other rooms big enough for Dragons to lounge while
still having a desk in one corner small enough for a humanoid.  The
floor along the north wall is lined with some kind of fabric and seems very
soft to walk on, it may be some kind of dragon lounge judging by how large an
area it covers.  There is a passage to the west."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"paintings","painting"}
"The paintings are of many dragons and riders in all kinds of tasks from
combat to look out.  All the figures seem to be staring at a staff
being held by a depiction of a wizard on the south wall."

extra {"wizard","staff"}
"The wizard has his hand stretched out and it seems there is a place
you can almost grab the staff. Maybe if you searched the staff you would
find it."

extra {"desk"}
"Its a desk alright but there doesn't seem to be any drawers and it
seems totally empty."

extra{"fabric"}
"Wussshhhhh you bound across the comfortable floor wasn't that fun."

west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

SECRET_DOOR_DIFFICULTY(SOUTH, 50)
```

```
south to portal_room descr
"You see what looks to be a portal room."
keyword {"air lock door","air lock","staff","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED,EX_HIDDEN};

end

 portal_room
 title "Green field room"
 descr
 "Like the other rooms on the station this one is large enough for
 dragons to comfortably fit in.  The strange thing about this room though
 is it is totally empty except for a green field right in the center.
 there is a door that leads to another room to the north."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

 extra {"green field","field"}
 "The field looks to be a green fog shifting and churning as you watch.
 if you are nuts you could probably enter it."

 north  to office descr
 "You see what looks to be an office."
 keyword {"air lock door","air lock","door"}
 key nokey
 open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

 //A link to the portal is also here from room_port
 end

 ship_port
 names{"green field", "field"}
 title "Green field"
 descr
 "Green Mist swirls about you."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

 in ship

 dilcopy force_move@function(
 //Time to activation
 4,
 //room and act
 "portal_room@dragonst!You feel your body dissolving for lack of a better
 description.&nYou appear on the deck of a ship.",
 //True or False for randomizing or not
 FALSE);
```

```
 end

room_port
names{"green field", "field"}
title "Green field"
descr
"Green Mist swirls about you."
movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

in portal_room

dilcopy force_move@function(
//Time to activation
4,
//room and act
"ship@dragonst!You feel your body dissolving for lack of a better
description.&nYou appear on the deck of a ship.",
//True or False for randomizing or not
FALSE);


end

disposal_room
title "Red field room"
descr
"Like the other rooms on the station this one is large enough for
dragons to comfortably fit in.  The strange thing about this room though
is it is totally empty except for a red field right in the center.
there is a door that leads to another room to the east."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"red field","field"}
"The field looks to be a red fog shifting and churning as you watch.
if you are nuts you could probably enter it."

east to chamber descr
"You see the main chamber.";

 //A link to the portal is also here from dis_port
end

dis_port
names {"red field","field"}
title "Red field"
descr
```

```
"Red Mist swirls about you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}
dilcopy force_move@function(
//how fast to force move in seconds
4,
//room to force move to and act
"deathspace@dragonst!You feel your body dissolving for lack of a better description.",
//true or false random move or not
0);
in disposal_room

end

ship
title "War dragon"
descr
"Blue light softly glows from con duets that line the walls of this ship.
The floors beside the east and west wall have what looks to be soft
fabric covering.  The south wall has small controls that seem to be made
for humanoids with two small chairs that look to be pilot seats.  view
portals are about 50 meters up the side of the ship on the west and east
wall and some kind of electronic screen covers the south wall.  The ship
seems to be a one room ship but there is a green field by the north
wall."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"view port"}
"Sorry your not 50 meters tall maybe it is made for a dragon?"

extra {"view screen","screen"}
"It seems to be the pilots view screen but you can't seem to see a way
to turn it on."

extra {"controls","control"}
"The controls are in some weird language and your afraid if you start
pushing buttons you might rocket in to the station or worse slam into
a planet."

extra {"soft fabric","fabric"}
"It looks to be a dragon lounge area."

 //A link to the portal is also here from ship_port
end

deathspace
title"Open space"
descr
```

```
"You see the ship and the station far off in the distance and you are in Space!"

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

dilcopy death_room@function (
//how often is damage done 4 would be 1 second
4,
//damage
400,
//act for the damage.
"You realize to late that was the trash disposal transporter and you feel
your lungs explode.");


end

%mobiles

bldragon

title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}

extra {}
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."

extra {"eye","eyes"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"

extra {"bat wings","wings"}
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."

M_DRAGON_BLACK_OLD(SEX_MALE)
```

```
end

janitor
names {"ugly janitor", "janitor", "hobgoblin"}
title "an ugly janitor"
descr "an ugly janitor is walking around, cleaning up."

extra{}
"This ugly green thing looks more goblin than hobgoblin but he seems intent on
cleaning everything around him."

M_AVG_HOBGOBLIN(6, SEX_MALE)

// he is sort of good for cleaning so much
alignment 900

//give him some money
money 5 IRON_PIECE

dilcopy janitors@function(15);

// only want him cleaning the station
dilcopy wander_zones@function("dragonst", 20, 1, 1);

end

bob

names {"Bob"}
title "Bob"
descr "Bob the Banker is here, sitting behind the counter."
extra {}
"He has a very serious look on his face."

// define from composed.h
M_SHOP_KEEPER(4, SEX_MALE, RACE_HUMAN)

//discourage people from killing banker
exp -500

flags {UNIT_FL_NO_TELEPORT}

special SFUN_BANK
end

%objects

info_board

title "a merchant information board"
descr "A merchant information Board is mounted on a wall here."
names {"merchant information board","information board","merchant
board","board"} extra {} "A large flashy black steal board."
```

```
MATERIAL_METAL("A very fine quality black steel")
type ITEM_BOARD
dilcopy board@boards("info","","rem_res@boards","",100);

end

w_stiletto
title "a stiletto"
names {"deadly looking stiletto","deadly stiletto","stiletto", "dagger"}
descr "A deadly looking stiletto has been left here."

MATERIAL_METAL("A very fine quality steel")
manipulate {MANIPULATE_TAKE, MANIPULATE_WIELD}
WEAPON_DEF(WPN_DAGGER, 1, 2)
weight 2
cost 2 GOLD_PIECE
rent 1 COPPER_PIECE

SKILL_TRANSFER(SKI_BACKSTAB, 2)
dilcopy abi_restrict@function (ABIL_DEX,10,0,25,"");

extra{}
"This looks like a thief dream come true. "

extra {"$identify"}
"The stiletto looks magical in nature and seems really sharp.  You can
tell if you wield it you would be able to backstab someone really easy."

extra{"$improved identify"}
"The stiletto gives you a magic bonus of +1 and has a quality of +2.
It also raises your back stabbing skill  by 2%.  You have to have at
least 10% in dex before you can wield this magical weapon."

end

wpn_locker

title "a weapons locker"
names {"weapons locker","weapon locker","locker"}

descr "a small weapons locker hangs on the wall here."
manipulate {MANIPULATE_ENTER}
CONTAINER_DEF(500)
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
weight 400
MATERIAL_METAL("A very fine quality steel")

extra {}
"It is an ordinary weapons locker that looks like it holds any illegal
weapons that are taken on the station."

end
```

```
pol_plate
names {"polished breast plate","polished plate","breast plate","plate"}
title "a polished breast plate"
descr "A polished breast plate has been left here."
extra{}
"This is one shiny plate it seems to be made out of one perfect piece of
metal.  There doesn't even seem to be any markings of owner ship."
manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_BODY} ARMOUR_PLATE(5,9)

MATERIAL_METAL("A high luster silver colored metal")

manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_BODY}
ARMOUR_DEF(ARM_PLATE,5,9)
dilcopy abi_restrict@function (ABIL_STR,40,0,25,"");
cost 2 GOLD_PIECE
rent 3 COPPER_PIECE
weight 25

extra{"$identify"}
"This is a high quality plate with a magical feeling."

extra{"$improved identify"}
"The plate has a magical bonus to your defence of a +5 and a quality of
+9. You need 40% in strength to be able to wear it."
end

liq_ration
names {"red bag", "bag", "wine"}
title "a red bag"
descr "A red bag has been gently placed here."

MATERIAL_ORGANIC("a soft plastic")
manipulate {MANIPULATE_TAKE}

LIQ_WINE(1,2,2,0)
cost 2 IRON_PIECE
extra {}
"A small label reads Tassel Grove's finest.  Year 321"

extra {"$identify"}
"Its the special wine from Tassel grove a small halfling village on the
planet Valhalla.  It seems like a great vintage wine."

end

beef_stick

title "a tough leathery stick"
descr "A tough leathery looking stick is laying here."
names {"tough leathery stick","tough leather stick","leathery stick",
"leather stick","tough stick","stick"}
```

```
extra {}
"This has the word BEEF burnt into it."

manipulate {MANIPULATE_TAKE}
FOOD_DEF(5,0)
weight 1
cost 1 COPPER_PIECE
MATERIAL_ORGANIC("tough beef")
end

  maskwa

names {"maskwa platinum ring", "platinum ring","maskwa ring","maskwa","ring"}
title "a Maskwa ring"
descr "A Maskwa platinum ring is laying here."
MATERIAL_METAL("Platinum, and other precious metals")
extra {}
"The ring has a large bear head.  Could this be the legendary head of
Maskwa?  Any thing formed with its head on it is said to strengthen the
wearer."
type ITEM_WORN
manipulate {MANIPULATE_TAKE, MANIPULATE_HOLD, MANIPULATE_WEAR_FINGER}
cost 100 COPPER_PIECE
rent 50 IRON_PIECE
weight 1
STR_TRANSFER(+1)
end

