Modding Guide

"It's how you use it!"

Using K1n9_Duk3's Enormous Tool to modify Duke Nukem II

Written by *K1n9_Duk3*
Published by *193 (Reloaded)*

Compiled on August 19, 2013

http://home.arcor.de/k1n9duk3
http://k1n9duk3.ohost.de

Contents

Preface

This guide was written for all those who want to use **K1n9_Duk3's Enormous Tool** (referred to as "the editor") to modify **Duke Nukem II** (referred to as "the game"). It is both a manual for the editor and a documentation of the limitations of the **Duke Nukem II** engine and its file formats.

You will find a chapter for each of the editor's tools, explaining how to use the tool and occasionally giving some hints on how to take full advantage of the game's capabilities.

Special Thanks

I would like to thank the following people for figuring out the structure of some of the **Duke Nukem II** data files:

Dave Bollinger

"Szevvy"

"Malvineous"

This project would not have been possible without your work.

Beta Information

The editor is still considered a beta version. Many tools contain some experimental features and sometimes even allow you to set some settings (like the level width) to values that are not "officially" supported. This was done to allow anyone testing the editor to experiment a little and maybe even discover undocumented features that were not even used in the original game.

If you have some unanswered questions or you find some errors in this document, feel free to contact me:

- via e-mail to k1n9duk3@arcor.de
- via the Public Commander Keen Forum
  (`http://www.pckf.com`, User: K1n9_Duk3)
- via the 3D Realms Forums
  (`http://forums.3drealms.com`, User: King Duke)

Part I.

The Tools

1. The Level Editor

1.1. Level Files

The filenames of the level files are hard-coded in the game's main executable an cannot be changed without modifying the executable. The filenames are:

- `L1.MNI`
- `L2.MNI`
- `L3.MNI`
- `L4.MNI`
- `L5.MNI`
- `L6.MNI`
- `L7.MNI`
- `L8.MNI`
- `M1.MNI`
- `M2.MNI`
- `M3.MNI`
- `M4.MNI`
- `M5.MNI`
- `M6.MNI`
- `M7.MNI`
- `M8.MNI`
- `N1.MNI`
- `N2.MNI`
- `N3.MNI`
- `N4.MNI`
- `N5.MNI`
- `N6.MNI`
- `N7.MNI`
- `N8.MNI`
- `O1.MNI`
- `O2.MNI`
- `O3.MNI`
- `O4.MNI`
- `O5.MNI`
- `O6.MNI`
- `O7.MNI`
- `O8.MNI`

`L1.MNI` through `L8.MNI` are the levels for the first episode. The others are for episodes 2 through 4 and therefore are only found in the registered version of the game.

1.2. Creating A New Level

When you want to create your own level, it is generally easier to start with a new level rather than loading an existing level and erasing everything in it. When you create a new level, you must set up the map settings before you can start editing the actual level. (Editing a level without a tileset would be rather pointless, don't you think?) See section 1.3 below for further information on the map settings.

If you've created a new level, you must enter a file name upon saving the level. You can select any file name you want, but the game will only be able to use the level if you save it as one of the files listed in section 1.1.

1.3. Editing Map Settings

This section of this document has yet to be written.

1.4. Selecting And Drawing Tiles

This section of this document has yet to be written.

1.5. The Planes

The level editor allows you to edit three different planes or layers:

1. solid (non-transparent) tiles
2. masked tiles
3. actors

The text in the bottom-left corner of the screen shows which plane is currently selected. You can select a different plane by:

- left-clicking on the name of the plane in the bottom-left corner of the screen
- pressing 1, 2 or 3 for solid, masked and actors, respectively
- selecting a tile or an actor from the side bar on the left side of the screen

1.6. Level Format & Limitations

1.6.1. Tiles And Tile Attributes

Even though the level editor allows you to edit two separate planes for solid and masked tiles, these planes are actually stored as one single plane in the level files. A solid tile is stored as an offset into the planar image data in memory (multiples of eight bytes), ranging from 0 to 7992. Masked tiles use values from 8000 to 14360 (multiples of 40 bytes). To get the tile attributes, the game appears to divide the cell value by eight and then use the result as an index into the tile attribute array stored in the CZone file.

If both a solid and a masked tile are present at one cell, the most significant bit in the cell value is set (0x8000) and the lower 10 bits store the tile number (possible values are 0 to 1023, but only 0 to 999 are valid tile numbers). This leaves 5 bits for the masked tile number, which would only allow tile numbers from 0 to 31. The level format actually stores two additional bits for each cell value, which allows the level format to store masked tile numbers from 0 to 127 (1000 to 1127 in the editor).

This means that you cannot use the masked tiles 1128 to 1159 in a cell that also stores a solid tile. The editor will detect this as an error. If you save the level anyway, the tile numbers will lose their most significant bit, which means that any masked tile number greater than 1127 will be decreased by 128.

Unfortunately, the game does not decode these combined cell values to get the proper tile attributes. This means that any cell that stores both a solid and a masked tile will have no tile attributes in the game. The only attribute bit that is actually used in the game is the foreground bit of the masked tile. All the other attribute bits, including the foreground bit of the solid tile, will be ignored. When a tile attribute is ignored, the editor displays the attribute information in the satus bar in red.

1.6.2. Masked Foreground Tiles

Be very careful when using masked tiles that have the foreground bit set. The game will *freeze* if there are more than 252 masked foreground tiles visible at the same time! Solid foreground tiles are not affected by this limit.

The game probably uses a fixed-length array to store any masked foreground tiles that are currently visible. Once that array is full, trying to find an empty slot in the array could cause a deadlock.

1.6.3. Limited Number Of Actors

The game will *crash* when you try to load a level that has more than 450 actors! The skill flags (82 and 83) and the block boudaries (103 and 104) do not count as actors.

However, even with less than 450 actors, there might be graphical issues like garbled actor sprites. The actual limit causing this issue is currently unknown.

