New `.BIN` records - for Monvert and any other model exporter

For: RuDeE, Jpez1432, and anyone else writing `.BIN` files. **From:** the MTM2 engine work, 2026-08-12.

Monvert already writes good models. This describes the records the engine has gained **since** Monvert was written, so an exporter can produce content that uses the new features instead of routing through BinEdit for them.

Nothing here is required. **A `.bin` that emits none of this keeps working exactly as it does today** - every new record is opt-in, and no existing model contains one.

> **These are beta formats.** They are in active use and settled enough to build against, but if testing turns something up, a record may change and models may need re-exporting. Everyone is testing right now and that is understood - this is published early so you are not building against a stale engine, not because it is frozen.

---

1. What Monvert emits today

For reference, so the gaps below are clear. Monvert writes three face types:

value	opcode	
`0x18`	`MRGL_ZFACETTMAP` (24)	standard textured
`0x19`	`MRGL_ZFACET` (25)	untextured flat
`0x0E`	`MRGL_FACETTMAP` (14)	textured, unsorted

Everything else has to be set in BinEdit. The two records below are the ones worth adding, because they unlock features that **cannot** be reached from BinEdit's face-type list at all.

---

2. `MRGL_TEXTURE64` (62) - texture names longer than 15 characters

The original texture record has a 16-byte name field, so a texture name is capped at 15 characters plus the terminator. The engine now also accepts a 64-byte form.

```
typedef struct {
    int  type;        // 62
    int  slot;        // as MRGL_TEXTURE - leave 0, the engine fills it
    char name[64];    // NUL-terminated, up to 63 chars
} texture64Struct;
```

`type` and `slot` sit at the **same offsets** as the classic `MRGL_TEXTURE` (13) record, which is what lets the engine walk either with one pointer.

The rule that matters

Emit `MRGL_TEXTURE64` only when the name actually needs it. A name of 15 characters or fewer must still emit the classic 16-byte `MRGL_TEXTURE` record, byte-identical to what you write now.

That is not a style preference - it is what keeps short-named models loading in **older engines and in BinEdit**. A model that gratuitously uses the long record stops working for anyone who has not updated. BinEdit follows exactly this rule.

Name budgets

Long names are limited by the **POD directory**, not by this record. The budget is per folder, because the folder prefix shares the field:

folder	max, including extension
`ART\`	23
`DATA\`	22
`WORLD\`	21
`MODELS\`	20

Models referenced by a keyframe animation stay at 15 characters total, permanently - that record's size is fixed in the file format.

3. `MRGL_MATERIAL` (63) + `MRGL_MATFACET` (64) - glass, foliage, and what comes next

This is the interesting one. Historically a facet's appearance came from **which opcode** it was, and the opcodes are fixed *combinations* - which is why a windshield could be transparent **or** shiny but never both. That combination simply had no opcode.

Materials make appearance **data** instead:

- `MRGL_MATERIAL` is a *state change*, exactly like `MRGL_TEXTURE`. It sets the current material.
- `MRGL_MATFACET` is a textured facet that uses the current material. It is **deliberately

identical in layout to a normal textured facet** (`type`, `n`, `a,b,c,d`, then 3 verts of 3 ints), so your existing facet-writing code works unchanged - only the opcode differs.

Facets inherit the current material, so they batch the same way texture selection does.

The record - 48 bytes, fixed

```
typedef struct {
    int  type;              // 63
    int  flags;             // MRGLMAT_* bitfield, see below
    int  reflectivity;      // 16.16, 0..1  - environment reflection weight
    int  fresnelBias;       // 16.16        - reflection/alpha at face-on
    int  fresnelStrength;   // 16.16        - how hard it ramps to edge-on
    int  baseAlpha;         // 16.16, 0..1  - alpha before Fresnel
    int  specPower;         // 16.16        - specular exponent
    int  emissive;          // 16.16, 0..1  - self-illumination
    int  tintR, tintG, tintB; // 16.16, 1.0 = unchanged. Only read when MRGLMAT_TINT
    int  foliage;           // packed, see below. Only read when its flags are set
} materialStruct;
```

All params are **16.16 fixed point** (`1.0` = `65536`), matching the rest of the format.

The record is full at 48 bytes. Do not extend it - the size *is* the stride the engine walks by. A future property gets a new record type instead.