%end
```

7.8. Suggested object exercises

1. Using information you learned in Section 7.2.1, Section 7.2.2, Section 7.2.5, and Section 7.6.6 create a large shield.
2. Using information you learned in Section 7.6.7 and Section 7.2.14 create a ring that gives a person 5% strength bonus and removes 5% magic ability.
3. Using information you learned from Section 7.2.7 and Section 7.6.3 create a beer or soda from your local area.
4. Using information you learned in Section 7.2.9; make an object that gives off a bright light and will never run out.
5. Using the macros found in Section 7.2.7, Section 7.2.8, Section 7.2.11, and Section 7.2.10 along with information found in Section 7.6.4 and Section 7.6.3; create a chest that can be locked that contains food, drink, a pile of silver pieces, and a pile of iron pieces.

Chapter 8. The reset section

Once you have learned to build rooms, objects and NPCs, you will find one main missing thing, while you have created NPCs and objects they don't exist in the game unless you load them. When developing the VME we found that logging on and loading everything for the players to interact with, became a very difficult thing to do when we got over 30 items. After many seconds of thought we came up with the idea of a reset section that would do all of this work for us. In fact the reset takes care of closing doors after players have opened them, loading NPCs and their equipment, loading objects by themselves in rooms or even loaded objects in objects.

Everything inside the reset section activates once at boot time and then again when the reset time is up and the reset flag is true. These two fields were described in Section 3.2 and are included in the zone header. The reset section is denoted by the symbol '%reset' and can placed anywhere but we try to keep it at the end of our zone files. There is no set order you must reset your doors, objects, and NPCs in but I like to do doors first, special reset commands second, objects in rooms third, objects with objects in them forth, NPCs fifth, and finally NPCs. You may find that you have a better way of sorting them and again it is up to you.

8.1. Door resets

To show how the door resets work we will revisit an old room example from Section 5.5.1. The following two rooms are linked with a door and at boot time they are reset to closed. When the mud boots the door flags set on the room are the door flags that are used. After boot up time the reset section is where the VME gets its information about what to do with the door.

```
hallway
title "Module tunnel"
descr "The hallway is about 50 meters long and around 100 meters from
side to side and top to bottom...."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to chamber descr
"The hallway opens up into a chamber.";

east to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

end

office
title "The station office"
```

```
descr
"Large paintings fill the walls of this part of the station...."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};
end
```

Now that we have two rooms lets define the reset command and how it works. All reset commands have a keyword and then a set of arguments. The door reset command is simply 'door' followed by a set of arguments that tell the VME where the door is, which door in that location, and what you want to do with the door. The command looks like this.

```
door <room symbol> <Direction number> {<door flags>}
```

Door argument explanation

room symbolic

As the name indicates this is the room that the door is located in. If you are resetting a door not in the zone the reset command is in you will need to use a full symbolic name with the zone extension. The following would be two valid examples.

```
//room symbolic in  this zone
door myroom ...

//room symbolic in another zone
door out_room@frogwart ...
```

direction number

The direction number can be one of the pre-defined direction numbers in the file `vme.h`. shown here so you don't have to go flipping file to file.

```
#define NORTH 0
#define EAST  1
#define SOUTH 2
#define WEST  3
#define UP    4
#define DOWN  5
#define NORTHEAST 6
#define NORTHWEST 7
#define SOUTHEAST 8
#define SOUTHWEST 9
```

door flags

These flags, surrounded by '{}', describe the state of the door after the reset. The following is the list of possible door flags.

EX_OPEN_CLOSE

Set this if you can open and close this exit, be it a door, gate or otherwise.

EX_CLOSED

Set this if you want the exit to be closed at reset time.

EX_LOCKED

Set this if you want the exit to be locked at reset time.

> **Note:** An interesting aspect is that if you do not specify a key, you can only unlock this door with the 'pick' skill, 'unlock' spell or from DIL with UnSet();

EX_PICK_PROOF

Using this flag renders the 'pick' skill and 'unlock' spell un useable on the lock of this exit.

EX_HIDDEN

If this bit is set, the exit is hidden until the mobile has successfully searched for it, using the 'search'-command.

Now that we have all the information we need we can close the door after the reset time expires. For our two rooms the door reset would look like this.

```
door hallway EAST {EX_OPEN_CLOSE, EX_CLOSED}
door office WEST {EX_OPEN_CLOSE, EX_CLOSED}
```

> **Note:** As you can see from the example it is very important to close both sides of the door. If you do not close both sides you will get very weird and undefined errors when players are trying to open and close them

Another thing that you can do with the door reset command is change the doors status. In our previous example we reset the door to its status that it has when it first is loaded into the game. If however we wanted to change the door to a locked door we could do that by adding the locked flag like this.

```
door hallway EAST {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
door office WEST {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
```

8.2. Loading objects and NPCs

Time to start loading your zone with its life and all the other strange things you have built. There is two commands that do all the loading and equipping of objects. Oddly enough the commands are called 'load' and 'equip'. The format of the commands are almost the same but equip must be used inside a NPC grouping. With that in mind lets start with simple loads and work our way up.

The command to load an object or an NPC into a room is as follows:

```
load <object or NPC> [into] [<room<] [<load amount>]
  [{Other loads and equip commands}]
```

object or NPC

The first argument to the load command is the object or NPC symbolic name that you want to load. The first argument is the only one that must be included in all load commands.

into

This is just a symbol that tells the reset that we are loading the object or NPC into some other unit.

object, NPC, and room

The third argument is the symbolic name of the place where you are loading the object and the NPC.

load amount

the fourth argument is an optional argument that tells the reset how many of the objects are allowed in the world, zone, or locally. The possible values for this field are as follows:

max <num>

This command is always part of another reset command (load, equip, etc.). At reset time the entire world is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

local <num>

This command is always part of another reset command (load, equip, etc.). At reset time the location of which the unit is to be loaded into is scanned for occurences of the loaded unit only if the currently existing number is less than <num> will the command be executed.

zonemax <num>

This command is always part of another reset command (load, equip, etc.). At reset time the entire zone being reset is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

Optional grouping

Any reset command may be followed by a pair of curly brackets { } containing more reset commands. The commands inside the brackets will only be executed in case the associated command was successful.

Don't be alarmed if this sounds a bit hard. It all gets much more clear as some examples are explained. Lets take a look at the following example and see if we can't make this much more clear.

```
load udgaard/fido into midgaard/temple max 1
```

This example is pretty simple it says load the fido into the temple only if there isn't already 1 in the world. Now lets get a bit more complicated.

```
load udgaard/fido into midgaard/temple max 1
{
   load bone
   load excrement into midgaard/temple
}
```

Now we have said again load the fido into the temple if there is not already one in the world. Then if fido loads fill his inventory with a bone and load excrement into the temple as well.

We can get even more complicated but still just using the load commands by doing the following

```
load udgaard/fido into midgaard/temple max 1
{
   load bone
   load excrement into midgaard/temple
      load bag
   {
      load apple max 1
   }
}
```

now we still have the fido loading if there isn't one already in the world then the bone and the excrement and finally we load a bag. If there isn't an apple already in the world we load the bag with a apple in it other wise the bag will be empty.

Well that should be enough load examples for now but we will get right back to them in a bit. Now we should introduce another reset command called the 'equip' command that we have already mentioned. The 'equip' command works a lot like load but has much simpler arguments. The 'equip' command is as follows.

```
equip <symbol> position [load amount <num>]
```

symbol

The first argument is just the symbolic name of the item being worn by the NPC.

position

The position is any of the positions available in the `vme.h`. The following are all the positions along side there defines as found in the `vme.h`.

Table 8-1. Wear positions

Position	Define(s)
head	WEAR_HEAD
hands	WEAR_HANDS
arms	WEAR_ARMS
body	WEAR_BODY
legs	WEAR_LEGS
feet	WEAR_FEET
cloak	WEAR_ABOUT
shield	WEAR_SHIELD
hold	WEAR_HOLD
wield	WEAR_WIELD
ear	WEAR_EAR_R and WEAR_EAR_L
neck	WEAR_NECK_1 and WEAR_NECK_2
wrist	WEAR_WRIST_R and WEAR_WRIST_L
finger	WEAR_FINGER_R and WEAR_FINGER_L
chest	MANIPULATE_WEAR_CHEST
back	MANIPULATE_WEAR_BACK
waist	MANIPULATE_WEAR_WAIST
ankle	WEAR_ANKLE_R and WEAR_ANKLE_L

load amount

the fourth argument is an optional argument that tells the reset how many of the objects are allowed in the world, zone, or locally. The possible values for this field are as follows:

max <num>

This command is always part of another reset command (load, equip, etc.). At reset time the entire world is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

local <num>

This command is always part of another reset command (load, equip, etc.). At reset time the location of which the unit is to be loaded into is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

zonemax <num>

This command is always part of another reset command (load, equip, etc.). At reset time the entire zone being reset is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

Now with the equipment command you can now get your NPCs dressed and ready for battle. The 'load' and 'equip' commands are not the easiest though so lets go through some simple examples.

```
load guard into jail
  {
  equip helmet WEAR_HEAD
  equip plate WEAR_BODY
  equip pants WEAR_LEGS
  equip specialsword max 1
  load brass_key
  }
```

This is how you would equip a NPC with all items from the current zone. As you can see we didn't need full symbolics because the server knows to grab the items from the zone the resets are in.

