Model Table of Parameters

This is a page that may get merged with the parent but is Jasperre messing around figuring out all the parameters in a "neater" table for easier reference mainly for himself but may be useful to others!

Parameters marked in orange ("Warning") are basically unused by the game but are "valid" for one reason or another (but you should consider using the better alternatives), ones in red ("Error/ignore") are not read by the engine at all. Blue ("unknown") coloured fields have some kind of use, possibly, but either Bioware never used it or it's not been tested much or at all/we don't know the parameters and full implementation, and may not be worth using it as a parameter anyway.

Please correct here and on the parent page where possible. Information sourced from:

- Torlacks information: https://neverwintervault.org/project/nwn1/other/torlaks-nwn-binary-model-files-basics
- The cleanmodelsee cleaner/decompiler: https://github.com/plenarius/cleanmodels
    - Additionally checking the decompilation of files previously compiled with NWN:EE, or with nwnexplorer a quick check of original old pre-EE models
- Discord and various people and the MDL documentation on this wiki
- The games console - which seems to have most if not all of these parameters listed (presumably leftovers from development since actually *using* the console to generate a model from scratch would be crazy, if it even would work! Possibly it uses these to load ASCII models though)
- Random testing

- Whitespace, Comments and Ignored Parameters
- Text, Integer and Float Limits
- MDL Parameters
    - filedependancy
    - newmodel and donemodel
    - newmodel Parameters
    - Model Geometry (beginmodelgeom) Parameters
        - Node Parameters for dummy
        - Node Parameters for Geometry (trimesh, animmesh, danglymesh, skin)
        - Node Parameters for Effects (emitter, light, reference)
        - Node Parameters for Axis Aligned Bounding Boxes (aabb)
    - Animation (newanim) Parameters
- MTR Parameters
- TXI Parameters

Whitespace, Comments and Ignored Parameters

All extra whitespace is ignored by the game, but most files have some indents for readability. Comments start with # and typically are only included only on their own dedicated lines. Compiling a model loses both of these making it a much more efficient binary file for loading purposes, and thus both are optional.

Note that whitespace between parameters (eg "ambient 1 1 1" has spaces between the 1's) is, of course, required - but a single space is fine. Many programs automatically format.

Any fields that the game doesn't recognised will be simply skipped, eg; if you can add some stuff to automate your own scripting of ASCII MDL. For instance a parameter field called "jasperrescoolfield reallycool" will not be compiled or used by the game.

Text, Integer and Float Limits

These values have these limits:

Type	Limitations	Notes
Text	Will be noted in each field as needed. Some may be stored in the compiled model as CHAR [32], some as CHAR [64] and some as CHAR [16] Some have set options. Note for file/parent references the keyword **NULL** is recognised by the game as "blank/not set/invalid/don't look up" and is safe to use to not load anything in that instance. Any references to a resman file need to be 16 characters or less to work properly.	UTF-8 windows CR LF is used by ASCII files by default although not sure what UTF-8 characters are valid for text fields. Note that file name references - ie things loaded into the game from resman - rely on OS limitations that might have their own restrictions (especially on Windows) **Recommendation:** Typically you should just use all lowercase for text fields simply for consistency, but it doesn't *really* matter - "classification ChArAaCtEr" is valid. Text ASCII parameters usually translate into a binary flag/integer as appropriate. See Torlacks documentation if you are interested in this.
Integer	To test maximum and minimum but may depend on parameter	The compiled model uses INT8, UINT8, INT16, UINT16, INT32 and UINT32, so...yeah!

Float	To Looks like 7 significant figures in most ASCII files, eg: -0.0934969 is valid. The game reads the float in converts it to the right number, and when compiling will save that number back in binary form. To test maximum and minimum	The binary stores floats as 4 bytes, which implies a standard float32 that NWN tends to use. The ASCII is in text form so many not accurately represent the float value the game can read if hand edited. Some decompilers *shouldn't* lose precision if it properly formats the float back into a string for viewing. This needs some testing!