1.6.4. Limited Amount Of Memory

The game appears to provide a fixed amount of memory to be used by sprites and music. This amount of memory appears to be independent of the amount of free conventional memory.

If the game runs out of memory, it will crash. Depending on the amount of memory used by the current level, the game may crash directly after loading the level, or when the user brings up an in-game message or menu. The editor tries to predict the amount of memory that will be used by the level and generate warnings or errors if certain limits are exceeded, but the results are not always reliable.

To make sure that your level will not run out of memory, you should playtest it on the hardest skill level and press H in the game to bring up the help screens. If that does not force the game to crash, your level should work just fine.

2. The CZone Editor

2.1. CZone Files

CZone files contain the game's tileset graphics and the attribute data for the tiles. Each tile is an 8×8 pixel image composed of four color planes. There are always 1000 'solid' (or 'unmasked') tiles and 160 'masked' tiles in a CZone file. The masked tiles have a fifth color plane that indicates which pixels are transparent.

CZones can have any filename that fits in the DOS naming scheme (see chapter A on page 47). The only safe way to identify a CZone file is by its file size: CZone files always have a size of 42000 bytes.

The original game uses the following CZone files:

- `CZONE1.MNI`
- `CZONE2.MNI`
- `CZONE3.MNI`
- `CZONE4.MNI`
- `CZONE5.MNI`
- `CZONE6.MNI`
- `CZONE7.MNI`
- `CZONE8.MNI`
- `CZONE9.MNI`
- `CZONEA.MNI`
- `CZONEB.MNI`
- `CZONEC.MNI`
- `CZONED.MNI`
- `CZONEE.MNI`

The files `CZONE6.MNI` through `CZONED.MNI` are only found in the registered version of the game.

2.2. Creating A New CZone

To create a new CZone, you need to import an image sized 320×232 pixels. You could use any other size that consitst of at least 1160 8×8 pixel tiles, but it is advisable to use 320×232 pixels, as this is the format the CZone will be presented to you in the level editor and the CZone editor.

The first 1000 tiles in the CZone image must be solid (non-transparent). Only the last 160 tiles in a CZone are masked and therefore can use transparency. Not every one of the masked tiles needs to have transparent pixels, though it is generally advisable not to waste any of the few masked tile slots for a tile image that does not use transparency at all. Drawing masked tiles is slower than drawing solid tiles (see section 5.2), which is why you should not use masked tiles when you could use a solid tile for the same purpose.

2.3. Editing A CZone

Note: The description for bit 10 originally was "fast animation", but that turned out to be wrong. If bit 10 is set, the animation will actually be slower than the normal animation.

This section of this document has yet to be written.

3. The Sprite Editor

3.1. Sprite Files

The sprite data is split across two files. `ACTORS.MNI` contains the actual image data for all sprites. `ACTRINFO.MNI` contains all information on the structure of the sprites, such as width and height and the offsets of the image data in `ACTORS.MNI`.

The game refers to the sprites and their frames by index. The sprites and frames have no name. Fortunately, both the shareware and the registered version use the same sprite file, so the indices are the same for both versions.

3.2. Editing The Sprites

When you start sprite editor, you must first load the sprites, using the "Load Sprites" option. If the editor doesn't already know the location of your **Duke Nukem II** directory, it will ask you to select it. If the sprites were read successfully, the editor will automatically go into edit mode.

The edit mode consists of three menu levels: the sprite list, the frame list (including sprite settings), and the frame settings (including the Import, Export and Edit Image options). You must select a sprite in the first menu level to access the second level, and you must select a frame in the second level to get to the third level.

3.2.1. Changing The Number Of Sprites And Frames

You can change the number of sprites by selecting the "Sprites:" entry in the first level and pressing Enter.

Be very careful when changing the number of sprites and frames! If you reduce their numbers, the game might crash or do other weird things, because the sprite and frame indices for many things are hard-coded in the game. If you increase the number of sprites or frames, the game might run out of memory and crash (which

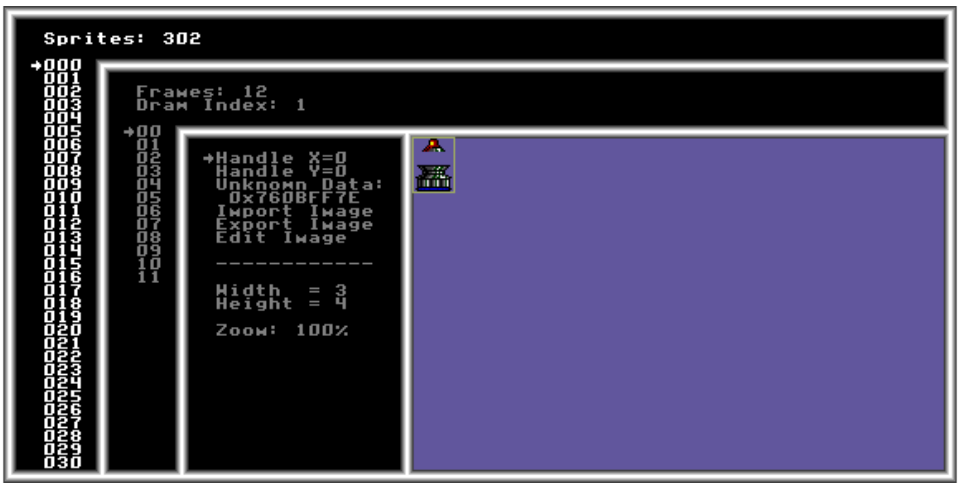

Figure 3.1.: The Sprite Editor's Edit Mode

is bad), or you might not be able to save the sprite file anymore (which might be worse). Make sure that the following is always true:

$$3 \times SpriteCount + 8 \times FrameCount < 65,536$$

(FrameCount is the total number of frames in the entire file.)

The number of sprites must not exceed 1000 and the number of frames for each sprite must not exceed 100. There is no way to access sprites beyond index 999 or frames beyond index 99 in the game, and therefore the editor will not allow you to go beyond these limits.