```
load guard into safe_room max 2
 {
 equip plate WEAR_BODY
 load powersword max1
  {
  load silver_pile into safe_room
  }
 }
```

In this example we only load the silver pile if the guard loads and if the power sword loads which may or may not happen since there is only a max of one sword allowed while there is a max of 2 guards allowed. What will happen in this case is at the first reset there will be one guard and one pile of silver. The next reset there will still only be one pile of silver but now there will be two guards.

Hopefully you have got the basic resets down. If not don't worry there are plenty more examples to come and we still have to make the resets for our dragon station zone.

8.3. Special reset functions

Now that we have gone over the basic load and equip commands we have some special commands that you can add to them to make them do more interesting things. Sometimes when doing resets you don't always want items or NPCs to load or sometimes you want them to load but only if a certain amount of other things correctly load. There are also times you want to clear the rooms or reload an entire object after removing the old one. All these things and more can be accomplished with the reset section.

8.3.1. The complete directive.

The 'load' and 'equip' commands have one more argument that can be placed at the end of them to make them act a bit different.the complete directive. In the case where this directive is placed at the end of a 'load or 'equip' command, the unit is only loaded in case all immediate commands inside its nesting are executed successfully. For example:

```
load captain into jail_room complete
{
   equip magic_sword position WEAR_WIELD max 1
   load bag
   {
      load ruby_ring max 1
   }
}
```

In this case the captain is only loaded if the objects magic_sword and bag are successfully loaded. if the ruby_ring is not loaded, it will have no effect on the complete directive. To make the ruby_ring affect to captains complete directive, the bag must also have specified a complete directive (because the bag would then not be complete, and thus the captain would not be complete).

8.3.2. The follow command

Once you load a NPC you may want that NPC to follow another NPC. That is what the 'follow' command is for. The following is the format of the 'follow' command

```
follow <symbol> <load amount #> <complete>
```

symbol

The first argument to the follow command is the symbolic name of the NPC to follow the NPC of the outer grouping.

load amount

the second argument is an optional argument that tells the reset how many of the NPC followers of this type are allowed in the world, zone, or locally. The possible values for this field are as follows:

max <num>

This command is always part of another reset command (load, equip, etc.). At reset time the entire world is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

local <num>

This command is always part of another reset command (load, equip, etc.). At reset time the location of which the unit is to be loaded into is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

zonemax <num>

This command is always part of another reset command (load, equip, etc.). At reset time the entire zone being reset is scanned for occurences of the loaded unit - only if the currently existing number is less than <num> will the command be executed.

complete

This only makes the NPC follow if all the other things in the grouping finishes completely. For a better description of how this directive works see Section 8.3.1.

The follow command is always used nested inside a loaded NPC to force the NPC <symbol> to follow the NPC of the outer grouping. The following would be a correct use of the follow command.

```
load captain into jail
  {
  follow guard max 4
    {
 equip guard_helmet WEAR_HEAD
 equip guard_plate WEAR_BODY
 equip guard_legs WEAR_LEGS
 equip guard_boots WEAR_FEET
 }
   follow guard max 4
    {
 equip guard_helmet WEAR_HEAD
 equip guard_plate WEAR_BODY
 equip guard_legs WEAR_LEGS
 equip guard_boots WEAR_FEET
 }
```

```
  }
```

This example would load two guards that are fully dressed and they would start following the captain which is also loaded.

8.3.3. The purge command

There are times when you want to clean up a room. This can be done very easy by using the **purge**. The following is the format of the purge command.

```
purge <symbol>
```

This command doesn't take much description. The symbol is the room you want to empty of all objects and NPCs. If you wanted to get rid of all objects and NPCs from a room with the symbolic name of jail it would look like this.

```
purge jail
```

8.3.4. The random command

If you ever want to load something only some of the time. There is a built in **random** command that allows you to pick the percentage of the time that the item will load. The random command has the following format.

```
random <num>
  {group or single set of resets}
```

It is important to point out this is done by a random percentage chance where as 1% of the time would be almost not at all and 100% of the time would be all the time. If we wanted to load a group of things only 80% of the time it would look like this.

```
random 80
  {
  load captain into jail_room complete
    {
    equip magic_sword position WEAR_WIELD max 1
    load bag
      {
      load ruby_ring max 1
      }
```

```
    }
}
```

8.3.5. The remove command

Many times players take items out of containers like chests or steal items from your NPCs and leave them naked. If the NPC is not dead the resets don't reload them therefore your NPCs will stand there empty and so will your chests. This is fine if that is what you want but sometimes you want them to get dressed or refilled again at reset time. that is what the **remove** command is for. The following is the format of the remove command.

```
remove <symbol1> in <symbol2>
```

Again the remove command is a simple command and it only has two arguments, the item and where it is to remove it from. If you wanted to have a cabinet that at every reset it would have a knife and a bag of sugar in it would look like this.

```
remove cabinet in kitchen
load cabinet into kitchen
 {
 load sugarbag
 load knife
 }
```

8.4. Reset walk through

The dragon station is almost finished. All you have to do now is create the resets for it. We don't have a lot of stuff in the zone but we have enough to make a decent example of how the resets work. As I mentioned at the start of the chapter on resets I like to do the doors first, the objects in rooms second, then the NPCs. again this is nothing we force you to do but I find it helps me keep track of my items and NPCs.

With that in mind we will start by resetting our doors. In the zone there is two doors. One is a regular door that is closed when the mud reboots and the other is a hidden and locked door when the mud reboots. We will not block the rooms in and show you them again but if you want to you can see the rooms in Section 8.5. the resets for these doors would look like this.

```
//Office door reset
door hallway EAST {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
door office WEST {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
```

```
//secret door reset
door office SOUTH {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED,EX_HIDDEN}
door portal_room   NORTH {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
```

> **Note:** Both sides of the door don't have to have the exact same flags the door in the office is hidden but the one on the ship is not.

The next thing to build resets for is the two items we are loading directly into rooms and their contents if they have any. The two items we are loading into rooms are the board and the weapons locker. I am just going to stick the board in the main chamber and the weapons locker in the office. The reset for these two items looks like this.

```
load info_board into chamber

load wpn_locker into office
  {
  load w_stiletto
  }
```

Notice we also loaded a stiletto into the weapons locker and it will only load once a reboot since the cabinet will never be removed and unless the cabinet reloads the stiletto will not reload.

finally we get to the NPcs and their equipment. We only have 3 NPCs in our zone so it shouldn't be any problem especially since we don't have that much close. I am going to load the dragon into the ship on a percentage chance basis and then load the Janitor into the station so he can get to cleaning it up. Finally I will load Bob into the office so he can sit and count his money.

```
load bob into office
  {
  equip pol_plate WEAR_BODY
  }

  load janitor into chamber
    {
 equip pol_plate WEAR_BODY
 }

 random 50
   {
   load bldragon
       {
    load maskwa
    }
  }
```

Only a couple final things to point out. It doesn't matter if you load the same type of plate or same type of clothing on every NPC. It would look pretty silly if everyone was wearing the same clothes but I wanted to make sure you understood you didn't have to make one set for everyone. Also you don't have to equip wearable things you can load them into the inventory like we did with the ring on the dragon. That about covers it all. The resets are now done and we can now put them all together with the zone and complete our dragon station.

8.5. The complete dragon station

```
#include <composed.h>
%zone dragonst
lifespan 20
reset RESET_ANYHOW
creators {"whistler"}

notes
"This is the dragon station I shortened it to dragonst for ease in
loading.  If you have  any questions email me at whistler@valhalla.com"

help
"Not sure what could help you now.  You are stuck on one of the
weirdest space stations you have ever seen and you smell burning
sulfur."

%rooms

chamber
title "The middle chamber of the station"
descr
"This chamber seems to have the entire station rotating around it.  It is
unbelievably large the ceiling seems to be a good 200 meeters high and
the room is perfectly cubic. Small human size ornate chairs with dragon
designs scrawled on the arms and back are arranged in a triangle like
setting with one large chair at the front.  This must be where all
station meetings are held. large pictures cover the walls depicting
dragons in all kinds of situations.  large passages lead of to the west
and the east.."

extra {"chair","chairs"}
"The chairs are made of some metal you don't recognize and every inch is covered
with some kind of dragon."

extra  {"dragon picture","picture"}
"Thousands of dragons dot the skies of this rather life like picture.  In the
center you see something move.  It looks to be a little green dragon."

extra{"green dragon","dragon","green"}
"An intellegence looking dragon is sitting perched on a large chair watching you."

movement SECT_INSIDE
```

```
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}


west to disposal_room descr
"You see a small room.";

east to hallway descr
"You see what looks to be a hallway.";

end

hallway
title "Module tunnel"
descr "The hallway is about 50 meters long and around 100 meters from
side to side and top to bottom.  The hallway seems to be dust free.  The
walls and the floors seem to be made out of the same sterile
metal-plastic that all space agencies uses.  There are large plate glass
windows that open up into space.  The hallway is filled with a dim light
that seems to come from everywhere yet no where all at once.  You notice
a glimmer of bright light coming from the windows.  To the east you see
an air lock and to the west the hallway opens up into a larger room."

extra {"windows","window"}
"Your eyes are drawn to a large ship lit up with running lights sitting
about 1 kilometer from the station."

extra{"floor","walls","wall"}
"Well what can be said it looks to be in perfect condition.  What else would
you want to know?"

extra {"large ship" ,"ship"}
"The ship looks really big and is shaped like a dragon.  The scales
sparkle and seem to be multiple colors."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

west to chamber descr
"The hallway opens up into a chamber.";

east to office descr
"You see what looks to be an office."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

end

office
title "The station office"
descr
"Large paintings fill the walls of this part of the station.  The room
```

```
is as large as the other rooms big enough for Dragons to lounge while
still having a desk in one corner small enough for a humanoid.  The
floor along the north wall is lined with some kind of fabric and seems
very soft to walk on, it may be some kind of dragon lounge judging by
how large an area it covers.  There is a passage to the west."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"paintings","painting"}
"The paintings are of many dragons and riders in all kinds of tasks from
combat to look out.  All the figures seem to be staring at a staff
being held by a depiction of a wizard on the south wall."

