MTM2 glTF `extras` - the round-trip contract

Status: proposed, 2026-08-13. Written by the engine side; **nothing here is agreed until Monvert's authors say so.** That is the point of the document. **Producer:** BinEdit, *File Export glTF (Blender)*. **Consumer:** any tool converting glTF back to `.bin`. **Background:** `BIN_GLTF_EXPORT.md` (why this exists and how the exporter works).

---

1. The problem this solves

glTF carries geometry, UVs, normals, textures and PBR materials. It has **no slot** for what makes an MTM2 face an MTM2 face: the face type byte, the `MRGL_MATERIAL` record, texture cycles with their rates, or the fact that two triangles used to be one quad.

So the exporter puts all of it in `extras`. Blender surfaces `extras` as **custom properties** and writes them back out, which makes this loop possible:

```
stock .bin BinEdit .glb  Blender  .glb converter .bin
```

It only closes if both ends use the same key names and the same meanings. Everything below is what BinEdit writes today. Where a decision looks wrong, say so - changing it now costs nothing and changing it once content exists costs everything.

---

2. Three rules before any key

2.1 Absent extras mean PLAIN GEOMETRY, never "assume defaults"

Blender discards every `extras` block on export unless the author ticks Include Custom Properties. Measured, not assumed: with the default settings a round trip returns material, node and scene extras as `null`.

Nothing announces the loss. The model looks right in Blender and imports fine as geometry; it only turns out plain when the `.bin` is written.

> **So a converter that meets a mesh with no `extras` must write a plain model.** It must **not** substitute defaults and record them as if the author chose them - that turns a forgotten checkbox into a file full of confident, wrong material data.

2.2 Validate, do not trust

These are editable widgets in Blender's property panel. A stray drag while scrolling changes a value - that happened within minutes of first use here, to `mtm2TextureSize`. Range-check anything you act on, and prefer the geometry over the metadata when they disagree.

2.3 Handedness and scale are the exporter's, and must be undone exactly

Source space	MTM2 model space is **Y-up, LEFT-handed**
What the exporter does	negates **Z**, negates the **normal's Z**, and **reverses triangle winding**
What a converter must do	the same three again - the transform is its own inverse
Scale	vertices are **raw engine integers, unscaled**. **1 foot = 256 units**

All three or none. Doing the mirror without the winding reversal gives a model whose faces point inward; doing neither gives a mirrored model. Both look plausible in a viewport with backface culling off, which is how the exporter shipped backwards once already.

If an author scales in Blender the vertices are no longer engine units. There is currently **no key recording an applied scale** - an open question for §7.

---

3. Node and scene extras - model level

Written identically to `nodes[0]` and `scenes[0]`. Blender reads both; it has no concept of document-root extras, so root is written for direct file readers only and does **not** survive Blender.

```
{
  "mtm2Source": "TVSW.BIN",
```

```
  "mtm2HellMagic": 65536,
  "mtm2Units": "engine integer units, unscaled",
  "mtm2Axes": "source is Y-up LEFT-handed; Z negated, normals negated, winding reversed",
  "mtm2WindingAgree": 28,
  "mtm2WindingDisagree": 0,
  "mtm2Vertices": 34,
  "mtm2Faces": 28,
  "mtm2Triangles": 56
}
```

key	type	meaning
`mtm2Source`	string	the `.bin` this came from
`mtm2HellMagic`	int	the header value; **write it back unchanged**
`mtm2Units`, `mtm2Axes`	string	human-readable notes. Informational - do not parse
`mtm2WindingAgree` / `Disagree`	int	how many source faces wound with / against their own normal. Diagnostic
`mtm2Vertices`, `mtm2Faces`, `mtm2Triangles`	int	source counts, for sanity checks

Optional, present only if the model had them:

key	type	meaning
`mtm2FaceGroups`	array of `{name, faces:[int]}`	named face selections, indices into the **source** face list
`mtm2VertexGroups`	array of `{name, verts:[int]}`	named vertex selections

