MTM2 Engine Content Limits - authoritative bookkeeping

Updated 2026-08-09 (commit `15da90a`). **This file exists so the authoring tools - Traxx, BinEdit, C-Pod - can enforce the same numbers the engine enforces.** Traxx hard-codes these caps to stop authors from overflowing; when its source is updated, take the numbers from HERE, and keep this file current whenever an engine define changes.

Per-level content limits

what authors see	engine define	file	old	now	tool should allow
Scenery objects (each placement counts, duplicates too)	`MAX_BOX_LIST`	`core\simobj.h`	800 (eff. 541, see below)	**4608**	**4096** (8-09 raise; engine margin for mo vers/ramps/checkpoin ts, which are also boxes)
Moving objects (trains/movers)	`MAX_TRAINS`	`core\Simobj.cpp`	50	**104**	**100** (engine-usable is cap–1 = 103; the count check runs after the store)
Ramps	`MAX_RAMP_LIST`	`core\simobj.h`	10	**100**	100
Top-crush objects (crush cars)	`MAX_TOP_CRUSH_L IST`	`core\simobj.h`	10	**100**	100 - Traxx has NO separate cap (it keeps them in the 4096-entry box array), so nothing to raise there
Unique models per level	`MAX_SCENERY_MOD EL`	`core\SIMMODEL.H`	1024 (orig. 256)	**8192**	4096 unique models is plenty; define is 8192 - MUST stay **above** `MAX_BOX_LIST` (eviction-dangling rule, see the define's comment)
Model textures	`TEXTURE_LIST_SI ZE`	`core\global.h`	1024	**4096**	4096
Simulated objects total	`MAX_SIM_LIST`	`core\sim.h`	550	**4864**	(internal - vehicles + RAMPS + boxes + top-crush, NOT cylinders; worst case 10+100+4608+100 = 4818, so keep >= that)
Ground boxes	tile grid	`core\World.h` (`maxGroundX/Z` 256)	2048-class	unchanged	unchanged per user decision
Vehicles	`MAX_VEHICLE_LIS T` / `MAX_RACE_VEH ICLES`	`core\simobj.h` / `core\sim.h`	10 / 8	unchanged	8 racers
Sounds per track	Traxx `MAXNPODSOUNDS`	(engine side: KASE lists, no matching hard define found - verify at test)	256	**512**	512 - Traxx used to drop sounds SILENTLY when full; now reports 'Max. 512 Sounds!' (fixed 8-09, Traxx `f8b4c97`)

what authors see	engine define	file	old	now	tool should allow
Lights (BOX_LIGHT) **IMPLEMENTED 8-11** (23e49c6)	MAX_LIGHT_PROPS	core\Simobj.cpp	-	**128**	**100**. Overflow is **non-fatal** - surplus lights are dropped and noted, never GTFO. Separately, only MAX_HEADLIGHT (128) lights can *illuminate* at once across the whole scene, shared with truck headlights; past that the engine keeps the **nearest** ones. DX9 policy (user, 8-09): render what fits, note the shortfall, keep the track playable.

The famous "541 scenery objects" was never a define - it was MAX_SIM_LIST(550) – 8 trucks – 1, an emergent cap. It no longer binds; MAX_BOX_LIST is the real scenery ceiling now.

DONE 8-11 (23e49c6) - two new box types the tools must offer: LIGHT and MOVING

BOX_TRAIN was doing three unrelated jobs because it was the only type that moved: movement, a train sound + horn, and a headlight. Now split three ways:

value	name	authoring label	behaviour
10	BOX_TRAIN	Train	unchanged - moves, train sound + horn, headlight
12	BOX_LIGHT	**Light**	a light. **Static.** Authored radius, height, colour, glow/cone
13	BOX_MOVING	**Moving**	movement only - no horn, no train sound, no light, **no** CONT_SOUND_MAX **slot**

The movers cap (100) covers TRAIN + MOVING combined. BOX_LIGHT does **not** consume it - lights have their own list and never enter the train integrator.

