TRN-Files

1. Introduction

The ".trn" files are an NWN2 addition to the area-files already known from NWN1. They contain all terrain-information of exterior areas, that is: visible terrain (surface), walkable terrain (walkmesh), lighting (color, shadows), texturing.

In addition there seems to be information about the gras and there has to be information about water - but both is still not fully understood at the time of writing.

2. Terrain-Layout

Each ".trn" file contains all data for one map, but is divided into packets describing "megatiles" (2x2 tiles). Each megatile can have a different set of up to 6 different textures and 4 water levels.

The internal representation of a megatile is a trimesh-like grid, build by 625 vertices (25x25) that define 1152 (24x24x2) triangles. This structure is saved in the terrain-file and *can* be modified, although it's unsure how toolset and game will react - so for now it should *not be touched*, except from changing the vertices' z-coordinate and thus modelling the height of the terrain. With each vertex there are a number of additional values stored, e.g. normals and tinting (color), but not all of them are known by now.

Texturing information is stored in 2 standard DDSs (direct draw surfaces) of 128x128 pixel dimension and 32 bit depth.

Gras is stored within the trn-file, all other placeable vegetation is stored elsewhere.

Water is stored within the trn-file.

Please bear in mind:

All information presented herein is the result of guesswork,
so don't take any of it for fixed!

If you find out more, please send mail to

tanita@elfenwald.homelinux.net

3. File-Format

For those in the know: it *seems* that the trn is a derivate of the mdb-format!

a) Header

At the beginning of the trn-file there is a simple header:

Header		
Type	***Label***	***Description***
`char[4]`	`magic`	always 'NWN2', identifying the fileformat
`int16`	`vminor`	minor of version information (v major.minor)
`int16`	`vmajor`	major of version information (e.g: v 3.2)
`int32`	`packets`	number of packets stored in this trn

b) Packet-Key Table

Immediately after the header there is a list of packet-keys, containing `packets` entrys of the following, very simple format:

Packet-Key		
Type	***Label***	***Description***
`char[4]`	`pcktype`	can be 'TRWH', 'TRRN', 'ASWM', 'WATR', ...
`int32`	`pckoffset`	start of this packet, beginning from start of file

The rest of the file is build of these packets, each starting at `pckoffset`. Each packet starts by its FourCC type, followed by an int32 giving the length of its payload data. So even if we do not know form or content of a packet we still can read, store and write it.

Packet, general form		
Header		
Type	***Label***	***Description***
`char[4]`	`pcktype`	same as in key-table
`int32`	`paylen`	length of payload data
Payload		
`byte[paylen]`	`payload`	payload, of course each has its own format...

c) Packet "TRWH"

This packet is first in all (examined) TRN-files. It holds the terrains dimensions:

Packet 'TRWH'		
Header		
Type	***Label***	***Description***
`char[4]`	`pcktype`	'TRWH'
`int32`	`paylen`	length of payload data
Payload		
`int32`	`width`	terrains width
`int32`	`height`	terrains height
`int32`	`reserved`	unused or unknown

d) Packet "TRRN"

This packet contains the visible terrain of one megatile. There are `width` x `height` terrain-packets in the file, starting at 00x00y (lower left), followed by 01x00y and so on.

The TRRN-packet is build around a standard-form trimesh of 25 x 25 = 625 vertices defining 24 x 24 x 2 = 1152 triangles (squares, divided into two triangles). It is possible to change the form of this mesh, but it's almost certainly a very bad idea... it is absolutely unknown by now how the client and/or toolset will cope with a nonstandard terrain mesh, but i'd bet one or both of them will just panic... :-)

Aside the terrain-mesh there are two DDS's describing how to layer and mix the six textures, and a special "gras" data block.