3.2.2. Sprite Properties

Each sprite has two properties: the number of frames and a draw index.

The draw index defines the order in which the actors are updated an drawn onto the screen during the actual game (i. e. in a level). The file format can store any signed 16-bit value (-32,768 to 32,767), but only the values -1 to 3 are supported by the game. The actors with index -1 are updated and drawn first, therefore actors with a higher draw index will appear in front of actors with a lower index.

Note: Actors with an unsupported draw index will not appear in the levels.

3.2.3. Frame Properties

Each frame has the following properties:

Width and Height (2 bytes each)

These properties define the width and height of the frame's image (in tile units). Each tile has a size of 8×8 pixels.

The width and height cannot be changed manually. They are updated automatically when importing an image for the current frame.

HandleX and HandleY (2 bytes each)

These properties define a handle for drawing the image. The default handle (0, 0) is the bottom left tile of the image. The image will be shifted HandleX tiles to the right and HandleY tiles down when the image is drawn. Both values can be negative if the image should be shifted to the left or up.

Unknown Data (4 bytes)

The purpose of these bytes is still unknown. These might be a checksum or simply bytes that were reserved for future enhancements of the sprite format. They are displayed and can be edited as a 32-bit hexadecimal value.

These bytes do not seem to be used by the game. Just leave them as they are.

3.3. Changing Images

Images can either be edited with QuickPaint, which is included in the editor itself, or exported to be edited with other programs.

3.3.1. Editing Images With QuickPaint

QuickPaint is a very primitive paint tool and works very much like the level editor. You can select a color by clicking the right mouse button and you can draw a pixel using the left mouse button. You can also use flood-fill by holding down the shift key and then pressing the left mouse button. However, you cannot change the size of the image.

3.3.2. Importing And Exporting Images

You can import and export individual frames by using the Import Image and Export Image options in the third level of edit mode (i. e. the context menu of a frame). The images can be exported to and imported from indexed images in BMP, GIF and PNG format.

You can also export and import all sprites using the respective function in the sprite editor's main menu. This will export all images to or import them from the selected folder. The files are exported as PNG images, using filenames of the form `S0_F0.PNG`.

3.3.3. Palettes

The game defines which palette will be used to display the sprite's images. Depending on the context, the same sprite image might be displayed using different palettes and therefore look differently. The editor tries to map each sprite to the correct palette as used by the game.

When exporting, the editor will write the same palette it uses to display the image to the exported file. It will include a palette of usually 32 colors, where the first 16 colors are not transparent and the later half of the colors are fully transparent. The sprites should only use the non-transparent colors and the first transparent color. Using the higher colors of the palette has some interesting, but mostly undesired, side effects (see section 5.2).

When a sprite is imported, the editor will ignore the actual colors in the palette. It simply reads the palette indices for each pixel and converts the data back to the 5-plane tile data used by the game. Therefore, you must make sure that whichever program you use to edit the exported images does not change the order of the colors in the palette. For example, some PNG compressors will change the order of the palette entries so that the transparent color entries appear first, because this results in a smaller file. If the order is chaged, the imported image might use the wrong colors in the game.

Note: Images exported as PNG images will have the upper half of the palette marked as fully transparent colors. That means you cannot edit them with Microsoft Paint, because Paint will save PNG images that include transparency as 32-bit images. The palette will be lost and the editor will not be able to import the modified file.

3.4. Image Format & Limitations

3.4.1. Sprite 29

Most of the sprites in **Duke Nukem II** store their image data as 5 bitplanes (4 planes of color information, plus a mask plane). The only exception is sprite number 29, which stores the bigger font used in the game's main menu. All frames of sprite 29 use only 2 bitplanes (a color plane and a mask plane). In addition, the width and height of the frames for sprite 29 are ignored by the game. The game assumes that all frames are one tile wide and two tiles high.

3.4.2. Image Sizes And Hitbox Sizes

The larger sprite images are, the more memory they take up. Increasing the size of a single sprite image could already cause the game to crash while loading a level which requires this sprite.

Another limitation of the sprite format is that the width and height of the images appear to also affect the size of the actor's hitboxes. For example, if you increase the height of the Duke sprites (sprites 5 and 6), players will not be able to walk through some of the smaller openings in the levels anymore.

In any way, you should not change the size of a sprite image unless you absolutely have to.

4. The VOC Converter

4.1. VOC Files

The digitized sounds (commonly referred to as wave sounds) used by **Duke Nukem II** are stored in Creative Voice Format (see subsection 4.1.1). The following files are VOC files:

- `INTRO3.MNI`
- `INTRO4.MNI`
- `INTRO5.MNI`
- `INTRO6.MNI`
- `INTRO7.MNI`
- `INTRO8.MNI`
- `INTRO9.MNI`
- `SB_1.MNI`
- `SB_2.MNI`
- `SB_3.MNI`
- `SB_4.MNI`
- `SB_5.MNI`
- `SB_7.MNI`
- `SB_8.MNI`
- `SB_9.MNI`
- `SB_10.MNI`
- `SB_13.MNI`
- `SB_17.MNI`
- `SB_19.MNI`
- `SB_20.MNI`
- `SB_21.MNI`
- `SB_22.MNI`
- `SB_25.MNI`
- `SB_28.MNI`
- `SB_30.MNI`

As the name suggests, the files `INTRO3.MNI` to `INTRO9.MNI` are part of the intro animation. The `SB_*.MNI` files are the SoundBlaster alternatives for the AdLib and PC Speaker sound effects. All file names are hard-coded. If one of these files is missing, the game will crash and it might even cause your system to freeze!

4.1.1. Creative Voice Format

Creative Voice Files were designed to be processed and played directly by a Creative SoundBlaster card. This means that all decoding of the data contained in the files is done in hardware. There is no reference software decoder, which makes is very difficult to decode some VOC files properly. The DOSBox emulator is able to emulate the SoundBlaster card, but there is no guarantee that the DOSBox emulator decodes the data correctly.

Part II.

