The Idiot's Guide To Mod Making

for

Diablo II Lord of Destruction

by

Myhrginoc

Original Concept by
Phrozen Heart & Mordini

Additional Credits

Mike Meneldil	Heynooffense	Peaceatlast20
Rich Grenfell	Soul Slayer	Rage AA
Paul Siramy	Alkalund	Fusman

Version History

v1.3	6/22/2002	Complete rewrite to provide examples and detailed explanations. Incorporates several exercises which were previously stand-alone tutorials.
v1.2x beta	4/18/2002	Pretty much a total rewrite to hopefully help iron out any previous confusion that might have occurred. These versions were all closed betas only.
v1.1	12/8/2001	Added a link on the 'Step 1' page to the latest datafile for use with MPQ Viewer as some people have had problems extracting string files.
v1.0	10/4/2001	First public release.

Getting Started

The journey of a thousand miles
begins with a single step

Welcome to Diablo II, the way *you* want it! You have played the standard game enough, the Realms lag so much, and maybe you downloaded a modified version and wondered how they could change things so much. Now you want to try yourself, but don't know where to begin. This guide will show you where to find what you need, and how to make simple changes. There are often several ways to accomplish the same things; any method written in this document is but one way to get there.

Source Files

Look in your Diablo II directory (usually C:\Program Files\Diablo II). You will find a lot of files, most of which we won't concern ourselves with. The ones of interest are Game.exe and the files with an .mpq extension. Game.exe actually runs the game; you will also find Diablo II.exe, but that is only a loader for Game.exe. The mpq files are archive files that contain thousands of data files within them, everything from item definitions to the movies between acts. These data files can be grouped by their extensions.

d2char.mpq	data files for Diablo II – characters
d2data.mpq	data files for Diablo II – everything else
d2exp.mpq	data files and sound effects for Expansion features
d2music.mpq	music files for Diablo II
d2speech.mpq	character and NPC speech for Diablo II
d2xtalk.mpq	character and NPC speech for Expansion
d2sfx.mpq	sound effects for Diablo II
d2video.mpq	movies for Diablo II
d2xvideo.mpq	movies for Expansion
patch_D2.mpq	all changes introduced by any patch, not included in release versions

Table 1 - Archive Files for Diablo II Lord of Destruction

txt	1: database files for static game information 2: descriptive files which are not used in game
bin	compiled versions of the txt database files used when the game runs
cof	control files for animations
dc6	graphics for inventory, background panels and certain animations
dcc	character, monster, object and effects animations
ds1	map files used for area definition
dt1	graphics tiles used for surface appearances
tbl	string tables for text seen in the game
wav	sound files for effects, speech and music
dat	1 : Diablo II palettes, used for converting dc6 & dcc 2 : Colormaps, to see the same monster in different colors

Table 2 - Diablo II File Types – Inside the Archives

Tools of the Trade

Before you can even start, you will need to get your hands on the basic tools used to access and modify all of these files. All of the basic tools are available from the Phrozen Keep File Center.

- MPQ Viewer – for viewing and extracting files from the .mpq archives. There is a separate data file for all known contents of the archives, without it MPQ Viewer won't know what to look for.
- Tab Delimited Text Editor – for editing the database txt files. D2Excel is designed specifically for these files, and is available for download. Microsoft Excel or other spreadsheet programs will work, but there are some tricks to know. Although the files have .txt extensions, they are ***not*** ordinary text files, so do not use a word processor or even Notepad as they will corrupt the databases.
- Baron Darkstorm's Table Editor – for editing string table files.
- DC6Maker – for viewing and converting dc6 type graphics files. Editing graphics files is outside of the scope of this guide.
- CV5 – for extracting, viewing and converting several types of graphics files. One component, CVDCC.DLL, is required for dcc files but may be a separate download. Editing graphics files is outside of the scope of this guide.
- MPQ2K and MPQ Stormless Editor – for packing your files into an mpq after you have changed them.

Another file that may need to be downloaded separately is patch_D2.mpq. This file is not included on the CD-ROM releases for either Diablo II or Lord of Destruction. You get them normally by logging on to Battle.net and getting the latest patch pushed to your machine. You can also download patches and install them separately. Patch_D2.mpq stores updated versions of any file that is located in *any* of the release mpq's. When you make your changes to a file from *any* mpq, you need to store it in patch_d2.mpq, so you should not try to modify a CD-only installation of the game. Run at least one patch to update the game and obtain patch_D2.mpq. Information in patch_D2.mpq will always override earlier versions of the same data files stored in the release archive files.

1.07	6/19/2001	Release version of Lord of Destruction
1.08	6/27/2001	Maintenance and features upgrade
1.09	8/21/2001	Major change in game features
1.09b	10/5/2001	Maintenance update
1.09c	11/29/2001	Maintenance update
1.09d	12/5/2001	Maintenance update
1.10	*?*	*Pending major feature changes*

Table 3 - Lord of Destruction Update History

Many mod makers are downgrading to version 1.09b because of a serious bug introduced with 1.09c and still present in 1.09d. Blizzard had provided certain skills on magical equipment that would be activated by combat events, these skills are known as "chance-cast skills." But with version 1.09c and later, the chance-cast skills are displayed but they cause no damage. You can downgrade to version 1.09b by replacing three program files with earlier versions: they are D2Client.dll, D2Common.dll and D2Game.dll. All three 1.09b dlls are available at the Phrozen Keep. Save your 1.09d dlls for use on Battle.net.

Now you need a way to keep track of all of the files you will be generating within your Diablo II directory. One method is to make subdirectories within your Diablo II directory, with one mod or test setup in each subdirectory. You can have a shortcut to Game.exe with a mod directory as your starting directory. There is also a package of Visual Basic scripts available at the Phrozen Keep which will run your mods in their own subdirectory and automate changing DLLs for those mods that provide custom DLLs. You can use the DLL management feature to run mods with 1.09b DLLs while keeping unmodded Diablo II at the current version for Battle.net play.

The very first thing you should do after downloading the tools is make yourself a workspace separate from the game files. Make a copy of patch_D2.mpq and save it someplace safe, that way you always have a clean version to work from. Make another copy and move it to your workspace. Install the tools in the workspace so they always look for files there. This is especially important for MPQ2K, which uses a file with the same name as one of the game components; you must not get these versions mixed up!

Exercise #1 – Increasing Quantities

We will start with a very simple modification, changing the number of keys that can stack together. The standard limit is twelve, which is a very low number for chest-studded dungeons like the Durance of Hate. We will make the upper limit fifty so we don't keep running back to town for more.

To begin, you will need to run MPQ Viewer. Start the program and click on the File menu.
Note there are two Open options. The first time you run MPQ Viewer you need to load a datafile, which tells MPQView what the structure of the archive files is and where to find the various internal files. To load a datafile, select the first Open option as shown below. You only need to do this once as MPQ Viewer will remember the last datafile used.

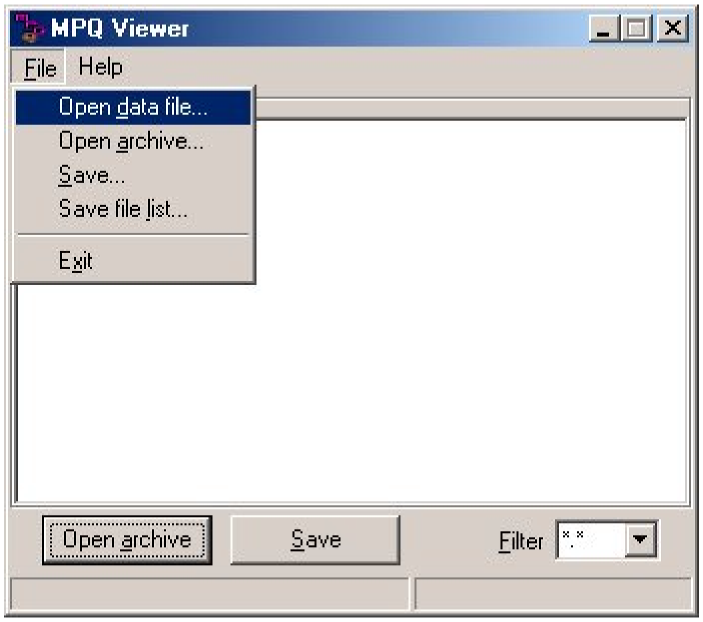
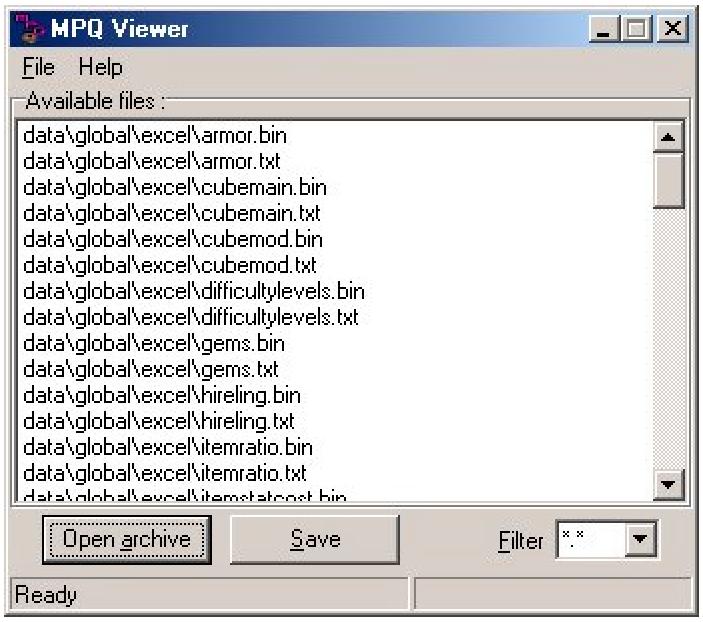


Figure 1 - MPQ Viewer Open Datafile

Next, click the Open Archive button, or select from the File menu. Select patch_D2.mpq in the File Open dialog, and the entire contents of the mpq will appear in the main window.