Flags

flag	value	meaning
`MRGLMAT_LIT`	`0x0001`	takes scene lighting at all
`MRGLMAT_GOURAUD`	`0x0002`	smooth-shaded; **also the gate for specular and reflection**
`MRGLMAT_BLEND`	`0x0004`	alpha blended
`MRGLMAT_ALPHATEST`	`0x0008`	cutout, clipped at 0.5
`MRGLMAT_ADDITIVE`	`0x0010`	additive instead of src-alpha
`MRGLMAT_REFLECT`	`0x0020`	environment reflection
`MRGLMAT_FRESNEL`	`0x0040`	view angle drives alpha **and** reflection weight
`MRGLMAT_TWOSIDED`	`0x0080`	no backface cull
`MRGLMAT_NOZWRITE`	`0x0100`	no depth write (translucency ordering)
`MRGLMAT_EMISSIVE`	`0x0200`	self-illumination term
`MRGLMAT_TINT`	`0x0400`	multiply texture by tintR/G/B
`MRGLMAT_ALPHAREF`	`0x0800`	use this material's alphaRef
`MRGLMAT_TRANSLUCENT`	`0x1000`	light bleeds **through** the surface (backlit leaves)
`MRGLMAT_TEXSOLID`	`0x2000`	texture alpha marks the **solid** areas - see below

`MRGLMAT_TEXSOLID` *(2026-08-21) - the one flag that changes how many draws a facet takes*

Every other flag above changes how a facet is *shaded*. This one makes the engine draw it **twice**: the normal blend pass, then an alpha-tested **opaque** pass over the top that keeps only the texels the alpha channel calls opaque and writes depth

for them. The result is solid where the texture is opaque and glass where it is not - a mesh grille or window guard, from one face and one texture.

For a converter this is still just a bit, and nothing about the record changes: same 48 bytes, same stride, same field order. But two things follow that a tool must not get wrong:

- **Never set it on a material whose texture is fully opaque.** Every `.RAW` with no black in it,

and every PNG without an alpha channel, is fully opaque - and this flag would make the whole facet **solid**, destroying the glass. It is meaningless without `MRGLMAT_BLEND` for the same reason.

- **Preserve it through a round trip.** A converter that rebuilds `flags` from a fixed list it

knows about will silently drop this bit and turn a mesh back into plain glass. Round-trip the integer, not your interpretation of it - which is the same rule the reserved fields already state.

Preset: `MRGLMAT_MESH` = `MRGLMAT_GLASS | MRGLMAT_TEXSOLID` = `0x2167`.

The packed `foliage` int:

```
bits  0..15   alphaRef     0..255   - read only when MRGLMAT_ALPHAREF
bits 16..31   translucency 0..65535 -> 0..1, read only when MRGLMAT_TRANSLUCENT
```

Every field is gated on its own flag, and that is deliberate. A record written before a field existed has zeros there, so the flag makes those zeros mean *"not used"* rather than *"black"* or *"fully transparent"*. Set the flag whenever you write the field.

The two presets that exist today

```
GLASS   = LIT | GOURAUD | BLEND | REFLECT | FRESNEL | NOZWRITE     // 0x01E7
FOLIAGE = LIT | ALPHATEST | TWOSIDED | TRANSLUCENT                 // 0x1089
```

Both are confirmed working in game on DX11 and Vulkan. **Glass is what a windshield wants**; foliage is for leaf cards, fronds and grass - `TWOSIDED` is the one authors notice, because a leaf card no longer vanishes when seen from behind.

`MRGLMAT_ALPHAREF` is deliberately **not** in the foliage preset. It works, but it is inert on colour-keyed `.RAW` art: transparency there is a colour key, so alpha is only ever 0 or 255 and no threshold between them moves a pixel. It is worth setting only for genuinely soft-alpha PNG art.

Presenting this to an artist

Please expose these as **presets** ("Glass", "Foliage"), not as thirteen checkboxes, and not as a "material type" dropdown. A type list would recreate exactly the combination explosion that materials exist to remove - the next material should cost one flag row, not a new enumeration.

---

4. Textures can be PNG or TGA now, and the recorded name is just a stem

This is probably the most useful thing here, and it changes the art workflow rather than the format.

The engine loads `.PNG` and `.TGA` directly. Same stem, new extension: `ART\MYSKIN.RAW` `ART\MYSKIN.PNG`. Square, power of two, **32 to 1024** pixels. Alpha is real alpha - no colour key. So an author no longer has to flatten everything through Triraw to 256-colour `.RAW`.