MDL Parameters

MDL files can be compiled in game, and when compiled if the field is required a default value is produced if the field is missing.

Note that there are some fields which are required for the model to even load, so won't have a "default" as such since the game either won't load the MDL file (with an error usually in the client log) or it will load it and crash the game to desktop.

filedependancy

This top level parameter typically comes first and is *entirely unused by the engine.* It appears to be for human readability only.

If it did anything at all it would have been part of the MDL pipeline at Bioware (who regularly shipped uncompiled models leaving the .max file defined here, as does the gmax and other exporters as a human readable reference).

The game compiler drops this and any decompiling efforts usually throw in "Unknown" just so there's something.

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes
filedependancy	`filedependancy <max_file_name>`	helm_001.max	n/a	Text	Text string	Safely leave this as NULL or leave out of definition - completely unused by the engine, doubtful used by any 3rd party tools

newmodel and donemodel

This top level parameter blocks out the entire model classification. Technically the definition "

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes
newmodel	`newmodel <model_name>`	helm_001	n/a	Text	Filename	This should match the .mdl file name, eg; helm_001.mdl helm_001 Each model in the game should have a unique mdl file and therefore newmodel entry While the compiled model has CHAR[64] as the length this should be kept to 16 characters or less (resman limitation).
donemodel	`donemode <model_name>`	helm_001	n/a	Text	Filename	This must match the newmodel name so put the same value. This is at the end of the file signifying the file is done with this model. It is likely only one model is loaded from a file so no newmodel is expected after this.

newmodel Parameters

These are top level parameters are under **newmodel** that either block a new set of parameters out (eg; **beginmodelgeom**) or specify one item for the entire model. It is in a rough order of how the game stores it (relying on a decompiler)

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes
setsupermodel	`setsupermodel <model_name> <supermodel_name>`	c_colemralds pfe0	NULL	Text	2 parts: • Current model name as defined by **new model** • Parent "super model" used to inherit animations or **NULL**	This command references other model files. A model may have a (super)model from which it inherits animations. If there is no supermodel the second parameter has to be NULL. The model may overwrite any animation from its supermodel. The structure of the two models must to match, i.e. the order of the objects in both files must be the same.

classification	`classification <classification>`	character	Not set (0 in binary)	Text	If not left out of definition it should be on of these: • effect • tile • character • door	If not included no special classification is applied. This is entirely valid and happens on things like GUI models where this flag is simply not set (in binary it's set to 0). The other 4 all have some (unknown) level of changes in the engine when loaded, so while you can reference a door object in placeables.2da it is likely better to first of all make a copy of the model and alter this classification to match the character one. The engine may make further assumptions about the model classification however, depending on how it is loaded (eg; via. an appearance.2da line, GUI model, placeables. 2da, visualeffects.2da, etc.) but this needs testing. • Not set - this is common among GUI elements, • effect or effects - These have some additional properties for effects • tile - these are used as tiles in a tileset • character - these are anything not covered by the other 3 (placeables, phenotype models, usual creature models, items,) • door - additional properties for doors only These classifications affects render order, hitchecks, highlighting and probably a bunch of other stuff. **If anyone has concrete information about what the engine does with this please let us know on Discord or get an account and edit this in.**
setanimationscale	`setanimationscale <scale>`	1.4	1	Float	Positive float values (eg; 0.1 to 3.0, but not a negative like -1.5)	Even if the model hasn't got animations, or the property isn't set, when compiled it will obtain a default value of 1 Obviously only applicable to things with animations, usually a creature model that is inheriting animations from a parent super model
beginmodelgeom	`beginmodelgeom <model_name>`	helm_001	n/a	Text	Filename	This starts the geometry definition of the model, reuse the filename for the parameter
endmodelgeom	`endmodelgeom <model_name>`	helm_001	n/a	Text	Filename	This ends the geometry definition of the model, reuse the filename for the parameter
newanim	`newanim <animation_name> <model_name>`	fog pln_fog	n/a	Text	Two parts which are likely limited to 16 characters: • Name of the animation (in the example: "fog") • The parent model name (in the example: "pnl_fog", this is the same as the **newmodel** name)	This starts an animation definition in the model. You can have multiple animations tied to the parent model. These can also override the super model used for animations set with the setsupermodel parameter.
doneanim	`doneanim <animation_name> <model_name>`	fog pln_fog	n/a	Text	Copy of newanim	Ends that animation definition. A new animation may occur afterwards.

