Contents

[Preamble 1](#_Toc195414448)

[Overview of the Lib/Edit Files 2](#_Toc195414449)

[Customising the Existing Edit Files 3](#_Toc195414450)

[Creating an New Edit File 4](#_Toc195414451)

[Creating an Unangband Mod 4](#_Toc195414452)

[Related Source Code Files 4](#_Toc195414453)

[Common Edit File Features 5](#_Toc195414454)

[Lib/Edit File Specifics 5](#_Toc195414455)

[Artifact.txt 5](#_Toc195414456)

[Overview 5](#_Toc195414457)

[General notes 5](#_Toc195414458)

[Artifact.txt entry format 5](#_Toc195414459)

[Understanding artifact.txt entries 6](#_Toc195414460)

[Customising this file 6](#_Toc195414461)

[Credits 6](#_Toc195414462)

[Ego\_Item.txt 7](#_Toc195414463)

[General notes 7](#_Toc195414464)

[Ego\_Item.txt entry format 7](#_Toc195414465)

[Understanding Ego\_Item.txt 8](#_Toc195414466)

[Credits 8](#_Toc195414467)

[Flavor.txt 9](#_Toc195414468)

[Understanding Flavor.txt entries 9](#_Toc195414469)

[Monster.txt 9](#_Toc195414470)

[P\_Class.txt 15](#_Toc195414471)

[P\_Race.txt 17](#_Toc195414472)

[Spell.txt 19](#_Toc195414473)

# Preamble

This document covers the text configuration files held in the lib/edit directory and how to modify them.

# Overview of the Lib/Edit Files

Unangband takes a data driven approach to much of the game design. May game rules are abstracted into external edit files which are contained in the lib/edit subdirectory. These files are parsed in using code within init1.c and then turned into binary data which is then reparsed in init2.c. The binary data is held in the lib/data subdirectory with a .raw file corresponding to each .txt file in lib/edit.

This allows high performance loading of the data contained in the edit files in a platform independent manner - they are only reparsed in the event the edit file date is newer than the data file date, or the Unangband version number increased. If you change the internal Unangband code to read these files or the type definitions of any data structure used in these files, you should delete the .raw files before reloading - otherwise you may end up with unusual errors due to bad data which could potentially crash the game and/or corrupt save files and other data.

Although game logic is contained in the edit files, it is not equivalent to scripting. The edit file language is simple, consisting of comment lines, starting with a symbol, blank lines and data definition lines. The data definition lines almost always act as defining part of a row of a data table. This is almost always a large array of elements of a C structure. e.g. the monster.txt file defines a large array of elements of monster\_race. The edit files are simply a way of extending the existing tables.c file to include a lot more elements, and should be read in conjunction with the relevant structures defined in types.h and with heavy use of grep to determine where the data and structures are used in the Unangband code.

The Unangband table based data design would be better supported in a SQL like language, or Lisp or even extended with Lua or another scripting language. Unangband may be rewritten using one of these languages at some point. The paper Scaling Games to Epic Proportions (http://www.cs.cornell.edu/%7Ewmwhite/papers/2007-SIGMOD-Games.pdf) discusses a SQL like language supporting game elements on a massive scale that may be of interest. Avoiding scripting is not without complications in the game design, but at the moment there are no compelling reasons to make Unangband support scripting.

A data definition line starts with a letter followed by a colon, followed by either one or more colon separated fields, or flags separated by white space and vertical bars (|). There are three basic types of edit file: indexed files, Markov chained files, and table files.

In the first type of file, the data is organised in blocks starting with a N: data line, which defines data block as corresponding to the nth element of the array, and usually the name of the element. e.g.

N:1:the first element

A:1:2:3

B:4:5

F:FIRST\_FLAG | SECOND\_FLAG

N:2:the second element

etc.

These are used where the index is important, such as when referring to a particular monster race or character class number.

The second type consists of array elements that are strung together using a Markov chain approach. The first line of each element usually defines the block number, and a simple algorithm goes through the array, finds elements corresponding to the same block and picks one, then chooses the next block based on selection criteria within the element chosen, and repeats until a block number of 0 is chosen. These files usually build a sentence such as a room description, or character history. Because of the design of these files, the unique element number is not as important, so it is not usually defined in the file.

The third type is where the file builds a table. Each line corresponds to one row of the table, with numeric values separated by colons corresponding to each column.

To emphasise: the data structure in all three types is an array of elements of a C structure - it's just a matter of how the information is presented in the file and parsed is slightly different.