Packet 'TRRN'		
Header		
Type	***Label***	***Description***
`char[4]`	`pcktype`	'TRRN'
`int32`	`paylen`	length of payload data
Payload		
Terrain-Header		
`char[128]`	`trnname`	name of the terrain (unique format)[1]
`char[32]`	`texture1`	name of the first texture layer
`char[32]`	`texture2`	name of the second texture layer
`char[32]`	`texture3`	name of the third texture layer
`char[32]`	`texture4`	name of the fourth texture layer
`char[32]`	`texture5`	name of the fith texture layer
`char[32]`	`texture6`	name of the sixth texture layer
`float[3][6]`	`txcolor`	RGB-color of the six texture layers
`int32`	`vercount`	number of vertices in terrain mesh (always 625)
`int32`	`tricount`	number of triangles in terrain mesh (always 1152)
`vertex[625]`	`vertices`	625 vertices, format see below
`triang[1152]`	`triangles`	1152 triangles, format see below
`int32`	`ddsAlen`	length of first DDS (always 65664)
`byte[ddsAln]`	`ddsA`	DDS A, see below
`int32`	`ddsBlen`	length of second DDS (always 65664)
`byte[ddsBln]`	`ddsB`	DDS B, see below
`int32`	`grascount`	number of gras-blocks
`gras[grscnt]`	`grasdata`	blocks of gras-data, number and size varies

[1] the terrain names format is build of three parts, e.g. "012345nameofmap00x00y" beginns with a 6 char number seemingly randomly assigned (or reflecting some internal id) to distinguish maps with identical names. then follows the maps name, and after that the position of this megatile within the map.

Most interesting might be the format of the vertex-blocks:

Vertex		
Type	***Label***	***Description***
`float[3]`	`position`	x/y/z-coordinates of this vertex
`float[3]`	`normal`	x/y/z-components of normalvector (used for lights)
`byte[4]`	`tinting`	x/r/g/b color for tinting the map
`float[2]`	`xy_0to10`	x/y-coordinates within megatile (range 0 to 10)
`float[2]`	`xy_0to1`	x/y-coordinates within megatile (range 0 to 1)

The last two sets of two floats are coordinates relativ to the megatile's borders, the first ones ranging from 0 to 10, the second ones ranging from 0 to 1. They might be coordinates in *texturespace* and thus affect mapping of textures to the tiles, i can imagine quite a number of usefull effects to achieve by tampering with these...

While you can change the x/y-coordinates of the vertices you should bear in mind that they are related to the vertices of the walkmesh, so it might be troublesome to move them around...

The (presumed) normalvector affects shading of the terrain, by perturbing it you can give a flat surface some visual "depth". Sadly the toolset will "adjust" it as soon as you touch the terrain with a terrainbrush. On the bright side this will also correct the walkmesh, so as long as we don't fully understand walkmesh format it's still possible to make the toolset recompute the walkmesh by "brushing over" the terrain with a very soft (0%) smoothing brush...

Texturing information is stored in two standard 32 bit deep DDS-bitmaps (refer to Nvidia for the specs). These bitmaps do not define rgba pixels as they usually do, but each color channel (red, green, blue and alpha) is used to alpha-mix the texture layers. In elder versions of the TRN-format there have been only 4 textures, so one DDS was enough, but now there are six textures so the fifth and sixth texture use the first two colorchannels of a second DDS. In the future this might be expanded to 8 textures, as there are still two unused channels...

Triangle		
Type	***Label***	***Description***
`short[3]`	`vertindex`	index of the vertices defining this triangle

The triangles are laid out regular, forming 24 rows and columns of squares, each square divided into two triangles. Just activate wiremesh view in toolset to get an impression of the grid's layout. As with vertices you can alter this structure completely, but toolset relies on it and will get confused if you do so.

The main reason for storing all of this redundant (as ist could be rebuilt algorithmically) information may be to speed up loading and viewing the 3d-surfaces. Probably what we have here is the very format you feed into DirectX? This could outline what can be done, and what can't by tweaking the meshes...

Gras		
Type	***Label***	***Description***
`char[32]`	`blockname`	name of this gras-block, containing tilecoordinates
`char[32]`	`grastex`	name of the used gras-texture
`int32`	`bladecount`	number of gras-blades
`blade[bldct]`	`blades`	gras-blades, see below

Blade		
Type	***Label***	***Description***
`float[3]`	`position`	x/y/z-position of this gras-blade
`float[3]`	`direction`	orientation of this gras-blade (sort of normalvector?)
`float[3]`	`dimension`	0, 0.5 or 1, perhaps sort of texture-mapping

Each megatile can have an optional datablock describing gras layout, but this block may be missing if there is no gras at all.

Gras is just a (sometimes huge!) number of rectangular, semi-transparent patches of texture (aka "blades") set at random positions and facing random directions, all together forming the impression of dense vegetation.