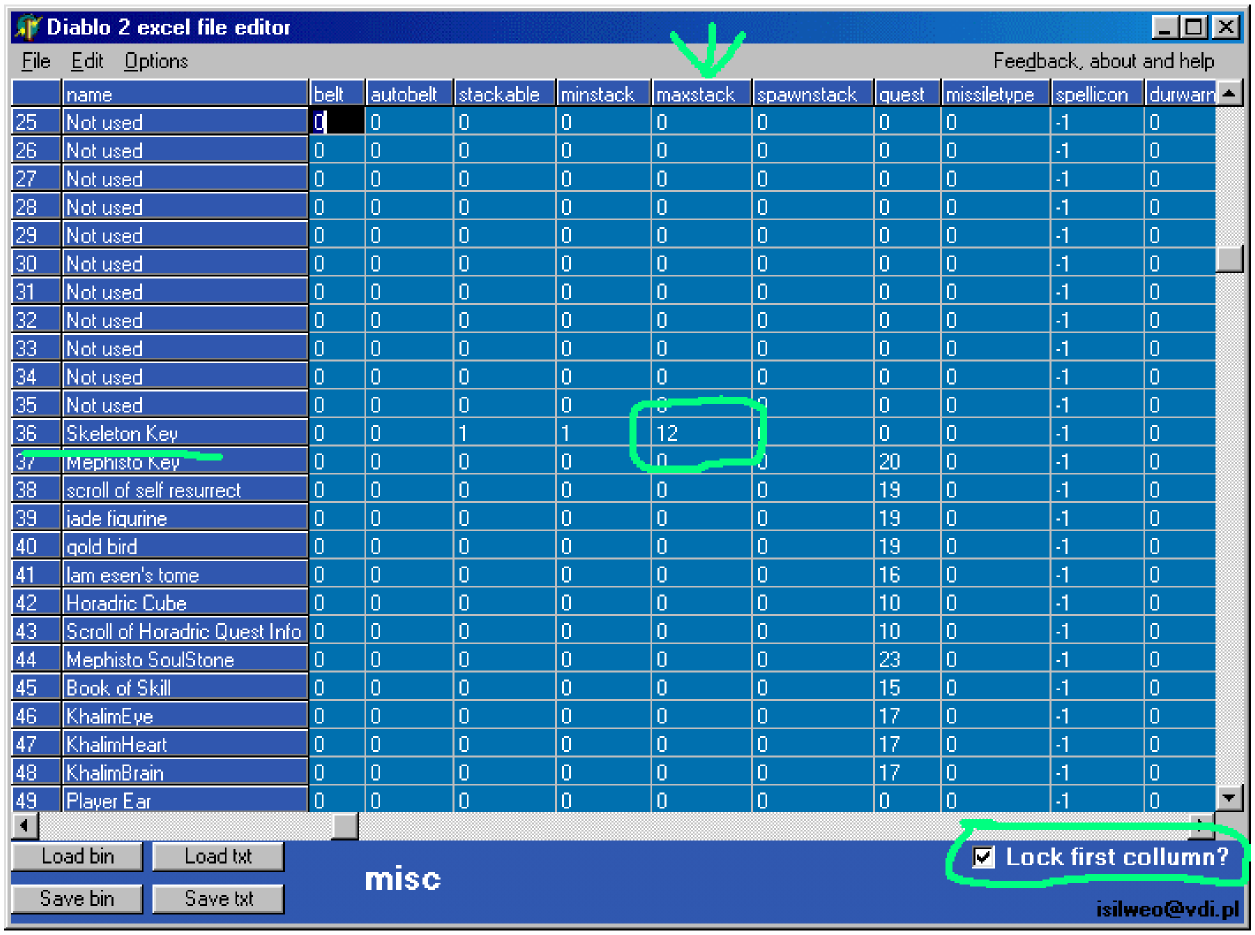


data\global\excel\armor.bin
data\global\excel\armor.txt
data\global\excel\cubemain.bin
data\global\excel\cubemain.txt
data\global\excel\cubemod.bin
data\global\excel\cubemod.txt
data\global\excel\difficultylevels.bin
data\global\excel\difficultylevels.txt
data\global\excel\gems.bin
data\global\excel\gems.txt
data\global\excel\hireling.bin
data\global\excel\hireling.txt
data\global\excel\itemratio.bin
data\global\excel\itemratio.txt

Figure 2 - MPQ Viewer Archive Listing

The paths you see in front of each filename are their position in the mpq archive, similar to how files and directorys work in Windows. Table 2 shows there are two types of txt files, database files and descriptive files. We are interested in the database files only, and they are all conveniently grouped together in a single internal mpq directory, data\global\excel. Scroll down until you see the files listed in this mpq directory. The file we want is called Misc.txt, so highlight that file and click on the Save button. Save this file to your workspace. Shut down MPQ Viewer, we are done with it for this exercise.

Open up D2Excel next. Then open Misc.txt using the Load Txt button or file menu option. You will see something like this screen below. Here is where you see how these text files are actually databases. Each row is a record, and each column is a field. We want the record for keys so we can adjust the value of the maximum stack field.



	name	belt	autobelt	stackable	minstack	maxstack	spawnstack	quest	missiletype	spellicon	durwarn
25	Not used	0	0	0	0	0	0	0	0	-1	0
26	Not used	0	0	0	0	0	0	0	0	-1	0
27	Not used	0	0	0	0	0	0	0	0	-1	0
28	Not used	0	0	0	0	0	0	0	0	-1	0
29	Not used	0	0	0	0	0	0	0	0	-1	0
30	Not used	0	0	0	0	0	0	0	0	-1	0
31	Not used	0	0	0	0	0	0	0	0	-1	0
32	Not used	0	0	0	0	0	0	0	0	-1	0
33	Not used	0	0	0	0	0	0	0	0	-1	0
34	Not used	0	0	0	0	0	0	0	0	-1	0
35	Not used	0	0	0	0	0	0	0	0	-1	0
36	Skeleton Key	0	0	1	1	12	0	0	0	-1	0
37	Mephisto Key	0	0	0	0	0	0	20	0	-1	0
38	scroll of self resurrect	0	0	0	0	0	0	19	0	-1	0
39	jade figurine	0	0	0	0	0	0	19	0	-1	0
40	gold bird	0	0	0	0	0	0	19	0	-1	0
41	lam esen's tome	0	0	0	0	0	0	16	0	-1	0
42	Horadric Cube	0	0	0	0	0	0	10	0	-1	0
43	Scroll of Horadric Quest Info	0	0	0	0	0	0	10	0	-1	0
44	Mephisto SoulStone	0	0	0	0	0	0	23	0	-1	0
45	Book of Skill	0	0	0	0	0	0	15	0	-1	0
46	KhalimEye	0	0	0	0	0	0	17	0	-1	0
47	KhalimHeart	0	0	0	0	0	0	17	0	-1	0
48	KhalimBrain	0	0	0	0	0	0	17	0	-1	0
49	Player Ear	0	0	0	0	0	0	0	0	-1	0

Figure 3 - Using D2Excel on Misc.txt

Now click on the Lock First Column box in the lower right corner. This will allow us to see the names of the records as we pan the field display. Scroll down until you see Skeleton Key, then scroll right until you see MaxStack in the header row. Where the row and column intersect is the value we want to change. You see it is already 12, that is what Blizzard assigned. Now click on that cell and change the number to 50. Save the file with the Save txt button.

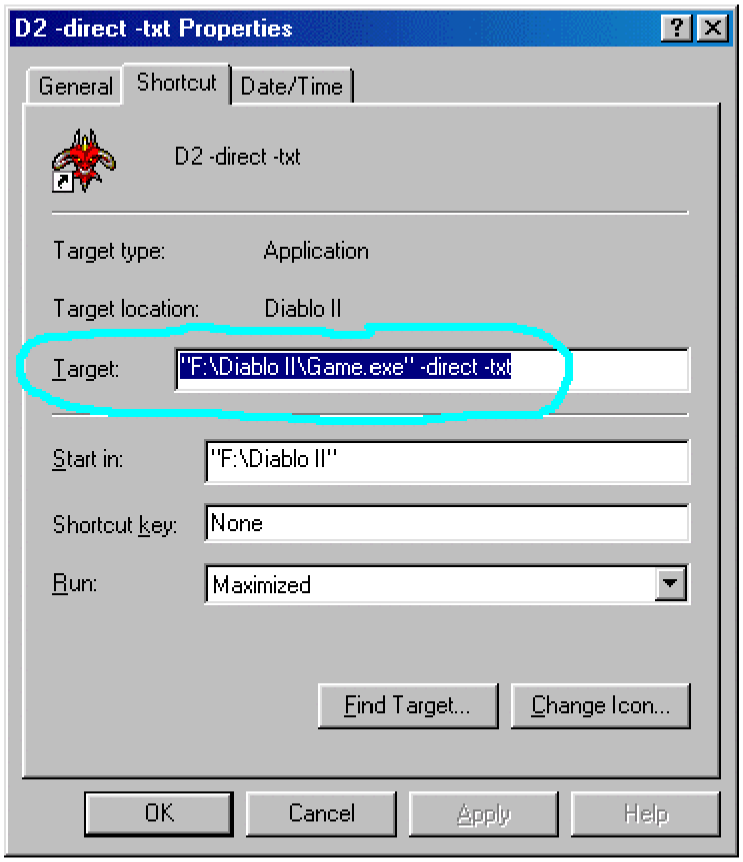
How did we know Skeleton Key was what we wanted to change, when in game all we see is Key? That is one of the hard parts of txt files, the name shown here is not necessarily what you would see in play. The names here are string keys, which are used to *point to* the display name in the tbl files. We will go into greater detail during the fourth exercise.

You can see quickly how this change could be applied to *any* stackable item. Do you want more arrows in your quivers? Do you want 40-scroll spell books? Just edit the same field in their records. But watch your numbers! The game will not accept any value over 511.

Making It Work

Now we have made a change, how do we get it into the game? At this point you have to decide if your mod will use mpq files or run in –direct –txt mode. What does this mean? In the first mode, you pack your changes into patch_D2.mpq, move patch_D2.mpq into your game directory, and start the game as usual. Also, if you are planning to share your mod with a Mac user, this is the only way that works on a Mac. The second mode doesn't require packing into the mpq, but you have to set up your system for it. You have to use this second mode to get the compiled database file in any case, so we will start there.

In MPQ Viewer you saw how the txt and bin files are stored in the mpq in a data\global\excel directory. We have to make our own copy of that directory structure in our game directory. In Windows Explorer, go to your game directory, and create a new directory called **data**. Inside that directory make one called **global**. Inside the global directory make one more called **excel**. Now go back to your game directory and highlight game.exe. Copy the file, switch to your desktop, and right click to paste a shortcut on the desktop. Next, right click on the new shortcut and choose Properties at the bottom of the popup menu. You should see this screen:



D2 -direct -txt

Target type: Application

Target location: Diablo II

Target: "F:\Diablo II\Game.exe" -direct -txt

Start in: "F:\Diablo II"

Shortcut key: None

Run: Maximized

Figure 4 - Making a -direct -txt shortcut

Here is where you enter the information that makes Diablo II run in –direct –txt mode. In the Target field, make sure the full path name (use your system's path) is in quotes as above. Then outside the quotes add –direct –txt ***spaced as above***. Save the shortcut.