The 'L' block - what a writer emits for a BOX_LIGHT

Optional block in the .sit box record, using the same peek-a-character pattern as the existing '!' (type/flags), 'p' (priority) and '@' (sounds) blocks. **Emit it only for** boxType == 12 - that is what keeps a legacy track byte-identical when loaded and re-saved. Two lines, immediately after the '@' sound block:

```
Llight rad,hgt,r,g,b,glow,glowRad,coneLen,coneRim,coneBase,bright,aimOff
64.000000,8.000000,1.000000,1.000000,1.000000,0,6.000000,125.000000,10.000000,3.500000,1.000000,0.000000
```

#	field	type	default	meaning
1	rad	float	64.0	**The pool radius ON THE GROUND**, feet. Mount height does NOT eat into it (see below). **Max 4095** (clamped; the packing needs it)

#	field	type	default	meaning
2	`hgt`	float	8.0	Height offset above **the box's own origin**, feet - how far up the pole the lamp head sits, NOT how high it is above the world. **Honoured exactly**, not snapped to the ground. Changes neither the pool size nor its strength, and does **not** set the vertical extent (that is measured - see below)
3-5	`r,g,b`	float	1,1,1	Colour, 0..1
6	`glow`	int	**0**	0 = no visible source, 1 = glow sprite, 2 = glow + cone
7	`glowRad`	float	6.0	Radius of the visible **glow sprite**, feet. Deliberately separate from `rad`: the lit pool is tens of feet across, the bright spot is a few
8	`coneLen`	float	125.0	Cone fade-out distance, feet (only read when `glow` = 2)
9	`coneRim`	float	10.0	Cone radius at the far end, feet
10	`coneBase`	float	3.5	Cone radius at the source, feet
11	`bright`	float	**1.0**	How **strongly** it lights the ground. **Range 0.25 … 3.75**, 0.25 steps; 1.0 = a truck headlight, so the scale reaches **down to a quarter of one**. See below
12	`aimOff`	float	**0.0**	How far **along the object's facing direction** the lit pool lands, feet. 0 = directly beneath the lamp, which is what every light did before this field. See below

The header line must stay **under 80 characters** - the engine consumes it with `fgets(text,80,f)`. The one above is **72**, so there are **7 characters left**. Count any future field name; do not eyeball it. (The *data* line has its own 256-char buffer and is not the constraint.)

A `BOX_LIGHT` written with **no** `'L'` block is valid and gets the defaults above.

THE FOUR KNOBS, AND WHAT EACH ONE ACTUALLY DOES

They are deliberately **orthogonal** - none of them secretly consumes another. This was not true before 2026-08-11 and the old behaviour is what made the field names misleading.

want	change	do NOT change
the lit area **WIDER**	`rad`	-
the ground **BRIGHTER** *or* **DIMMER**	`bright` (0.25 … 3.75; **below 1.0 is where the useful range is**)	`rad` does not brighten; it only moves the edge
the lamp head **HIGHER**	`hgt`	it costs you nothing - the pool keeps its size **and** its strength at any height, and the vertical reach grows automatically because it is measured
the pool **somewhere other than under the lamp**	`aimOff`	it moves the pool, it does not resize or dim it

want	change	do NOT change
a **visible bulb / beam**	`glow` (1 or 2)	separate system; `bright` does not scale it

`hgt` **used to cost you dearly and no longer does.** Falloff was measured in 3D, so a light mounted 60 ft up with `rad` 64 reached the ground at **6% strength across a 22 ft pool**. `rad` is now measured **horizontally**, so the same light gives a full 64 ft pool at full strength - whether the lamp head is 10 ft up or 200 ft up.

The vertical extent - measured, not authored

A horizontal falloff on its own is an infinite column, so the light is bounded in the vertical. **The bound is derived from how high the light actually is above the terrain beneath it** - the engine measures this each frame. It is *not* `hgt`, and there is no field you set:

```
H = the light's real height above the ground below it        (measured, feet)
extent = max(2 x H, rad)     full strength out to ¾ of it, then linear to zero at the extent
```

the light is…	extent	full strength holds to	so…
8 ft up (bollard on the ground)	64 ft	48 ft	lights its own pool and **cannot** wash a cliff above it
60 ft up	120 ft	90 ft	pool fully lit, 30 ft to spare for uneven ground
200 ft up	400 ft	300 ft	pool fully lit, **100 ft** to spare
200 ft up, `rad` 16	400 ft	300 ft	a tight spot from high up works - the extent follows height, not `rad`

`hgt` **is NOT this height.** `hgt` is the light's offset above **the box's own origin** - 8 ft up a lamppost. For a lamppost standing on the ground the two are the same, which is why using `hgt` looked right. **They are not the same for a light riding a prop at altitude:** a light on a model floating 150 ft up still has `hgt` 8. Deriving the extent from that gave 64 ft over a 158 ft drop and **the ground pool vanished entirely** - measured in game, 2026-08-11.