Other than the positioning coordinates i didn't investigate any further as i do not intend to tweak gras in any way, but i'm pretty sure this is again some kind of direction and a set of texture coordinates.

Thinking of it as i write this, these might as well be only two floats for orientation and thus four floats for texture-mapping - i'll have to look into this again, sometimes!

e) Packet "WATR"

This packet holds water information. It is uncertain wether you can have multiple levels of water in one megatile, and a lot of information is still unknown. What is known is pointing at a lot more functionality than what is exposed by the toolset, for example it is definitely possible to modell water almost like terrain...

Packet 'WATR'		
Header		
Type	***Label***	***Description***
`char[4]`	`pcktype`	'WATR'
`int32`	`paylen`	length of payload data
Payload		
Water-Header		
`byte[128]`	`unknown`	probably name?
`float`	`red`	RGB red as fraction of 1.0
`float`	`green`	RGB gree as fraction of 1.0
`float`	`blue`	RGB blue as fraction of 1.0
`float`	`ripple_x`	like in toolset
`float`	`ripple_y`	like in toolset
`float`	`smoothness`	like in toolset
`float`	`ref_bias`	like in toolset
`float`	`ref_power`	like in toolset
`float`	`unknown`	always 180f
`float`	`unknown`	always0.5f
`texture[3]`	`layer[3]`	water texture, scroll direction, scroll rate
`float`	`offset_x`	x in water-space (1/8 of megatiles x coordinate)
`float`	`offset_y`	y in water-space (1/8 of megatiles y coordinate)
`int32`	`vercount`	number of vertices in terrain mesh (always 625)
`int32`	`tricount`	number of triangles in terrain mesh (always 1152)
`vertex[625]`	`vertices`	625 vertices, format see below
`triang[1152]`	`triangles`	1152 triangles, format see below
`int32[1152]`	`triflags`	0 = water, 1 = no water
`int32`	`ddslen`	length of DDS (always 16512)
`byte[ddsln]`	`ddsA`	DDS, see below
`int32[2]`	`megatile`	x, y indices of the associated megatile

The three texture-structures hold the names of the textures (32 chars), plus 4 floats defining the scrolling direction (sin & cos of a scrolling angle), scrolling rate and scrolling angle. I just don't know why there are two different definitions of the scrolling direction in the toolset, but they are both here as well...

Please note that dir_x and dir_y are not exactly the values you enter in the toolset, but are normalized to represent a vector of length 1.

Texture		
Type	***Label***	***Description***
`char[32]`	`texturname`	name of the texture
`float`	`dir_x`	amount of scrolling into x-direction
`float`	`dir_y`	amount of scrolling into y-direction
`float`	`rate`	speed of scrolling
`float`	`angle`	scrolling angle, as multiple of pi (not degree!)

Please note that the vertex-structure is ***not*** the same as in terrain-packets (triangle-data is the same though).

Vertex		
Type	***Label***	***Description***
`float[3]`	`position`	x/y/z-coordinates of this vertex
`float`	`x_0to5`	x in [0..5] coordinates in texture-space of the waters
`float`	`y_0to5`	y in [0..5] normal map (wobbling the surface)
`float`	`x_0to1`	x in [0..1] coordinates in texture-space of the waters
`float`	`y_0to1`	y in [0..1] highlights map (sparkling highlights)

Triangle		
Type	***Label***	***Description***
`short[3]`	`vertindex`	index of the vertices defining this triangle

f) Packet "ASWM"

This packet holds walkmesh data in zipstream compressed format.

Packet 'ASWM'		
Header		
Type	***Label***	***Description***
`char[4]`	`pcktype`	'ASWM'
`int32`	`paylen`	length of payload data
SubHeader		
`char[4]`	`compid`	'COMP' indicating compressed data to follow
`int32`	`paylen`	length of compressed data
`int32`	`datalen`	length of uncompressed data
Payload		
`byte[paylen]`	`payload`	walkmesh in zipstream compressed form

So this is the great mystery...

Here we have the walkmesh, but (to me) the format is unknown.

Please bear in mind:

All information presented herein is the result of guesswork,
so don't take any of it for fixed!

If you find out more, please send mail to

tanita@elfenwald.homelinux.net