String values in the files are stored as indexes into a separate chunk of contiguous zero byte delimited strings. Each string currently has a unique index even though the string may be potentially duplicated elsewhere in the file. Usually two separate chunks of strings are used for a file: one for names, and one for descriptions. During the parsing process in init1.c, a maximal size chunk is allocated using malloc, the size of which is defined in lib/edit/limits.txt - and then once the size is known, it is written to the .raw file so that the correct size is allocated when the .raw file is read in.

It is possible to build a distribution of Unangband which does not allow the lib/edit files to be parsed. To do this, disable the ALLOW\_TEMPLATES define in config.h and distribute Unangband with the .raw files in lib/data. Note that this distribution is now heavily dependent on the endianess and integer size of the machine it is created on, so you should ensure that this is a binary only distribution.

There is some in game calculations used in the lib/edit files as well. For instance, level, rarity and experience reward for monsters is algorithmically determined. To output a new copy of the edit files containing the generated values, enable ALLOW\_TEMPLATES\_OUTPUT and the edit files will be output to the lib/user directory when the game launches. Note that e.g. level is dependent on power, which can depend on level, so that the depths for monsters you have not changed will also vary in the output file.

# Customising the Existing Lib/Edit Files

The edit files are a gateway drug: they exist to allow players to modify the game without requiring an understanding of coding or compilation. The edit files in Angband are the reason there are so many variants of Angband – it is very easy for a player to modify one or more entries in an edit file to customise the game to their liking, and then progressively change the game so that it becomes a full variant. Outside the Angband community, these would be called mods or total conversions.

An understanding of the ability to read code is highly recommended if you wish to customise the existing edit files. This is for several reasons:

* The edit files themselves are incompletely documented. This file is an attempt to rectify part of that issue.
* Unangband is a work-in-progress and constantly evolving. This means that the SVN repository is the best place to get a view of the developer intentions, and the released source code for the version you are playing the best reference for specific implementation of the edit files.
* Documentation can at only at best discuss the code at a high level. For an exact understanding of how the edit files interact and features of these files, the source code and generated executables are your primary reference.

Luckily the Angband code base is well documented, a tradition that Unangband attempts to continue, although has done so in fits and starts. Refer to the ‘Related Source Code Files’ section below for a guide to reading the source code in conjunction with the edit files.

Customising the edit files is ‘simply’ a matter of making a change to the .txt file, saving it and restarting Angband. This will automatically update the corresponding .raw file as a part of the game start up. Various checks are made to ensure that the changes parse correctly, but not a lot of data bounds checking is enabled, so it is relatively easy to crash the Unangband executable, by putting invalid entries in the edit files.

If you wish to add entries to the end of a particular edit file, you’ll need to edit limits.txt. This file covers the initialisation of a number of maximum entries in each of the other edit files, as well as upper bounds for dynamically allocated memory for holding the text and descriptions for the majority of these files, and a few other values (e.g. the maximum numbers of monsters and objects on a dungeon level).

You should familiarise yourself with the game-play first, and then reading the edit files, before you make any changes. This will give you a good feeling for how changing the edit files makes corresponding changes in game. A common first attempt may be to make easy monsters have powerful or valuable drops, by adding various DROP\_ flags to the monster type. Equally, Morgoth can be made very weak and easy to defeat as well as appearing much early in the dungeon, allowing you to achieve impressive speed runs.

You’ll quickly learn (unless you are of a certain frame of mind), that changing the edit files to win the game quickly ruins the experience for you, and you should refrain from trying this, or even peeking at the information contained inside, to discover the surprises ahead. Unfortunately, this temptation is often too great – and a challenge that few open source games are likely to be able to overcome. But with the rise of sites like gamefaqs.com and programs like Glider, the same information sharing challenge is faced by commercial games as well. Luckily, the procedural generation, randomness and other roguelike features such as perma death ensure that Unangband remains a challenge, even with complete knowledge of the game rules.

The edit files are not as expressive as a full scripting language and you’ll find that there are a number of cases where various values in the edit file have hard-coded dependencies in the game code, or interact in unexpected ways. Unangband has tried to reduce the number of instances where this happens in the ‘core’ Angband edit files, but at the same time feature creep has meant that overall it is probably just as bad, if not worse than Angband itself. This documentation attempts to address this. Each file has a corresponding section in this document, which discusses the specifics of customising a file and how the file interacts with other lib/edit files and the game engine (compiled source code).