Model Geometry (beginmodelgeom) Parameters

The model geometry is one section that has all the nodes (parts) of a model defined. It must begin with a dummy with no parent.

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes

node	node <node_type> <node_name> node dummy helm_001 node trimesh helm_001_g	dummy (if not a valid value)	Text	node_type must be one of these: • dummy • trimesh • animmesh • danglymesh • skin • emitter • light • reference • aabb node_name can be any text, but likely should be a relevantly named node. At least one dummy one usually is named identically to the model name with nothing defined in it. **Note** the node_name really should NOT be longer than **32 characters** since it truncates and this can cause to unforseen issues with a variety of engine situations - just... don't do it.	Each of these node types have a different set of valid parameters. Notably some of these are undocumented or unused by Bioware. The main ones are: **Dummy:** dummy **Geometry:** trimesh, animmesh, danglymesh, skin (skinmesh) **Effects:** emitter, light, reference **Axis Aligned Bounding Boxes:** aabb Any others like "patch" (used in some old Bioware models) default back to "dummy" it seems so are safely ignored or see the dummy section for them. The order nodes are added to the MDL is important and defines the internal "part number" and thus if you use inherited animations the order becomes important for that.
endnode	endnode			n/a		This ends the current node, there is no reference to what it is closing weirdly compared to other begin/end and new/done.

Node Parameters for dummy

Dummy nodes don't have a model, but are used to position parts of the model for animations or for attachment nodes (like "headconjure" or similar).

The default dummy node just has a parent NULL defined and nothing else - although some models add a position and orientation to it. All models should have this.

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes
parent	`parent <node_parent>`	parent rootdummy	n/a	Text	Another node name defined in the MDL or NULL for no parent	Most newanim and beginmodelgeom sections contain one null parent node, usually with no position or orientation, that is used to parent other nodes.
wirecolor	`wirecolor <R> <G> <B>`	wirecolor 0.5 0.5 0.5	n/a	Float	Float values, but since this property does nothing it can be safely ignored.	This is unused by the engine and can be safely ommitted from MDL ASCII and it is not added to compiled models. Blender or NWMax may use it though.
position	`position <X> <Y> <Z>`	position 0.0 1.12739 1.99808	n/a	Float	Float values	Position of the node relative to its parent. This property can be animated.
orientation	orientation <X> <Y> <Z> <A>		0.0 0.0 0.0 0.0 (if not dummy)	Float	Float values	The rotation of this node relative to its parent, in axis-angle representation. The first three values define the axis, the fourth defines an angle in radian. This property can be animated.
scale	scale <scale_multiplier>	scale 1.5	1.0	Float	Float value, although obviously negative would be odd/non-functional.	Uniform scale of this model node. This property can be animated. Note that for static placeables this is likely ignored (so can't be a shortcut to making a model bigger and smaller). There may be other limitations such as the type of models that are supported using it. Given you can resize a model by...resizing the verts and so forth, in blender or gmax or even with a script if you're proficient, this seems to have limited value in the first place. Bioware *didn't really ever use this.* The only base game model with it not set to 1.0 is tn_sdoor_24...which may just be an error in the first place.

Node Parameters for Geometry (trimesh, animmesh, danglymesh, skin)

You can (and should) use the parameters for **dummy** above here as well. Scale, orientation and position is relative to the parent node.