There is no height at which the ground goes dim, because the reach grows with the height. It bites only in two places, both of which take deliberate effort to reach:

- terrain inside the pool sitting more than **½ x H** *below* the light starts to fade, and is gone

at H below it;

- above the light, anything higher than the same extent is dark - which is the point. A lamp at the

bottom of a canyon is low above its own ground, so it has a short reach and leaves the clifftop alone.

The floor at `rad` means a light sitting on the ground behaves exactly as it did before any of this: the extent can only grow, never shrink.

Consequence for existing content: a light authored before this that sat high up will now be **wider and brighter**. That is the fix working, not a regression.

`bright` *- what it actually does, and its quantisation*

The engine sums `65535*(rad-r)/rad` per light, divides by 16 and **clamps** the result, so a single light **already saturates at its own centre**. `bright` therefore does **not** raise the peak - there is no headroom. It **widens the saturated core and lifts the falloff**: at `2.0` the light holds full output out to *half* its radius instead of only at the exact centre.

So `bright` and `rad` are complements, not duplicates - `rad` **moves the edge of the pool,** `bright` **fills the middle.** It affects the **ground/prop illumination only**; the glow sprite and the cone are sized by `glowRad` / `coneRim` and are unchanged by it.

Quantised to 0.25 steps the usable values are 0.25, 0.50, 0.75 … 3.75. Anything below 0.25 clamps to 0.25 and anything above 3.75 clamps to 3.75; in between, the engine rounds to the nearest step. A tool may present a free-text float - it will simply snap.

The scale reaches BELOW a truck headlight, and that is the point of it (2026-08-12)

It used to be **1.00 … 4.75**, i.e. the *dimmest* light an author could ask for was a full truck headlight. Measured in game on a moonlit track at `rad` 64: `bright` 1.0 and 4.0 were reported as *"both almost full bright"*, and *"it never goes low enough."* The arithmetic says exactly why - at the darkest night ambient the clamp leaves only 1792 of headroom while one light at 1.0

puts 4095 into the centre, so the pool is **already saturated out to 36 of its 64 feet** before the author turns anything up. Everything above ~1.0 was spending steps on a range the clamp had flattened.

The same 15 steps were therefore slid down. **The ceiling was not raised** - it would buy nothing.

`bright`	**pool centre (night ambient 2304, `rad` 64)**	**saturated core**
0.25	0.81 of full, against a 0.56 background	**none - a clean gradient from centre to rim**
0.50	full	8 ft of 64
1.00	full	36 ft of 64
3.00	full	55 ft of 64

No existing authored value changes. `1.0` mapped to the old level 0 (multiplier `(4+0)/4`) and now maps to level 4 (multiplier `4/4`) - both exactly 1. Every 0.25 step from **1.00 to 3.75 lands on the identical multiplier it did before**; the remap is a pure extension downward. Only values above 3.75 move, and they clamp to 3.75, which at any night ambient is indistinguishable from 4.75 because both flatten the pool well past half its radius.

> Why the cap and the steps: the value rides to the GPU packed into the spare high range of the light's radius word (`w = rad + level*4096 + heightQ*65536`), which is also why `rad` **is clamped to 4095** and why `hgt` is quantised to 4 ft steps up to 1020 ft. All three fields have to fit inside 2^24, the last integer a float carries exactly - and with the height tier they now fill it exactly. **The word is full**: anything further (coloured pools, for one) needs its own component on the cbuffer-backed renderers rather than another tier here. **Packing level 0 is RESERVED and an authored light never gets it.** It means exactly 1.0x and belongs to the engine's own lights - truck headlights, the near fill light, the train engine and every legacy track, none of which carry a `bright` at all. Their packed word stays bit-for-bit the plain radius. A tool must therefore never write a `bright` it expects to land on level 0; the engine clamps the authored range to levels 1..15 for exactly this reason.