# Creating an New Edit File

The data driven design process often extends

# Creating an Unangband Variant

# Related Source Code Files

You should download and use the tool grep for determining where in the source code a particular value is used. For historical reasons, the data structure corresponding to an edit file has a different name. The lib/edit file and corresponding data structures and data types are in the table below:

|  |  |  |
| --- | --- | --- |
| **Lib/Edit file** | **Data Structure(s)** | **Data Type(s)** |
| Artifact.txt | a\_info | artifact\_type |
| Blows.txt | blow\_info | blow\_type |
| Cost\_adj.txt |  |  |
| Dungeon.txt | t\_info | town\_type |
| Ego\_item.txt | e\_info | ego\_item\_type |
| Flavor.txt | flavor\_info | flavor\_type |
| Limits.txt | z\_info |  |
| Monster.txt | r\_info |  |
| Names.txt |  |  |
| Object.txt | k\_info | object\_kind, object\_type |
| P\_class.txt | c\_info | player\_class |
| P\_hist.txt | h\_info |  |
| P\_race.txt | p\_info | Player\_race |
| Quest.txt | q\_info |  |
| Room.txt | d\_info |  |
| Rune.txt | y\_info |  |
| Shop\_own.txt |  |  |
| Spell.txt | s\_info |  |
| Store.txt |  |  |
| Style.txt | w\_info |  |
| Terrain.txt | f\_info | feature\_type |
| Vault.txt | v\_info | Vault.txt |

# Common Edit File Features

# Lib/Edit File Specifics

## Artifact.txt

### Overview

Artifact.txt defines the list of artifacts in the game. An artifact is a unique object that can be found and identified only once in the game. Once the artifact name has learnt and subsequently left on a dungeon level, it can never be found again. Note that this identification requirement corresponds to the old “preserve” mode from Angband.

Artifacts are typically more powerful than the equivalent object or ego item found on the level (permitted artifact power is twice that of an object and one and a third that of an ego item on the equivalent level – see randart.c for details of how this is computed for artifacts and object\_power in object2.c for others). Artifacts are divided into three distinct types: “special” artifacts which consist of items other than armor or weapons, normal artifacts, and random artifacts.

The random artifact set is generated from the base artifact set at the start of each game, and tops up the total number of artifacts in the game to 255. The random artifact algorithm is complex, and covered in randart.c. You can choose various birth options to allow you to share the same artifact set between multiple games, as well as ensure all artifacts, including the special artifacts, are randomly generated.

Generation of artifacts in game occurs in two different places in the code in object2.c, depending on whether the artifact is a “special” artifact or not. Special artifacts are created when the ‘base item’ from object.txt is created according to the standard object creation rules, and instantly turned into an artifact. This means that every “special” artifact needs a corresponding unique object defined in object.txt.

Otherwise, artifacts are created if a created armour or weapon passes several checks, and then all artifacts from this file that match the weapon or armour type must pass a rarity test to see if the object in question converts into an artifact. The match is determined on the tval and sval values specified for the artifact matching a generated base object, so that deep artifacts which depend on shallow base objects are less likely to be created than their rarity suggests.

Note that monsters with DROP\_GOOD and particularly DROP\_GREAT are much more likely to carry artifacts than other monsters. This has led to various types of artifact scumming behaviour in the past which relies on the fact that artifacts cannot be destroyed or picked up by monsters. While cloned and breeding monster offspring no longer carry additional objects to restrict many of this type of exploit, it is still possible with various summoned monsters.

The player can determine an object is an artifact in a number of ways: examples include sensing that it is an artifact or special item, noting that it cannot be destroyed, seeing that it has a Unique rune and identifying the name and recognising it as a name of an artifact that they have seen or heard of previously. It is important to be careful of the interaction between item sensing and artifact “preservation” so that the name of an artifact is not learned prior to the player actively identifying the artifact, and therefore allowing the artifact to be lost.

### General notes

Note for historical reasons the misspelling of artefact as artifact throughout Unangband, that has been inherited from Angband.

The artifact indexes are defined in "defines.h", and must not be changed (This may no longer apply).

Hack -- "Grond" and "Morgoth" MUST have a rarity of one, or they might not be dropped when Morgoth is killed. Note that they, like the "special" artifacts, are never created "accidentally".

### Artifact.txt entry format

N: serial number : item name

I: tval : sval : pval

W: depth : rarity : weight : cost

P: base armor class : base damage : plus to-hit : plus to-dam : plus to-ac

F: flag | flag | etc

A: activation : recharge time : recharge time dice

D: description

### Understanding artifact.txt entries

'N' indicates the beginning of an entry. The serial number must increase for each new item.

'I' is for basic information. The tval is for the type of item, the sval identifies the subtype and the pval indicates the amount of effect the item has, if applicable.