Now copy the newly-changed Misc.txt to the data\global\excel directory you made in your game directory. Start Diablo II with the new shortcut and send a test character into a game. You should always use test characters in any new mod so if you crash and somehow scramble your character file, you won't have lost something important. Quit this game, unless you want to go buy some keys and see if you get more than twelve.

Each time you start a game in –direct –txt mode, you compile a new bin file in the data\global\excel directory for each available txt file unless one is already there. It doesn't matter if the text file is located in an external directory or in one of the mpq files, in any event you will end up with about seventy bin files. If later you change txt files after generating bin files, it is a good idea to delete all of the bin files in data\global\excel before running Diablo II in –direct –txt mode. This ensures you will start with bin files that correspond with the latest txt files.

You can continue to play mods using the directories you created, or you can pack into patch_D2.mpq. We will do the latter now. Go into the data\global\excel directory you made. Find misc.bin and copy it to your workspace.

MPQ2K is a DOS program

Before we can run MPQ2K we have to make a couple of supporting files. MPQ2K doesn't run under Windows, so if you were to click on the file in Windows Explorer, you would see a DOS window flash on your screen and disappear. What you need to do is make a script file with MPQ2K commands and a batch file to run the program. Fire up Windows Notepad; both of these new files are ordinary text files.

We will start with the script file. The important commands in MPQ2K are Open an Archive, Add to an Archive, and Close an Archive. In Notepad, make a four-line file as below and save it as MPQLoad.txt.

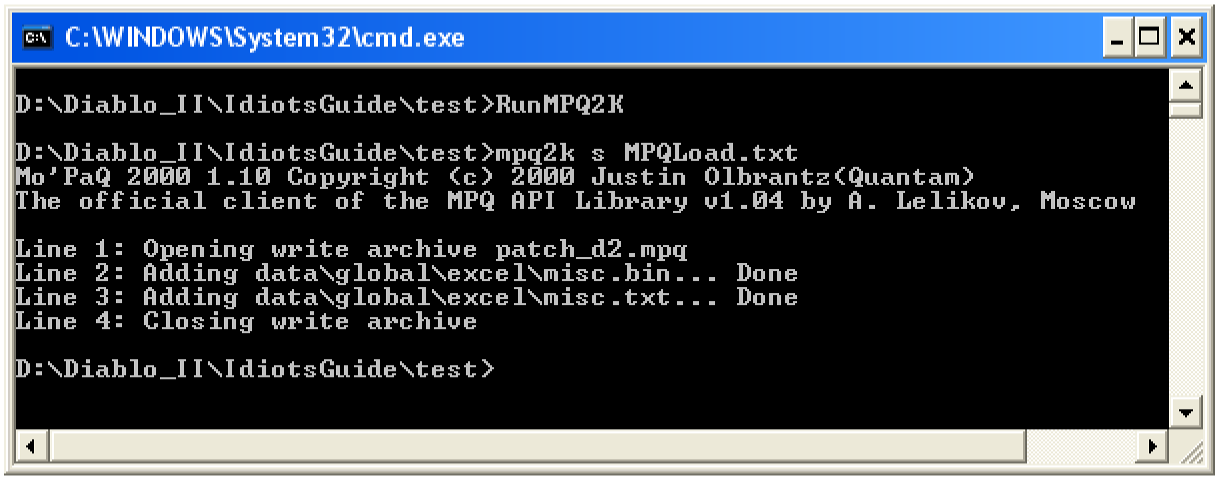
```
O patch_D2.mpq
a misc.txt data\global\excel\misc.txt
a misc.bin data\global\excel\misc.bin
C patch_D2.mpq
```

The Open and Close archive commands must have capital letters. For the add file commands, there is an upper case form and a lower case form. The only difference between the two is the lower case form also compresses the file it inserts, a useful way to minimize mpq size as you pack more and more files into it.

Now for the batch file. Type the single line in Notepad, and save this file as RunMPQ2K.bat. Be sure to replace "Text Document" with "All Files" in the Save As Type dropdown or your files will be named RunMPQ2K.bat.txt.

```
mpq2k s MPQLoad.txt
```

You are ready to pack the mpq at last. Make sure you have your new bin and txt files, your script and batch file, and MPQ2K's four files (mpq2k.exe, staredit.exe, storm.dll, lmpqapi.dll) in your workspace directory. Double-click on the batch file in Windows Explorer. You will still see a DOS window flash and disappear, but at least it is doing something. If you open a DOS window, you can run the batch file and see the results, as below:



```
D:\Diablo_II\IdiotsGuide\test>RunMPQ2K

D:\Diablo_II\IdiotsGuide\test>mpq2k s MPQLoad.txt
Mo'PaQ 2000 1.10 Copyright (c) 2000 Justin Olbrantz(Quantam)
The official client of the MPQ API Library v1.04 by A. Lelikov, Moscow

Line 1: Opening write archive patch_d2.mpq
Line 2: Adding data\global\excel\misc.bin... Done
Line 3: Adding data\global\excel\misc.txt... Done
Line 4: Closing write archive

D:\Diablo_II\IdiotsGuide\test>
```

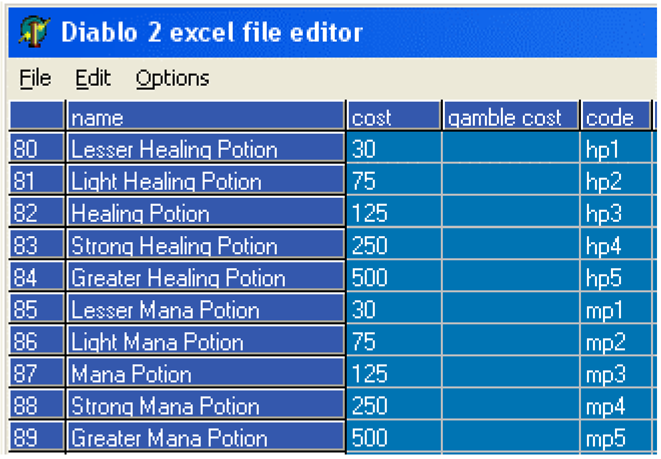
Figure 5 - MPQ2K in a DOS Window

Check one more time to be sure you copied the original patch_D2.mpq to a safe place. Now copy the modified patch_D2.mpq to your game directory. Run the game normally, don't use the –direct –txt shortcut you made earlier. Play for a while and see that you can stack up more than twelve keys in one slot. Congratulations on making your first D2 mod!

Exercise #2 – Buyable Mana Potions

In the first exercise you learned how to extract files, edit txt spreadsheets, run in –direct –txt mode and pack the mpq with your changes. For this exercise we will return to Misc.txt and learn how to make vendors sell additional goods. The techniques are the same; we are just changing different records and fields.

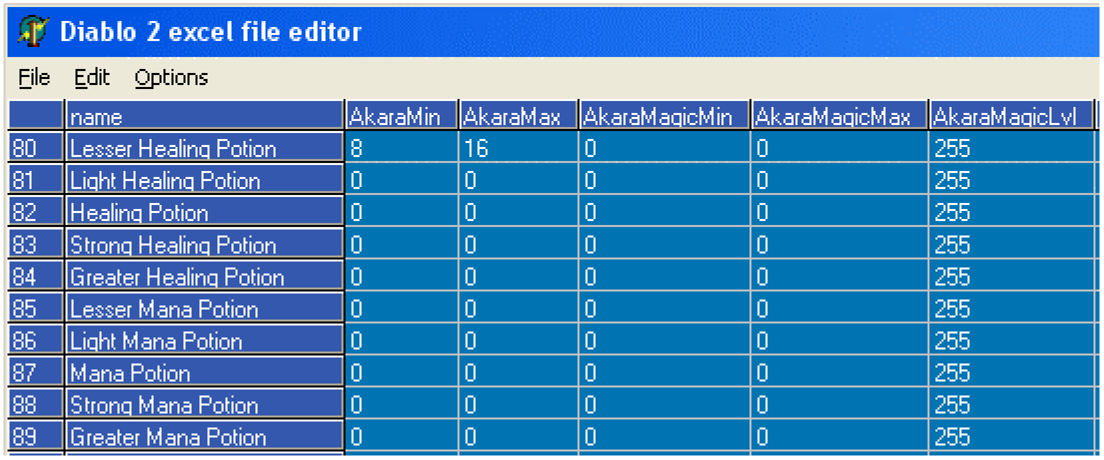
Open Misc.txt and find the lines for healing and mana potions. Scroll right until you see the Code field. You can see how the names in the Name column don't agree with the display names for these potions. For example, Strong Healing Potion here shows in game as Greater Healing Potion, while Greater Healing Potion here is the Super Healing Potion in game.



	name	cost	gamble cost	code
80	Lesser Healing Potion	30		hp1
81	Light Healing Potion	75		hp2
82	Healing Potion	125		hp3
83	Strong Healing Potion	250		hp4
84	Greater Healing Potion	500		hp5
85	Lesser Mana Potion	30		mp1
86	Light Mana Potion	75		mp2
87	Mana Potion	125		mp3
88	Strong Mana Potion	250		mp4
89	Greater Mana Potion	500		mp5

Figure 6 - Potions in Misc.txt

Now scroll right to the many fields where vendors are given the ability to sell any items. Each vendor has five fields. We will start with Akara, since she already sells healing potions. Getting her to sell mana potions also is as simple as using mana potion codes in the same way healing potion codes are used. In this case, we would enter 8 in AkaraMin and 16 in AkaraMax for the Lesser Mana Potion record. The other three fields refer to magical items, which is outside of the scope of this exercise.

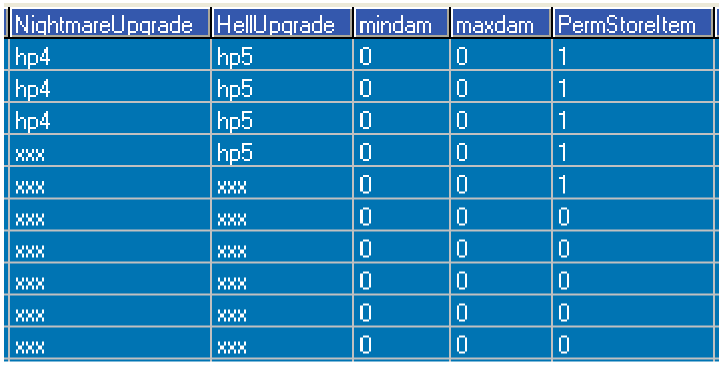


	name	AkaraMin	AkaraMax	AkaraMagicMin	AkaraMagicMax	AkaraMagicLvl
80	Lesser Healing Potion	8	16	0	0	255
81	Light Healing Potion	0	0	0	0	255
82	Healing Potion	0	0	0	0	255
83	Strong Healing Potion	0	0	0	0	255
84	Greater Healing Potion	0	0	0	0	255
85	Lesser Mana Potion	0	0	0	0	255
86	Light Mana Potion	0	0	0	0	255
87	Mana Potion	0	0	0	0	255
88	Strong Mana Potion	0	0	0	0	255
89	Greater Mana Potion	0	0	0	0	255

Figure 7 - Potion Selling by Akara

You can go from act to act making the same changes for Lysander, Alkor, Jamella and Malah, giving each vendor the ability to sell mana potions just like they already sell healing potions.