`aimOff` *- putting the pool where the beam points*

Every light used to pool directly at its own feet, however the cone above it was aimed. `aimOff` throws the lit pool **along the object's facing direction**, so an angled lamp lights the ground it is actually pointing at:

```
poolX = ipos.x + aimOff * dirmatrix.m3        poolZ = ipos.z + aimOff * dirmatrix.m9
```

That is the **same forward vector the cone is drawn with** and the same one Traxx's direction arrow points along, so rotating the prop turns the pool, the cone and the arrow together. They cannot disagree - which is the point of deriving it rather than giving it its own angle.

Two properties that look like oversights and are deliberate. **A tool must mirror this formula exactly rather than re-deriving it**, or its preview will lie:

- **The vertical component (`m6`) is not used.** The shift is purely horizontal; the lamp head stays

on the model. Folding it in would drag the source down through its own post when you tilt it, and `hgt` is decoupled from the pool on purpose.

- **The horizontal part is not normalised.** Tilt the prop toward vertical and `|(m3,m9)|` shrinks,

so a steeply-aimed lamp throws its pool proportionally closer and one aimed straight down keeps it underneath - the offset fades out exactly as the geometry says. An **untilted** prop has a unit horizontal forward, so the ordinary case is simply *"`aimOff` feet along the arrow"*.

It costs the renderers **nothing**: the shift is applied to the position handed to the light list, so the shaders see a light standing somewhere else. No new component, no fourth packing tier - which is what made it possible at all, the radius word being full (above).

A lamp with a non-zero `aimOff` stops lighting its own post, because the post is no longer inside the pool. That is intended - a light shines away from its source - but it is visible, so choose it deliberately.

The **vertical extent still measures from the lamp head**, not from the displaced pool. The light still comes from the lamp, which is the right way round - but `hgt` is how high the lamp sits above **its own** ground, and a long offset can land the pool on ground at a quite different height. On a slope the drop from the lamp head to the *lit* ground can therefore exceed `hgt` by a lot.

Worked example: `hgt` 20, `rad` 64 extent `max(40, 64)` = **64 ft**. Throw the pool into a dip 60 ft below the post's own ground and the lamp head is 80 ft above what it is lighting - past the extent, so the pool fades out or vanishes.

The fix is `hgt`, and it is free. It buys extent at 2 ft per foot and costs nothing in pool size or strength, so a lamp throwing a long offset over uneven ground should simply be mounted higher - `hgt` 50 gives 100 ft of extent and the example above lights normally. If a pool looks dimmer than its `bright` says it should be, this is the first thing to check.

Parser rule for writers (added with `bright`)

The engine now reads this block **line-at-a-time and tolerantly**: it accepts **fewer** fields than listed (missing trailing fields keep their defaults) and ignores surplus ones. That means a tool emitting only the first 10 fields still loads correctly on a build that knows 12.

This is why `bright` and `aimOff` could both be appended without a format break, and it is the mechanism every future field of this block should rely on: **append at the end, never insert.**

It did **not** used to. Before this, the reader was a single `fscanf` with a fixed 10-field format string, and `.SIT` is parsed **positionally** - emitting an 11th value to a 10-field reader would have left it in the stream and shifted **every field of every record after it**, silently corrupting the track. If you are writing a `.SIT` for an *older* engine, emit exactly 10 fields.

Any track using either new type is `formatVersion=2` - see `TRACK_VERSIONING.md`.

Texture caches (engine-side pools, not authoring caps)

cache	define	file	value
32x32 slots	`TEXTURE_SIZE_32`	`core\global.h`	256 (was 64)
64x64 slots	`TEXTURE_SIZE_64`	`core\global.h`	2048
256x256 slots	`TEXTURE_SIZE_256`	`core\global.h`	1024 (was 512; hard ceiling `MAX_CACHE_SIZE` 2048)
512/1024 HD	byte-budgeted LRU arena (pipeline stage 2)	-	no slot cap by design

Canary: a `CACHE WRAP` line in `texcache.txt` next to the exe means a working set outgrew its pool.