---

4. Material extras - per primitive

One glTF material per distinct draw state: the tuple **(face type, texture, material record, colour)**. Faces are deliberately **not** grouped by texture alone - a flat face and a shiny face sharing art are different surfaces.

```
{
  "mtm2FaceType": "TRANS_MTM2",
  "mtm2FaceTypeId": 51,
  "mtm2Quads": 4,
  "mtm2FaceCorners": [4,4,4,4],
  "mtm2Texture": "BAN01.RAW",
  "mtm2TextureSize": 64,
  "mtm2TextureHD": false
}
```

key	type	meaning
`mtm2FaceType`	string	`NORMAL_TEX` `SHINY_TEX` `TRANS_MTM` `TRANS_MTM2` `IGNORE_TEX` `HELLBENDER` `UNKNOWN`
`mtm2FaceTypeId`	int	the raw byte - **authoritative**. The name is for humans and reads `UNKNOWN` for anything that is not a legacy `FT_` type. **Observed: a glass facet is `64` (`0x40`, `MRGL_MATFACET`) with the name `UNKNOWN`** - so key on the number, and expect a material face to arrive this way
`mtm2Quads`	int	how many source faces were quads
`mtm2FaceCorners`	array of 3/4	**§5** - the pairing itself
`mtm2Texture`	string	the name the `.bin` **records**, extension included (`FOO.RAW`)
`mtm2TextureSize`	int	source pixel size
`mtm2TextureHD`	bool	true if the source was a true-colour PNG/TGA rather than an 8-bit RAW
`mtm2Color`	`[r,g,b]` 0-255	present only for `IGNORE_TEX` - the flat face colour

Animated texture cycles - present only for an animated texture:

key	type	meaning
`mtm2AnimFrames`	array of string	frame titles **in order** - these are the real files
`mtm2AnimRate`	number	frames per second
`mtm2AnimName`	string	the cycle's label. May be a **generated placeholder** (`ANIMTEXTURE1`, `ANIMTEXTURE2` ...) when the model carries no recorded name

> **FOR AN ANIMATED TEXTURE, `mtm2Texture` IS THE CYCLE'S NAME, NOT A FILE NAME.** A converter that treats it as art to load will look for a file that does not exist. The files are the entries of `mtm2AnimFrames`. Detect the case by the presence of `mtm2AnimFrames`, not by inspecting the name.

Observed on `FILADV~1.BIN`:

```
mtm2Texture "ANIMTEXTURE1"       <- the cycle, NOT a file
mtm2AnimName "ANIMTEXTURE1"      mtm2AnimRate 6.04     faceType TRANS_MTM2 (51)
mtm2AnimFrames  TXFLAME0.RAW ... TXFLAMEF.RAW   (16 frames)
```

The embedded image for such a primitive is the **first frame only**.

Material record - present only for a face using `MRGL_MATERIAL` (glass, foliage, mesh, tint):

```
"mtm2Material": {
  "flags": ["LIT","GOURAUD","BLEND","REFLECT","FRESNEL","NOZWRITE","TINT"],
  "flagBits": 1383,
  "reflectivity": 0.5, "fresnelBias": 0.08, "fresnelStrength": 5.0,
  "baseAlpha": 0.35, "specPower": 70.0, "emissive": 0.0,
  "tint": [0.984, 0.0157, 0.0157],
  "alphaRef": 0, "translucency": 0
}
```

key	type	meaning
`flags`	array of string	`LIT GOURAUD BLEND ALPHATEST ADDITIVE REFLECT FRESNEL TWOSIDED NOZWRITE EMISSIVE TINT ALPHAREF TRANSLUCENT TEXSOLID`
`flagBits`	int	the same set as bits - **authoritative**
`reflectivity` ... `emissive`	number	16.16 fixed unpacked to real
`tint`	`[r,g,b]`	multipliers, 1.0 = unchanged. Only meaningful with `TINT` set
`alphaRef`	int 0-255	on-disk units, gated by the `ALPHAREF` flag
`translucency`	int 0-65535	on-disk units, gated by `TRANSLUCENT`

`flags` **and** `flagBits` **are both written on purpose.** The names make a material reviewable in a property panel and survive a bit being renumbered; the bits are what to act on.