The last part of this exercise is making vendors sell better potions in higher difficulties, and never running out of potions. To do this, we will scroll to the far right of the database.



NightmareUpgrade	HellUpgrade	mindam	maxdam	PermStoreItem
hp4	hp5	0	0	1
hp4	hp5	0	0	1
hp4	hp5	0	0	1
xxx	hp5	0	0	1
xxx	xxx	0	0	1
xxx	xxx	0	0	0
xxx	xxx	0	0	0
xxx	xxx	0	0	0
xxx	xxx	0	0	0
xxx	xxx	0	0	0

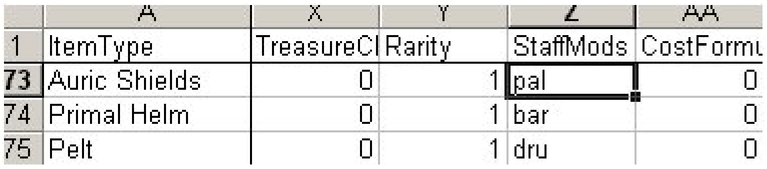
Figure 8 - Better Potions, Always Potions

Observe that the better grades of healing potions are listed in the upgrade fields, and use the same pattern in the mana records. Code “mp4” is Greater Mana Potion and code “mp5” is Super Mana Potion. Code “xxx”means not applicable, so we want to change the corresponding xxx values to mp4 and mp5. Last of all, change the PermStoreItem field to 1 for the mana potion records to make them always available.

As in the first exercise, save the edited file to your workspace directory. Copy the new Misc.txt to your data\global\excel directory, use your –direct –txt shortcut to start a game and generate bin files. Then copy misc.bin back to your workspace and pack patch_D2.mpq. You can use the same MPQ2K script as last time, since these changes were in the same database file.

Exercise #3 – Enchanted Paladin Shields

In this exercise we will look at the automatic modifiers some item types have. The automatic mods are usually bonuses to skills, such as sorceress skills on staves. Because sorceresses always have had skills available on staves, these automagic mods are often called Staff Mods. You can see this in the column header too. Except as detailed below, all steps are the same as in the first exercise. For this exercise we will need ItemTypes.txt.



	A	X	Y	Z	AA
1	ItemType	TreasureCl	Rarity	StaffMods	CostFormu
73	Auric Shields	0	1	pal	0
74	Primal Helm	0	1	bar	0
75	Pelt	0	1	dru	0

Figure 9 - Paladin Shields

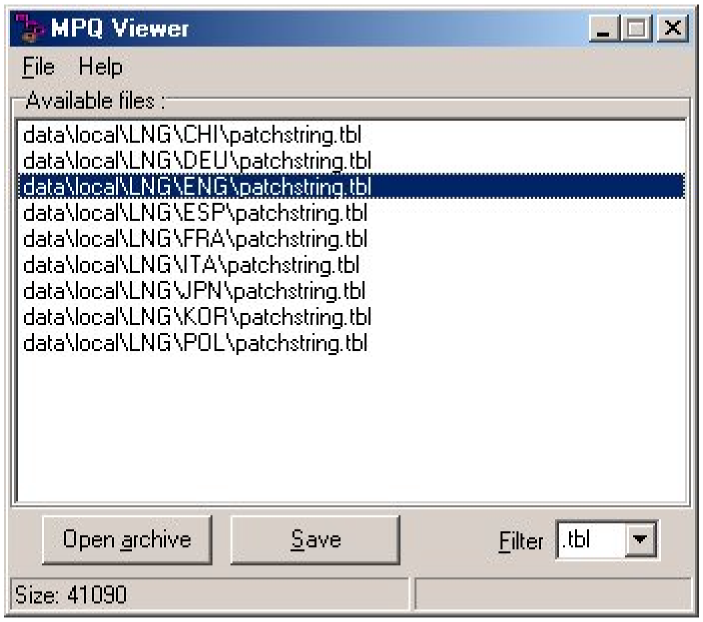
This view shows Microsoft Excel using the Freeze Panes display mode, which you set by positioning the cursor at Cell B2 and then selecting menu option Window | Freeze Panes—this has the same effect as locking the first column in D2Excel. Scroll down until you see the Auric Shields record, then scroll right until you see the StaffMods field. Enter the class code for Paladin in the field as shown above. This will make all Auric Shields spawn with Paladin skills, similar to scepters, in addition to their inherent resistances. Save this file to your workspace, test and insert to mpq as usual.

This example shows applying Paladin skills to a Paladin shield (remember "Auric Shield" is not what gets displayed). In our case, you could have put other Class skills (like Sorc skills) than the Paladins ones, but only Paladins can use this item type so that would be useless. But you *can* apply any class to the StaffMods field for any item type. Remember *all* items spawned from an item type will get the bonus skills, so setting the "tors" item type with "dru" staff mods would result in every single body armor having druid skills, which is rather extreme.

Exercise #4 – Modifying an Item

In this exercise, you will make property changes and learn about the tbl files. We are going to replace the unique ring Manald Heal with a much more powerful unique, The Eye of Mordini. Except as detailed below, all steps are the same as in the first exercise.

Open MPQ Viewer and extract UniqueItems.txt from the mpq's data\global\excel directory and save to your workspace directory. Then look for the string table named patchstring.tbl. Here we make use of the filter function in MPQ Viewer, but you have to edit the default .TBL filter to lower case for it to work. (The filters are buggy and in some cases will miss files. The only way to assure yourself of seeing every file is to leave the filter as *.* and scroll down the entire list.)



```
data\local\LNG\CHI\patchstring.tbl
data\local\LNG\DEU\patchstring.tbl
data\local\LNG\ENG\patchstring.tbl
data\local\LNG\ESP\patchstring.tbl
data\local\LNG\FRA\patchstring.tbl
data\local\LNG\ITA\patchstring.tbl
data\local\LNG\JPN\patchstring.tbl
data\local\LNG\KOR\patchstring.tbl
data\local\LNG\POL\patchstring.tbl
```

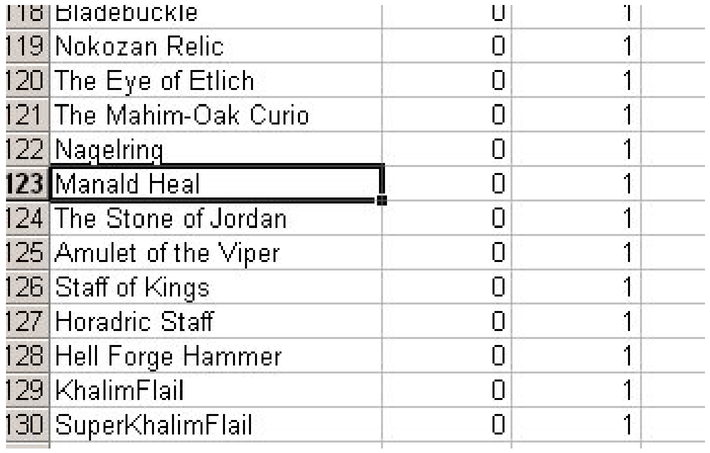
Filter: .tbl

Size: 41090

Figure 10 - Choosing a PatchString.tbl File

You will see there are a number of patchstring.tbl files. Pick the file that is in the directory corresponding to the language you run Diablo II in. For example, if you have a Spanish installation, choose the string table in the data\local\LNG\ESP directory. Extract the appropriate file to your workspace directory.

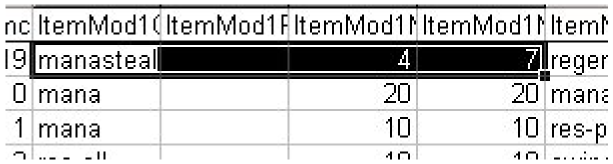
Now load UniqueItems.txt for editing as below. This example shows the database file in Microsoft Excel. Scroll down until you see the record for Manald Heal. Remember from the first exercise this name is *not* the one that displays in the game!



118	Bladebuckle	0	1
119	Nokozan Relic	0	1
120	The Eye of Etlich	0	1
121	The Mahim-Oak Curio	0	1
122	Nagelring	0	1
123	Manald Heal	0	1
124	The Stone of Jordan	0	1
125	Amulet of the Viper	0	1
126	Staff of Kings	0	1
127	Horadric Staff	0	1
128	Hell Forge Hammer	0	1
129	KhalimFlail	0	1
130	SuperKhalimFlail	0	1

Figure 11 - Finding Manald Heal in UniqueItems.txt

Scroll right through the fields until you get to the one labeled ItemMod1Code (column O in Excel). This is the first magical property that the ring possesses, "manasteal" which is obviously Steal x% Mana. Download the Magic Code Master List from the Phrozen Keep to see what all of the magic modifiers are, or you can extract Properties.txt from the mpq for yourself.

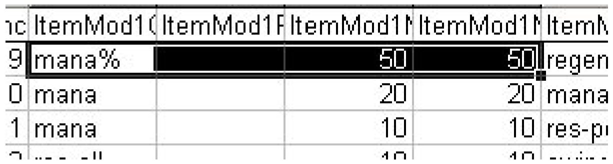


nc	ItemMod1(	ItemMod1F	ItemMod1N	ItemMod1N	ItemN
19	manasteal		4	7	reger
0	mana		20	20	mana
1	mana		10	10	res-p

Figure 12 - First Property of Manald Heal

There are four fields associated with each property. The first field is the Magic Code, followed by three fields (ItemMod1Param, ItemMod1Min and ItemMod1Max) that define the strength of its effects according to the Master List. The first property for Manald Heal is shown above. ItemMod1Param is blank, but the other two fields are used. With ItemMod1Min less than ItemMod1Max, when the game spawns a Manald Heal the ring will appear with a random % Mana Steal between the two values.

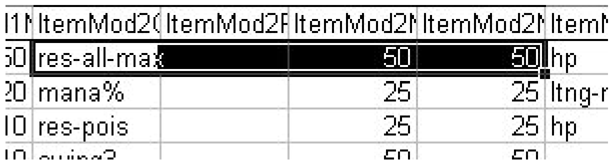
Change the ItemMod1Code field from "manasteal" to "mana%" and change both ItemMod1Min and ItemMod1Max to 50. You have now removed the mana stealing ability from the ring and added +50% to your maximum mana. With both ItemMod1Min and ItemMod1Max at the same value, that property no longer has a range of possible values; it will always spawn the same.