Advanced Modding Techniques

5. Graphics

5.1. Color Palettes

This section of this document has yet to be written.

5.2. Masked Images

Masked images usually consist of five planes:

1. Mask plane
2. Blue plane
3. Green plane
4. Red plane
5. Intensity plane

The video mode used by the game provides four separate color planes for blue, green, red and intensity (which means the game is basically an EGA game hiding behind a VGA palette). Unmasked images have no mask plane, so they can be drawn by simply copying the byte values for each plane from the image data into the video buffer.

To draw a masked image, the game applies the value from the mask plane to the data stored in each plane of the video buffer using a bitwise AND function. This means that the pixels, for which the bits in the mask plane are set, will be kept as they are, while the other pixels are set to color 0 (black). The actual pixel values of the masked image are then applied to the corresponding planes using a bitwise OR function.

If the masked image's plane data has some pixels that are not set to color 0 and the mask bits for these pixels are set, the data will be applied to the non-zero value in the video buffer and therefore modify the color value of the pixel instead

of replacing it. This technique can be used to create some primitive lighting effects.

This section of this document has yet to be written.

6. Scripts

6.1. Script Files

The game uses a pretty sophisticated scripting language to define what the menues, help texts, story, ordering information etc. look like.

The scripts are stored as (mostly) plain ASCII text in the following files:

- `TEXT.MNI`
- `HELP.MNI`
- `OPTIONS.MNI`
- `ORDERTXT.MNI`

The scripts can be edited with any text editor such as Notepad. However, there might be some issues because the `//SETKEYS` command (see section 6.3.23) may require the use of control characters (ASCII codes 0 to 31). The version of Notepad bundled with Windows XP seems to be able to correctly read and write all the control characters that are actually used in the game's original script files. Other text editors might not be able to handle the control characters correctly, so be careful!

You should *definitely not* edit the script files with Microsoft Word or similar programs! Those programs might not write plain ASCII text when you save the file, since they usually store additional typesetting information such as font, size and color in the saved file. This will make the game unable to read the script files properly.

6.2. Script Sections

Each script file contains one or more script sections. Each script section starts with the name of the section and ends with the `//END` command. An empty script section looks like this:

```
My_Script
//END
```

When the game tries to execute a script, it will open the script file and search for the name of the desired section. It will then interpret any command until it encounters the `//END` command. If it cannot find a command in a line of text, the line will be treated as a comment.

The name of each section as well as all commands usually start directly at the beginning of a line. You can add spaces in front of a section name or a command, but not tabs. Tabs are treated as normal characters and as such will become part of the name or command string, thus preventing the game from recognizing them.

It is possible to have more than one command in one line. Using more than one command per line, however, may have undesired side effects such as commands being drawn onto the screen as part of a text.

These are the script sections used by the game:

6.2.1. &Calibrate

Found in: `OPTIONS.MNI`

This defines the background for calibrating the joystick. It simply draws an empty window.

6.2.2. &Credits

Found in: `TEXT.MNI`

This defines what will be displayed when the user selects “Credits” in the main menu.

6.2.3. &Instructions

Found in: `TEXT.MNI`

This defines what will be displayed when the user selects “Instructions” in the “Instructions And Story” menu.

6.2.4. &Load

Found in: TEXT.MNI

This defines the message that will be shown while loading a game. This message is drawn on top of the "Restore Game" menu.

6.2.5. &Save

Found in: TEXT.MNI

This defines the message that will be shown while saving a game. This message is drawn on top of the "Save Game" menu.

6.2.6. &Story

Found in: TEXT.MNI

This is the script for the story cutscene that will be displayed during the inro or when the user selects "Story" in the "Instructions And Story" menu.

6.2.7. 2Quit_Select

Found in: TEXT.MNI

This defines the "Are you sure you want to quit?" windows for the actual game. The first menu item is "Yes", everything else means "No".

6.2.8. BAD_GAME

Found in: TEXT.MNI

This defines what will be displayed when the "copy protection" of the registered version fails.

6.2.9. Beta_Only

Found in: TEXT.MNI

Leftovers from the beta version. Not used by the final game.

6.2.10. Both_S_I

Found in: TEXT.MNI

This defines all menu items for the "Instructions And Story" menu.

6.2.11. Episode_Select

Found in: TEXT.MNI

This defines all menu items for the "Select An Episode" menu. (For starting a new game or viewing the high scores.)

6.2.12. Full_Health

Found in: TEXT.MNI

This defines the message for the EAT cheat. Registered version only.

6.2.13. Game_Speed

Found in: TEXT.MNI

This defines all menu items for the "Game Speed" menu.

6.2.14. God_Mode_Off

Found in: HELP.MNI

This defines the message for turning god mode off.

Note: I have no idea how to access god mode in the game, so don't bother asking me about it.

6.2.15. God_Mode_On

Found in: `HELP.MNI`

This defines the message for turning god mode on.

Note: I have no idea how to access god mode in the game, so don't bother asking me about it.

6.2.16. Hints

Found in: `HELP.MNI`

This defines the hint messages for each level. It consists entirely of `//HELPTEXT` commands (see section 6.3.14 below).

6.2.17. HYPE

Found in: `TEXT.MNI`

This is the script for the hype cutscene that will be displayed when you start the game for the first time (i. e. the config file `NUKEM2.-GT` does not exist).

6.2.18. Key_Config

Found in: `OPTIONS.MNI`

This defines all menu items for the "Game Controls" menu.

6.2.19. Main_Menu

Found in: `TEXT.MNI`

This defines all menu items for the main menu.

6.2.20. Music_Off

Found in: `TEXT.MNI`

This defines the message for turning the music off in-game.

6.2.21. Music_On

Found in: TEXT.MNI

This defines the message for turning the music on in-game.

6.2.22. My_Options

Found in: OPTIONS.MNI

This defines all menu items for the "Game Options" menu.

6.2.23. New_Highscore

Found in: TEXT.MNI

This defines the background for entering a name into the highscore list.