extra {"wizard","staff"}
"The wizard has his hand stretched out and it seems there is a place
you can almost grab the staff. Maybe if you searched the staff you would
find it."

extra {"desk"}
"Its a desk alright but there doesn't seem to be any drawers and it
seems totally empty."

extra{"fabric"}
"Wussshhhhh you bound across the comfortable floor wasn't that fun."

west to hallway descr
"You see what looks to be a hallway."
keyword {"air lock door","air lock","door"}
open {EX_OPEN_CLOSE, EX_CLOSED};

SECRET_DOOR_DIFFICULTY(SOUTH, 50)
south to portal_room descr
"You see what looks to be a portal room."
keyword {"air lock door","air lock","staff","door"}
key nokey
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED,EX_HIDDEN};

end

 portal_room
 title "Green field room"
 descr
 "Like the other rooms on the station this one is large enough for
 dragons to comfortably fit in.  The strange thing about this room though
 is it is totally empty except for a green field right in the center.
 there is a door that leads to another room to the north."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}
```

```
 extra {"green field","field"}
 "The field looks to be a green fog shifting and churning as you watch.
 if you are nuts you could probably enter it."

 north  to office descr
 "You see what looks to be an office."
 keyword {"air lock door","air lock","door"}
 key nokey
 open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED};

 //A link to the portal is also here from room_port
 end

 ship_port
 names{"green field", "field"}
 title "Green field"
 descr
 "Green Mist swirls about you."

 movement SECT_INSIDE
 ALWAYS_LIGHT
 flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

 in ship

 dilcopy force_move@function(
 //Time to activation
 4,
 //room and act
 "portal_room@dragonst!You feel your body dissolving for lack of a better
 description.&You appear on the deck of a ship.",
 //True or False for randomizing or not
 FALSE);


 end

room_port
names{"green field", "field"}
title "Green field"
descr
"Green Mist swirls about you."
movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

in portal_room

dilcopy force_move@function(
//Time to activation
4,
//room and act
"ship@dragonst!You feel your body dissolving for lack of a better
```

```
description.&You appear on the deck of a ship.",
//True or False for randomizing or not
FALSE);


end

disposal_room
title "Red field room"
descr
"Like the other rooms on the station this one is large enough for
dragons to comfortably fit in.  The strange thing about this room though
is it is totally empty except for a red field right in the center.
there is a door that leads to another room to the east."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"red field","field"}
"The field looks to be a red fog shifting and churning as you watch.
if you are nuts you could probably enter it."

east to chamber descr
"You see the main chamber.";

 //A link to the portal is also here from dis_port
end

dis_port
names {"red field","field"}
title "Red field"
descr
"Red Mist swirls about you."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}
dilcopy force_move@function(
//how fast to force move in seconds
4,
//room to force move to and act
"deathspace@dragonst!You feel your body dissolving for lack of a better description.",
//true or false random move or not
0);
in disposal_room

end

ship
title "War dragon"
descr
"Blue light softly glows from con duets that line the walls of this ship.
```

```
The floors beside the east and west wall have what looks to be soft
fabric covering.  The south wall has small controls that seem to be made
for humanoids with two small chairs that look to be pilot seats.  view
portals are about 50 meters up the side of the ship on the west and east
wall and some kind of electronic screen covers the south wall.  The ship
seems to be a one room ship but there is a green field by the north
wall."

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

extra {"view port"}
"Sorry your not 50 meters tall maybe it is made for a dragon?"

extra {"view screen","screen"}
"It seems to be the pilots view screen but you can't seem to see a way
to turn it on."

extra {"controls","control"}
"The controls are in some weird language and your afraid if you start
pushing buttons you might rocket in to the station or worse slam into
a planet."

extra {"soft fabric","fabric"}
"It looks to be a dragon lounge area."

 //A link to the portal is also here from ship_port
end

deathspace
title"Open space"
descr
"You see the ship and the station far off in the distance and you are in Space!"

movement SECT_INSIDE
ALWAYS_LIGHT
flags {UNIT_FL_NO_WEATHER, UNIT_FL_INDOORS}

dilcopy death_room@function (
//how often is damage done 4 would be 1 second
4,
//damage
400,
//act for the damage.
"You realize to late that was the trash disposal transporter and you feel your lungs explod


end

%mobiles

bldragon
```

```
title "a black dragon"
descr "A big ugly black dragon is clawing the ground here."
names {"big ugly black dragon","ugly black dragon","big black dragon",
"black dragon","dragon"}

extra {}
"The black dragons scales glitter like black granite that has been
polished for years by water.  He has a large neck and huge bat like
wings.  his eyes watch you as you stand before him.  One claw seems to be
tapping slightly on the ground as if the dragon is waiting for
something."

extra {"eye","eyes"}
"The dragons eyes seem to follow you no matter where you go in the room
nothing seems to escape the dragons attention."

extra {"claws","claw"}
"The claw is big black and it looks very deadly.  It seems like the
dragon has two sets of 4 large claws and 2 sets of 4 smaller claws which
to say means the claws are about the size of short swords and long
swords."

extra {"scales","scale"}
"Its a scale!  Haven't you ever seen a dragon before!"

extra {"bat wings","wings"}
"The dragon sees you looking and flaps his wings creating one heck of a
wind blast."

M_DRAGON_BLACK_OLD(SEX_MALE)

end

janitor
names {"ugly janitor", "janitor", "hobgoblin"}
title "an ugly janitor"
descr "an ugly janitor is walking around, cleaning up."

extra{}
"This ugly green thing looks more goblin than hobgoblin but he seems intent
on cleaning everything around him."

M_AVG_HOBGOBLIN(6, SEX_MALE)

// he is sort of good for cleaning so much
alignment 900

//give him some money
money 5 IRON_PIECE

dilcopy janitors@function(15);
```

```
// only want him cleaning the station
dilcopy wander_zones@function("dragonst", 20, 1, 1);

end

bob

names {"Bob"}
title "Bob"
descr "Bob the Banker is here, sitting behind the counter."
extra {}
"He has a very serious look on his face."

// define from composed.h
M_SHOP_KEEPER(4, SEX_MALE, RACE_HUMAN)

//discourage people from killing banker
exp -500

flags {UNIT_FL_NO_TELEPORT}

special SFUN_BANK
end

%objects

info_board

title "a merchant information board"
descr "A merchant information Board is mounted on a wall here."
names {"merchant information board","information board","merchant
board","board"} extra {} "A large flashy black steal board."

MATERIAL_METAL("A very fine quality black steel")
type ITEM_BOARD
dilcopy board@boards("info","","rem_res@boards","",100);

end

w_stiletto
title "a stiletto"
names {"deadly looking stiletto","deadly stiletto","stiletto", "dagger"}
descr "A deadly looking stiletto has been left here."

MATERIAL_METAL("A very fine quality steel")
manipulate {MANIPULATE_TAKE, MANIPULATE_WIELD}
WEAPON_DEF(WPN_DAGGER, 1, 2)
weight 2
cost 2 GOLD_PIECE
rent 1 COPPER_PIECE

SKILL_TRANSFER(SKI_BACKSTAB, 2)
dilcopy abi_restrict@function (ABIL_DEX,10,0,25,"");
```

```
extra{}
"This looks like a thieves dream come true. "

extra {"$identify"}
"The stiletto looks magical in nature and seems really sharp.  You can
tell if you wield it you would be able to backstab someone really easy."

extra{"$improved identify"}
"The stiletto gives you a magical bonus of +1 and has a quality of +2.
It also raises your back stabbing skill  by 2%.  You have to have at
least 10% in dex before you can wield this magical weapon."

end

wpn_locker

title "a weapons locker"
names {"weapons locker","weapon locker","locker"}

descr "a small weapons locker hangs on the wall here."
manipulate {MANIPULATE_ENTER}
CONTAINER_DEF(500)
open {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
weight 400
MATERIAL_METAL("A very fine quality steel")

extra {}
"It is an ordinary weapons locker that looks like it holds any illegal
weapons that are taken on the station."

end

pol_plate
names {"polished breast plate","polished plate","breast plate","plate"}
title "a polished breast plate"
descr "A polished breast plate has been left here."
extra{}
"This is one shiny plate it seems to be made out of one perfect piece of
metal.  There doesn't even seem to be any markings of owner ship." manipulate
 {MANIPULATE_TAKE, MANIPULATE_WEAR_BODY} ARMOUR_PLATE(5,9)

MATERIAL_METAL("A high luster silver colored metal")

manipulate {MANIPULATE_TAKE, MANIPULATE_WEAR_BODY}
ARMOUR_DEF(ARM_PLATE,5,9)
dilcopy abi_restrict@function (ABIL_STR,40,0,25,"");
cost 2 GOLD_PIECE
rent 3 COPPER_PIECE
weight 25

extra{"$identify"}
"This is a high quality plate with a magical feeling."
```

```
extra{"$improved identify"}
"The plate has a magical bonus to your defence of a +5 and a quality of
+9.  You need 40% in strength to be able to wear it."
end

liq_ration
names {"red bag", "bag", "wine"}
title "a red bag"
descr "A red bag has been gently placed here."

