MTR

MTR files define a material which can be referenced from within models (MDL files) that are linked to a node on a model. They essentially can define any custom shaders and shader parameters to use, any additional texture references such as normal or specular textures, and even replace the base texture (the diffuse one) by referencing another texture file.

Loading MTR Files

There are two ways to reference an mtr file. The first method is to use the new **materialname** field in the MDL file, usually with a file name such as m_name.mtr, since this this removes any ambiguity and for the case of PLT textures makes it clear what MTR file is in use, due to the bugs below.

```
materialname myMaterialFile
```

The second way is to reference it in a **bitmap** or **texture0** field:

```
bitmap myMaterialFile
```

The latter enables using materials with existing textures and models without model edits. If an mtr file is present it takes precedence over image textures - be it DDS, PLT or TGA.

PLT and MTR Usage Note

Please note that MTR files are PLT files need to be done in a very specific way. The **texture0** line in the MTR file must be omitted entirely. It cannot be null, or even the original PLT texture name, since this does some odd things around what PLT colour are chosen and can lead to colour bleeding between models.

The best bug free method is to have the **bitmap** (or texture0) to be the PLT texture name - mainly as a point of reference (it is actually ignored) - and **materialfile** line to be the material name, eg:

```
  bitmap pmh0_belt003
  materialname M_pmh0_belt003
```

Then the file **m_pmh0_belt003.mtr** contains just this line (since it is just a normal file):

```
texture1 pmh0_belt003_N
```

Note the omission of the texture0 line.

MTR File Format

MTR files are made up of a number of simple text settings one per line. They are outlined below. Comments can be added with two slashes (eg: "// comment").

A sample completed MTR file:

```
// Specify it is normal and spec mapped
renderhint NormalAndSpecMapped

// Define textures
texture0 plc_tree_d // Diffuse
texture1 plc_tree_n // Normal
texture2 plc_tree_s // Specular
texture3 plc_tree_r // Roughness
texture4 plc_tree_h // Height
texture5 plc_tree_i // Illumination
```

Shaders

Defining shaders is optional. If omitted, a default shader will be picked depending on content type. See here for more about shaders.

Defining Custom Shaders

```
// Specifies a custom vertex shader: vsMyVertexShader.shd
customshaderVS vsMyVertexShader

// Specifies a custom fragment shader: fsMyFragmentShader.shd
customshaderFS fsMyFragmentShader
```

Renderhint

Specifying a **renderhint** in the material file will instruct the game to handle the associated content in a certain way. Currently, only two renderhint tokens are supported, which have identical function and can be used interchangeably : "**NormalAndSpecMapped**" and "**NormalTangents**". Specifying either of these will make the game generate vertex tangent and handedness information *(provided that the associated model is not already compiled)*, as well as picking a shader variant with normal mapping if no custom shader is specified.

Defining Renderhint

```
renderhint NormalAndSpecMapped
```

The renderhint can also be added directly to a MDL file.

Note that NormalAndSpecMapped and NormalTangents do seem to have different effects if you're directly altering a flat amount of specular or roughness set in the MTR file itself. See Shaders for an example.

Textures

Texture definitions are optional. Any texture defined in the mtr file takes precedence over the one in the model/ mdl file. Up to 11 textures are supported: **texture0** - **texture10** (numbering doesn't need to be continuous). Specifying a texture makes it available in the shaders in the corresponding uniform sampler (texUnit*).

As noted above texture0 should be omitted for PLT mtr files due to bugs.

Defining Textures

```
// Binds a texture to texUnit0 in the fragment shader. Overrides bitmap/texture0 in the model.
texture0 myFirstTexture

// Binds a texture to texUnit1 in the fragment shader. Overrides texture1 in the model.
texture1 mySecondTexture
```

Default textures are as per below if using Standard material inputs:

```
texture0 - Diffuse
texture1 - Normal
texture2 - Specular
texture3 - Roughness
texture4 - Height
texture5 - Illumination
```

To have a set of texture0, texture1...texture10 lines but miss some out, use the keyword **null**, eg to miss off the specular and roughness map but have the height map:

```
renderhint NormalAndSpecMapped
texture0 object_tex
texture1 object_tex_n
texture2 null
texture3 null
texture4 object_tex_h
texture5 object_tex_i
```

Additional texture slots are available from 11 to 15 which previously had similar functionality, however they are currently unavailable for use in the MTR file using the default shaders. Each slot is now internally used for other purposes as listed below.