Renderer-side: the clean-mip terrain BLOCK cache (all three DLLs)

cache	define	file	value
clean-mip terrain blocks	`TERRCLEAN_MAX`	`wincore\RENDD3D11.CPP`, `RENDVK.CPP`, `RENDD3D9.CPP`	**1024** (was 256)
…its byte ceiling	`TERRCLEAN_BYTES`	same three	192 MB

Each terrain block gets its OWN texture with a clean mip chain, because mipping the shared atlas page averages in neighbouring tiles - that bleed IS the distance banding. **One entry per track texture**: a track whose Traxx "Track Statistics" reports 303 textures produces a measured peak of exactly 303.

`TERRCLEAN_MAX` **must be >= the engine's texture cap (1024).** At 256 it failed at a quarter of what a legal track may contain, and the failure is SILENT - blocks just fall back to the bleeding page. A renderer cache must never give up before the documented content limit.

Raising it means raising the BACKING POOL too. Vulkan's `g_terrCleanPool` carried three hardcoded 256s (`maxSets` + both `VkDescriptorPoolSize`s); with the array at 1024 the cache still stopped dead at 256 because `_allocImgSet` ran dry. All three now derive from `TERRCLEAN_MAX`. When raising any cache bound here, **grep for the old number** - pools are sized beside the array.

Canary: `[Graphics] hdLog=1` makes each DLL append `(terrClean) peak N of 1024 entries | X KB` per track to `hdtex.txt`, plus a `FULL`/`EXHAUSTED` line if it ever runs out. Cost is ~21 KB per 64x64 block, ~341 KB per full-page 256x256 one - measured peak on the heaviest known track is 6.3 MB.

MRGL opcodes - the live ceiling, and one number that must stay free

`MAX_3D_PRIM` (`engine\3D.H`) bounds the `flist[]` dispatch table in `3d.cpp`. **The two move together or a valid opcode indexes off the end of the table.**

opcode	record	added
62	`MRGL_TEXTURE64` - `name[64]`, 72 bytes	2026-08-09
63	`MRGL_MATERIAL` - 48 bytes	2026-08-09
64	`MRGL_MATFACET` - facet-shaped, 3 ints/vertex	2026-08-09
65	`MRGL_KEYFRAME64` - `modelFile[64][64]`, stride 4376	2026-08-11
66	`MRGL_MATERIAL2` - 32 bytes: `flags2`, `normalStrength` (16.16), `reserved[5]`	2026-08-13
67	- **FREE. Keep it free.**	

`MAX_3D_PRIM` **is currently 67.**

67 was briefly `MRGL_NORMALMAP` **and was reverted the same day (2026-08-15).** Nothing ever wrote one, so the number is clean. Normal maps are found by the `<diffuse>_N` file convention instead - a single fixed convention is what lets the engine and all three tools agree without negotiating. **Do not re-mint 67 for this without revisiting that decision deliberately.**

The rules that apply to every new record here

- **A record's size is FIXED FOREVER once anything has written one.** Its `sizeof` is the on-disk

stride, asserted at compile time. New capability = a **new opcode**, never a wider old record - `MRGL_TEXTURE64` and `MRGL_MATERIAL2` both exist for exactly this reason.

- `reserved[]` **must be written as ZERO.** A later field will mean "absent" by being zero, so junk

there is indistinguishable from meaning.

- **Every walker needs an arm, and the arms are MANDATORY**: `MRGLSize` / `MRGLSizeRaw`

(`engine\Model.cpp`), the byte-swap switch - **name-bearing records go with the NAME arm**, since swapping the whole record scrambles the characters - and `flist[]`. A missing stride does not degrade; the walk desyncs into the middle of the next record.