ıc	ItemMod1(	ItemMod1F	ItemMod1N	ItemMod1N	ItemN
9	mana%		50	50	regen
0	mana		20	20	mana
1	mana		10	10	res-p

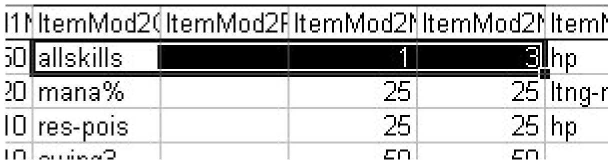
Figure 13 - Changing to Increased Maximum Mana

UniqueItems.txt has fields for up to ten separate properties, each with its four-field definition; all ten work the same way. Below are two examples of other changes you can make. Once you are done, save the file. If you are using Microsoft Excel or other spreadsheet, be sure to save the file as *tab delimited text file* or it won't work in the game.



I1N	ItemMod2(	ItemMod2F	ItemMod2N	ItemMod2N	ItemN
50	res-all-max		50	50	hp
20	mana%		25	25	ltng-r
10	res-pois		25	25	hp

Figure 14 - Add 50% to All Maximum Resists



I1N	ItemMod2(	ItemMod2F	ItemMod2N	ItemMod2N	ItemN
50	allskills		1	3	hp
20	mana%		25	25	ltng-r
10	res-pois		25	25	hp

Figure 15 - Add +1 to +3 to All Skills

Editing String Tables

We are not done yet. We changed the properties of the ring, but it is still called Manald Heal. So we have to change the string tables to show the new name for the item. Before we roll up our sleeves and get to work, we need to discuss what string tables are and how they function.

Almost all text seen in the game is stored in .tbl files. This includes items and their properties, monsters, NPC names, Battle.net logging and error messages, and all of the scrolling text seen when you talk to a NPC or click on one of the tomes. There are three .tbl files for each language. The oldest file, String.tbl, is located in d2data.mpq; it contains all of the original game's strings. Expansion strings are stored in ExpansionString.tbl located in d2exp.mpq. And most recent changes are stored in PatchString.tbl in patch_D2.mpq.

The string tables have two fields, regardless of which file you are using. The left field is the string key and the right field is what displays on screen. In order to properly show an item, the string key in the source database must match exactly the string key entered into a tbl file. In the case of Manald Heal, the name is used in both fields, so the string key in UniqueItems.txt was easy to decipher. But it isn't obvious that a Hand Axe is called a Hand Axe when its string key is the three-letter code "hax".

Sometimes a string is defined in more than one file. If so, any definition in ExpansionString.tbl will override any definition in String.tbl. Any definition in PatchString.tbl will override any definition in either of the other two files. You might think, why not make changes in PatchString.tbl and not worry about the other two? There are two reasons why you should not do that. The first reason is that every time Blizzard releases a patch, there will be a new PatchString.tbl. If you defined five hundred strings for a major mod, it is *not easy* to move those strings to the newer version of the file. The second reason is that mod makers have seen instability in PatchString.tbl that is not apparent in the other two; some people have no difficulty altering the file and others get bizarre errors. For those two reasons, many mod makers choose to extract String.tbl from d2data.mpq and pack it into patch_D2.mpq after they finish all of their changes. String.tbl is not changed at all by patches or the Expansion, so additions there are safe. But if you make a change to a record that appears in either of the other .tbl files, you have to change the record in those files. If you change String.tbl and not the others, you will never see your new string displayed.

Enough theory for now, let's change Manald Heal to The Eye of Mordini. Manald Heal is named in String.tbl since it is from the original Diablo II, so we can either change the display text in String.tbl or we can add a new record to PatchString.tbl. We will look at how to do both methods, even though you only extracted PatchString.tbl earlier.

Run the Darkstorm Table Editor (file name tbled102.exe). You will get this opening screen:

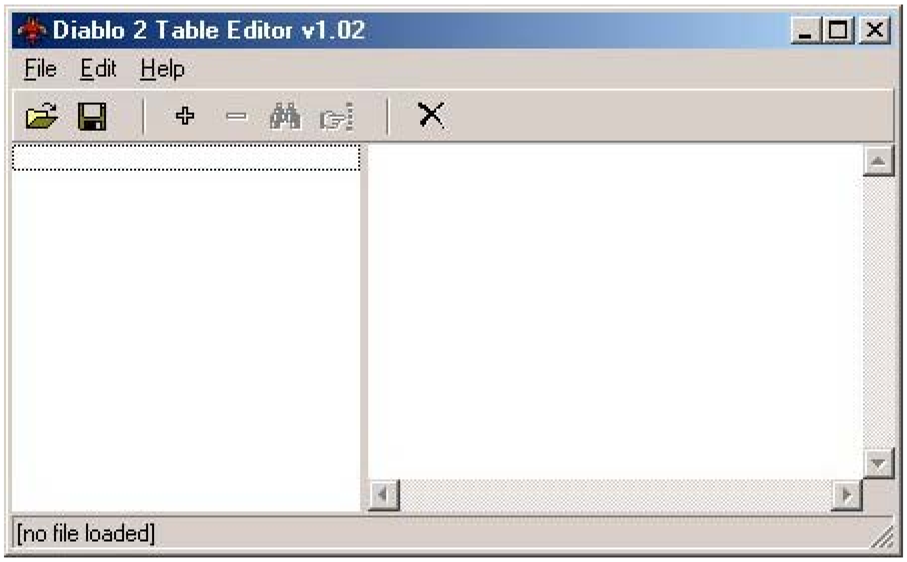
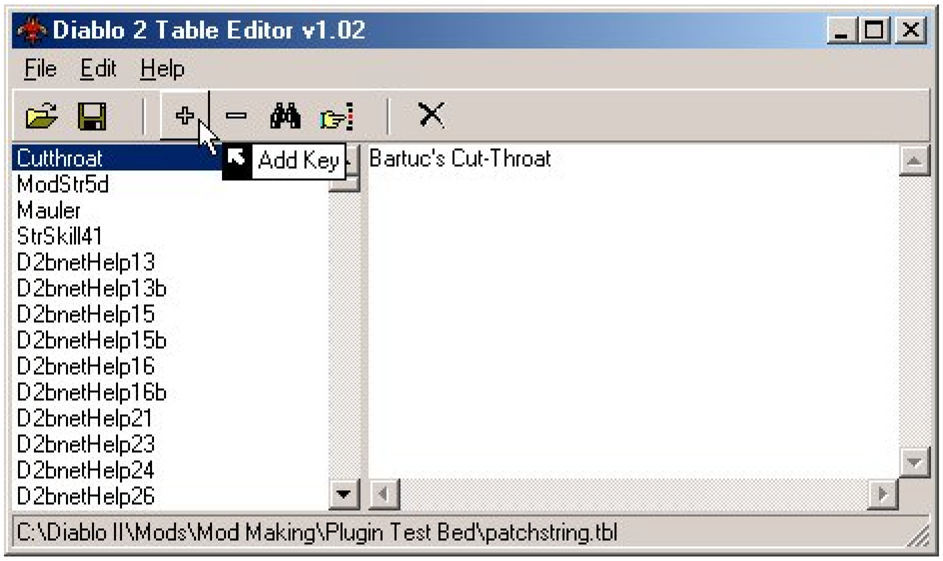


Figure 16 - Darkstorm Table Editor

Open PatchString.tbl and scroll down towards the bottom. You will notice the display string in the right field is only displayed when you have its string key selected. Also, some keys are nothing more than an "x" or occasionally a capital "X". These keys do not correspond to records in any of the txt files; they are special application strings directly referenced from the executable files. You will also see strings where the display text reads "All Resistances +%d" – this is where the text includes a variable that is filled in when the game is running.



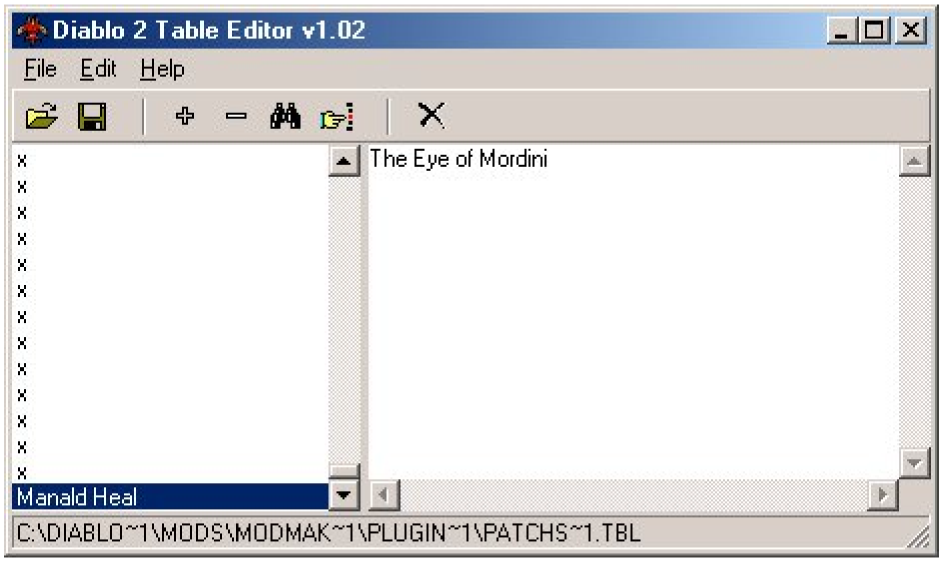
Cutthroat
ModStr5d
Mauler
StrSkill41
D2bnetHelp13
D2bnetHelp13b
D2bnetHelp15
D2bnetHelp15b
D2bnetHelp16
D2bnetHelp16b
D2bnetHelp21
D2bnetHelp23
D2bnetHelp24
D2bnetHelp26

Bartuc's Cut-Throat

C:\Diablo II\Mods\Mod Making\Plugin Test Bed\patchstring.tbl

Figure 17 - Table Editor with PatchString.tbl Loaded

Since we are adding a new key to PatchString.tbl, click on the Add Key button as shown above. You will jump to the end of the table and your cursor will be in the left field. You have one chance now to enter the new string key; you must match exactly the key for Manald Heal in UniqueItems.txt. Fortunately, the key is the same as the display name, so enter "Manald Heal". On the right, enter "The Eye of Mordini" as shown below. It doesn't matter to the user that the display name doesn't match the internal name at all. If you make a mistake, you must delete the key (button next to Add Key) and start over. Be sure you *never delete any of the existing keys*, if you won't be using one then delete the display text for that key only!