Texture Slot	Current Use
texture11	Depth Buffer [used for post processing]
texture12	Screen Buffer Color [used for reflections and post processing]
texture13	Cube Environment Map
texture14	Flat Environment Map
texture15	Noise Map [used for water]

Parameters

Optional definitions of parameters, which will be passed to the shader.

Defining Parameters

```
// Creates a float parameter that will be bound to a shader uniform of the same name (if it exists).
parameter float fModulator 0.5

// Creates an integer parameter that will be bound to a shader uniform of the same name.
parameter int iCount 2

// Creates a 4-float parameter that will be bound to a shader uniform of the same name.
// Float parameters may have up to 4 dimensions (not supported for int)
parameter float vDiffuse 0.8 1.0 0.65 1.0
```

See an example of passing parameters on the Standard material inputs page.

You can also set parameters on objects in NWN (so not individual tiles, but say a creature, item or placeable) for shaders with nwscript, such as with the function **SetMaterialShaderUniformVec4**, see Shaders for some more info on this.

Examples

Overwriting Textures

The following example overrides the standard armoire placeable (plc_a01.mdl). The new placeable is a simple square and references a mtr file with three textures. The files have to be placed into the override folder:

The MDL file has these fields:

plc_a01.mdl

```
node trimesh a01_planeA
  parent plc_a01
  ...
  bitmap ttr01_grass02           # Old style texture reference, overwritten in mtr
  texture1 null                  # texture will be overwritten in mtr
  texture2 null                                    # texture will be overwritten in mtr
  renderhint NormalAndSpecMapped # Enables normal and specular mapping
  materialname testmat           # Reference to mtr file
  ...
endnode
```

For information on PBR texture references in .mtr files, see here.

testmat.mtr

```
// textures0 overwrites bitmap or texture0 parameter in mdl
texture0 testtx
// Used by normal and spec mapping shaders
texture1 testtx_n
texture2 testtx_s
```

Roughness, Heightmaps and Illumination are supported as well, the fuller usually used list are as follows:

```
renderhint NormalAndSpecMapped
texture0 * (Diffuse)
texture1 *_n (Normal)
texture2 *_s (Specular)
texture3 *_r (Roughness)
texture4 *_h (Height)
texture5 *_i (Illumination)
```

Remember: if the model uses PLT textures make sure you do **not** include the texture0 line at all, even if it says "texture0 null", since it can bug out the PLT loading.

Accessing Textures Within Shader Code

Each texture 0-10 comes with two uniforms accessible within shader code.

The first is **texture<X>Bound** where <X> is the index of the texture. If **texture1Bound** is not equal to 0, then a texture is available in that slot to the shader. If not, then reading that slot may instead print waste data from the object's last purpose. Errors include printing a ghost spec map from another material in the scene if a spec map is not explicitly given in the MTR file or when texture2 is set to null.

The second uniform is **tex<X<Unit**, again where <X> is the index of the texture. This uniform holds the Sampler2D object. Using the texture2D function, you can ask for a texel form the sampler2D object by specifying uv coordinates and an optional bias. See external link for more details.

Textures 11-15 use other names as listed below.

Texture Slot	Availability Flag	Shader Uniform or Method
texture11	N/A	uniform sampler2D texFBDepth
texture12	N/A	uniform sampler2D texFBColor
texture13	texEnvCubeBound	uniform samplerCubeMap texUnitEnvCube and function SampleEnvironmentMapCube()
texture14	texEnvBound	uniform sampler2D texUnitEnv and function SampleEnvironmentMap()
texture15	texNoiseBound	uniform sampler2D texUnitNoise