The different types are:

- trimesh - your basic static shape, won't change as the model moves
- danglymesh - Danglymeshes are Trimeshes which are "bouncy" or "dangly" and are affected by environmental effects like wind, character movement and visual effects. Bioware used it on creatures for some bits of hair, ghosts, clothing bodyparts and...jiggly breasts (sigh), flags and grass on tilesets some other minor things (some flappy bits of creatures). In addition to their own set of attributes, Danglymeshes inherit all attributes from Trimeshes.
- animmesh - Animeshes are Trimeshes with animated UV coordinates or vertices. In the geometry section of a model they have exactly the same properties as Trimeshes. See animation parameters below.

- skin - Skinmeshes are Trimeshes with additional weight parameters to support skeletal animations. As such they possess all properties of a Trimesh. However they have some oddities like not generating shadows.

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes
center			n/a	Float	n/a	Unused by the game. Would possibly have been the pivot point of the object. Instead the nodes determine the object layout and attach points/center points.
ambient	`ambient <R> <G> <B>`		1.0 1.0 .1.0	Float		OpenGL GL_Ambient material parameter. If it is lower then the texture on the model becomes quite dark. The default is a sane 1.0 1.0 1.0 so it can be "safely" left out, but better to set it manually. Note the old Bioware default was 0.2 0.2 0.2 where usually you'd want it set to 1.0 1.0 1.0 so if a model looks dark decompile it and change it to 1.0 1.0 1.0 (this was updated in patch 1.81.8193.17)
diffuse	`diffuse <R> <G> <B>`	`diffuse 1.0 1.0 1.0`	1.0 1.0 1.0	Float	0.0 to 1.0 for each R, G and B	Material diffuse colour, ie the RGB applied to the main texture0 file. It is a relatively cheap way to get a different coloured creature; say you wanted a "red" goblin you'd set it to 1.0 0.0 0.0. Frankly in this day and age creating a better replacement texture on a new model, or even using Replace ObjectTexture, or shaders is a much better idea. The default is a sane 1.0 1.0 1.0 so it can be "safely" left out, but better to set it manually. Note the old Bioware default value were 0.8, 0.8, 0.8 which are awful. Use 1.0, 1.0, 1.0 by default in your own models else your texture will come out weird (this was updated in patch 1.81.8193.17)
specular	`specular <R> <G> <B>`	`specular 0.5 0.5 0.5`	0.0 0.0 0.0 (But can be ignored)	Float		Material specular color, however this has no effect. Compiled models do include it and default to 0.0 0.0 0.0
shininess	`shininess <shininess_value>`	`shininess 1.0`	1.0 (But ignored by engine)	Float		This appears to be now unused as of NWN:EE, if it was ever used before the current renderer at least now ignores it, but it is still saved to the compiled model if left out (at least with a sane default 1.0)
render	`render <on_off_boolean>`	`render 1`	1	Boolean	1 - this model is rendered in the game world 0 - this model is not rendered in the game world	Controls whether this object should be rendered. The geometry will still be loaded to the memory and casts shadows, even if this is unchecked. This is frequently used in conjunction with the Shadow attribute to create a low-poly shadow volume object. Valid values are either 0 (disabled) or 1 (enabled). Please use 1 and 0 instead of things like "true" and "false" here.
shadow	`shadow <on_off_boolean>`	`shadow 0`	1	Boolean	1 - shadows are attempted to be rendered for this object 0 - shadows are not rendered for this object ever	**Shadow rendering only works with trimesh**, not with skinmesh (and likely not animmesh or danglymesh). It also doesn't work well on models with high face counts - best idea to really do shadows well on a model is to have the usual model with **render 0** and with **shadow 0** set, and shadow version created as a much simpler trimesh with **shadow 1** and **render 0** set. Keep the new one much lower on the faces count (sub-100 or even lower) since it helps not have errors in the resulting shadows and keeps things running fast. See this Model Shadows page for more information. It needs expanding to cover the ins and outs and all the different object types using them and how the game renders it, as well as debugging them and best practices. Disabling shadows is also recommended for things generating their own light and which casting a shadow makes no sense (like ghosts or other see through things) since the shadows are quite "hard" and cannot be altered to be more transparent per-creature.

renderhint	`renderhint <renderhint_option>`	`renderhint`
		
		`NormalAndSpecMapped`
			Not set - ""	Text (set options)	NormalAndSpecMapped
					
					NormalTangents
						EE only.
						
