Designing a 125 GB Monolithic Software Package — First 125 MB as Executable; Remainder as Assets/Data — Target Environments: Windows 10/11 (native) and Browser■only WebAssembly

1. Concept Overview

This document proposes a packaging convention in which a single 125 GB file contains two logical parts:
• The first 125.00 MB is the executable (the OS■loadable program and its core runtime code).
• Everything after 125 MB consists of assets (models, media, maps), databases, and configuration payloads.

On Windows 10/11, this can be implemented using the Portable Executable (PE) format with an appended overlay region.
In the browser, a WebAssembly (WASM) core (~125 MB) boots the app, which then streams and manages the remaining assets under a virtual filesystem backed by the web platform (Cache Storage/OPFS/IndexedDB).

2. Windows 10/11 Native Implementation

2.1. PE/COFF and the “Overlay”
• Windows loads only the sections declared in the PE headers. Any bytes after the last section—often called the overlay—are ignored by the OS loader.
• Therefore, a single file can contain a normal signed PE at the start, followed by arbitrary data. Installers and self■extracting archives already use this pattern at smaller scales.

2.2. Enforcing the 125 MB Rule
• Place the entire PE (headers + .text/.rdata/.data/.rsrc/.reloc, etc.) within the first 125 MB. The rest of the file is the overlay.
• Inside the executable, embed a tiny, early■load manifest (TOC) that describes where assets live in the overlay (offsets, sizes, hashes, MIME, target install path, compression, encryption).
• Alignment: PE SectionAlignment and FileAlignment govern mapped code/data; the overlay does not need PE alignment but should keep large■block alignment (e.g., 4 KiB or 64 KiB) to improve paging and I/O.

2.3. Startup Flow
1) OS maps the 125 MB PE into memory; overlay remains on disk.
2) Your bootstrap code parses the embedded TOC and validates the package signature.
3) It either (a) memory■maps slices of the overlay on■demand, or (b) extracts assets into proper locations, e.g.:
• Immutable binaries: %ProgramFiles%\Vendor\App\bin
• Large content: %ProgramData%\Vendor\App\assets (shared) or %LOCALAPPDATA%\Vendor\App\assets (per■user)
• Mutable DBs: %LOCALAPPDATA%\Vendor\App\data

4) Verify integrity (SHA■256 per■chunk), decompress (zstd/LZMA), and decrypt (if used) before first use.
5) Start the main app with a deterministic search path (private “DLL directory”, SxS assemblies) to avoid DLL hijacking.

2.4. DLLs and Side■by■Side (SxS)
• Prefer statically linking critical runtime or package private■DLLs placed next to the EXE (or in an app■local SxS). Use an app manifest to pin versions and enable SafeDllSearchMode.
• Avoid placing DLLs into system directories. Keep everything app■local to ease updates and reduce hijacking risk.

2.5. Code Signing, SmartScreen, and AV
• Authenticode covers the entire file (excluding the signature blob). Appending an overlay must be done *before* signing; any post■signing modification invalidates the signature.
• Submit to Microsoft for reputation building to avoid SmartScreen prompts. Expect long scans for a 125 GB file; prefer delta updates to keep downloads smaller.

2.6. Updating a 125 GB Package
• Use block■level deltas (Rsync/Rolling hash; bsdiff/xdelta; or a custom chunker like 4–16 MiB content■defined chunks).
• Maintain a content■addressed store on disk so unchanged chunks are reused across versions.
• Use BITS or a resilient updater service to handle resume and throttling.

2.7. Filesystem, Limits, and Performance
• NTFS supports 125 GB files; avoid FAT32 media. Enable LargePages only for mapped sections that benefit.
• Memory footprint remains modest because the OS maps only PE sections; overlay is streamed.
• Consider per■title caching and prefetching hints (SuperFetch is deprecated, but your bootstrap can model usage to prefetch critical ranges).

3. Browser■Only WebAssembly Implementation

3.1. What “Executable” Means on the Web
• There are no Windows DLLs in a browser. Your 125 MB ‘executable’ is a WASM core module (possibly split into multiple modules) plus a minimal JS bootstrap.
• Use streaming compilation/instantiation so the runtime begins before the full 125 MB arrives.

3.2. Module Layout and Linking
• Split into components: core.wasm (~125 MB) + feature modules (graphics.wasm, audio.wasm, sim.wasm). Use the WebAssembly Component Model or dynamic linking conventions supported by your toolchain.
• Threads: use WASM threads via SharedArrayBuffer; ensure COOP/COEP headers and cross■origin isolation.
• Memory: today’s WASM32 limits single■module linear memory to ~4 GiB; WASM64 (memory64) lifts this ceiling as browsers adopt it. Architect for tiling/streaming so you never require multi■GiB contiguous buffers.

3.3. Delivering the Remaining ~125 GB of Assets
• Do not ship a literal 125 GB single HTTP object. Instead:
– Chunk assets into 4–16 MiB (CDN■friendly). Name by content hash (e.g., SHA■256) for dedup

and integrity.
– Use Range requests and parallel fetch for large objects.
– Compress with zstd/brotli; serve over HTTP/3 with proper prioritization.
• Storage:
– Cache Storage for immutable chunks.
– OPFS (Origin Private File System) for DBs and write■heavy content.
– IndexedDB for metadata/TOC if convenient.
• Include a signed manifest (TOC) with per■chunk size/hash and a top■level signature (Detached CMS or COSE). Verify in the WASM core before accepting data.