MATERIAL_ORGANIC("a soft plastic")
manipulate {MANIPULATE_TAKE}

LIQ_WINE(1,2,2,0)
cost 2 IRON_PIECE
extra {}
"A small label reads Tassel Grove's finest.  Year 321"

extra {"$identify"}
"Its the special wine from Tassel grove a small halfling village on the
planet Valhalla.  It seems like a great vintage wine."

end

beef_stick

title "a tough leathery stick"
descr "A tough leathery looking stick is laying here."
names {"tough leathery stick","tough leather stick","leathery stick",
"leather stick","tough stick","stick"}

extra {}
"This has the word BEEF burnt into it."

manipulate {MANIPULATE_TAKE}
FOOD_DEF(5,0)
weight 1
cost 1 COPPER_PIECE
MATERIAL_ORGANIC("tough beef")
end

  maskwa

names {"maskwa platinum ring", "platinum ring","maskwa ring","maskwa","ring"}
title "a Maskwa ring"
descr "A Maskwa platinum ring is laying here."
MATERIAL_METAL("Platinum, and other precious metals")
extra {}
"The ring has a large bear head.  Could this be the legendary head of
Maskwa?  Any thing formed with its head on it is said to strengthen the
wearer."
type ITEM_WORN
```

```
manipulate {MANIPULATE_TAKE, MANIPULATE_HOLD, MANIPULATE_WEAR_FINGER}
cost 100 COPPER_PIECE
rent 50 IRON_PIECE
weight 1
STR_TRANSFER(+1)
end

%reset

//Office door reset
door hallway EAST {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}
door office WEST {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}


//secret door reset
door office SOUTH {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED,EX_HIDDEN}
door portal_room   NORTH {EX_OPEN_CLOSE, EX_CLOSED,EX_LOCKED}


load info_board into chamber

load wpn_locker into office
  {
  load w_stiletto
  }

  load bob into office
  {
  equip pol_plate WEAR_BODY
  }

  load janitor into chamber
    {
 equip pol_plate WEAR_BODY
 }



 random 50
   {
   load bldragon
       {
    load maskwa
    }
  }

%end
```

Chapter 9. Color and formatting codes

Now that you have got a handle on how to build rooms, objects, and NPCs, we will now give you the ability to format text the way you want it formatted and color it as well. Currently the VME doesn't support all the ASCII characters but if we get enough people wanting this ability it will be added.

9.1. The escape character

When working with colors or formatting there are always two parts to a formatting command. The first part we call the escape character which is the '&' character. Thus all formatting and color commands would look as follows:

```
&<color or formating command>
```

9.2. Formatting codes

As you may have noticed the string fields on the VME are formatted in an english paragraph style. What that means is all text is formatted with he following rules:

- All leading blanks are stripped away. For room descriptions 3 leading spaces are added. This way we ensure a consistent formatting of the displayed room descriptions.
- Spaces and blanks (newlines) are contracted to single spaces yielding a correctly 'wrapped' text for any sentence.
- If a '.' is encountered followed by a blank, a total of two spaces are inserted after the '.'.

These formatting rules are great for normal descriptions and extras but what if you want to make a sign or a map. You don't always want the text to be rapped, if the server rapped your sign it would turn out looking like a jumble of punctuation in the form of a paragraph. Table 9-1 contains a list of all formatting characters and a short description of them. Section 9.3 contains a more in depth discussion of each format command with examples and Section 9.4 deals with the color commands.

Table 9-1. Formatting and color codes

Code	Description
&	Used to make an & character instead of an escape code.
l	Text from this point forward will not be formatted. It will be interpreted literally with newlines, spaces, etc. Useful when making a sign or a map.

Code	Description
f	Text from this point forward will be formatted. This option is the reverse of &l and is default on any section of text.
s<#>	Force a singe space character (or <#> if specified, may come in handy, instead of having to toggle formatting).
n	Force a new line, very handy instead of toggling the &l formatting switch.
h	Clears the screen if the terminal of the user supports it.
x	A file new line used for split so that you can split a file into lines
c<color>	Set the foreground color.
b<color>	Set the background color.
[<color>]	Set a preset foreground and background color.

9.3. Formatting code descriptions and examples

&&

If you want a single '&' you must let the VME know that you don't want a formatting or a color code. You do this by doubling the '&' sign. The following is a couple examples:

text	Result
&&	&
&&&&	&&
&&&&&&	&&&

&l

When you want to turn off the formatting you use this formatting code. Everything after the '&l' will be shown exactly as you put it in the string.

```
&l
*       *
 *     *
  *   *
   * *
    *
   * *
  *   *
 *     *
*       *
```

&f

The formatted text as we have already said is default. If you want to turn the formatted text back on after some literal text you will have to use the '&f' code. The following is an example of some literal text followed by a short bit of formatted text.

```
&l
*     *
 *   *
  * *
   *
  * *
 *   *
*     *

&fThe X marks the spot!
```

&s<#>

If you want to input extra spaces in a sentence with out using the '&l' you can add them one at a time or multiple by using the '&s' code.

```
This sentence has 10 spaces&s10before the first word before.
```

&n

If you want to input some blank lines with out using the literal code you can add a '&n'for each line you want.

```
This sentence&n&n&n would look like this:


This sentence


would look like this:
```

&x

The line break is made for use with the DIL language. You will not need it to do regular text formatting. It was added so a DIL could split a string that is loaded from a file. If you don't understand the following example don't worry it is explained more in the DIL reference.

```
mystrlist:=split(string,"&");
```

&h

On terminals that can handle it the '&h' will clear the screen. If you wanted a sign that would clear the screen before displaying when a character looked at it would look like this.

```
&h&l
*     *
 *   *
  * *
   *
  * *
 *   *
*     *
```

```
&fThe X marks the spot!
```

9.4. Color code descriptions and examples

&c and &b

In order to allow you to change the colors there are two codes. One is for the foreground color (&c) and the other is for the background color (&b). '&c' is used with one or two arguments depending on brightness, while the '&b' s only used with one because it has only one brightness. They both have the forms as follows:

```
&c<bright><color>
&b<color>
```

It is important to set both the foreground and background color because if a player has his default background color set to blue and you use blue as a foreground color it will make the letters invisible to the player. It is also important to set the colors back to the default color when done. this is done by using the following command:

```
&[default]
```

> **Note:** The '&[default]' command will be described in the next section. It is enough to know for now that it will return the players colors to their default colors.

Before we give some color examples we should now define the symbols for brightness and the symbols for each color and what they are.

Table 9-2. Colors

Code	Color
n	Black
r	Red
g	Green
y	Yellow
b	Blue
m	Magenta
c	Cyan
w	White

Table 9-3. Sample Color codes

Code	Resulting Colors

Code	Resulting Colors
&cb&bw	
&+g&bn	
&c+w&r	
&c+w&bn	

&[<color>]

As we have said in the previous section if you are not careful you can make your text not visible by the player by setting the foreground to the same color as the background. In order to make it possible for you to easily change how the color looks and to even match it with the way players have their colors set already we have created colors that the players can set. and you can use. The VME comes with a default list of colors which can be added to by either the `color.def` or even by a DIL program on line. The default colors are as follows:

death	default	exit
group	hit_me	hit_opponent
hit_other	immort_descr	immort_title
log	miss_me	miss_opponent
miss_other	nodam_me	nodam_opponent
nodam_other	npc_descr	npc_title
obj_descr	obj_title	pc_descr
pc_title	prompt	respond
room_descr	broom_title	say_other
say_self	shield_me	shield_opponent
shield_other	shout_other	shout_self
social_other	social_self	spells
tell_other	tell_self	time
weather	whisper	who
who_guild	who_inv	who_name
who_title	wiz	xpgain

To use these colors all you have to do is use the following formatting command:

```
&[color]
```

The color that will be shown is the color that the player has set for the color in question. If for example the player has his or her 'death' color set to bright red with a black background and you have a description as follows:

```
descr
"This is a &[death]death&[room_descr]room"
```

The description would be in the players 'room_descr' color while the word death would be in his or hers 'death' color. You should note we had to set the color back to the room description color so that the rest of the description was not in the 'death' color.

To change the players color to the default out put color which is the color that is used when no color is specified by the server then you use 'default'. You probably won't use this in normal zone building but it is very important to know it exists when you start making spells, skills, and commands with DIL

Chapter 10. The DIL section

When I first thought of writing this manual I had planned to leave DIL totally out of it. The DIL language always confuses new Builders and complicates teaching simple rooms, objects, and NPCs. It became clear to me though that DIL is such a part of the server that I at least had to mention it hear so builders would know where to look and what DIL could do for them when they got to the point where they were ready to use it.

10.1. What is DIL?

DIL is short for, Data-based instructional language. Not to be confused with Database since DIL is nothing like a database. Data-based means that the language works on a fixed set of units like objects, NPCS, and rooms and is designed to give them a life of their own. Unlike on many of the mud servers on the internet DIL is not an interpreted language it is a compiled language which gives you the user much more safe guards against crashes and slow code.

DIL also provides a full set of data types to allow you to do calculations or store information from players. The DIL language can even deal with file access and it can add fields and information to the players if needed. In short the VME server has its own internal functional language that will allow you to do just about anything you want to.

10.2. What can DIL be used for?

...

It is hard to explain what all DIL can be used for with out just writing a list of things that have already been done in DIL so here is the list.

- All spells
- 60% of the commands and skills and growing
- Administrator commands
- Movement commands
- look commands
- 200+ quests and growing
- Message boards
- Mail system
- Clan system
- Automated Newbie guides

- object restrictions
- Death Sequence
- Magical combat system
- NPC agressive functions
- personalized Familiars
- personalized pets
- Deck of cards
- Chess board
- dice
- online AD&D game playing system
- Communication channels
- automated wedding chapel

Realize this is only a small list of things that can be done in DIL. In the future we hope to be able to add the ability to easily change combat and all the character update features using DIL. These things can be currently done with DIL but it takes a lot of knowledge and work which we hope to simplify.

10.3. Where do I get more information on DIL

The online DIL reference guide is the most authoritative guide currently for DIL. It can be found at http://www.valhalla.com and follow the builders links.

In the near future we hope to have an entire new DIL manual that will teach, a person who has never coded all the way to people who are already professional coders, to use DIL. For now if you can not find what you need in the DIL reference manual you can join the DIL email list at <dil-request@valhalla.com> or you can send a mail to <whistler@valhalla.com>. Until and even after the new DIL manual is written we will always try to help you as much as possible while you are getting started. It is important that you at least try and look through the DIL reference before asking a lot of questions because many of the questions may be answered already.

Appendix A. VMC command line options

The argument string is processed from left to right. Options may appear between filenames, but it should be noted that an option only takes effect when it is encountered. In most cases, options should be placed to the left of the filename arguments.

Option	Description
-m	The location of the money file. Normally etc/money.
-v	Verbose compiler output shows much more information about objects and NPC
-M	Make option. Only compile source files if they have been modified more recently than the corresponding output files.
-p	Pre process the file only
-s	Suppress the generation of output files.

These options are not available to the email or FTP compiler, but normally you should not bother with them - they are probably set automatically.

Appendix B. Reserved keyword listing

ability	affect	alignment	applyf	armour	attack	bits	bright
capacity	complete	cost	creators	data	default	defensive	descr
dilcopy	door	duration	end	equip	exit	exp	extra
firstf	flags	follow	height	help	hit	id	in
inside_descr	into	key	keyword	lastf	level	lifespan	light
link	load	local	mana	manipulate	max	minv	money
movement	names	nop	notes	npcflags	offensive	open	outside_descr
position	purge	race	random	remove	rent	reset	romflags
sex	special	speed	spell	string	tickf	time	title
to	type	value	weapon	weather	weight	zonemax	

Appendix C. Race Definitions in values.h

The following list was extracted from the `values.h`