- **Traxx has its OWN transcription** of the stride table (`TrackPOD\TrackPODModel.cpp`,

`mrglSkipInts`). It reports an unknown opcode and **stops**, so a model carrying an untranscribed record loses every polygon after it with no reason shown. `MRGL_MATERIAL2` sat untranscribed there from 8-13 until 8-15 - the gap was live for any material model BinEdit wrote. **When the engine gains an opcode, that table must follow in the same change.**

Truck DISPLAY name - 31 characters, and it used to corrupt the file at 30

`vehicle::truckName` is `char[32]` (`core\sim.h`), so **31 characters + terminator**. This is the name on line 2 of a `.TRK`, not a filename - the section below is about filenames and does not cover it.

TOOL CONTRACT: BinEdit / Traxx should cap the truck name field at **31 characters**. Longer names are now truncated cleanly by the engine, but the author never sees what the game will show.

Fixed 2026-08-25. The loader did `fgets(v->truckName,31,f)` then `truckName[strlen-1] = 0` to strip the newline. `fgets(buf,31,..)` stops after **30 characters** and only consumes the newline if it fits - so a name of 30+ left the `\n` in the stream and **every subsequent line was off by one**: the next `fgets` ate the stray newline instead of the `truckModelBaseName` LABEL, and the model name became the label itself:

```
Unable to open model: truckModelBaseName.bin
```

The truck still appeared in the 3D preview and died at race start. The old strip also ate the last real character of the name. Reported on **"Chevy Herzog Trophy Truck Evo2"** - exactly 30 characters; one shorter and it works, which is why it survived since 1998 (no stock truck name is that long).

Loader now reads the whole line into an 80-byte buffer, consumes any overflow to end-of-line, strips CR/LF, and copies bounded. Line 1449 was the ONLY tight `fgets` in that parser - every other read already used 80.

Name lengths - DESIGN LOCKED, ENGINE SIDE DONE 2026-08-09

STATUS: engine reads long names; BinEdit writes them. Traxx in progress. Landed: `3b4eb17` - `MRGL_TEXTURE64` *(opcode 62), the texture record with* `name[64]`*. A new opcode, because* `textureStruct` *is a direct overlay on raw* `.bin` *bytes and its* `name[16]` *is the on-disk layout of every existing model. Compile-time assertions pin the prefix offsets and both strides (24 / 72); widening the old record now fails the build.* `a93adb1` - ~25 in-memory name buffers `[16][64]`, each classified in-memory-only with evidence first (nothing crosses the DirectPlay wire; nothing is written as a raw struct). Closed four live overflows on the way. `0235ae6` *- the six fields that were transitively frozen by* `textureStruct`*, plus the records that receive them converted to* `texture64Struct`*.* ***Still capped by their own formats - these are decisions, not resizes:*** `keyFrameStruct::modelFile` *(stride pinned 344) ·* `textureCycleStruct`*'s 32-byte-strided trailing names.* ***TRUCK LOD NAMES - UNCAPPED 2026-08-21 (was the third entry on that list).*** *The 1997 rule was "stem <= 7: APPEND the detail digit; longer: write it* ***at offset 7****", which silently destroys the 8th character -* `BIGFOOTX` *and* `BIGFOOTY` *both resolve to* `BIGFOOT1.BIN`*. Never a buffer limit (*`truckModelName` *is* `[64]`*, the locals are* `[80]`*), which is why the 8-09 widening did not move it and why BinEdit still reported LOD stems as capped.* ***Now: APPEND first, fall back to the offset-7 name only if the appended file does not exist.*** *Existing trucks authored under the overwrite name miss the first probe and resolve exactly as they always did; new content ships* `<full stem><digit>.bin` *and is found first; a truck may carry both during a transition (appended wins). Stems <= 7 are byte-identical and take the same single probe - the appended form IS the legacy form there, so the fallback cannot fire.* ***Two call sites, keep them in step:*** `core\Truck.cpp` *(the loader) and* `core\fileman.cpp` *(the EXTRACT MANIFEST). Fixing only the loader gives a truck that runs in game and silently loses its LODs when packaged.* ***TOOL CONTRACT:*** *emit* `<full stem><digit>.bin`*, no truncation; when RESOLVING an existing truck's LODs, do the same two-step probe or new content gets mislabelled. The remaining bound is the* ***POD entry****, not the engine:* `podItem.name[32]` *must hold* `MODELS\<stem><digit>.BIN`*, so a LOD-bearing stem fits in 24 – 4 (*`.bin`*) – 1 (digit) =* ***19 characters*** *- just above the* ***18-char*** *stem contract in* `CPOD_LONG_NAMES.md` *§1, so a stem authored to that contract fits with one to spare. (This line previously called 19 "comfortably above" a 20-char ceiling, which it is not; the contract is 18 and the comparison now holds.) It remains the packer's job to enforce, not the engine's.* ***TOOLS - BinEdit*** `91d4fc1` ***(long names) +*** `c6be80e` ***(materials); Traxx*** `b46d03a`***.*** **Long names:** BinEdit's stem cap is 11 -> 20, still bounded by the per-prefix POD1 budget, and it WRITES `MRGL_TEXTURE64` - but only when a name needs it. A name <=15 chars still emits the byte-identical 24-byte record, so short-named models keep loading on older engines. ***Animated frame names stay capped at 15 SEPARATELY.*** `textureCycleStruct` *is 32-byte strided as* `name[16]` *followed by* `sound[16]`*, so an over-long frame name overwrites that frame's SOUND. Sharing one constant with still-texture names would have raised it silently.* **Materials:** BinEdit authors `MRGL_MATERIAL` / `MRGL_MATFACET`. Glass is a PRESET over the ten flags, deliberately NOT a "material type" dropdown - a type list would reinstate exactly the product-of-combinations problem materials exist to remove. Adding the next material is one flag row plus one checkbox. * **Traxx** merged `limits` into `hd-textures`; stems computed per prefix from one place. Do not confuse this with the **content COUNT** raises (`15da90a`, `5e4f583` - scenery, movers, ramps, models). Different axis.