6.2.24. No_Can_Order

Found in: TEXT.MNI

This defines the message that will be displayed when the user selects Episode 2, 3 or 4 in the "Select An Episode" menu. Shareware version only.

6.2.25. No_Game_Restore

Found in: OPTIONS.MNI

This defines the message that will be displayed when the user selects an empty slot in the "Restore Game" menu. It is drawn on top of the menu.

6.2.26. Not_In_Game

Found in: OPTIONS.MNI

Not used by the final game.

6.2.27. Now_Ch

Found in: `TEXT.MNI`

This defines the message for the NUK cheat. Registered version only.

Note: Unlike the EAT and GOD cheats, the game will FadeOut after executing the script and FadeIn back to the game screen, switching back to the game's palette.

6.2.28. Ordering_Info

Found in: `ORDERTXT.MNI`

This defines what will be displayed when the user selects "Ordering Information" in the main menu. Shareware version only. The registered version uses `V4ORDER` instead (see section 6.2.39 below).

6.2.29. Password

Found in: `TEXT.MNI`

Leftovers from the beta version. Not used by the final game.

6.2.30. Paused

Found in: `TEXT.MNI`

This defines the message for pausing the game. This must have a `//WAIT` command or the game will not stay paused!

6.2.31. Q_Order

Found in: `TEXT.MNI`

This basically shows the first screen of the "Ordering Information". Shareware version only.

Note: I have not yet been able to figure out when this script is actually executed.

6.2.32. Quit_Select

Found in: TEXT.MNI

This defines the "Are you sure you want to quit?" windows for the main menu. The first menu item is "Yes", everything else means "No".

6.2.33. Restore_Game

Found in: OPTIONS.MNI

This defines all menu items for the "Restore Game" menu.

6.2.34. Save_Game

Found in: OPTIONS.MNI

This defines all menu items for the "Save Game" menu.

6.2.35. Skill_Select

Found in: TEXT.MNI

This defines all menu items for the "Select Skill" menu.

6.2.36. Sound_Off

Found in: TEXT.MNI

This defines the message for turning the sound off in-game.

6.2.37. Sound_On

Found in: TEXT.MNI

This defines the message for turning the sound on in-game.

6.2.38. The_Prey

Found in: `TEXT.MNI`

This defines the message for the GOD "cheat".

6.2.39. V4ORDER

Found in: `TEXT.MNI`

This defines what will be displayed when the user selects "Ordering Information" in the main menu. Registered version only. The shareware version uses `Ordering_Info` instead (see section 6.2.28 above).

6.2.40. Volume1

Found in: `TEXT.MNI`

This defines the background of the highscore list for Episode 1.

Note: The game draws the names *after* the script is executed, and then calls FadeIn.

6.2.41. Volume2

Found in: `TEXT.MNI`

This defines the background of the highscore list for Episode 2.

Note: The game draws the names *after* the script is executed, and then calls FadeIn.

6.2.42. Volume3

Found in: `TEXT.MNI`

This defines the background of the highscore list for Episode 3.

Note: The game draws the names *after* the script is executed, and then calls FadeIn.

6.2.43. Volume4

Found in: `TEXT.MNI`

This defines the background of the highscore list for Episode 4.

Note: The game draws the names *after* the script is executed, and then calls FadeIn.

6.2.44. Warp

Found in: `HELP.MNI`

This defines the menu items for the level warp cheat.

Note: I have no idea how to access this cheat in the game, so don't bother asking me about it.

6.2.45. Weapon_Select

Found in: `HELP.MNI`

This defines the menu items for the weapon select cheat.

Note: I have no idea how to access this cheat in the game, so don't bother asking me about it.

6.3. Script Commands

These are the script commands used by the game:

6.3.1. APage

Usage: `//APAGE`

The `//APAGE` command marks the beginning of a new page. See section 6.3.20 for further details.

6.3.2. BabbleOff

Usage:
`//BABBLEOFF`

The `//BABBLEOFF` command turns the "babble" animation off and draws the first frame of sprite 297 (closed mouth). This is required in case the user skips a `//DELAY` command by pressing a key or the "babble" animation takes longer than the scripted delay.

This command is only used in the section `&Story` in `TEXT.MNI`.

6.3.3. BabbleOn

Usage:
`//BABBLEON <t>`

The `//BABBLEON` command turns the "babble" animation on. The parameter `<t>` defines how long the animation is being displayed (total number of babble frames, 11 frames per second).

The "babble" animation uses sprite 297 to animate the mouth of the TV anchorman. Each babble frame draws a random image of this sprite. The animation stays active until all frames have been played or a `//BABBLEOFF` command is being processed.

This command is only used in the section `&Story` in `TEXT.MNI`.

6.3.4. CenterWindow

Usage:
`//CENTERWINDOW <y> <h> <w>`

The `//CENTERWINDOW` command creates a window that is `<w>` tiles wide and `<h>` tiles high. The upper border of the window will be `<y>` tiles below the top of the screen. The window will be horizontally centered on the screen, unless it was shifted using the `//SHIFTWIN` (see section 6.3.24). The game will animate the window (i.e. the window gets larger until it reaches the desired size).

If the window should be dawn on top of the in-game screen, you must use the `//SETCURRENTPAGE` command first. That's because the game uses multiple screen buffers while the menus only use a single buffer.

Note that the width and height of the window include its borders! The window must be 3 tiles high to hold one line of text. The `//CWTEXT` and `//SKLINE` commands are used for drawing text inside a centered window.

6.3.5. CWText

Usage:
`//CWTEXT <text>`

The `//CWTEXT` command draws `<text>` centered in a centered window. The text may contain special characters (see section 6.3.29 for further details), but it is best not to use them in a centered window.

6.3.6. Delay

Usage:
`//DELAY <t>`

The `//DELAY` command causes the script to wait until `<t>` *ticks* have passed or a key has been pressed. 1 second should be about 140 ticks.

6.3.7. End

Usage:
`//END`

The `//END` command marks the end of a script section. Any commands following the `//END` command will not be interpreted.