```
#define RACE_HUMAN           0     /* PC race */
#define RACE_ELF             1     /* PC race */
#define RACE_DWARF           2     /* PC race */
#define RACE_HALFLING        3     /* PC race */
#define RACE_GNOME           4     /* PC race */
#define RACE_HALF_ORC        5
#define RACE_HALF_OGRE       6
#define RACE_HALF_ELF        7
#define RACE_BROWNIE         8
#define RACE_GROLL           9
#define RACE_DARK_ELF       10
#define RACE_SKAVEN        120
#define RACE_GNOLL         121
#define RACE_GOBLIN        122
#define RACE_HOBGOBLIN     123
#define RACE_KOBOLD        124
#define RACE_NIXIE         125
#define RACE_NYMPH         126
#define RACE_OGRE          127
#define RACE_ORC           128
#define RACE_SATYR         129
#define RACE_FAUN          130
#define RACE_SPRITE        131
#define RACE_DRYAD         132
#define RACE_LEPRECHAUN    133
#define RACE_PIXIE         134
#define RACE_SYLPH         135
#define RACE_HERMIT        136
#define RACE_SHARGUGH      137
#define RACE_GIANT         138
#define RACE_WARDEN        139   /* Warden???             */
#define RACE_TROLL         140
#define RACE_NORSE_GOD     142   /* Hmmmm. probably need better categories */
#define RACE_MERMAID       145
#define RACE_SIREN         146
#define RACE_NAIAD         147
#define RACE_MERMAN        148
#define RACE_MINOTAUR      149
#define RACE_YETI          150
#define RACE_OTHER_HUMANOID 999

#define RACE_BEAR          1000
#define RACE_DOG           1001
#define RACE_WOLF          1002
#define RACE_FOX           1003
#define RACE_CAT           1004
#define RACE_RABBIT        1005
```

```
#define RACE_DEER           1006
#define RACE_COW            1007
#define RACE_HARE           1008
#define RACE_GOAT           1009
#define RACE_EAGLE          1010
#define RACE_PIG            1011

#define RACE_DUCK           1100  /* This will interest the biologists... */
#define RACE_BIRD           1101  /* This will interest the biologists... */
#define RACE_RAT            1102
#define RACE_HORSE          1103
#define RACE_BADGER         1104
#define RACE_SKUNK          1105
#define RACE_BOAR           1106
#define RACE_MOUSE          1107
#define RACE_MONKEY         1108
#define RACE_PORCUPINE      1110
#define RACE_ELEPHANT       1112
#define RACE_CAMEL          1113
#define RACE_FERRET         1114
#define RACE_VULTURE        1115
#define RACE_SQUIRREL       1116
#define RACE_OWL            1117
#define RACE_LEMURE         1118  /* Half-monkey (Makier) */
#define RACE_ELK            1119  /* Larger deer (Whapiti-deer) */
#define RACE_LION           1120
#define RACE_TIGER          1121
#define RACE_LEOPARD        1122
#define RACE_OTHER_MAMMAL   1999

#define RACE_TREE           2000
#define RACE_VINE           2001
#define RACE_FLOWER         2002
#define RACE_SEAWEED        2003
#define RACE_CACTUS         2004

#define RACE_OTHER_PLANT    2999

#define RACE_MAGGOT         3000
#define RACE_BEETLE         3001
#define RACE_SPIDER         3002
#define RACE_COCKROACH      3003
#define RACE_BUTTERFLY      3004
#define RACE_ANT            3005
#define RACE_WORM           3006
#define RACE_LEECH          3008
#define RACE_DRAGONFLY      3009
#define RACE_MOSQUITO       3010

#define RACE_OTHER_INSECT   3999

#define RACE_LIZARD         4000
#define RACE_SNAKE          4001
```

```
#define RACE_FROG           4002
#define RACE_ALLIGATOR      4004
#define RACE_DINOSAUR       4005
#define RACE_CHAMELEON      4006
#define RACE_SCORPION       4007
#define RACE_TURTLE         4008
#define RACE_BAT            4009
#define RACE_TOAD           4010

#define RACE_OTHER_REPTILE  4999

#define RACE_CAVE_WIGHT     5001  /* Some kind a creature... */
#define RACE_UR_VILE        5002  /* Some kind a creature... */
#define RACE_STONE_RENDER   5003  /* Some kind a creature... */
#define RACE_VAMPIRE        5005
#define RACE_SLIME          5006
#define RACE_WYRM           5007
#define RACE_AUTOMATON      5008
#define RACE_UNICORN        5009

#define RACE_DRAGON_MIN     5010  /* For use with special object */
#define RACE_DRAGON_BLACK   5010
#define RACE_DRAGON_BLUE    5011
#define RACE_DRAGON_GREEN   5012
#define RACE_DRAGON_RED     5013
#define RACE_DRAGON_WHITE   5014
#define RACE_DRAGON_SILVER  5015
#define RACE_DRAGON_TURTLE  5016
#define RACE_DRAGON_LAVA    5017
#define RACE_DRAGON_SHADOW  5018
#define RACE_DRAGON_LIZARD  5019
#define RACE_DRAGON_MAX     5020  /* For use with special object */

#define RACE_LESSER_DEMON   5020  /* Approx. Level < 100          */
#define RACE_GREATER_DEMON  5021  /* Approx. Level > 100          */
#define RACE_SERVANT_DEMON  5022  /* Approx. < level 20           */
#define RACE_PRINCE_DEMON   5023  /* Almost god, max level 149 (no more!) */
#define RACE_LESSER_DEVIL   5025  /* Approx. Level < 100 */
#define RACE_GREATER_DEVIL  5026  /* Approx. Level > 100 */
#define RACE_SHADOW_DEVIL   5027
#define RACE_ARCH_DEVIL     5028

#define RACE_MEDUSA         5030
#define RACE_WINGED_HORSE   5031
#define RACE_GARGOYLE       5033
#define RACE_GOLEM          5034
#define RACE_YOGOLOTH       5035
#define RACE_MIST_DWELLER   5036

#define RACE_WEREWOLF       5037
#define RACE_WERERAT        5038

#define RACE_ELEMENTAL_AIR   5040
```

```
#define RACE_ELEMENTAL_EARTH 5041
#define RACE_ELEMENTAL_FIRE  5042
#define RACE_ELEMENTAL_FROST 5043
#define RACE_ELEMENTAL_WATER 5044
#define RACE_ELEMENTAL_LIGHT 5045

#define RACE_DEVOURER       5600
#define RACE_DANALEK        5601

#define RACE_FAMILIAR       5900 /* Weirdo race... */
#define RACE_OTHER_CREATURE 5999

#define RACE_ZOMBIE         6000
#define RACE_LICH           6001
#define RACE_GHOUL          6002
#define RACE_SKELETON       6003
#define RACE_GHOST          6004
#define RACE_SPIRIT         6005
#define RACE_MUMMIE         6006
#define RACE_BANSHEE        6007
#define RACE_NAGA_SOUL      6008
#define RACE_OTHER_UNDEAD   6999

#define RACE_CRAB           7000
#define RACE_SAND_SPIDER    7002
#define RACE_RIVER_LEECH    7003
#define RACE_SAND_CRAWLER   7004
#define RACE_SEA_HORSE      7005
#define RACE_SHARK          7006
#define RACE_LAMPREY        7007
#define RACE_MANTA_RAY      7008
#define RACE_CLIFF_HUGGER   7009
#define RACE_ALGAE_MAN      7010
#define RACE_WHELK          7011
#define RACE_OYSTER         7012
#define RACE_KRAKEN         7013
#define RACE_CAVE_FISHER    7014 /* Tiamat: lobster / spider breed */
#define RACE_OCTOPUS        7015
#define RACE_WHALE          7016
#define RACE_DOLPHIN        7017
#define RACE_EEL            7018

#define RACE_FISH           7998
#define RACE_OTHER_MARINE   7999
```

Appendix D. weapon definitions in values.h

The following list was extracted from the `values.h`