The recorded name is a STEM plus an ignorable hint - it no longer has to say `.RAW`.

```
model says:   MYSKIN.RAW  ·  MYSKIN.PNG  ·  MYSKIN.TGA  ·  MYSKIN
all resolve:  ART\MYSKIN.PNG    ART\MYSKIN.TGA    ART\MYSKIN.RAW
```

Changed 2026-08-12 (engine `80a3e68`, Traxx `2d87772`, BinEdit `43679f9`) after exactly the objection you would have raised: *"it makes no sense to name a texture that won't actually be there."* The PNG/TGA probe had always been stem-based; only the legacy `.RAW` open still took the name literally. Now all four tools resolve the same way, whatever the record says.

So **write whatever extension is natural** - your material name plus `.RAW`, plus `.PNG`, or no extension at all. Current tools do not care.

One reason remains to prefer `.RAW`, and only one: an **older engine or an older BinEdit** still reads the name literally. If you want a model to open on a 1998 install or on a build from before today, `.RAW` is the safe spelling. If you have already decided to ship PNG-only art - which is a legitimate choice - that compatibility is gone anyway and the point is moot.

A pod may ship the PNG **and** the `.RAW`/`.ACT` (works everywhere), or the PNG alone (smaller - often dramatically, the fallback is typically 3-11x the size of the HD source - but then the track or truck is invisible to a 1998 install rather than broken, which is deliberate).

Full detail for artists is in `core/AUTHORING_HD_ART.md`.

---

5. The texture-name glitch - reproduced, then fixed on our side

The readme describes textures that BinEdit accepts but Traxx and the game reject, worked around by replacing each texture in BinEdit and replacing it back.

Reproduced here on 2026-08-12, round-tripping a stock `DODGERAM.BIN` through Monvert v3 on Blender 5.2. The exported `.bin` contains the texture names **without their extension**:

	original	Monvert re-export
texture names	`RAM1.RAW`, `BLACK.RAW`, `SIENG.RAW` …	`RAM1`, `BLACK`, `SIENG` …

The readme says the name is written "with .raw extension" - in this run it was not.

Precisely when this bites (checked in the engine's loader, not assumed):

recorded name	HD art (`.PNG`/`.TGA`) present	only legacy `.RAW` present
`MYSKIN.RAW`	resolves	resolves
`MYSKIN` (no extension)	**resolves**	**fails**

The PNG/TGA probe is **stem-based** - it replaces whatever follows the last dot, and appends one if there is none - so a bare name finds HD art perfectly well. The **legacy `.RAW` load uses the recorded name verbatim**, so a bare name has no extension to open.

That matches the symptom exactly: it fails on RAW-only art, which is most existing content and certainly most test cases. BinEdit does not care because it matches on the stem, and the workaround works because BinEdit rewrites the name with its extension when you replace a texture.

So this was **not** "the game is stricter than BinEdit" in general - it was one specific path.

Fixed on our side - you need not change anything

Rather than ask you to append `.RAW`, we removed the asymmetry: **the legacy open is stem-based too** now, in the engine (`80a3e68`), Traxx (`2d87772`) and BinEdit (`43679f9`). A bare name resolves against RAW-only art from the 2026-08-12 build onward, so your current output works as-is.

Older builds still have the old behaviour, so if you want models that open on anything earlier, appending `.RAW` remains the safe spelling - and it costs nothing. Your call, not a requirement.

Three name rules that still matter

1. **NUL-terminate within the field** and zero the remainder. A name written to the full width with

no terminator reads as garbage downstream - genuine corruption, not a lookup miss.

2. **Keep the name <= 15 characters** unless you are deliberately emitting `MRGL_TEXTURE64` (§2).
3. **Upper-case is reliable.** Mixed case has caused pod-lookup trouble before.

---

6. Two other findings from the same round-trip

Both from importing and re-exporting a stock `DODGERAM.BIN` (12 objects, 1705 verts, 2054 faces, 12 materials - the import itself was clean and named every material correctly).

The addon does not load its own preferences unless the folder is named `mtm2_bin_prefs`

Export fails outright with:

```
KeyError: 'bpy_prop_collection[key]: key "mtm2_bin_prefs" not found'
```

`MTM2BinPreferences.bl_idname` is `"mtm2_bin_prefs"` (line 40) and `_prefs()` looks up `bpy.context.preferences.addons["mtm2_bin_prefs"]` (line 61) - but that collection is keyed by the addon's **module name**, which is `monvert` as the zip installs. Renaming the installed folder to `mtm2_bin_prefs` makes export work immediately; that is how this was confirmed rather than guessed.

One-line fix: `bl_idname = __name__`, and read the prefs back with the same value.

Reproduced on **Blender 5.2**. If it does not happen for you, you are likely installed under a folder name that happens to match, or on a Blender version that resolved it differently - worth fixing regardless, since it depends on install layout rather than on anything the user does.