						Either option provides the same results. The model compiler seems to sets a flag to 1 if either are set.
						
						Instructs the game to generate tangent and handedness vertex data for the associated model(s), provided that they are uncompiled and no tangent and handedness data exist. Also makes the game choose a shader with normal mapping enabled to render this content. Note: The renderhint can also be specified in a MTR material file.

beaming	`beaming <on_off_boolean>`	`beaming 1`	0	Boolean	1 - beaming is on 0 - beaming is off	For light-rays. Light rays cannot be made specifically for a tile, as tiles can rotate. Instead a different kind of shadow was created where the shadow volume is rendered at an alpha value and a specific color. To create shadow volumes, make a group of simple triangles above the camera level and enable this flag.
inheritcolor			0	Boolean	1 - inherit colour 0 - do not inherit colour	Apparently not used by the game (in fact **no** base game Bioware model sets this to 1 on any node!) but if you think it is used provide an example. Might have been so nodes that have a parent inherit their trimesh "colors" value from them (presumably that colour being the ambient or diffuse numbers? or the "colors" array under trimesh?). To be honest even if it did work there is little reason to use it today and you can simply define colours per-node anyway which is safer. Note the binary stores this weirdly as a integer, even though it should be a boolean.
alpha	`alpha <alpha_amount>`	`alpha 0.5`	1.0	Float	0.0 to 1.0	Alpha value controlling transparency. Valid values are floating point numbers within [0.0, 1.0] - 0.0 being fully transparent and 1.0 fully opaque. This property can be animated.

transparencyhint	`transparencyhint <transparency_priority>`	`transparencyhint 1`	0	Integer	Values 0 through 9 0 - Transparency hint disabled - will be rendered before transparent items 1 - 9 - Transparency hint enabled - and will display in order, so 9 will be rendered last.	As transparency can be extremely costly to calculate from alpha values on a texture, this property gives the engine priority values of transparent objects. An object with a transparency hint of 1 will be rendered after an object that has a value of 0. The values appear to work up to 9 (10 values total) but this needs fully testing since the value stored in the compiled MDL is a UINT32 (then again boolean values have UINT32...silly Bioware) **(NEEDS TESTING MAY BE WRONG)** This does not work on dynamic objects, it only affects static objects (*this might mean only works with trimesh?* - Jasperre)
selfillumcolor	`selfillumcolor <R> <G> <B>`	`selfillumcolor 1.0 0.5 0.5`	0 0 0	Float	Colour values 0.0 to 1.0	Makes the mesh glow without actually acting as a light source. It most likely corresponds to the GL_EMISSION material parameter in OpenGL. This property can be animated. **Note:** In older mdl files this parameter might be spelled wrongly as `seTfillumcolor`. The mispelling of this parameter is one of the primary reasons that older models fail to compile using nwnmdlcomp.
Textures	**Definition**	**Example Contents**	**Game Default**	**Type**	**Valid Values**	**Description and Notes**
materialname	`materialname <material_filename>`	`materialname m_helm_002`	Not set - ""	Text	MTR file name so 16 characters maximum (excepting PLT files)	EE only. The MTR file is able to define textures, or override textures defined in the MDL (but you can specify textures0 through 2 in the MDL, so the MTR can partially define textures) for use with Standard material inputs (fancymaps). Therefore a good use case is set texture0 for the diffuse in the MDL and have texture1 through texture4 defined in the MTR for fancymapping purposes, allowing one MTR file to be shared by multiple of the same model, but with different colours etc. - very useful for phenotype based armor.
bitmap	`bitmap <texture_filename>`	`bitmap c_aribeth_evil`	Not set ("") if not defined (equivalent to NULL)	Text or NULL	Texture file name (Priority of loading: DDS / PLT / TGA), so 16 characters maximum Note MTR files take precedence	This is an alias for texture0. While it is very common (most decompilers use it by default as the name of the field) texture0 is a bit more sensible with the advent of MTR files properly using the texture0 through texture10 options in that file as per below for fancymaps.
texture0	`texture0 <texture_filename>`	`texture0 c_aribeth_evil`	Not set ("") if not defined (equivalent to NULL)	Text or NULL	Texture file name (Priority of loading: MTR / DDS / PLT / TGA), so 16 characters maximum Note MTR files take precedence (excepting PLT files)	Usually texture0 is the base texture used on this node of the model. Can be omitted or preferably set to **NULL** if the texture isn't going to be rendered (see **render** above). Textures not found (or no texture defined) default to blank white usually in game. Cloaks have a special 2da that loads an overriding texture0 - see cloakmodel.2da (the prefix of the file is hardcoded and the ID number is from the 2da "TEXTURE" column). **There may be also default loading of texture0 for parts-based phenotype models. Needs checking.** As noted above bitmap is an alias but with MTR files using texture0 through texture15 the saner thing is to define just texture0 so it fits in and is more easily identifiable. As per the fancymap specs if you use them; you could potentially have the MDL apply the Diffuse (base texture), Normal and Specular per-model and the MTR apply the rest.