```
#define WPN_BATTLE_AXE    7  /* Two Handed */
#define WPN_HAND_AXE      8
#define WPN_WAR_MATTOCK   9  /* Two Handed */
#define WPN_WAR_HAMMER   10
#define WPN_GREAT_SWORD  11  /* Two Handed */
#define WPN_SCIMITAR     12
#define WPN_KATANA       13
#define WPN_FALCHION     14
#define WPN_KOPESH       15
#define WPN_BROAD_SWORD  16
#define WPN_LONG_SWORD   17
#define WPN_RAPIER       18
#define WPN_SHORT_SWORD  19
#define WPN_DAGGER       20
#define WPN_BATTLE_MACE  21  /* Two Handed */
#define WPN_MACE         22
#define WPN_BATTLE_CLUB  23  /* Two handed */
#define WPN_CLUB         24
#define WPN_MORNING_STAR 25
#define WPN_FLAIL        26
#define WPN_QUARTERSTAFF 27
#define WPN_SPEAR        28
#define WPN_HALBERD      29
#define WPN_BARDICHE     30
#define WPN_SICKLE       31
#define WPN_SCYTHE       32  /* Two handed */
#define WPN_TRIDENT      33
#define WPN_FIST         34
#define WPN_KICK         35
#define WPN_BITE         36
#define WPN_STING        37
#define WPN_CLAW         38
#define WPN_CRUSH        39
#define WPN_WHIP         40
#define WPN_WAKIZASHI    41
#define WPN_BOW          42 /* Here down to Staff are Rangers Guild Jan 98 */
#define WPN_CROSSBOW     43
#define WPN_SLING        44
#define WPN_FIGHTING_STAFF 45 /* Two handed */
#define WPN_SABER        46
#define WPN_CUTLASS      47
#define WPN_MACHETE      48
#define WPN_LANCE        49
#define WPN_SHOCK_LANCE  50
#define WPN_PIKE         51
#define WPN_GREAT_AXE    52
#define WPN_BATTLE_SWORD 53
```

Appendix E. Liquid macros file

```
#define LIQ_WATER(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("clear", WEIGHT,CAPACITY,INSIDE,10,1,0,POISON)
#define LIQ_BEER(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("brown", WEIGHT,CAPACITY,INSIDE,5,2,3,POISON)
#define LIQ_WINE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("clear", WEIGHT,CAPACITY,INSIDE,5,2,5,POISON)
#define LIQ_ALE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("brown", WEIGHT,CAPACITY,INSIDE,5,2,2,POISON)
#define LIQ_DARK_ALE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("dark brown", WEIGHT,CAPACITY,INSIDE,5,2,1,POISON)
#define LIQ_WISKEY(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("golden",WEIGHT ,CAPACITY,INSIDE,4,1,6,POISON)
#define LIQ_WHISKY(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("golden",WEIGHT ,CAPACITY,INSIDE,4,1,6,POISON)

#define LIQ_LEMONADE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("red", WEIGHT, CAPACITY, INSIDE, 8, 1, 0,POISON)
#define LIQ_FIREBRT(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("green", WEIGHT,CAPACITY,INSIDE,0,0,10,POISON)
#define LIQ_LOCALSPC(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("clear", WEIGHT, CAPACITY, INSIDE, 3, 3, 3,POISON)
#define LIQ_SLIME(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("light green", WEIGHT,CAPACITY,INSIDE,8,4,0,POISON)
#define LIQ_MILK(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("white", WEIGHT, CAPACITY, INSIDE, 6, 3, 0,POISON)
#define LIQ_TEA(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("brown", WEIGHT, CAPACITY, INSIDE, 6, 1, 0,POISON)
#define LIQ_COFFEE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("black", WEIGHT, CAPACITY, INSIDE, 6, 1, 0,POISON)
#define LIQ_COFFE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("black", WEIGHT, CAPACITY, INSIDE, 6, 1, 0,POISON)

#define LIQ_BLOOD(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("red", WEIGHT, CAPACITY, INSIDE, -1, 2, 0, POISON)
#define LIQ_SALTWAT(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("clear", WEIGHT, CAPACITY, INSIDE, 2, 1, 0, POISON)
#define LIQ_COKE(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("black", WEIGHT, CAPACITY, INSIDE, 5, 1, 0,POISON)
#define LIQ_VODKA(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("clear", WEIGHT,CAPACITY,INSIDE,0,0,10,POISON)
#define LIQ_BRANDY(WEIGHT,CAPACITY,INSIDE,POISON) \
    LIQ_DEF("golden", WEIGHT,CAPACITY,INSIDE,4,1,6,POISON)
```

Appendix F. Complete magical transfers macros listing

This listing of macros was taken from `wmacros.h`. When building your objects you should check the macros file to make sure you have the most up to date macros.

```
#define CHAR_FLAG_TRANSFER(_MFLAGS) \
flags {UNIT_FL_MAGIC}       \
affect                          \
   id ID_TRANSFER_CHARFLAGS     \
   duration -1                  \
   data[0] _MFLAGS              \
   firstf TIF_EYES_TINGLE       \
   tickf TIF_NONE               \
   lastf TIF_EYES_TINGLE        \
   applyf APF_MOD_CHAR_FLAGS;

/* skill MUST be one of SKI_XXX, amount in -10 to +10 */
#define SKILL_TRANSFER(skill, amount) \
flags {UNIT_FL_MAGIC}       \
affect                      \
   id ID_SKILL_TRANSFER     \
   duration -1              \
   data[0] skill            \
   data[1] amount           \
   firstf  TIF_SKI_INC      \
   tickf   TIF_NONE         \
   lastf   TIF_SKI_DEC      \
   applyf  APF_SKILL_ADJ;

/* weapon MUST be one of WPN_XXX, amount in -10 to +10 */
#define WEAPON_TRANSFER(weapon, amount) \
flags {UNIT_FL_MAGIC}       \
affect                      \
   id ID_WEAPON_TRANSFER    \
   duration -1              \
   data[0] weapon           \
   data[1] amount           \
   firstf  TIF_WPN_INC      \
   tickf   TIF_NONE         \
   lastf   TIF_WPN_DEC      \
   applyf  APF_WEAPON_ADJ;

/* spell MUST be one of SPL_XXX, amount in -10 to +10 */
#define SPELL_TRANSFER(spell, amount) \
flags {UNIT_FL_MAGIC}       \
affect                      \
   id ID_SPELL_TRANSFER     \
   duration -1              /* Must be permanent in the object */  \
   data[0] spell            /* It is a spell SPL_XXX transfer  */  \
```

```
   data[1] amount            /* Amount of better spell skill    */  \
   firstf  TIF_SPL_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_SPL_DEC       \
   applyf  APF_SPELL_ADJ;

#define STR_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}       \
affect                       \
   id ID_TRANSFER_STR        \
   duration -1                /* Must be permanent in the object */  \
   data[0] ABIL_STR           /* It is a strength function       */  \
   data[1] amount             /* Amount of better strength       */  \
   firstf  TIF_STR_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_STR_DEC       \
   applyf  APF_ABILITY;

#define DEX_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}       \
affect                       \
   id ID_TRANSFER_DEX        \
   duration -1                /* Must be permanent in the object */  \
   data[0] ABIL_DEX           /* It is a dex function            */  \
   data[1] amount             /* Amount of better dex            */  \
   firstf  TIF_DEX_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_DEX_DEC       \
   applyf  APF_ABILITY;

#define CON_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}       \
affect                       \
   id ID_TRANSFER_CON        \
   duration -1                /* Must be permanent in the object */  \
   data[0] ABIL_CON           /* It is a con function            */  \
   data[1] amount             /* Amount of better con            */  \
   firstf  TIF_CON_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_CON_DEC       \
   applyf  APF_ABILITY;

#define CHA_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}        \
affect                       \
   id ID_TRANSFER_CHA        \
   duration -1                /* Must be permanent in the object */  \
   data[0] ABIL_CHA           /* It is a cha function            */  \
   data[1] amount             /* Amount of better cha            */  \
   firstf  TIF_CHA_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_CHA_DEC       \
   applyf  APF_ABILITY;
```

```
#define BRA_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}       \
affect                       \
   id ID_TRANSFER_BRA        \
   duration -1                 /* Must be permanent in the object */  \
   data[0] ABIL_BRA            /* It is a bra function            */  \
   data[1] amount              /* Amount of better bra            */  \
   firstf  TIF_BRA_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_BRA_DEC       \
   applyf  APF_ABILITY;


#define MAG_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}       \
affect                       \
   id ID_TRANSFER_MAG        \
   duration -1                 /* Must be permanent in the object */  \
   data[0] ABIL_MAG            /* It is a mag function            */  \
   data[1] amount              /* Amount of better mag            */  \
   firstf  TIF_MAG_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_MAG_DEC       \
   applyf  APF_ABILITY;


#define DIV_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}       \
affect                       \
   id ID_TRANSFER_DIV        \
   duration -1                 /* Must be permanent in the object */  \
   data[0] ABIL_DIV            /* It is a div function            */  \
   data[1] amount              /* Amount of better div            */  \
   firstf  TIF_DIV_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_DIV_DEC       \
   applyf  APF_ABILITY;

#define HIT_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}       \
affect                       \
   id ID_TRANSFER_HPP        \
   duration -1                /* Must be permanent in the object */  \
   data[0] ABIL_HP            /* It is a hit point function      */  \
   data[1] amount             /* Amount of better strength       */  \
   firstf  TIF_HIT_INC       \
   tickf   TIF_NONE          \
   lastf   TIF_HIT_DEC       \
   applyf  APF_ABILITY;

#define SPEED_TRANSFER(newspeed) \
flags {UNIT_FL_MAGIC}       \
```

```
affect                        \
   id ID_TRANSFER_SPEED       \
   duration -1                 /* Must be permanent in the object */  \
   data[0] newspeed           \
   firstf  TIF_SPEED_BETTER   \
   tickf   TIF_NONE           \
   lastf   TIF_SPEED_WORSE    \
   applyf  APF_SPEED;