ROUND-TRIP `flagBits`, NOT YOUR LIST OF NAMES. A converter that rebuilds the integer from the flags it happens to recognise will silently DROP any bit added after it was written - and bits do get added: `TEXSOLID` (`0x2000`) arrived on 2026-08-21, three flags after this document was first drafted. Losing it turns a mesh grille back into plain glass, with no error anywhere. Carry the integer through unchanged and treat `flags` as commentary, exactly as the reserved fields elsewhere in this spec are carried rather than interpreted.

`TEXSOLID` **costs a facet TWO DRAWS**, which is unlike every other flag here - they all change how a facet is *shaded*, this one changes how many times it is drawn (blend pass, then an alpha-tested opaque pass that writes depth). The record is unchanged - same 48 bytes, same stride - so a converter needs to do nothing special beyond preserving the bit. But do not set it on a material whose texture is fully opaque: the facet would render **solid** and its glass would vanish. See `MONVERT_HANDOVER.md` for the mechanism.

Observed on `BROKEN2.BIN` (a truck body carrying a glass windscreen), exported with the build this document describes:

```
mtm2FaceTypeId 64        mtm2FaceType "UNKNOWN"      alphaMode BLEND
flags   LIT, GOURAUD, BLEND, REFLECT, FRESNEL, NOZWRITE, TINT
flagBits 1383            baseAlpha 0.35    reflectivity 0.5
tint    [0.984, 0.0157, 0.0157]
```

`1383` is exactly those seven bits, so the two representations agree - which is the property a converter should assert on rather than assume.

And `test_FTree05.bin`, a foliage leaf card, giving the other half of the material system:

```
mtm2FaceTypeId 64        alphaMode MASK        doubleSided true
flags   LIT, ALPHATEST, TWOSIDED, ALPHAREF, TRANSLUCENT
flagBits 6281            alphaRef 64           translucency 32768
```

`6281` = `1 + 8 + 128 + 2048 + 4096`, again exact. Note the two mappings this one pins down, which the glass example could not: `TWOSIDED` `doubleSided: true` and `ALPHATEST` `alphaMode: MASK`.

> **The embedded image's alpha channel does NOT tell you a material is transparent.** One texture can be shared by several primitives, and the exporter keeps the alpha channel if **any** of them needs it. In `test_FTree05` the single image `MPLRED.RAW` serves both the foliage material above **and** an ordinary `NORMAL_TEX` face that is fully opaque - so the PNG is RGBA and the second material is not transparent at all. **Read transparency from `alphaMode` and the material flags, never from the image.**

---

5. `mtm2FaceCorners` - how quads survive

glTF has no quad, so each quad ships as two triangles. Recording only the *count* means an **unedited** model returns as pure triangles: the author loses something by doing nothing.

`mtm2FaceCorners` is the corner count of every source face, **in emission order**:

```
[4,4,3,4,...]     4 = the next TWO triangles were one quad
                  3 = the next ONE triangle was a triangle
```

Triangles are emitted `0-2-1` then `0-3-2` for a quad, so corners `0,1,2,3` rebuild directly.

Validation, which is mandatory:

```
sum(4  2 triangles, 3  1 triangle) == triangle count of this primitive
```

If that fails the author changed the mesh, or nudged the property. **Ignore the list and treat the primitive as triangles** - the geometry is still correct, only the quads are gone.

Verified byte-identical through BinEdit Blender Blender on `BIGFOOT.BIN` (188 quads), `ADOBE2.BIN` (300 quads, 620 triangles) and `REX2.BIN` (29 primitives).