texture1
 	
texture2
 	
texture3
	`texture1 <texture_filename>`
	`texture2 <texture_filename>`
	`texture3 <texture_filename>`
		Untested	Not set ("") if not defined (equivalent to NULL)	Text or NULL		The original 1.69 format allows texture1, texture2 and texture3 defined as well although Bioware never used it in their models and it might not have done anything useful and it might just be broken.
						
						If it is usable it needs fully testing and the fancymap specs go up to texture5 (6 textures total) meaning 2 of them always have to be in the MTR file anyway, making it a bit useless.

Tile Specific	**Definition**	**Example Contents**	**Game Default**	**Type**	**Valid Values**	**Description and Notes**

tilefade	`tilefade <fade_option>`	`tilefade 1`	0	Integer (set values)	Options: • 0 Disabled This object will never fade • 1 Fade This object will fade • 2 Base ??? • 3 Neighbor This object will fade together with objects in neighboring tiles	This property is only relevant for tiles. It controls whether this mesh will turn invisible to offer a clear line of sight on the character.
rotatetexture	`rotatetexture <on_off_boolean>`	`rotatetexture 1`	0	Boolean	1 - rotate texture on this tile 0 - don't rotate this texture	Only relevant for tilesets and controls ground mapping: When a model with this flag is rotated, the engine rotates the uv coordinates back to the original position. This avoids texture seams on tile edges.

trimesh	**Definition**	**Example Contents**	**Game Default**	**Type**	**Valid Values**	**Description and Notes**

verts	`verts <vertex_total>`
	`<X> <Y> <Z>`
		`verts 4`
		`-1.120000 0.2400000 0.0000000`
		`-1.120000 -0.2400000 0.0000000`
		`1.120000 0.2400000 0.0000000`
		`1.120000 -0.2400000 0.0000000`
			n/a	Integer for vertex_total
				
				Float for X/Y/Z
					Positional data can be negative, positive or 0.0 for X/Y/Z	Vertices.
						
						Every lines defines a single vertex. The **faces** section on a node references the indices of three vertices for every face.
						
						It is recommended (and possibly required) to keep the vertex_total number the same across all of the geometry definitions in this section except the faces.

tverts	`tverts <vertex_total>`
	`<U> <V> 0`
		`tverts 4`
		`0.062500 0.4062500 0`
		`0.062499 0.2187500 0`
		`0.937500 0.4062500 0`
		`0.937500 0.2187500 0`
			n/a	Integer for vertex_total
				
				Float for U/V
				
				Last parameter is always 0
					Positional data can be negative, positive or 0.0 for U/V
					
					Usually defined the same amount as the verts.
						Texture Vertices.
						