6.3.8. ETE

Usage:
`//ETE`

The `//ETE` command is only used in `ORDERTXT.MNI`, which is only used by the shareware version. Since the command cannot be found as a string in the main executable of either version, it is probably not used by the game at all.

This might have been intended to signal the interpreter to advance to the next page when the user presses Enter instead of stopping and returning the current page number to the game. However, the final implementation in the game appears to be hard-coded to ignore the Enter key while interpreting certain script sections. Known sections that ignore the Enter key are `Ordering_Info` and `&Instructions`.

6.3.9. ExitToDemo

Usage:
`//EXITTODEMO`

The `//EXITTODEMO` command causes the game to go into demo mode after 30 seconds without any input from the user.

This command is only used in the section `Main_Menu` in `TEXT.MNI`.

6.3.10. FadeIn

Usage:
`//FADEIN`

The `//FADEIN` command causes the game to fade the entire screen from black to the normal state. It does so by manipulating the palette until all colors are back at their original value. This command is only ever used after a raw screen or a palette was loaded (see sections 6.3.16 and 6.3.13).

6.3.11. FadeOut

Usage:
`//FADEOUT`

The `//FADEOUT` command causes the game to fade the entire screen to black. It does so by manipulating the palette until all colors are black. This command is often used before a raw screen is being loaded (see section 6.3.16).

6.3.12. GetNames

Usage:
`//GETNAMES <n>`

The `//GETNAMES` command draws the names of all savegames. The names will be drawn at x coordinate 14 (112 pixels) and y coordinates 6 (42 pixels) to 20 (152 pixels), using the big font and color 2. Only the name whose index matches `<n>` is drawn using color 3.

This command is only used in `Save_Game` and `Restore_Game` in `OPTIONS.MNI`.

6.3.13. GetPal

Usage:
`//GETPAL <file>`

The `//GETPAL` command loads a 16-color palette from `<file>`. The file must be a palette file (the first 48 bytes of the file are loaded as the palette), not an image file. This will also set all pixels in the screen buffer to color 0.

The palette will not be applied directly, so you may have to use the `//FADEIN` command if you faded to black before loading the palette.

6.3.14. HelpText

Usage:
`//HELPTEXT <e> <l> <text>`

The `//HELPTEXT` command defines the `<text>` that will be displayed when the player returns the hint globe to its pedestal on Episode `<e>` Level `<l>`. Only the ASCII characters from A to Z (upper and lower case), the space, comma, period, exclamation mark and question mark are printable. The asterisk character (`*`) is used as a 'stop' marker in the text. Each segment of the text should not be more than 37 characters long.

This command is only used in the section `Hints` in `HELP.MNI`.

6.3.15. Keys

Usage:
`//KEYS`

The `//KEYS` command draws the names of the keys that are currently used to control Duke. The names will be drawn at x coordinate 26 (208 pixels) and y coordinates 7 (56 pixels) to 17 (136 pixels). The order of the keys is Fire, Jump, Up, Down, Left, Right.

This command is only used in the section `Key_Config` in `OPTIONS.MNI`.

6.3.16. LoadRaw

Usage:
`//LOADRAW <file>`

The `//LOADRAW` command loads a 16-color image and its palette from `<file>`. The file must be 32048 bytes in size.

The palette will not be applied directly, so you may have to use the `//FADEIN` command if you faded to black before loading the image.

6.3.17. Menu

Usage:
`//MENU <n>`

The `//MENU` command defines a unique number for every menu (1 for the main menu, 2 for the episode select menu etc.). The game uses these numbers to store and restore the last cursor position for each menu.

6.3.18. NoSounds

Usage:
`//NOSOUNDS`

The `//NOSOUNDS` command disables the sound that is usually played when the user navigates to the next or the previous page.

6.3.19. PagesEnd

Usage:
`//PAGESEND`

The `//PAGESEND` command marks the end of an environment consisting of multiple pages.

6.3.20. PagesStart

Usage:
`//PAGESSTART`

The `//PAGESSTART` command starts an environment that consists of multiple pages.

The first page begins after the `//PAGESSTART` command, all following pages must start with the `//APAGE` command. The last page must end with the `//PAGESEND` command before the end of the script section (see section 6.3.7).

If the user should be able to navigate through these pages with the arrow keys, you must put a `//WAIT` command at the end of every page (before the `//APAGE` or the `//PAGESEND` command). Otherwise the game will simply cycle through all pages once and exit the script.

6.3.21. PAK (Press Any Key)

Usage:
`//PAK`

The `//PAK` (Press Any Key) command is used to draw the first frame of sprite 146. The image contains the text "Press any key to continue" and is usually drawn near the bottom-right corner of the screen.

6.3.22. SetCurrentPage

Usage:
`//SETCURRENTPAGE`

The `//SETCURRENTPAGE` command is often used before a centered window (see section 6.3.4) is created. It is required for all message windows that are drawn on top of the in-game screen, like `2Quit_Select` and `Paused`. If this command is missing, the window and its text may not show up on the screen at all.

6.3.23. SetKeys

Usage:
`//SETKEYS <keys>`

The `//SETKEYS` command defines a key for each menu item, so that the menu items can be accessed by pressing a single key instead of navigating to the menu item and pressing Enter. It is also used to define the Y and N keys for the "Are you sure you want to quit?" windows. `<keys>` is a sequence of ASCII characters, representing the low-level scan code of each key (see figure 6.1).

The scan codes 1 to 31 must be handled very carefully, because the ASCII characters 0 to 31 are control characters. The control characters 9 (Horizontal Tab = HT), 10 (Line Feed = LF) and 13 (Carriage Return = CR) have a special meaning in text files. The other control characters are usually considered to be non-textual and therefore might not be processed properly by some text editors. The sequence CR LF is used in MS-DOS text files to indicate a line break and cannot be used in a script file, because the sequence will be interpreted by the game as the end of the text line.