'W' is for extra information. Depth is the depth the object is normally found at, rarity determines how common the object is, weight is in tenth-pounds and cost is the item's value.

'P' is for power information. The items base armor class, its base damage and pluses to-hit, to-dam and to-ac.

'F' is for flags. These are common to the object.txt, artifact.txt and ego\_item.txt files. As many F: lines may be used as are needed to specify all the flags and flags are separated by the '|' symbol. Note that all artifacts cannot be destroyed and so implicitly have the IGNORE\_ type flags.

'A' is for activation. Activation is the effect the artifact activates for, referring to an entry number in spell.txt. The recharge time calculates from the recharge time plus a random value between 1 and the recharge time dice (if not 0). If an activation is given then the artifact must also have the ACTIVATE flag. Artifact weapons with activations can also be inscribed with =A by the player which will mean that the activation is used against the monster that the artifact hits, provided the activation has recharged.

‘D’ is for description. The artifact description should provide information about the history and/or owner of the artefact, as opposed spoilerish information about its powers. Contrast this with e.g. the artifact descriptions in T.o.M.E.

### Customising this file

Any changes or additions to the file will have influence on randarts and may break savefile compatibility for old savegames with randarts. If using random artifacts, finish your existing game before making any changes to the artifact.txt file at all.

Removing any artifact, or changing its "base" type (as defined by the tval and sval entries in the "I:" line), will break savefile compatibility for ALL savefiles.

Adding a new artifact, or changing the powers of existing ones, will NOT affect savefile compatibility for games with the standard artifact set: it is perfectly safe to do this. (If your new artifact is a new kind of object, or a new one at the end of the file, you must also amend object.txt or limits.txt.)

“Special” artifacts are artefacts that are not weapons or armor. Only the first 15 artifacts may be of these types currently: artifacts from index number 16 onwards must be Armor or Weapons. (This may no longer apply).

Note a few entries are used in hard-coded ways: namely the entries for Grond and the Iron Crown of Morgoth are dropped specifically.

When designing an artifact, consider strongly how it will be uniquely useful in the game. There are probably already too many artifacts in Angband and Unangband, and additions are likely to be either overpowered or not distinguishable from the large selection already available, or lesser powered ego or magic items. Artifacts with a mixture of good abilities with some negative side effects, or interesting activations, are worth considering, as are artifact staffs or diggers, and armour other than body armour.

### Credits

7: The Palantir of Westernesse (original creation by JLE)

14: The Elfstone 'Elessar'

15: The Jewel 'Evenstar'

18: The Balance Dragon Scale Mail 'Mediator' (original JLE creation)

26: The Hard Leather Armor of Himring (adapted from OAngband)

29: The Shield of Deflection of Gil-galad (adapted from OAngband)

33: The Metal Cap of Celebrimbor (suggested by Matthias Kurzke)

43: The Jewel-Encrusted Crown of Numenor (adapted from OAngband)

51: The Gauntlets of Eol (adapted from OAngband)

63: The Soft Leather Boots of Wormtongue (original JLE creation)

92: The Spear of Melkor (adapted from KAngband)

110: The Beaked Axe of Hurin (adapted from KAngband)

123: The Whip of Gothmog (original JLE creation)

127: The Heavy Crossbow of Umbar (adapted from OAngband)

128, 129: The Short Bows of Amrod and Amras (original JLE creations)

130: The Mattock of Nain (original JLE creation)

131: The Dwarven Pick of Erebor (adapted from SAngband)

132: The Ball-and-Chain of Fundin Bluecloak (suggested by Skylar Thompson)

133: The Main Gauche of Azaghal (adapted from KAngband)

134: The Large Leather Shield of the Haradrim (suggested by Skylar Thompson)

135: The Broken Sword 'Narsil' (original JLE creation)

136: The Bastard Sword of Eowyn (original JLE creation)

## Ego\_Item.txt

### Overview

Ego\_item.txt defines the list of ego items in the game. An ego item is an object that has additional flags applied to it beyond that possible by applying runes or spells to the item, without having the power of an artifact. Note that it is possible to use runes to upgrade an item to an ego item.

Ego items are typically more powerful than the equivalent object or found on the level (permitted ego item power is one and a half times that of a equivalent object – see object\_power in object2.c for details). There is also code to ensure that the ego item reaches a minimum power level if generated, including selection of the appropriate random ego ability and boosting the to hit, to damage, to ac and item pval as appropriate.