Unexplained header differences - not diagnosed, just reported

The first ints of the two files differ in ways that need a proper record walk to interpret, so this is an observation and **not** a claim that anything is wrong:

int	original	re-export
`[0]`	`20` (`MRGL_MAGNIFY`)	`20`
`[1]`	`65536`	`0`
`[2]`	`2`	`0`
`[4]`	`1675` (vertex count)	`1705`

`magnifyStruct` is `{type, power}`, so `[1]` is the magnify power. The vertex growth (1675 1705) is expected - UV seams split vertices - but the other two are worth checking from your side, since you know what the exporter intends to write there.

---

7. What NOT to change

Do not change what an existing record means. Around 200 shipped PODs depend on the current derivations, and the engine deliberately keeps reading them exactly as it always has. New capability goes in new records - which is why materials are a new opcode rather than new flags on an old facet type.

Do not widen the classic 16-byte texture record. It is the on-disk layout of every existing model. `MRGL_TEXTURE64` exists precisely so it does not have to move.

---

8. Reference

The authoritative numbers live in the engine source and in these files, which ship with it:

file	holds
`engine/3D.H`	every opcode, every struct, every flag - the live values
`core/ENGINE_LIMITS.md`	content caps the tools should enforce
`core/TOOL_CONTRACT.md`	the legacy-frozen / new-declares rule
`core/AUTHORING_HD_ART.md`	the artist-facing guide (HD textures, materials, lights)

If something here disagrees with `engine/3D.H`, **3D.H is right** - say so and it will be fixed.

---

8. `MRGL_MATERIAL2` (66) - the second material record

Added 2026-08-13. `MRGL_MATERIAL` (63) ran out of room, and the rule in this codebase is that a record's size is fixed forever once anything has written one - so new properties get a **second record**, never a wider first one.

32 bytes, fixed:

offset	field	meaning
0	`type`	66
4	`flags2`	`MRGLMAT2_NORMALMAP` = `0x0001`
8	`normalStrength`	16.16 fixed point. `0x00010000` (= 1.0) is "as authored"; 0 is flat

offset	field	meaning
12	`reserved[5]`	**zero in every record ever written**

It is a **state change**, like `MRGL_MATERIAL` - it selects state that following facets inherit.

> `reserved` **must be written as zeros.** A future field will mean "absent" by being zero, so a record with junk there becomes indistinguishable from one that meant something. Same rule as `MRGL_MATERIAL`'s own reserved tail.

You do not have to emit this record. It only *tunes* a normal map; a model with no `MRGL_MATERIAL2` behaves as though strength were 1.0. Emit it only if you want a per-material strength other than "as authored".

---

9. Normal maps are a FILE CONVENTION - there is no record to emit

This is the part most likely to be got wrong, because the obvious design is the wrong one.

A normal map is found by name. For a texture the model already names - say `ROCK.RAW` - the engine probes `ROCK_N.PNG`, then `ROCK_N.TGA`. That is the whole mechanism.

- **Nothing in the `.bin` names the map.** No opcode, no string, no flag is required.
- Therefore **a model authored years ago gains a normal map with no re-export at all** - the artist

drops `ROCK_N.PNG` beside `ROCK.RAW` and it works. That property is deliberate and is the reason the convention was chosen.

- The suffix is defined once in the engine (`ART_NORMAL_SUFFIX`, `engine\TEXLOAD.H`) and every tool

derives it the same way.

> **DO NOT invent a record that names the map.** One was written and reverted the same day (`MRGL_NORMALMAP`, opcode 67, 2026-08-15). It worked, but a single fixed convention is what lets the tools agree without negotiating: with a free-form name, every tool must read the record, agree precedence against the convention, and stay in step with the others forever. **Opcode 67 is free and should stay free unless this decision is revisited deliberately.**

What this means for an exporter

1. **Emit nothing.** Keep naming textures exactly as you do now.
2. **Do not rename a texture to `<stem>_N`.** That suffix is reserved; a texture called `FOO_N` would

be read as the normal map of `FOO`.

3. **If you bundle art with a model, include the `_N` files.** They are ordinary PNG/TGA files and

travel like any other texture.

4. Optionally emit `MRGL_MATERIAL2` (§8) if a material wants a strength other than 1.0.

Where it currently works

Objects, hardware-instanced props and terrain, on **DX11 and Vulkan**. DX9 is parked. Camera-facing billboards are deliberately **refused** - a tangent-space map describes detail a facing card does not have. Terrain is **HD art only**, because a map is registered per texture page and only HD art fills a page on its own.