3.4. Database Options
• SQLite■WASM or DuckDB■WASM with OPFS backends provide robust embedded DBs. Keep write■amplification in mind; prefer append■only or columnar formats for huge analytical data.

3.5. Packaging and Install Experience
• Make it a PWA: installable, offline■capable. A Service Worker manages caching, updates, and background sync (when the app is open).
• First■run wizard lets users choose storage quota (where supported), data packs (language, regions), and download schedules.

3.6. Security
• Use HTTPS + HSTS. Set strict CSP to block unexpected script execution. Use Subresource Integrity (SRI) for static JS; for WASM and chunked assets, verify hashes and a signed manifest inside the app.
• Cross■origin isolation (COOP/COEP) for threads and high■resolution timers.

3.7. Updates in the Browser
• The Service Worker fetches a new manifest; only changed chunks download. Maintain rolling caches; evict least■recently■used packs.
• Offer fast■path micro■updates for the WASM core (e.g., separate small patch modules) to avoid re■downloading the full 125 MB core.

3.8. What You Cannot Do in Browser■Only Mode
• No direct use of Windows DLLs or registry. No arbitrary file system access outside the origin sandbox. No kernel drivers. Integrations must use web APIs (WebGPU/WebAudio/WebUSB where available).

4. Shared Container Format (Native & Web)

Define a single content format so both the Windows loader and the WASM runtime read the same manifests and chunks.

Header (within first 125 MB):
• Magic: QPF\0; Version; Flags (endianness, compression, encryption)
• TOC offset/length (points into the overlay)
• Public key ID(s) and signature block offset for whole■file or per■chunk signing
• Bootstrapping table (names of primary executable/WASM module(s), entrypoints)

Overlay (after 125 MB):
• Chunks: [hash | offset | length | compression | encryption | targetPath | MIME]
• Optional pack files (e.g., .pak) with internal LZ4/zstd and their own mini■TOC

• End■of■file trailer repeating critical metadata for recovery

Operational notes:
• Place a small TOC cache in the first 125 MB for cold■start discovery; the full TOC can live in the overlay.
• Keep chunk sizes CDN■friendly; align to 1–4 MiB boundaries to optimize HTTP/3 and disk I/O.

5. Practical Constraints and Recommendations

• Distribution: 125 GB is immense. Prefer optional content packs and regional/language downloads. Use a CDN with partial■object and range■request optimization.
• Telemetry: Log chunk cache hits/misses (privacy■preserving) to tune prefetch and storage budgets.
• Legal/Compliance: Shipping cryptography may trigger export rules; include a crypto notice if using encryption.
• Testing: Build a synthetic content generator to fuzz TOC and chunk boundaries; test power■loss recovery and partial downloads.
• Accessibility: Ensure the bootstrap supports screen readers and keyboard navigation even before full assets are present.

6. Sample TOC Entry (JSON)

{
"version": 3,
"chunks": [
{"path":"assets/terrain/region-17.pack","sha256":"","size":16777216,
"offset": 134217728, "compression":"zstd","encryption":"none","mime":"application/octet-stream"},
{"path":"bin/plugins/physics.wasm","sha256":"","size":24431872,
"offset": 987654321, "compression":"none","encryption":"none","mime":"application/wasm"}
],
"signatures": {"algorithm":"ed25519","manifestSig":""}
}

7. Implementation Checklist

■ Keep the PE within 125 MB; everything else is overlay.
■ Embed a minimal TOC pointer in the executable; store the full TOC in the overlay.
■ Sign after the overlay is appended; verify signatures at startup.
■ Use app■local DLLs (or static link); lock DLL search path via manifest.
■ In the browser, ship a ~125 MB core WASM with streaming instantiation and split feature modules.
■ Store assets via Service Worker in Cache Storage/OPFS; verify per■chunk hashes.
■ Use chunked, content■addressed assets with delta updates.
■ Provide a first■run downloader with quotas, selectable packs, and resume.
■ Test crashes/power■loss during extraction and during web cache population.
■ Monitor performance; keep hot assets near the start of the overlay or in separate packs.

Appendix A: Registry and Configuration on Windows

• Prefer per■user configuration under HKCU\Software\Vendor\App and files under %LOCALAPPDATA%.
• Avoid HKLM writes unless the installer is elevated (MSIX or MSI). If using MSIX, you can keep the one■file packaging for download, but install into MSIX layout at rest.
• Use Event Tracing for Windows (ETW) channels for diagnostics without excessive disk logging.

Appendix B: Example Startup Pseudocode

```
if !verify_signature(exe_and_overlay): fail("Signature invalid")
toc = read_toc(pointer_in_exe)
for chunk in required_boot_chunks(toc):
map_or_extract(chunk)
launch_main()
background_prefetch(usage_model, network_budget)
```