#define SLOW_TRANSFER(amount) \
flags {UNIT_FL_MAGIC}          \
affect                         \
   id ID_TRANSFER_SPEED        \
   duration -1                 \
   data[0] newspeed            \
   firstf  TIF_SPEED_WORSE     \
   tickf   TIF_NONE            \
   lastf   TIF_SPEED_BETTER    \
   applyf  APF_SPEED;
```

Appendix G. Skill definitions in values.h

The following list was extracted from the `values.h`

```
#define SKI_TURN_UNDEAD        0
#define SKI_SCROLL_USE         1
#define SKI_WAND_USE           2
#define SKI_CONSIDER           3
#define SKI_DIAGNOSTICS        4
#define SKI_APPRAISAL          5
#define SKI_VENTRILOQUATE      6
#define SKI_WEATHER_WATCH      7
#define SKI_FLEE               8
#define SKI_SNEAK              9
#define SKI_BACKSTAB          10
#define SKI_HIDE              11
#define SKI_FIRST_AID         12
#define SKI_PICK_LOCK         13
#define SKI_STEAL             14
#define SKI_RESCUE            15
#define SKI_SEARCH            16
#define SKI_LEADERSHIP        17
#define SKI_KICK              18
#define SKI_SWIMMING          19
#define SKI_BASH              20
#define SKI_CLIMB             21
#define SKI_SHIELD            22
#define SKI_TRIP              23
#define SKI_DUAL_WIELD        24
#define SKI_CUFF              25
#define SKI_RESIZE_CLOTHES    26
#define SKI_RESIZE_LEATHER    27
#define SKI_RESIZE_METAL      28
#define SKI_EVALUATE          29 /* "Fake skill to simulate combinations */
#define SKI_PEEK              30
#define SKI_PICK_POCKETS      31
#define SKI_FILCH             32
#define SKI_DISARM            33
#define SKI_SKIN            34
#define SKI_SHELTER           35
#define SKI_SOOTHE            36
#define SKI_AMBUSH            37
#define SKI_CURARE            38
#define SKI_FASHION           39
#define SKI_BUTCHER           40
#define SKI_LAY_TRAP          41
#define SKI_SHOOT             42
#define SKI_HERBS             43
#define SKI_FORAGE            44
#define SKI_DOWSE             45
#define SKI_TRACK             46
```

```
#define SKI_HUNT            47
#define SKI_THROW           48
#define SKI_COOK            49
#define SKI_SCAN            50
#define SKI_SLIP            51
#define SKI_PALM            52
#define SKI_PLANT           53
#define SKI_STALK           54
#define SKI_KNEE            55
#define SKI_ELBOW           56
#define SKI_HIT             57
#define SKI_PUNCH           58
#define SKI_GLANCE          59
```

Appendix H. Spell definitions in values.h

The following list was extracted from the `values.h`

```
#define SPL_CALL_LIGHTNING    12  /* Cell Group  */
#define SPL_BLESS             13  /* D I V I N E */
#define SPL_CURSE             14
#define SPL_REMOVE_CURSE      15
#define SPL_CURE_WOUNDS_1     16
#define SPL_CURE_WOUNDS_2     17
#define SPL_CURE_WOUNDS_3     18
#define SPL_CAUSE_WOUNDS_1    19
#define SPL_CAUSE_WOUNDS_2    20
#define SPL_CAUSE_WOUNDS_3    21
#define SPL_DISPEL_EVIL       22
#define SPL_REPEL_UNDEAD_1    23
#define SPL_REPEL_UNDEAD_2    24
#define SPL_BLIND             25
#define SPL_CURE_BLIND        26
#define SPL_LOCATE_OBJECT     27
#define SPL_LOCATE_CHAR       28

#define SPL_RAISE_MAG         29  /* P R O T E C T I O N */
#define SPL_RAISE_DIV         30
#define SPL_RAISE_STR         31
#define SPL_RAISE_DEX         32
#define SPL_RAISE_CON         33
#define SPL_RAISE_CHA         34
#define SPL_RAISE_BRA         35
#define SPL_SUN_RAY           36
#define SPL_DIVINE_RESIST     37
#define SPL_QUICKEN           38
#define SPL_HASTE             39
#define SPL_RAISE_SUMMONING   40
#define SPL_AWAKEN            41
#define SPL_MIND_SHIELD       42
#define SPL_HEAT_RESI         43
#define SPL_COLD_RESI         44
#define SPL_ELECTRICITY_RESI  45
#define SPL_POISON_RESI       46
#define SPL_ACID_RESI         47
#define SPL_PRO_EVIL          48
#define SPL_SANCTUARY         49
#define SPL_DISPEL_MAGIC      50
#define SPL_SUSTAIN           51
#define SPL_LOCK              52
#define SPL_UNLOCK            53
#define SPL_DROWSE            54
#define SPL_SLOW              55
#define SPL_DUST_DEVIL        56
#define SPL_DET_ALIGN         57  /* D E T E C T I O N */
```

```
#define SPL_DET_INVISIBLE     58
#define SPL_DET_MAGIC         59
#define SPL_DET_POISON        60
#define SPL_DET_UNDEAD        61
#define SPL_DET_CURSE         62
#define SPL_SENSE_LIFE        63
#define SPL_IDENTIFY_1        64
#define SPL_IDENTIFY_2        65

#define SPL_RANDOM_TELEPORT   66  /* S U M M O N I N G */
#define SPL_CLEAR_SKIES       67
#define SPL_STORM_CALL        68
#define SPL_WORD_OF_RECALL    69
#define SPL_CONTROL_TELEPORT  70
#define SPL_MINOR_GATE        71
#define SPL_GATE              72
#define SPL_CREATE_FOOD       73  /* C R E A T I O N */
#define SPL_CREATE_WATER      74
#define SPL_LIGHT_1           75
#define SPL_LIGHT_2           76
#define SPL_DARKNESS_1        77
#define SPL_DARKNESS_2        78
#define SPL_STUN              79
#define SPL_HOLD              80
#define SPL_ANIMATE_DEAD      81
#define SPL_LEATHER_SKIN      82
#define SPL_BARK_SKIN         83
#define SPL_CONTROL_UNDEAD    84
#define SPL_BONE_SKIN         85
#define SPL_STONE_SKIN        86
#define SPL_AID               87
#define SPL_COLOURSPRAY_1     88  /* M I N D */
#define SPL_COLOURSPRAY_2     89
#define SPL_COLOURSPRAY_3     90
#define SPL_INVISIBILITY      91
#define SPL_WIZARD_EYE        92
#define SPL_FEAR              93
#define SPL_CONFUSION         94
#define SPL_SLEEP             95
#define SPL_XRAY_VISION       96
#define SPL_SUMMER_RAIN       97
#define SPL_COMMAND           98
#define SPL_LEAVING           99
#define SPL_FIREBALL_1       100  /* H E A T */
#define SPL_FIREBALL_2       101
#define SPL_FIREBALL_3       102

#define SPL_FROSTBALL_1      103  /* C O L D */
#define SPL_FROSTBALL_2      104
#define SPL_FROSTBALL_3      105

#define SPL_LIGHTNING_1      106  /* C E L L */
#define SPL_LIGHTNING_2      107
```

```
#define SPL_LIGHTNING_3      108

#define SPL_STINKING_CLOUD_1 109  /* I N T E R N A L */
#define SPL_STINKING_CLOUD_2 110
#define SPL_STINKING_CLOUD_3 111
#define SPL_POISON           112
#define SPL_REMOVE_POISON    113
#define SPL_ENERGY_DRAIN     114
#define SPL_DISEASE_1        115
#define SPL_DISEASE_2        116
#define SPL_REM_DISEASE      117
#define SPL_ACIDBALL_1       118  /* E X T E R N A L */
#define SPL_ACIDBALL_2       119
#define SPL_ACIDBALL_3       120
#define SPL_FIND_PATH        121  /* Divine   */
#define SPL_DISPEL_GOOD      122
#define SPL_PRO_GOOD         123
#define SPL_TRANSPORT        124
#define SPL_FIRE_BREATH      125
#define SPL_FROST_BREATH     126
#define SPL_LIGHTNING_BREATH 127
#define SPL_ACID_BREATH      128
#define SPL_GAS_BREATH       129
#define SPL_LIGHT_BREATH     130
#define SPL_HOLD_MONSTER     131
#define SPL_HOLD_UNDEAD      132
#define SPL_RAISE_DEAD       133
#define SPL_RESURRECTION     134
#define SPL_UNDEAD_DOOR      135
#define SPL_LIFE_PROTECTION  136
#define SPL_ENERGY_BOLT      137
#define SPL_CLENCHED_FIST    138
#define SPL_METEOR_SHOWER    139
#define SPL_SUN_BEAM         140
#define SPL_SOLAR_FLARE      141
#define SPL_SUMMON_DEVIL     142
#define SPL_SUMMON_DEMON     143
#define SPL_SUMMON_FIRE      144
#define SPL_SUMMON_WATER     145
#define SPL_SUMMON_AIR       146
#define SPL_SUMMON_EARTH     147
#define SPL_CHARGE_WAND      148
#define SPL_CHARGE_STAFF     149
#define SPL_MENDING          150
#define SPL_REPAIR           151
#define SPL_RECONSTRUCT      152
#define SPL_SENDING          153
#define SPL_REFIT            154
#define SPL_FIND_WANTED      155
#define SPL_LOCATE_WANTED    156
#define SPL_SUN_GLOBE        157
#define SPL_MAGIC_CANDLE     158
#define SPL_SONIC_BREATH     159
```

```
#define SPL_SHARD_BREATH     160
#define SPL_CONE_SHARD       161
#define SPL_RAISE_HPP        162
#define SPL_MANA_BOOST       163  /* Creation */
#define SPL_TOTAL_RECALL     164
#define LAST_SPELL           165
```