						Every lines defines a single texture vertex (tvert). They are 2-dimensional - the last value is always 0. The **faces** section on a node references the indices of three tverts for every face.

tverts1
 	
tverts2
 	
tverts3
	`tverts1 <vertex_total>`
	`<U> <V> 0`
		`tverts1 4`
		`0.0625000 0.4062500 0`
		`0.0624999 0.2187500 0`
		`0.9375000 0.4062500 0`
		`0.9375000 0.2187500 0`
			n/a	Integer for vertex_total
				
				Float for U/V
				
				Last parameter is always 0
					Positional data can be negative, positive or 0.0 for U/V
					
					Usually defined the same amount as the verts.
						Previously only one texture coordinate stream (named “tverts” in the model file) was supported per mesh. We now support three additional texture coordinate streams, named “tverts1”, “tverts2”, and “tverts3”, which will be linked to any shaders with the named attributes “vTcIn1”, “vTcIn2”, and “vTcIn3”.

faces	`faces <faces_total>`
	`<V1> <V2> <V3> <S> <UV1> <UV2> <UV3> <M>`
		`faces 2`
		`0 1 2 1 0 1 2 1`
		`3 2 1 1 3 2 1 1`
			n/a	Integer for faces_total
				
				Integer for all indexes (V1/2/3, S, UV1/2/3, M)
					All the entries are indexes, starting from 0.	Each line defines a triangle:
						• The first three values (v_1 v_2 v_3) each reference a vertex from the vertex list. This is the index, not the actual coordinate.
						• The 4th value (s) defines the shading group the face belong to. All neighboring faces belonging to same group will be rendered as smooth.
						• The next three values (uv_1 uv_2 uv_3) each reference a texture vertex (uv-vertex) from the tvert list. Again is the index, not the actual coordinate.
						• The last value (m) defines a material index. This is only relevant when creating walkmeshes for tiles and determines the surface type of this face.
						
						This number is primarily what you can use to check the model complexity.

colors	`colors <vertex_total>`
	`<R> <G> <B>`
		`colors 4`
		`1 1 1`
		`1 1 1`
		`0.85882 0.70191 0.47843`
		`0.18823 0 0.47843`
			n/a	Integer for vertext_total
				
				Float for R/G/B
					Float for R/G/B between 0.0 and 1.0	Vertex colors. Each line contains the color for the corresponding vertex in the vertex list, as such the number of colors must match the number of vertices. **Vertex colors are not used by the default shaders.** Bioware rarely used this so there isn't much loss to not having it. Changing the diffuse texture is recommended instead.

tangents	`tangents <vertex_total>`
	`<X> <Y> <X> <S>`
		`tangents 4`
		`0.001087 -0.92979 0.38287 1.0`
		`-0.04183 -0.99999 -0.00166 1.0`
		`0.00174 -0.92377 0.38285 1.0`
		`-0.00425 -0.99999 0.0 1.0`
				Integer for vertex_total
				
				Float for X/Y/Z and S
						Tangents can be generated by the in game compiler (which are added to compiled models automatically to save time generating them again next time the model is loaded), but can be included in the ASCII model if fine tuned. Most of the time it is much easier to leave them out of the ASCII MDL and compile it in game just before releasing the model.
						
						Per-vertex tangent data. Each line contains the value for the corresponding vertex in the vertex list, as such the number of tangents must match the number of vertices.
						• The first three X/Y/Z values defines the tangent vector
						• The fourth value S defines the binormal/ bitangent sign

normals	`normals <vertex_total>`
	`<X> <Y> <Z>`
		`normals 4`
		`0.208259 -0.78341 -0.58556`
		`0.34848 -0.65194 -0.67344`
		`0.087436 -0.52056 -0.84933`
		`0.25736 -0.83923 -0.47900`
			Autogenerated	Integer for vertex_total
				
				Float for X/Y/Z
						Per-vertex normals. Each line contains the normal for the corresponding vertex in the vertex list, as such the number of normals must match the number of vertices.
						