x
x
x
x
x
x
x
x
x
x
x
x
x
Manald Heal

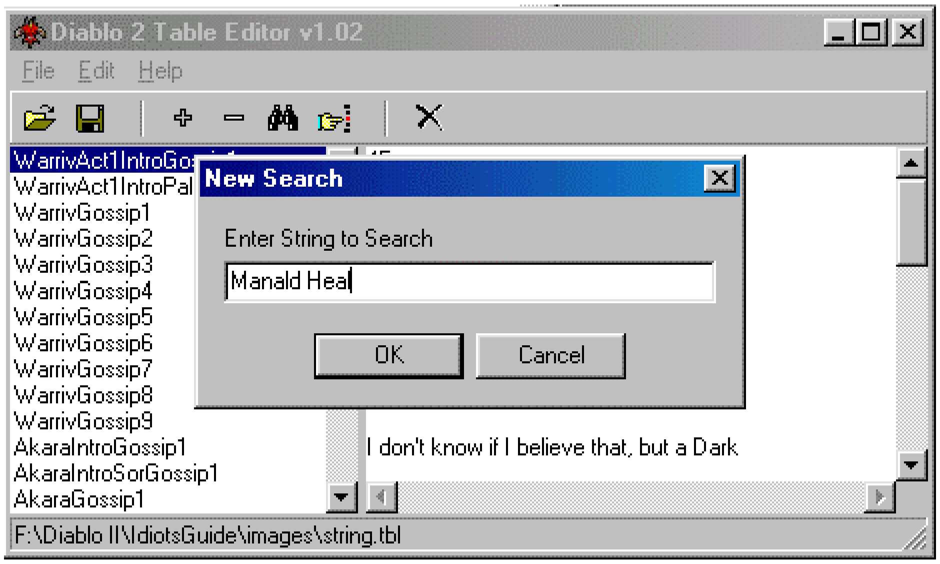
The Eye of Mordini

C:\DIABLO~1\MODS\MODMAK~1\PLUGIN~1\PATCHS~1.TBL

Figure 18 - Adding a New Key and String

Now save the .tbl file to your workspace directory.

If you were changing String.tbl instead of adding to PatchString.tbl, the process is similar. But instead of clicking on the plus for Add String, you would click on the binoculars for Find String. This function only works on the right field; you can't search by string key. Search for "Manald Heal", and when that record is found edit the right field to read "The Eye of Mordini".



Manald Heal

WarrivAct1IntroGo
WarrivAct1IntroPal
WarrivGossip1
WarrivGossip2
WarrivGossip3
WarrivGossip4
WarrivGossip5
WarrivGossip6
WarrivGossip7
WarrivGossip8
WarrivGossip9
AkaraIntroGossip1
AkaraIntroSorGossip1
AkaraGossip1

I don't know if I believe that, but a Dark

F:\Diablo II\IdiotsGuide\images\string.tbl

Figure 19 - Using the Search Function

As before, copy your modified UniqueItems.txt file to your data\global\excel directory. But now you need a place for the modified PatchString.tbl. So enter the data directory, and next to global add a new directory called **local**. Inside that directory make one for **LNG**, and inside there make one for the language code of your choice – ENG for English, DEU for German, etc. Copy your PatchString.tbl to the new directory.

Using your –direct –txt shortcut, start a game and generate bin files in your data\global\excel folder. Find your new UniqueItems.bin and copy it to your workspace. Now make a new version of your MPQ2K script file so you add the three changed files to patch_D2.mpq:

```
O patch_D2.mpq
a uniqueitems.txt data\global\excel\uniqueitems.txt
a uniqueitems.bin data\global\excel\uniqueitems.bin
a patchstring.tbl data\local\LNG\ENG\patchstring.tbl
C patch_D2.mpq
```

You have now completed the second exercise in elementary mod making!

Divide and Conquer!

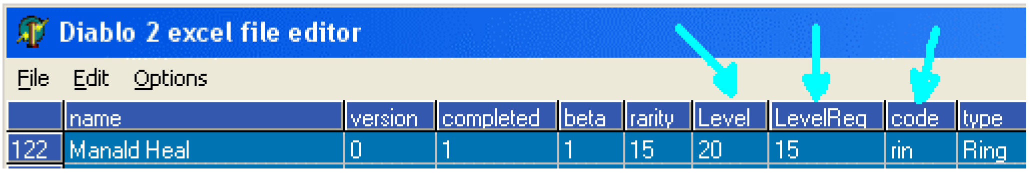
These two exercises show some simple changes. A full-blown mod has hundreds and even thousands of changes in it. Don't change everything at once! Start out making one change at a time, test and correct, then make the next change. As you gain experience you will feel more comfortable making more than a few changes between tests. But the more changes you test at one time, the harder it is to find the change that isn't working right.

In the next exercise you will learn how to test your new items.

Exercise #5 - Forcing an Item Drop

By now you have realized it is easy to find keys and arrows, but Paladin shields and Manald Heal rings don't fall out of the sky very often. There are several techniques available for forcing a specific drop, but going into their theory requires a certain familiarity with mod making. The method presented here is the Quill Rat Drop, named after those ubiquitous pests in the Blood Moor. It is easy enough if you don't worry too much about how it works. And here we make the distinction between a debugging version of a modified file and a release version.

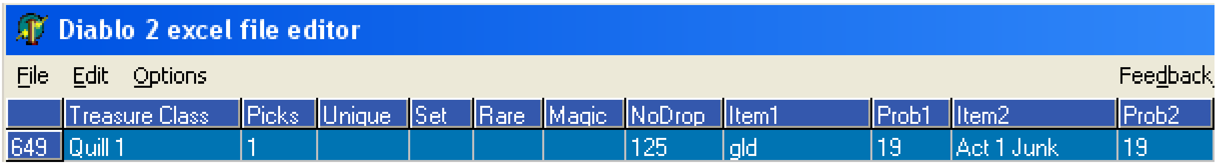
For this test we will extract and modify the file TreasureClassEx.txt. This is a fearsome-looking database containing most of the mechanism for selecting what gets dropped when you kill a monster or pop a chest. For now we will look at a very small part of this file, since Blizzard was kind enough to provide a set of records specifically for Quill Rats.



	name	version	completed	beta	rarity	Level	LevelReq	code	type
122	Manald Heal	0	1	1	15	20	15	rin	Ring

Figure 20 - Manald Heal Revisited

We will use the files we modified for the last exercise; we want to see what The Eye of Mordini actually looks like, and what it does for our characters. Make a copy of UniqueItems.txt so any changes we make can be reversed quickly; this edit is for debugging only. In UniqueItems.txt, look for the Manald Heal record again, this time viewing the first few fields. We are interested in three fields as shown above. First, change both Level and LevelReq to 1. This will allow a Quill Rat to drop the ring, and for our test character to wear it. Next, make note of the three-letter code "rin". We will need this code to tell the game we want Quill Rats to drop it. Save the file and copy it over to your data\global\excel directory.



	Treasure Class	Picks	Unique	Set	Rare	Magic	NoDrop	Item1	Prob1	Item2	Prob2
649	Quill 1	1					125	gld	19	Act 1 Junk	19

Figure 21 - Quill Rat Record in TreasureClassEx

Now open TreasureClassEx and scroll down to the first Quill Rat row. You will see these fields used, and everything to the right is blank until you get far over. This part shown is all we need though. First, delete all information from the NoDrop, Item2 and Prob2 fields. Next, replace "gld" with "rin" so the Quill Rat will drop a ring. Now go to the Unique field and enter the number 1024. Without going into a lot of detail, this forces the drop to be a unique ring. Quill Rats are very low level beasts in Normal difficulty, so the only unique ring they can drop will be our modified Manald Heal. Save this file and copy it to your data\global\excel directory.

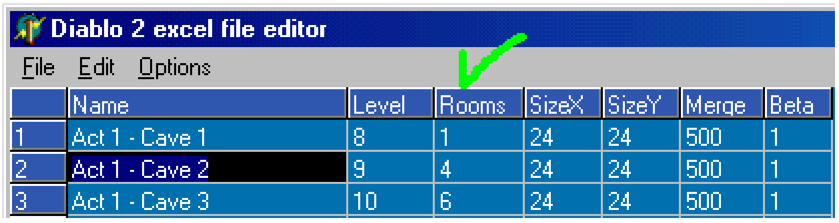
Now start a game using the –direct –txt shortcut. Go out into the Blood Moor and kill a few Quill Rats. Check their drops and you should find several copies of The Eye of Mordini. Identify them and they should be wearable. Put one on and look over your stats. Compare the visible properties and effects against what you intended with your changes to UniqueItems.txt. If everything looks fine, you have made your changes successfully and that feature is ready for release. Remember not to use your debugging files for mod release!

This method only works where the base item or magical property doesn't depend on the level of the character or monster, such as magic and rare items. To make the test more effective, you need to create a Level 90+ test character with access to Hell difficulty. You can use Jamella's Hero Editor or Shadowmaster to create high-level test characters. You won't need to change Level and LevelReq for most items when testing with a high-level character in Hell difficulty. Change the Quill 1 (H) record in TreasureClassEx, ten records past the one shown here, to force high-level drops.

Exercise #6 – Increasing Area Size

So far, all of the mods have been changes to item definitions and availability. There are many more ways to change the game! Changing and importing monsters, altering quests, changing graphics and NPC speech are features in many mods available at the Phrozen Keep, but many of these features are beyond the scope of this introductory guide. One change which is not difficult is to modify how certain levels are generated.

There are three types of areas in Diablo II. Outdoor levels are composed of many sections which are organized somewhat randomly. Each section is a separate .ds1 file. These areas are not expandable (yet). Other areas are "hard-coded" into one or more configurations, with special monsters and objects included as part of the .ds1 file. These areas are very difficult to expand.