Generation of ego items in game occurs in object2.c, and occur if a created armour or weapon passes several checks, and then all ego items from this file that match the weapon or armour type must pass a rarity test to see if the object in question converts into the ego item. The match is determined on the tval and sval ranges specified for the ego item matching a generated base object, so that deep ego items which depend on the creation of shallow base objects are less likely to be created than their rarity suggests.

Note that monsters with DROP\_GOOD are much more likely to carry ego items than other monsters, and weapons or armour generated by a DROP\_GREAT monster will always at least be an ego item, if there is a corresponding match for that weapon or armour tval and sval in this file.

The player can determine an object is an ego item in a number of ways: examples include sensing that it is an excellent, superb, ego item or high ego item, or using Rune Magic to determine the number of runes on a weapon or armour. Superb and high ego items are distinguished from excellent and regular ego items, by having a random extra flag as specified in the corresponding ego\_item.txt entry. This means that in addition to the flags specified here, that the ego item has a randomly chosen flag from a possible list in tables.c, corresponding to the value of xtra.

There is a tier below egos for many items, including weapons, armour and lights, called magic items. Magic items are normal items which have the random extra flag only, and no ego abilities. Magic items will sense as excellent items, and appear to be mini-ego items. But they do not have a corresponding entry in the ego\_item.txt file, and will be generated based on object boost code to ensure that they reach the mandated object power level for the particular level (see object2.c for details). Magic items are used to try to ensure that all weapon, armour and light drops are useful for the current level.

Some magic items will be generated as racial ego items. These are intermediate again between magic items and ego items. These are distinguished by having an xtra flag of 16. Racial ego items are part of an unfinished racial drop system. They have the random flag of a magic item but the negative racial flag they have allows the item to boost its power beyond what is normally permitted on the dungeon level. The racial drop system is intended to have e.g. orcs drop orcish weapons, armour and ego items, but causes too much of an impact on what equipment is found in the dungeon at this stage.

### General notes

The ego-item indexes are defined in "defines.h", and must not be changed. (This is probably no longer correct).

Ego-items 1-63 are "armor", and 64-127 are "weapons" for historical reasons, but no other requirement.

Note that every "ego-item" type has a different "index", and can only be created from items that belong to a certain "slot" in the equipment, if one assumes that "ammo" belongs to an imaginary slot (23). (Note that the above is not true and was not true even before my patch. The 'slot' seems very useless to me right now. -MWK-)

Note that several "ego-item" types may share the same "textual name", such as with "Armor of Resistance" and "Shield of Resistance". (This may no longer be true, although duplicate names are permitted).

### Ego\_Item.txt entry format

N: serial number : ego type

X: inventory slot : rating : xtra

C: max to-hit : max to-dam : max to-ac : pval

W: depth : rarity : weight : cost

T: tval : min\_sval : max\_sval

F: flag | flag | etc

Y:rune type : rune count

### Understanding Ego\_Item.txt

'N' indicates the beginning of an entry. The serial number must increase for each new item.

'X' is for extra information - inventory slot and rating. Slots range from 23 (for ammunition) to 35 (for boots). Rating affects level feelings. The xtra value determines random extra flags for the ego-item. Nothing happens with a value of 0, 1 stands for "one random extra sustain", 2 is "one random extra resist", and 3 is "one random extra ability". For a full list of xtra abilities, see tables.c.

'C' is for creation bonuses - maximum to-hit, to-dam, to-ac, and pval. Cursed ego-items will negate the given value.

'W' is for extra information. Depth is the depth the object is normally found at, rarity determines how common the object is, weight is in tenth-pounds and cost is the items value.

'F' is for flags. These are common to the object.txt, artifact.txt and ego\_item.txt files. As many F: lines may be used as are needed to specify all the flags and flags are separated by the '|' symbol.

‘O’ is for Obvious flags. These are flags that are learned when the ego item is ‘named’ e.g. when the IDENT\_NAME flag is applied. The Obvious flags should be obvious from the ego item name, and be a subset of the ‘F’ flags. As many O: lines may be used as are needed to specify all the flags and flags are separated by the '|' symbol.

'T' is for possible 'tval' and 'sval' values of the base-item. Up to three 'T' entries are possible for a single object. The ego-item can only be generated from items of this tval, with sval in the specified range.

‘Y’ is for runes. When count number of type of runes are applied to an object, that is permitted to appear as an ego item of the corresponding type, then the object will be converted to this ego item. Objects and ego items which already have a count listed for the number of runes, are treated as having that many runes already, when a rune of the same type is applied. This allows runes in particular to be used to upgrade e.g. a weapon of Frost to a ‘Frostbite’ using two runes of Frost. Note that post conversion, the runes are stripped off, so that the additional rune abilities specified in rune.txt do not apply until a further rune is added again. Ego items will identify as having so many runes of a particular type as listed here even if they do not ‘internally’ have the runes applied.