						Normals can be generated by the in game compiler (which are added to compiled models automatically to save time generating them again next time the model is loaded), but can be included in the ASCII model if fine tuned. Most of the time it is much easier to leave them out of the ASCII MDL and compile it in game just before releasing the model.
						
						Note: will not be regenerated for already compiled models, you must have the file loaded from ASCII to generate them afresh (say if you alter the normal map file).

danglymesh	**Definition**	**Example Contents**	**Game Default**	**Type**	**Valid Values**	**Description and Notes**
displacement	displacement <displacement_amount>	`displacement 0.035000001`	0	Float	Positive non-zero float to get an effect	Determines show far the vertices are allowed to move. This is further restricted by the constraints. The value appears to be approximately 1/2 meter per unit. Note: In animation files and others which don't really *use* the danglymesh defined, this and period /tightness are just set to 0 and no constraints list is present. Example: a_fa.mdl with head_g for instance.
period	period <period_amount>	`period 8`	0	Float	Positive non-zero float to get an effect	Determines how quickly the vertices move and how fast they come to a rest. This value can be up to 59 before the mesh begins to jiggle incorrectly. At 59.5 the mesh may spasm sporadically. At 60 or above, the mesh will jiggle insanely for up to 1 second and then no longer jiggle, becoming locked at the starting position. Tile grass (as created by walkmesh surface material 3) uses effects equivalent a hardcoded value greater than 60. It appears to be the only thing which can use this high a value.
tightness	tightness <tightness_amount>	`tightness 3`	0	Float	Positive non-zero float to get an effect	The 'tension' in the object, restricting movement. A tight object is difficult to move by wind. Tile grass may have an approximate tightness of 50 or more.

constraints	constraints <constraints_total> <list_of_constraints>	constraints 4 0 50 100 255	Not included if none defined (ie blank)	Array of Integers	Not sure there is a limit to the number of constraints since it is per-vertex. Each value needs to be on a new line and be between 0 and 255 (byte size).	Each line contains the constraints for the corresponding vertex in the vertex list, as such the number of constraints must match the number of vertices. Constraints act as a multiplier for the displacement value and range from 0 to 255, with their value equating to the multiplier range of 0 to 1. If the maximum displacement is set to 2, then the associated vertex painted with 255 will be allowed to move approximately 1 meter during maximum strength wind.
animmesh	**Definition**	**Example Contents**	**Game Default**	**Type**	**Valid Values**	**Description and Notes**
sampleperiod						
animverts						
animtverts						
skin (skinmesh)	**Definition**	**Example Contents**	**Game Default**	**Type**	**Valid Values**	**Description and Notes**
weights	weights <vertex_total> <N1> <W1> <N2> <W2> <N3> <W3> <N4> <W4>	weights 4 torso_g 1.0 torso_g 0.9 pelvis_g 0.1 torso_g 0.98 pelvis_g 0.02 torso_g 0.5 pelvis_g 0.5	n/a	Integer for vertex_total N1 through N4 need to be MDL Text, node names W1 through W4 are Float	N2 - N4 and W2 - W4 are optional N1 through N4 need to be MDL Text, node names W1 through W4 are Float between 0.0 and 1.0 (Sum of line must be 1.0)	Every skinmesh must specify weights which influence its movements. Each line contains the weights for the corresponding vertex in the vertex list, as such the number of entries must match the number of vertices. • Every line consists of `(name, weight)` tuple, each defines the influence a node has over a single vertex. • A line MUST have at least one `(name, weight)` tuple. It mustn't be empty. • A line MUSTN'T have more than four `(name, weight)` tuples • The weights are floating point numbers in the range of [0.0, 1.0] • The sum of weights of must be 1.0 for each line.

Node Parameters for Effects (emitter, light, reference)

Parameter Name	Valid For	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes

Node Parameters for Axis Aligned Bounding Boxes (aabb)

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes

Animation (newanim) Parameters

Parameter Name	Definition	Example Contents	Game Default	Type	Valid Values	Description and Notes

MTR Parameters

May follow but see MTR for now.

TXI Parameters

May follow but see TXI for now.