	Name	Level	Rooms	SizeX	SizeY	Merge	Beta
1	Act 1 - Cave 1	8	1	24	24	500	1
2	Act 1 - Cave 2	9	4	24	24	500	1
3	Act 1 - Cave 3	10	6	24	24	500	1

Figure 22 - Editing Maze Size

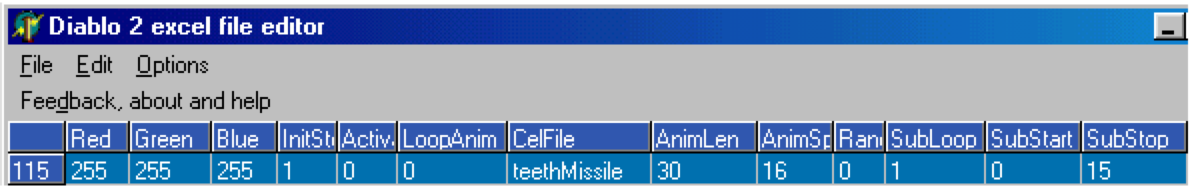
The third area type is the Maze, which is easy to expand. All you need to do is open LvlMaze.txt and edit the Room field. Mazes are found in most Acts:

Act 1	First floor of all caves (excluding Den of Evil) Crypt and Mausoleum underneath the Rogue Cemetary First four levels of the Forgotten Tower Jail and Catacomb levels of the Corrupted Monastery, except Andariel's Lair
Act 2	Sewers under Lut Gholein Stony Tomb Halls of the Dead Maggot's Lair Ancient Tunnels First floor of the Claw Viper Temple
Act 3	both Spider caves Flayer Dungeon Swampy Pit first level of Sewers under Kurast first two levels of the Durance of Hate
Act 4	{none}
Act 5	First floor of all Ice Caves

Other areas are listed in LvlMaze.txt. Some of them are known not to work with this method, such as the second (or "Treasure") levels of Act 1 caves or the three mini-Hells of Act 5. Others *might work* as they haven't been tested.

Exercise #7 – Editing Skills

Skills editing is a feature that covers the spectrum of mod making difficulty. Some changes can be accomplished by editing the skills.txt table, whereas other changes involve editing graphic files and even the game code itself. We will limit ourselves in this exercise to txt file changes. We will change the Necromancer Teeth skill into Ice Bolts. For this exercise, extract missiles.txt, skills.txt and string.tbl from the mpqs. In the figures below, certain fields have been squeezed to bring important fields together.



	Red	Green	Blue	InitSt	Activ	LoopAnim	CelFile	AnimLen	AnimSp	Ran	SubLoop	SubStart	SubStop
115	255	255	255	1	0	0	teethMissile	30	16	0	1	0	15

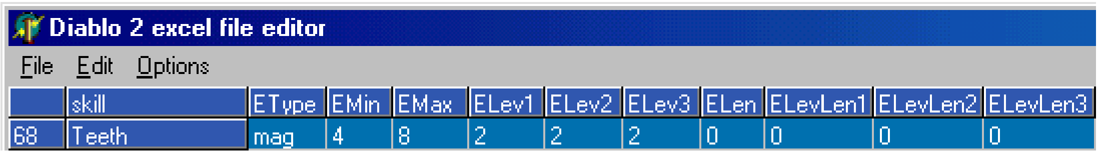
Figure 23 - Teeth in Missiles.txt

Start with missiles.txt, and scroll down to the Teeth record. Scroll right to the RGB color fields as shown above. The default for Teeth is 255/255/255, which is bright white. To give the new ice bolt a bluish cast, change these fields to 81/81/255.

CelFile is the name of the missile animation without the dcc extension. Change this field to “icebolt” (without the quotes). Each dcc file has a specific number of animation frames, and that number must match the number in the AnimLen field. Change this field to 6.

LoopAnim, SubLoop, SubStart and SubStop are fields which control how animations are repeated while the missile is in flight. Change LoopAnim to 1 and the other three fields to 0.

Check your changes against the existing record for Ice Bolt to confirm they match in the specific fields affected. Then save your work in missiles.txt.



	skill	EType	EMin	EMax	ELev1	ELev2	ELev3	ELen	ELevLen1	ELevLen2	ELevLen3
68	Teeth	mag	4	8	2	2	2	0	0	0	0

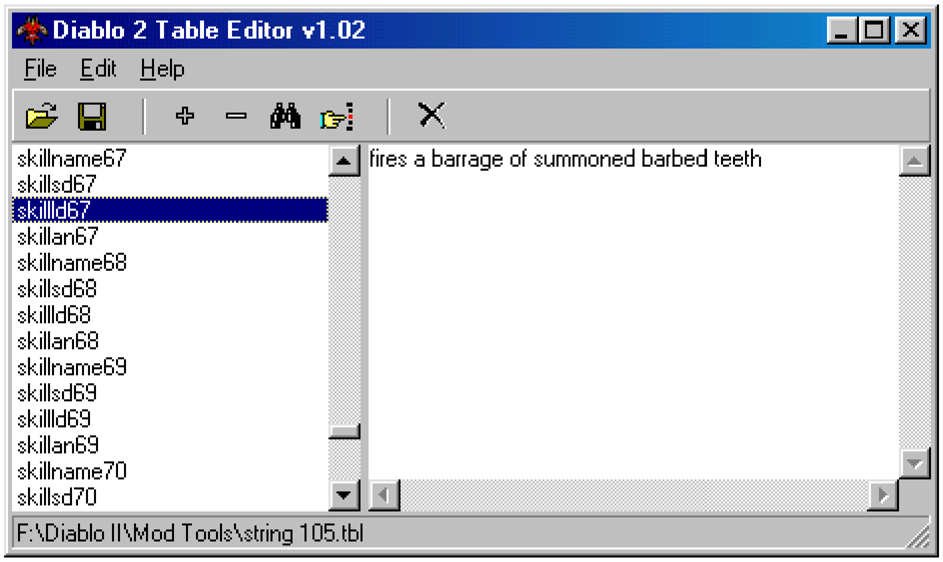
Figure 24 - Teeth in Skills.txt

Open skills.txt next and find the row for the Teeth skill. Scroll right until the elemental damage fields are visible, as shown above. Teeth is damage type magic but Ice Bolt is magic type cold, so make the following changes. First, change the Etype field to cold. Then, the damage per bolt is shown in Emin and Emax, for now we will leave the damage alone. The next three fields control the rate the skill potency increases per skill level, which we will leave alone as well. If we were to change these fields, the first field affects changes in skill potency for skill levels 1–8, the second field affects 9–16, and the third field affects all higher skill levels.

The last four columns control the duration of the skill effect. The basic unit of time in Diablo II is the frame rate. There are 25 and only 25 frames per second, this is hard-coded into the game. You may be using the “fps” display option (type “fps” in the in-game chat window, without quotes and without any other text, not even a space), this shows all sorts of performance metrics. But the frame rates shown in the fps display are <u>not</u> the frame rates used for game calculations. Cold is one of the damage types that is time-dependent, so we need some value other than 0 in the Elen field. For two seconds of cold duration, put 50 in the Elen field. The other three fields are similar to the potency/level fields described earlier. We will leave these three fields at 0.

Save your changes in skills.txt. Now we will change the strings associated with the newly-altered skill.

In the Table Editor, open string.tbl and use the search function (binoculars icon) to find instances of "teeth" in the file. The first instance is at string key strSkill28, which is a generic listing of the missile type. Change this value to "bolts". The next instance is at the head of the Teeth Skill key block, shown below.



skillname67
skillsd67
skillld67
skillan67
skillname68
skillsd68
skillld68
skillan68
skillname69
skillsd69
skillld69
skillan69
skillname70
skillsd70

fires a barrage of summoned barbed teeth

F:\Diablo II\Mod Tools\string 105.tbl

Figure 25 - Teeth Skill String Block

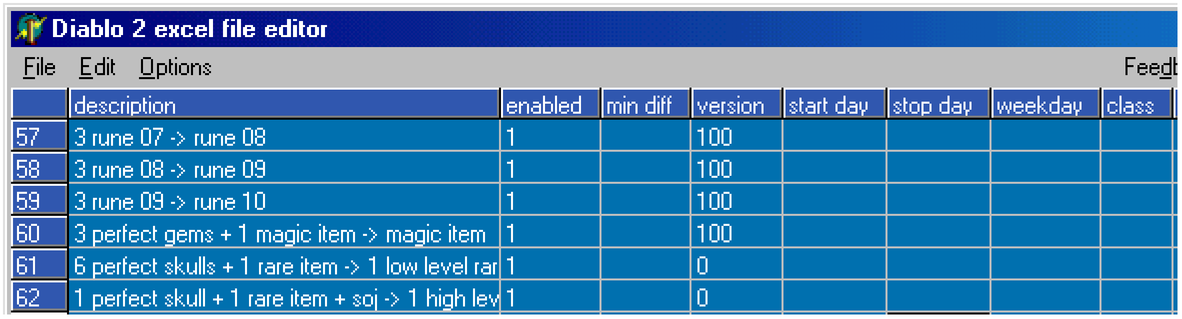
The first key of this block, SkillName67, is the display name for the skill. You may want to see "Ice Bolts" or "Ice Teeth" or something completely different; this is the record to change. Below that are two records for describing the new skill. Skillsd67 is the short description, it currently reads "fires barbed teeth". The next record is the long description, shown above. Edit these two fields to provide suitable descriptions, such as "fires ice bolts" for the short form and "fires a sleeting barrage of deadly frigid shards" for the long form. The last record of the skill block is the alternate name, skillan67. In most cases the alternate name is the same as the display name, so change this copy from "Teeth" to whatever you have in the skillname67 record.

Save these changes to string.tbl. Generate bin files for missiles.txt and skills.txt, and insert all five files into patch_D2.mpq. Now play your Necromancer and see how the skill has changed. You should now see cold damage against your foes. When you highlight your skill, the name and description changes should read as you entered them in string.tbl.

Many skills have information which is linked through code to the values in skills.txt. A more advanced technique where you swap skills between classes requires DLL editing. But this exercise shows you can change the behavior of skills considerably without knowing anything about graphics or assembly language.