One way to prevent issues with text editors is to edit the script file with a hex editor before you start editing it with a text editor. Search for `//SETKEYS` commands and replace the scan codes with the hexadecimal value 30 (ASCII code for the digit 0). The resulting text in the script would be `//SETKEYS 00000000`. Then you should be able to open the script with any text editor, edit it, and save it correctly. After you saved your modified script, you can use the hex editor again to change the parameter of all `//SETKEYS` commands to the desired scan codes.

The other way is to use the customized `TEXT.MNI` file that comes with this document as a base for your modifications. In the customized script file, all `//SETKEYS` commands have been removed and the user can select "Yes" or "No" in the "Are you sure you want to quit?" windows using the arrow keys.

The `//SETKEYS` command is only used in the file `TEXT.MNI`.

6.3.24. ShiftWin

Usage:
`//SHIFTWIN`

The `//SHIFTWIN` command shifts the next centered window (see section 6.3.4) and all following text three tile units to the left. In some cases, this command is followed by a parameter in the script files (usually `-3`). That parameter is ignored.

This command is used to draw the centered windows centered in the in-game screen area, as shown in figure 6.2.

6.3.25. SkLine

Usage:
`//SKLINE`

The `//SKLINE` command is used to skip a line in a centered window (see section 6.3.4). Most message windows skip the first line of their centered window.

6.3.26. Toggs

Usage:
`//TOGGS <x> <n> <y1> <t1> ... <yn> <tn>`

The `//TOGGS` command turns the first `<n>` entries of a menu into switches for the sound and music settings. The parameter `<x>` defines the x coordinate (in tile units) of the switch images. As implied before, `<n>` defines the number of switches to follow. Each switch is then given an y coordinate followed by a single character that defines the type of the switch. Supported types are `P` (PC Speaker), `S` (SoundBlaster), `L` (AdLib) and `M` (Music).

One noticeable side effect of the `//TOGGS` command is that it overrides the effects of the first n menu items. For example, if you put the `//TOGGS` command in the script for the main menu and define at least one switch, you cannot start a new game by selecting the item "Start A New Game" and pressing Enter. However, because of the `//SETKEYS` command, you will still be able to start a new game by pressing S.

This command is used in the section `My_Options` in `OPTIONS.MNI`.

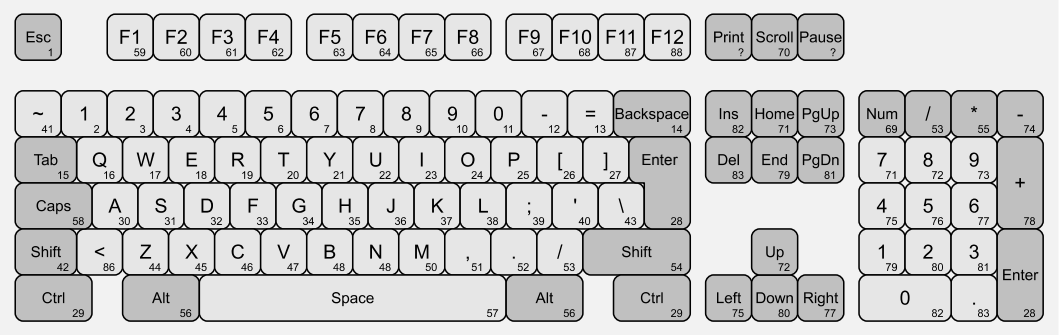

Figure 6.1.: Keyboard Scan Codes

Figure 6.2.: Shifted And Unshifted Centered Windows

Note: The `//TOGGS` command is also used in the `Warp` section in `HELP.MNI` to select an episode ("volume") number. The switch types for the episodes are the numbers `1` to `4`.

6.3.27. Wait

Usage:
`//WAIT`

The `//WAIT` command waits until the user presses a key.

6.3.28. WaitCursorEnd

The `//WAITCURSOREND` command can be found as a string in the game's executable, but it is not used in any of the script files. Syntax and semantics of this command are unknown.

6.3.29. XYText

Usage:
`//XYTEXT <x> <y> <text>`

The `//XYTEXT` command draws text or sprites at the given position (`<x>` and `<y>` are in tile units). Using the default font, only the ASCII characters 32 (space) to 90 ('Z'), the underscore ('_') and the lower case letters 'a' to 'z' are printable. The `<text>` parameter may contain the special ASCII characters listed in table 6.1. All unprintable characters, including the special characters themselves, will be drawn as a space.

If the big font is used, only the ASCII characters from A to Z (upper and lower case), the digits from 0 to 9, the space, comma, period, exclamation mark and question mark are printable. Unprintable characters will be displayed as random garbage. Keep in mind that each glyph of the big font is two tiles high. The lower tile of the glyph will be drawn at the given y coordinate.

You can type in any ASCII character by holding down the Alt key and then typing in the desired ASCII code using the "Num block". If you're using Windows, you must prefix the code with a zero. For example, you hold down Alt, then type 0 2 3 9 and release the Alt key to type the "draw sprite" special character into your script file.

Note: Sprites will not necessariy be drawn exactly at the given location. It's x coordinate will be `<x>`+1+`HandleX` and the y coordinate of the *bottom* of the sprite image will be `<y>`+1+`HandleY`. The handle values are defined for each frame in the sprite files (see chapter 3). The sprite image might be shifted further to the left if there is more than one space between the `<y>` parameter and the "draw sprite" special character.

Note: The special characters are sometimes displayed as random garbage when using the big font, so you should not use another special character after switching to the big font. It is probably better to split your text into multiple `//XYTEXT` commands if you want to use multiple colors in one line of text.