Two-tier scheme, format chosen per-file by the name itself:

1. **Engine buffers to be raised ONCE to the final size (**`char[64]`**-class) in Stage 1** - so the

audit never repeats; NOT a statement that it has happened. Applies to every name field: textures, KASE samples/songs, model refs, track fields.

2. **Stems <=20 plain POD1** (`name[32]`: 31 usable incl. dir prefix + extension; `WORLD\`

leaves 21). All existing third-party POD tooling keeps working on these volumes.

3. **Any stem >20 C-Pod writes the EXTENDED directory** (64-byte names, version-tagged header;

~50-char stems). Only the new engine mounts these - no new lock-in, since new content already requires the new exe (limits, PNG, materials). Authors never pick a format; the name decides.

4. Engine: the extended mount is a version tag + second entry size in the one loader (pod.cpp,

just hashed). Tool caps: Traxx/BinEdit/C-Pod allow 20 for POD1 output, more when writing extended.

5. **Keyframe exception stands regardless:** models referenced by keyframe records stay <=15 total

incl. extension (`keyFrameStruct::modelFile[16]` is on-disk, stride pinned 344) until side-tabled.

(superseded) Name lengths (until the long-filename work lands)

field	limit
POD directory entry	31 chars TOTAL incl. internal dir prefix + extension (`char name[32]`)
Current engine name fields	8.3 stems (the long-filename work will raise engine buffers; practical stem target ~ **22**)
Models referenced BY KEYFRAME records	<=15 chars total incl. extension (`keyFrameStruct::modelFile[16]` is on-disk, stride pinned at 344)

Rules for future raises

1. Every cap above fails **loud** (`GTFO`) on overflow - keep it that way; never let a raise

introduce a silent-drop path for authored content.

2. `MAX_SCENERY_MODEL > MAX_BOX_LIST` always (dangling-model crash otherwise - see SIMMODEL.H).
3. `MAX_SIM_LIST >= MAX_VEHICLE_LIST + MAX_RAMP_LIST + MAX_BOX_LIST + MAX_TOP_CRUSH_LIST` (10+100+4608+100 = 4818 today). Cylinders are NOT in buildSimList - dead feature. The old '+ 40' margin was wrong from 8-09: it counted 10 ramps, not the 100 they were raised to in that same pass, so 4650 was already 78 short.
4. All dependent arrays size off the defines (audited 8-09 - nothing hardcoded); keep it that way.
5. After any change here: update this file, then Traxx / C-Pod / BinEdit enforcement to match.