Exercise #8 – Cube Recipes

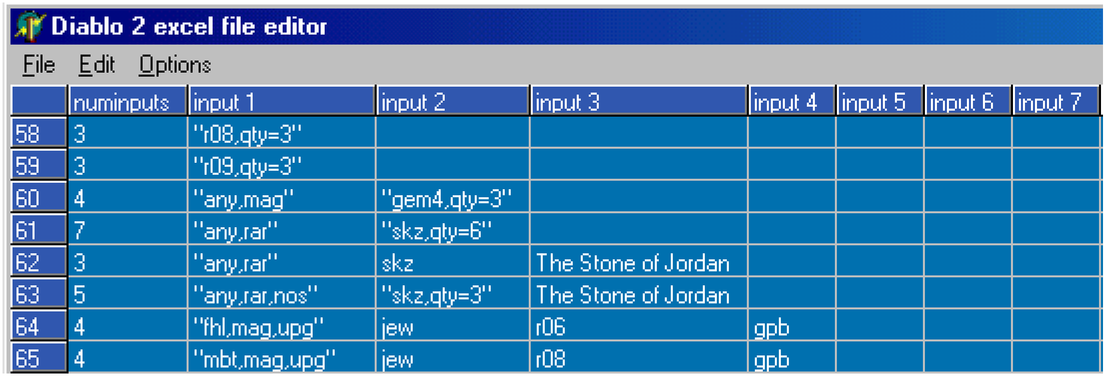
One of the most entertaining facets of the game is the Horadric Cube. Blizzard provided a few useful recipes to get each of us started, but the Cube is capable of far more than that. In this exercise, we will cover the basic structure of a cube recipe, and correct a deficiency in the game. Start by extracting cubemain.txt from the mpq. The structure of cubemain.txt was radically altered with version 1.09, so this exercise requires you be running 1.09 or any of the later sub-releases (1.09b, 1.09d). A good way to learn cube recipe structure is to look at the Blizzard recipes. For example, if you want to have all runes be upgradeable, not just the first nine, then complete the series which is shown in part below.



	description	enabled	min diff	version	start day	stop day	weekday	class
57	3 rune 07 -> rune 08	1		100				
58	3 rune 08 -> rune 09	1		100				
59	3 rune 09 -> rune 10	1		100				
60	3 perfect gems + 1 magic item -> magic item	1		100				
61	6 perfect skulls + 1 rare item -> 1 low level rar	1		0				
62	1 perfect skull + 1 rare item + soj -> 1 high lev	1		0				

Figure 26 - Cubemain.txt Character Qualification section

The first part of the database is the character qualification section. The first field is a descrition of the recipe, it is only for reference and has no game impact. A recipe must be enabled to be usable, this is a Boolean field so set it to 1. Min diff is the minimum difficulty required: 0 or blank for normal, 1 for Nightmare and 2 for Hell. Version is 0 for either Diablo II or expansion games, 100 is expansion games only. The next three fields compare these values to the day of the week and day of the month retrieved from your system clock. Class is the three-letter character class code; if you want to restrict your recipe to Necromancers, put "nec" in this field.



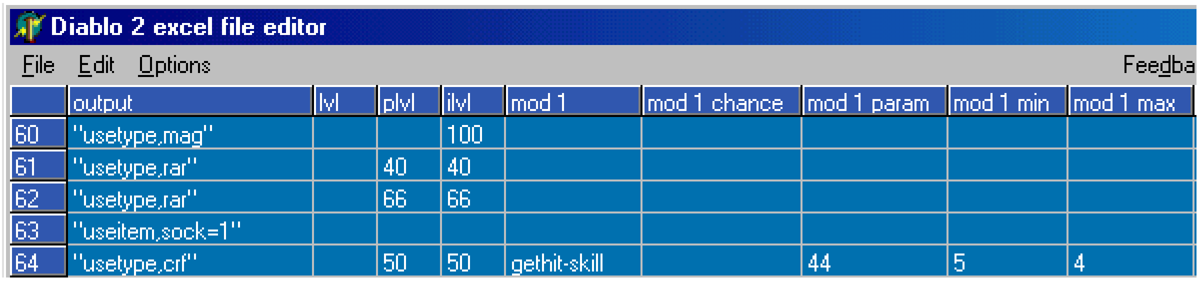
	numinputs	input 1	input 2	input 3	input 4	input 5	input 6	input 7
58	3	"r08,qty=3"						
59	3	"r09,qty=3"						
60	4	"any,mag"	"gem4,qty=3"					
61	7	"any,rar"	"skz,qty=6"					
62	3	"any,rar"	skz	The Stone of Jordan				
63	5	"any,rar,nos"	"skz,qty=3"	The Stone of Jordan				
64	4	"fhl,mag,upg"	jew	r06	gpb			
65	4	"mbt,mag,upg"	jew	r08	gpb			

Figure 27 - Input Section of Cubemain.txt

The second part of Cubemain.txt is the input section. These fields are the ingredients used by the recipe. Some Blizzard recipe ingredients are shown above. The first input is important, any recipe that transforms an object works on whatever object is in this first input field. Also, the number of ingredients must equal the sum of all inputs including the individual quantities. Quotes are used for any input with sub-fields, such as "fhl,mag,upg". Three-letter item codes or four-letter type codes are permitted.

qty	quantity	rar	rare item	crf	crafted item
nor	normal item	set	set item	nos	no sockets allowed
hiq	superior item	uni	unique item	eth	ethereal item required
mag	magical item	upg	upgradeable recipe, also works for exceptional and elite versions	any	any item, but this parameter sometimes doesn't work

Table 4 – Some cube recipe parameters



	output	lvl	plvl	ilvl	mod 1	mod 1 chance	mod 1 param	mod 1 min	mod 1 max
60	"usetype,mag"			100					
61	"usetype,rar"		40	40					
62	"usetype,rar"		66	66					
63	"useitem,sock=1"								
64	"usetype,crf"		50	50	gethit-skill		44	5	4

Figure 28 - Cubemain.txt output section

The third part of Cubemain.txt is the output section. These fields describe the properties of the newly-generated item. The output field is similar to the first input field, and many of the same parameters work here too. Additionally, there are three important parameters that are output parameters only: useitem, usetype, and sock = #. Useitem will take whatever item is listed in the first input field, including all properties already present, and apply whatever properties are assigned in the output section. You can use the useitem parameter to make recursive recipes, where you keep improving an item up to game limits by reapplying the recipe over and over. Usetype will take the base item from the first input field, strip all existing properties away (including ethereality!), "re-roll" the item's characteristics and apply output properties. Finally, sock will apply the designated number of sockets to an item, up to game-defined limits.

The three level fields (lvl, plvl and ilvl) control the level requirements for random properties. If lvl only is used, all random properties will be that level or higher. If lvl and ilvl are defined, the levels of the properties on the item will be between lvl and ilvl. If only plvl is defined, then the maximum level for random properties is plvl% of player's level, and there is no minimum. If plvl and ilvl are defined, then plvl% of the player's level is the minimum, and ilvl is the maximum. The character level required to use the new item will be the level requirement for the highest mod on the item, *plus* 3 for each random mod.

There are five property blocks after that, the first one of which is shown above. These blocks are used the same way as the property blocks in UniqueItems.txt, with one addition. If a value is used in Mod # Chance, that property has the value percent chance of being generated. So if you want a property to appear about a third of the time the recipe is used, put 33 in this field.

Okay, we have thrown a lot of information out about cube recipes, how do we use it? Let us correct a deficiency Blizzard coded into the game. After Act 2 normal difficulty, it is impossible to buy normal items from the vendors, everything is magical. So we will make two recipes, one to convert magical weapons to superior socketed weapons, and one to convert magical armor to superior socketed armor. This is one of those cases where the "any" parameter does not work, so we need a recipe for each. We will allow any magical, non-socketed item plus two Identify Item scrolls create the maximum number of sockets possible in the new item. See the previous figures for locating the right fields to input data.

Make two new rows at the bottom of Cubemain. When adding rows, it is important to remember never to insert a row, always add it on the bottom. Inserting rows may work in a few txt files, but causes severe propblems in most. Type a brief description of the recipe function in the first field for each row. Then skip over to the input section and enter 3 in the NumInputs field. In the first input field of the weapons record, enter "weap,nos,mag" for magical, non-socketed weapon. For the armor record, "armo,nos,mag" is put in the first input field. In both records, the second field should read "isc,qty=2". Now scroll to the output section, and put "usetype,hiq,sock=6" in the output field for both records.

Save the file, make bins, pack to mpq as in previous exercises. Go into a game and find or buy a magical weapon and a magical armor. Get some identify scrolls and test the recipes. Remember you won't get six sockets for many items, and in the lower difficulties no more than three. You will also see the new items can get superior grade properties where those are generated. With multiple recipe pairs, you can control the number of sockets more precisely, and have any item available for runewords!

Conclusion

The staff here at the Phrozen Keep hope you find this guide a useful introduction to the world of Diablo II mod making. There are many changes to the game possible through txt file editing alone, the exercises included here are merely the tip of a big iceberg. Use the other tutorials, file guides and resources to learn more about the files. And remember the Phrozen Keep Forums are always open for business, should you have questions that aren't answered in these sources.

Here is a recap of some important tips for making effective and functional mods.

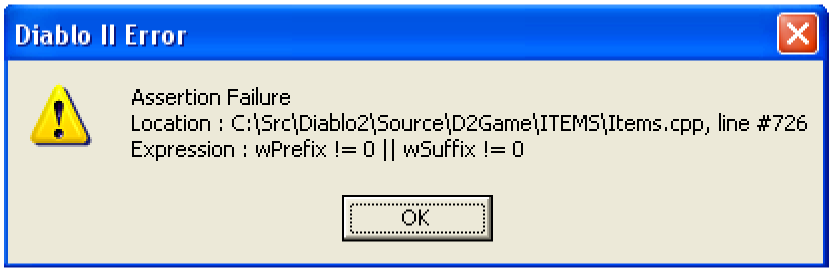
1. Always make incremental changes. The fewer alterations between tests, the faster you will find out what went wrong if something isn't working correctly.

2. If you add new lines to a txt file database or a string table, always add lines at the end of the file. Never insert a line, and never remove a line that you didn't add yourself. If you don't want something to display in a string table, just blank out the field on the right. There are some txt files which cannot be expanded, but of the files mentioned in this guide only LvlMaze is restricted in this way.

3. Use the Quill Rat Test to see if items spawn the way they should. If you are testing high-level items, and reducing the level and levelreq fields for testing purposes is not enough, use the first Quill Rat (H) line in TreasureClassEx.txt and test in Hell difficulty with a (generated) high-level test character.

4. Use existing similar records to see how a new record in any txt file should be organized.

5. If you crash in the game, there are two ways you can get information about what went wrong. The first way is the Assertion popup, as shown below. This message is usually very cryptic, but if you ask about it on the Phrozen Forums, somebody can help you better if you can describe the error using the information in the popup. The messages are easier to retrieve if you run the game windowed. Similar to –direct –txt, run the game with the –w switch for windowed mode.

   ```
   Assertion Failure
   Location : C:\Src\Diablo2\Source\D2Game\ITEMS\Items.cpp, line #726
   Expression : wPrefix != 0 || wSuffix != 0
   ```

   Another location for finding error messages is the debug file produced each time you run the game. This file is located in the default directory, and has a name like D2020508.txt, where the name is D2 followed by two-digit year, two-digit month and two-digit day, it is the day you ran that session. Open this file, and scroll up from the end of the file until you see lines similar to

   ```
   18:38:26.480  ***** UNHANDLED EXCEPTION: ACCESS_VIOLATION (c0000005)
   18:38:26.480  Fault address:  6FD83438 01:00042438 F:\DIABLO II\D2COMMON.DLL
   ```

   If you get an Assertion popup, the information will also appear in the debug file. And this source is the only way to find out more about the Unhandled Exception, which is a generic Windows error usually caused by Diablo II trying to access restricted memory locations.

6. *Above all else, have fun!*