---

6. Emitting a `.bin` - block order and material identity

Added 2026-08-13, in answer to a converter author's question. The earlier revision defined the records but never stated their ORDER, and the natural reading - header, then a texture *section*, then a material *section*, then a face *section* - **is wrong** and will produce broken models.

6.1 A `.bin` is a command stream, not sections

There are no sections. The engine walks the file start to end, and **state records apply to everything that follows them until changed** - exactly as `MRGL_TEXTURE` has always worked.

```
MRGL_MAGNIFY  (20)   optional; MUST be first if present (8 bytes: type + scale)
MRGL_VLIST    (2)    must precede any facet that indexes it
   ... then a FREE INTERLEAVING of:
MRGL_TEXTURE  (13) / MRGL_TEXTURE64 (62)    state: sets the current texture
MRGL_MATERIAL (63)                          state: sets the current material (48 bytes, fixed)
   <facets>                                 consume whatever state is current
MRGL_EOL      (0)    terminator
```

So the normal, expected shape is `TEXTURE faces TEXTURE faces MATERIAL matfacets MATERIAL matfacets 0`.

Interleaving is not merely tolerated, it is what real content does: of the two authored test models measured for this section, one carries **7** `MRGL_MATERIAL` **records interleaved among 290** `MRGL_MATFACET`**s**.

Who consumes what:

record	uses current texture	uses current material
classic textured facets (`14`, `17`, `24`, `30`, `34`, `41`, `52`, ...)		**ignored**

record	uses current texture	uses current material
MRGL_MATFACET (64)		

Only MRGL_MATFACET reads the material (core/Ground.cpp:17001). A classic facet emitted after a MRGL_MATERIAL does **not** pick it up - if a face needs material behaviour it must be a MATFACET.

MRGL_TEXTURE64 is a layout-compatible prefix of MRGL_TEXTURE (type and slot at identical offsets, engine/3D.H:104), so it is valid anywhere a texture record is, and is only needed for names longer than 15 characters. MRGL_MATFACET strides exactly like a classic textured facet: 24 + 12·n bytes (engine/Model.cpp:525) - it deliberately shares that arm.

6.2 A malformed stream TRUNCATES the model - it does not skip the bad block

Correct a common assumption: the engine does not "silently skip" a block it cannot parse. loadModel validate-walks with MRGLSizeRaw and **cuts the model off at the first bad-type or buffer-overrunning record** (engine/Model.cpp:254). Everything after that point is gone.

That is more dangerous than skipping, because the model still loads and can look *partly* right - a truncated model is exactly the failure that once cost 34 core models silently. The non-validating runtime walkers are harsher still: they abort. **A converter must therefore emit exact strides; a one-word error does not cost one face, it costs the entire remainder of the file.**

6.3 Emit ONE MRGL_MATERIAL per distinct material

Not a correctness rule - a **performance** one, and it is invisible until a track is built.

The static-instancing bake groups facets by **(texture name, cutout, material POINTER)** (core/Ground.cpp:15130):

```
const materialStruct *km = (code == MRGL_MATFACET) ? curMat : NULL;
... && km == utex[g].mat && ...
```

That is a **pointer** comparison, not a value comparison. **N byte-identical material records therefore create N separate groups**, fragmenting or entirely defeating instancing.

Measured on two versions of the same tree model:

	materials	MATFACET	classic facets	instanced?
version A	**7**	290	21 x ZFACETTMAP	**no**
version B	**1**	311	0	**yes**

Both are valid, both render - B instances and A does not. **Emit each distinct material once and order the facets to sit beneath it.** Leftover classic facets hurt too: they carry no material (km = NULL), so they form their own group and cannot express the material's behaviour at all.

6.4 Material state is per-model - no reset record needed

A converter does **not** need to emit a trailing "clear material". The engine clears the current material at the head of every model (core/Ground.cpp:16700), specifically so a model ending under a material cannot leak it into the next one drawn.