### Customising this file

There are probably already too many ego items, particularly weapons, in Unangband, particularly with the addition of magic items. Care should be taken when adding additional ego types, that the ego type does not match or exceed the power of an equivalent artifact. Note that many top end ego items, particularly weapons, do this already. Consider restricting an ego type to a particular subset of armour or equipment that make it less useful.

Note that due to the way ego items are created, that ego item dragon armour is highly likely, and therefore no ego items for dragon armour should be created.

### Credits

Credits listed here are JLE unless otherwise noted. All other credits are either noted in the file or from Angband.

10: Armor of Vulnerability (cursed ego-type for armor)

12: Dwarven Armor (heavy metal armor only)

21: Shield of Elvenkind (returning from Angband 2.7.8 and previous versions)

22: Shield of Preservation

23: Shield of Vulnerability (cursed ego-type for shields)

36: Crown of Serenity (adapted from OAngband: replaces Helm of Stupidity)

37: Crown of Night & Day (partially adapted from an OAngband ego-item)

43: Cloak of the Magi

52: Gloves of Thievery (leather gloves only)

53: Gauntlets of Combat (gauntlets and cesti only)

60: Boots of Stability (from OAngband)

61: Boots of Elvenkind (replaces Boots of Noise - too many cursed types of boot)

67: Weapon of Gondolin (rarer and more powerful than Westernesse)

70: Weapon of Fury (nastier than a Weapon of Extra Attacks, but also aggravates)

76: Weapon of Venom (Poison Brand: does NOT, however, provide poison resist)

101: Shovel/Pick of Impact

106: Bow of Lothlorien

107: Crossbow of the Haradrim

110: Sling of Buckland

111: Bow/Crossbow of the Nazgul (cursed bow ego-type)

120: Ammo of Holy Might (seeker, silver and mithril stuff only)

121: Ammo of Venom

## Flavor.txt

### Flavor.txt entry format

N:<index>:<tval>:<sval>

G:<char>:<attr>

D:<text>

### Understanding Flavor.txt entries

## Monster.txt

This file describes the monster.txt file in the lib\edit folder. The monster.txt file is used to initialise almost all information about the monster races in the game. The file is parsed into the monster.raw file in lib\data which is a raw data file which is loaded into memory. The raw file is created if it does not exist, or if the date on the monster.txt file is later than the raw file. It is best practise to always delete the raw file if you

The monster.txt file consists of one entry for each monster race, which is laid out as follows:

N: serial number : monster name

G: symbol : color

I: speed : hit points : vision : armor class : alertness

W: depth : rarity : group index : experience for kill

M: innate frequency : spell frequency : spell power : mana

B: attack method : attack effect : damage

S: spell type | spell type | etc

F: flag | flag | etc

D: Description

'N' indicates the beginning of an entry. The serial number must increase for each new item. Entry 0 is used for the player.

'G' is for graphics - symbol and color. There are currently 16 colors, as follows:

D - Dark Gray w - White s - Gray o - Orange

r - Red g - Green b - Blue u - Brown

d - Black W - Light Gray v - Violet y - Yellow

R - Light Red G - Light Green B - Light Blue U - Light Brown

(note that "black" is the same color as the screen background, and thus the monster will appear to be an empty black space if its color is "d".)

'I' is for information - speed, health, vision in tens of feet, armor class, and alertness. 110 is normal speed. Alertness ranges from 0 (ever vigilant for intruders) to 255 (prefers to ignore intruders).

'W' is for more information - level, rarity, group index and experience for killing. The group index is index to associate monsters of a particular type together. Monsters with the same group index will often be generated on the same level together. Group indexes if defined will also be used for escorts and for the summon group summoning spell. The current group indexes in use are:

1 Apprentices

10 Mimic

11 - 19 Elementals

20 Golem

21 Ghost

22 Spirit

23 Golem

24 Faerie

25 Ant

26 Termite

'B' is for blows - method of attack, effect of attack, and damage from attack. There may be up to four of these lines; effect and damage are optional.

'M' is for magic information - cast innate, cast spell, spell power, and mana. All breath weapons and a few other spells are 'innate' and use the innate percentage chance to determine how frequently they are cast. In addition, some blow types may be used at range (such as arrow, lash, spit and so on) and also use the cast innate to determine how frequently they may be used.