ASCII	Effect
239	Draws a sprite. The character must be followed directly by *exactly five decimal digits*. The first 3 digits are the sprite number, the last 2 digits are the sprite's frame number.
240	Switches to big font with color 0
241	Switches to big font with color 1
242	Switches to big font with color 2
243	Switches to big font with color 3
244	Switches to big font with color 4
245	Switches to big font with color 5
246	Switches to big font with color 6
247	Switches to big font with color 7
248	Switches to big font with color 8
249	Switches to big font with color 9
250	Switches to big font with color 10
251	Switches to big font with color 11
252	Switches to big font with color 12
253	Switches to big font with color 13
254	Switches to big font with color 14
255	Switches to big font with color 15

Table 6.1.: Special ASCII Characters

6.3.30. Z

Usage:
`//Z <y>`

The `//Z` command defines the y coordinate (in tile units) of the menu cursor. This is almost always the y coordinate of the selected menu item's text. The x coordinate of the cursor is always 8 tile units (64 pixels).

7. Breaking Shareware Compatibility

7.1. Why?

In November 1995, Apogee released TED, the editor that was used to create the levels for **Rise of the Triad** and a ton of other games. One of the files relased along with the editor contains the following message:

> We do respectfully request that you do not modify the levels for the shareware version of Rise of the Triad. The authors worked hard on the game and if there are lots of free levels available for the shareware version, a user will have far less incentive to order the full game. So please respect our wishes, and only create levels for the registered version. (And we took so much trouble to put those handy Alternate Level selections in the Registered Setup, too!)

I believe that Apogee would have requested the same for **Duke Nukem II** had there been any editors available for it back then.

7.2. How?

There are many ways to make sure that your Levelpack/Mod/Total Conversion will only work with the registered version of **Duke Nukem II**. The easiest and most boring way is to only modify the levels for Episode 2, 3 and 4.

If you want to modify all four episodes, you should force the game to crash when a user attempts to play it using the shareware version of **Duke Nukem II**. This can be done by using files that are only included in the registered version as the Backdrop, CZone or Music files of the first level of episode 1. See section 7.3 for a list of suitable files.

This obviously limits the creative freedom for Episode 1 Level 1, especially if you want to create a Total Conversion that replaces every image in the game. One quick way to avoid having noticeable materials from the original registered version is to use

the alternate backdrop. The game will load the alternate backdrop even if none of the backdrop-switching flags are set, so there is no way to ever switch to the alternate backdrop. If it cannot load the alternate backdrop, it will crash.

Now, if you want *complete* creative freedom (i. e. you want to use backdrop-switching in Episode 1 Level 1) you can modify the game's script files. Simply load one of the registered-only fullscreen images somewhere in the script before the actual background image is loaded – preferably in the `Main_Menu` section of `TEXT.MNI`. The best image to use for this method is `END4-2.MNI` because that image is never used by the game, so there is no need to ever replace it with a custom image.

The last method also prevents the user from loading a saved game to skip the unplayable first level.

7.3. Registered-Only Files

7.3.1. Backdrops

- `DROP3.MNI`
- `DROP4.MNI`
- `DROP8.MNI`
- `DROP15.MNI`
- `DROP16.MNI`
- `DROP19.MNI`
- `DROP23.MNI`

7.3.2. CZones

- `CZONE6.MNI`
- `CZONE7.MNI`
- `CZONE8.MNI`
- `CZONE9.MNI`
- `CZONEA.MNI`
- `CZONEB.MNI`
- `CZONEC.MNI`
- `CZONED.MNI`

7.3.3. Music

- `DEPTHSA.IMF`
- `KISGIRLA.IMF`
- `NUKEMANA.IMF`
- `WINNINGA.IMF`

7.3.4. Fullscreen Images

- `END2-1.MNI`
- `END3-1.MNI`
- `END4-1.MNI`
- `END4-2.MNI`
- `END4-3.MNI`
- `LOAD2.MNI`
- `LOAD3.MNI`
- `LOAD4.MNI`

Appendix

A. MS-DOS Filenames

Most of the information in the following text was taken from a german MS-DOS 5.0 manual. It may not be translated correctly, but it should tell you everything you need to know.

Every file has a *Name* and most files also have an *Extension*. The name is always displayed first and the extension is always separated from the name by a period, like in the following example:

`name.ext`

If the file has no extension, then the period is optinal: `name` and `name.` both refer to the same file.

The term *Filename* includes both the name and the extension of a file.

A.1. Names

Rules for the name of a file are:

- Names must contain at least one character (names must not be empty).
- Names must not be longer than eight characters.
- Names may only use the letters `A` to `Z`, the digits `0` to `9` and the following characters: `_ ^ $ ~ ! # % & - { } ( ) @ ’ ‘`
- Names must not include spaces, commas, slashes or periods (except for the period that separates name and extension).
- The following names are reserved and must not be used as a name of a file: `CLOCK$`, `CON`, `AUX`, `COM1` to `COM4`, `LPT1` to `LPT3`, `NUL` and `PRN`.

Note: You may use extended characters (ASCII codes 128 to 255) in a name. In this case you should use code page 850, since code page 437 provides only limited support for extended characters.

A.2. Extensions

The rules listed above also apply to the extension of a file. The only difference is that an extension may be empty and must not consist of more than three characters.

Note: Most modern versions of Windows do not show the extension of most files by default.

B. FAQ

Q *How did you come up with this stupid name for the editor?*

A I once read that 3D Realms named their version of UnrealEd "Duke's Enormous Tool". That's how.

Q *Where can I get the source code for the editor?*

A Nowhere! At least not right now. The code is a huge mess and I'm not gonna release it until it's cleaned up properly.

Q *Where can I get the source code for the game?*

A Nowhere! (I think there might be a pattern here.) Apogee/3D Realms never released the source code for Duke Nukem II. It was said that the source code was lost over the years.

Q *The editor crashed! What do I do now?*

A First, have a look at the `log.txt` file (to be found in the same folder as `Duke2Edit.exe`). If there is nothing useful in that file, feel free to contact me (see page VIII) and tell me about the issue. I won't be able to check these messages every day, so be prepared to wait for a few weeks (or months).

To identify the problem, I have might need you to send me whichever file you were working on. You could speed things up a little by uploading the files in question (level files, CZone files etc.) to an online file host of your own preference and adding a link to the files in your post/message. Please *do not* attach these files to an email – I will delete such mails without even reading them.

List of Tables

List of Figures