A MRGL_MATFACET that appears before any MRGL_MATERIAL in its model degrades safely to a plain lit textured facet (MRGLMAT_LIT, core/Ground.cpp:17158) - it does not inherit another model's material and does not read uninitialised state. It simply loses its material behaviour, so it is still an authoring mistake, just not a corrupting one.

---

7. Open questions for Monvert's authors

1. **Prefix.** Everything is mtm2* on the assumption that a bare extras namespace is shared with

whatever else touches the file. Is that the right prefix?

2. **Scale.** If an author scales in Blender the vertices leave engine units and nothing records it.

Options: a mtm2Scale key written by whoever applies one; a documented rule that the converter divides by the object's transform; or "don't scale" as an authoring rule.

3. **Face groups** index the **source** face list, which an edit invalidates. Is that worth keeping,

or should they become Blender vertex groups and come back by another route?

4. **Materials by value or by identity?** BinEdit already merges identical `CMaterial` records so two

faces share one. A converter re-merging on the way back would be equivalent; relying on primitive count would not.

5. **Does anything else need to survive** that is currently dropped - LOD chains, the `.bin`

texture-cycle sound? Neither is in a single model's export today.

6. **Model sequences - the RETURN trip.** BinEdit now exports one as a single animated `.glb`

(§8), so the outward direction is done and verified. Coming back is not, and it needs decisions rather than code:

- A `.glb` with **N–1 morph targets** should presumably become **N `.bin` files plus a `0x20`

control file**. Who names them, and how is the sequence name recovered?

- The weights are **baked samples**, so a converter should read the **targets**, not try to

invert the animation curve.

- An author who adds a shape key in Blender has added a frame. Is that supported, or refused?
- **Nothing currently marks a `.glb` as having come from a sequence.** If that matters, it wants

a key - say `mtm2Sequence` naming the control file and its rate - and this is the moment to agree it, before either side ships something that has to be migrated.

---

8. What is NOT in this contract

- **Keyframes, and the animation-control `.bin`.** **Two different things are called "animation"

here, and only one of them is in this contract.**

Texture cycle	inside one model, `mtm2AnimFrames` above. **Covered.**
Model sequence	a separate `.bin` whose header type is `0x20`, holding **no geometry at all** - only the list of the other `.bin` files that make up the sequence. **Not covered.**

`REX.BIN` is the second kind: 348 bytes, type `0x20`, no model data, carrying the information for `REX1`-`REX4` which are the actual models. BinEdit says so when you open it. A per-model exporter has nothing to export from such a file, and it is not a failure when nothing comes out - **do not treat a `0x20` file as a broken model.**

But this one is worth solving, and glTF already has the shape for it. A model sequence is a **deforming 3-D structure**, not a texture effect. Measured on the four REX frames:

	REX1	REX2	REX3	REX4
vertices	822	822	822	822
faces	746	746	746	746
primitives	29	29	29	29
per-primitive vertex counts	\=\=	\=\=	\=\=	\=\= identical

Same topology throughout, differing only in vertex **positions** - which is precisely glTF's **morph targets** (`primitives[].targets[]` plus an animation driving `weights`). Blender imports those as **shape keys** and plays them. So a sequence could ship as **one `.glb`** that animates, instead of four unrelated files plus a `0x20` file nothing reads. **IMPLEMENTED AND VERIFIED - BinEdit, *Bin Animator "Export to Blender…"***, or `--gltfsequence <control.bin> <out.glb>` headless.

It **reads** the control file (never loads it as a model), loads the frames it names, checks their topology matches, and writes **one `.glb`**: frame 0 as the mesh, the rest as POSITION morph targets, and the spline above baked into a weights animation. `REX.BIN` 29 primitives x 3 targets, 32 samples over a 4-second loop, confirmed animating in Blender.

So the export direction is settled and is NOT an open question. What remains open is the return trip - see §7.6.