All other spells use the cast spell percentage chance to determine their frequency. In addition, the spell power determines the damage multiplier for these spells (but not breath weapons, which depend on hit points).

Monster mana determines how many spells may be cast before having to rest to recover mana, or cast a spell to add mana. Each monster spell has a mana cost that is associated with this.

'S' is for spells. These are explained in more detail below. As many S: lines may be used as are needed to specify all the spells, spells

are separated by the '|' symbol.

'F' is for flags. These are explained in more detail below. As many F: lines may be used as are needed to specify all the flags, flags are separated by the '|' symbol.

'D' is for description. As many D: lines may be used as are needed to describe the monster.

=== Understanding monster race flags ===

To understand the additional monster flags, you should grep the code base using the regular expression RF[0-9]\_ followed by the monster flag as shown in the monster.txt file.

Brief explanations of a number of flags follow:

UNIQUE monsters are just "special" monster races, with the requirement that only one monster of that race can exist at a time, and when it is killed, it can never again be generated.

ATTR\_CLEAR monsters appear the same color as the floor, terrain feature or object underneath them.

ATTR\_MULTI monsters are "multi-hued" - they change color, every player turn.

ATTR\_METAL monsters have a occasionally "flickering" attr which is one step lighter than their base color.

CHAR\_CLEAR monsters are totally invisible: the symbol shown on the screen is the same as the floor (or other terrain feature) below it, or the item that is lying on that floor. These monsters can currently be targetted as normal monsters: perhaps this can be changed so that the player must KNOW the monster is there (by bumping into it, or having the monster wake up and attack the player?)

CHAR\_MULTI monsters are those which look like objects, and use the symbols of objects. They are in plain sight, but the character should not know that a monster is there. At the moment, such monsters can be targetted normally (which means that this flag currently has no effect), but this may change in the future, as with CHAR\_CLEAR monsters.

(Note that currently both CHAR\_CLEAR and CHAR\_MULTI monsters are treated as normal monsters that are just a little hard to see... if you "l"ook for them, you will find them unless they are invisible and you cannot see invisible, and if you use the \* command for targetting mode you can pick them as a target.)

=== Understanding object.txt ===

Add some "IGNORE\_XXX" flags to Rings, Amulets, etc. XXX XXX

=== Understanding k\_info.txt ===

N: serial number : & object name~

G: symbol : color

I: tval : sval : pval

W: depth : charges : weight : cost

P: base armor class : base damage : plus to-hit : plus to-dam : plus to-ac

A: depth/rarity : depth/rarity : etc

F: flag | flag | etc

Y: runest | runesc

'N' indicates the beginning of an entry. The serial number must

increase for each new item. The '&' and '~' symbols are use to

include articles and pluralization as necessary to ensure

grammatical correctness in object descriptions.

'G' is for graphics - symbol and color. There are 16 colors, as

follows:

D - Black w - White s - Gray o - Orange

r - Red g - Green b - Blue u - Brown

d - Dark Gray W - Light Gray v - Violet y - Yellow

R - Light Red G - Light Green B - Light Blue U - Light Brown

'I' is for basic information. The tval is for the type of item, the

sval identifies the subtype and the pval indicates the amount of

effect the item has, if applicable.

'W' is for extra information. Depth is the depth the object is

normally found at, rarity determines how commoAn the object is,

weight is in tenth-pounds and cost is the items value.

'P' is for power information. The items base armor class, its base

damage and pluses to-hit, to-dam and to-ac.

'A' is for allocation - depth and rarity, in pairs. This allows an

item to have multiple natural depths and rarities. It is used to

ensure that certain vital items such as food and identify scrolls

are found throughout the dungeon.

Now identify scrolls are not found throughout the dungeon... however

detect magic scrolls are. Identify scrolls and \*identify scrolls

are much rarer. Items of detect magic and detect curse are as frequent

as the old identify scrolls.

Staves of perception are replaced by staves of detect curse. Rods of

perception are replaced by rods of detect magic.

Brought back staves of perception. Nice and deep (Level 45).

'F' is for flags. These are fairly self-explanatory. As many F:

lines may be used as are needed to specify all the flags and flags

are separated by the '|' symbol.