"But BinEdit refuses to open `REX.BIN`" - correct, and it should: there is no model in it. A sequence export never loads the control file **as a model**; it **reads** it, then loads the models it names. The layout is small and already known:

```
[0] 0x20      type - animation control
[2] 4         frame count
[3] 65536     delay; rate = 65536 / delay, the same 16.16 encoding texture cycles use
              then 16-byte name slots: rex1.bin  rex2.bin  rex3.bin  rex4.bin
```

348 bytes = 4 + 344, and 344 is BinEdit's keyframe record stride (`24 + 20x16`) - so a `0x20` file is exactly one keyframe record, the same structure `BinAnimatorDlg` already edits. The four models it names all export cleanly today.

THE TWO ANIMATIONS INTERPOLATE DIFFERENTLY, AND GETTING THIS BACKWARDS IS THE EASY MISTAKE.

texture cycle	**swaps** frames - a discrete texture change, glTF `STEP`
model sequence	**blends** - a looping **Catmull-Rom** over four controls

`engine\Keyframe.cpp` is explicit: `t = 0.0; s = (1.0 - t)/2.0 s = 0.5`, with controls `i-1, i, i+1, i+2` all wrapping, and the four `car0..car3` basis coefficients applied to **vertex positions**.

That maps onto morph weights exactly, not approximately. Morph output is `Basis + Σ w·(Target − Basis)`, which is linear in the weights, and the four coefficients sum to 1 - so feeding each frame's coefficient in as its weight reproduces the engine's spline precisely. The coefficients **overshoot below 0 and above 1**; anything clamping weights to 0..1 silently flattens the motion. (Blender does clamp by default - a shape key's slider range has to be widened.)

Demonstrated end to end: `REX.BIN` its four models one `.glb` with 3 morph targets per primitive and a 32-sample weight animation over a 4-second loop, overshoot intact, plays in Blender.

- **LOD chains.** Also a file-naming convention, not in-file data.
- **Texture pixels as authored.** Textures are embedded as PNG, converted from the 8-bit RAW plus its

palette, with the engine's own alpha rule applied. That is what the game draws, not the original file. A converter should treat the embedded image as a preview and the `mtm2Texture` name as the reference to the real art.

- **Normal maps. Deliberately not in this contract, and not a gap.**

A normal map is found **by file-name convention**: for a texture named `ROCK.RAW` the engine probes `ROCK_N.PNG`, then `ROCK_N.TGA`. **Nothing in the `.bin` names it, and there is no `extras` key for it.**

That is what lets a model authored years ago gain a normal map with no re-export - the artist drops `ROCK_N.PNG` beside the diffuse. Adding an extras key here would create a second source of truth that every reader would then have to reconcile against the convention.

> A record naming the map (`MRGL_NORMALMAP`, opcode 67) was written and **reverted the same day**, 2026-08-15. Opcode 67 is free and should stay free unless that decision is revisited.

For a round trip this means: nothing to read, nothing to write, nothing to preserve. Carry the `_N` files with the rest of the art and treat them as ordinary textures. Do not let an exporter invent a material named `<stem>_N` - that suffix is reserved and such a texture would be read as another texture's normal map.

The one related record that *does* exist is `MRGL_MATERIAL2` **(66)**, which carries only a **strength** (16.16) and a flag - never a filename. It is optional; absent means 1.0. See `MONVERT_HANDOVER.md` §8.

The `_N` ALPHA channel is a per-texel SPECULAR MASK (2026-08-23): 0 = full highlight, 255 = none, and only Gouraud/shiny faces are affected. It is likewise **nothing a tool writes** - it is painted in the image editor, and the shiny-vs-flat selectivity comes from the FACE TYPE, not from a material flag. Do not add a "specular map" extras key or a `MRGLMAT2` bit for it: legacy faces carry no material record at all, so a flag would have excluded exactly the art it targets. A source with **no alpha channel** must stay distinct from one painted white - the engine forces alpha to 0 for 1- and 3-channel sources so legacy maps keep their highlight. An exporter that re-saves an RGB `_N` as RGBA with alpha=255 would silently flatten it.