## P\_Class.txt

=== Understanding p\_class.txt ===

Format:

N:class number:class name

S:str:int:wis:dex:con:chr:agi:siz

C:dis:dev:sav:stl:srh:dig:tht:thn:thb

X:dis:dev:sav:stl:srh:dig:tht:thn:thb

I:exp:sense\_base:sense\_div:sense\_type:sense\_squared

A:max\_attacks:min\_weight:att\_multiply:chg\_weight

M:spellbook tval:spell-stat-study:spell-stat-mana:spell-stat-fail:first-level:max weight:high-power

B:spell number:level:mana:fail:exp

T:title

E:tval:sval:min:maxef

F:school flags

'N' indicates the beginning of an entry. The serial number must

increase for each new class.

'S' is for the stat modifiers - strength, intelligence, wisdom,

dexterity, constitution, charisma, agility, size.

'C' is for the base skills - disarming, magic devices, saving throws,

stealth, searching ability, digging, to hit (throws), to hit (normal),

to hit (bows).

'X' is for the skill modifiers - disarming, magic devices, saving

throws, stealth, searching ability, digging, to hit (throws), to hit

(normal), to hit (bows). The modifiers are added to the base skill

every ten levels.

'I' is for extra info - experience factor in percent,

base chance for pseudo-id (smaller is better),

level dependent divisor for the pseudo-id chance, type of sensing

(see object2.c), sense squared.

'A' is for attack info - maximum possible number of attacks, minimum

weapon weight for calculating the number of attacks, multiplier for

attack calculations, weight divisor for charging.

'M' is for magic info - tval of the used spellbook, number of the

spell-stat, level of the first spell, weight that hurts spells.

'T' is for class titles (Up to 10 entries).

'E' is for starting equipment - tval of the item, sval of the item,

minimum amount, maximum amount.

'F' is for class flags.

## P\_Race.txt

=== Understanding p\_race.txt ===

Format:

N:race number:race name

S:str:int:wis:dex:con:chr:agi:siz

R:dis:dev:sav:stl:srh:dig:tht:thn:thb

X:expbase:infra:monster race equivalent

I:history:agebase:agemod:hometown

H:hgtmale:modhgtmale:hgtfemale:modhgtfemale

W:wgtmale:wgtfemale

F:racial flags

C:classes (numeric)

O:wield:bow:left\_ring:right\_ring:neck:lite:body:outer:arm:head:hands:feet

D:adjective

'N' indicates the beginning of an entry. The serial number must

increase for each new race.

'S' is for the stat modifiers - strength, intelligence, wisdom,

dexterity, constitution, charisma, agility, size.

'R' is for the base skills - disarming, magic devices, saving throws,

stealth, searching ability, digging, to hit (throws), to hit (melee),

to hit (bows).

'X' is for extra info - experience factor in percent,

range of infravision in 10 feets.

'I' is for info - starting point of the player history, base age,

random age modifier, home town.

'H' is height info - base height (males), mod height (males),

base height (females), mod height (females).

'W' is weight info - base weight (males), base weight (females).

'F' is for racial flags.

'C' is for class choices.

'O' is for objects. This defines if the class has a built in object

in a particular equipment slot.

XXX We also use this file to define possible shapes that a player can shape

change into as an effect, in addition to their starting race. To define that

a race is selectable as a starting race, it must have an index less than

the number of 'playable' races defined in the limits.txt file.

## Spell.txt

=== Understanding spell.txt ===

N: number : name

A: tval : sval : slot

C: class : level : mana : failure : min failure

F: flags

B: attack method : attack effect : damage

L: duration

S: other effect : parameter

D: description

'N' indicates the beginning of an entry. The serial number must

increase for each new item.

'A' is for appears in which objects, referenced by tval and sval.

Slot must be unique for objects which have multiple spells

associated with them, and slots must be numbered from 1 upwards.

The exception is rune stones, where the slot corresponds to the

second ('verb') runestone sval.

'C' indicates the casting details listed by class. Note that currently

no more than 5 classes can cast a particular spell.

'F' flags indicating many effects that take place when the spell is

cast.A

'B' indicates the attack effect of the spell similar to monster attacks.

Note that blow specifies whether a directional parameter is needed,

and the increase in spell effect based on player level.

Note that we can specify a constant bonus to damage in this file, and

up to 4 blows are permitted.

If dice is > 0 and sides == 0, we use dice \* level + bonus, or

dice \* 25 + bonus, if the spell is the result of using an object.

Note that if you are applying blows to a player, you should apply the blow

that blinds them last, if possible, because they will not know the results

of the other blows.

'L' indicates how long the duration component of the spell lasts.

'S' indicates another parametised effect. The parameter value

depends on the effect in question.

'D' description indicates the description display when the spell

takes effect. If an item has a description, it will be identified

as a part of the activation of the item.

ToDo: Explain the other monster and spell flags. But maybe in another file.