PRD — Windows Standalone Executable & Installer Wizard + Technical Design
Project: JPE Sims 4 Mod Translation Suite — JPE Studio (Windows)
Source of truth: JPE Sims 4 Mod Translation Suite SOP

---
1. Purpose & Goals

This PRD defines how to ship the JPE Sims 4 Mod Translation Suite as:

1. A standalone Windows desktop executable ("JPE Studio for Windows") that:
   - Runs fully offline.
   - Bundles the core translation engine and JPE/JPE-XML language stack.
   - Provides a safe, friendly environment for working with Sims 4 mods.

2. A Windows installer wizard that:
   - Is "dummy-proof" for non-technical modders.
   - Handles all dependencies and configuration (paths, file associations, etc.).
   - Aligns with the SOP's safety, diagnostics, and versioning requirements.

High-level objectives:

- Make it trivial for a Sims 4 modder to install, launch, and start translating mods without touching Python, VS Code, or command lines.
- Preserve the core architecture: /core, /languages, /desktop as per SOP, packaged into a Windows-native experience.
- Enforce safety (no touching saves, no overwriting original mods), and provide structured diagnostics from installer to app.

---
2. Scope

2.1 In Scope

- Windows-only desktop distribution (v1):
  - Single JPE Studio executable (or minimal EXE + support files).
  - Traditional installer wizard (EXE/MSI) with:
    - Install, repair, modify, uninstall.
    - Optional silent/CLI installation.
- Packaging into a Windows build:
  - core/ translation engine, parsers, validators, diagnostics.
  - languages/ (JPE DSL + JPE-XML schema).
  - desktop/ GUI editor.
  - plugins/ that ship in v1 (file-type adapters, version packs).
- Integration of diagnostics and error reporting into:
  - Installer (install logs, friendly messages).
  - App startup (clear errors if engine or language packs fail to load).
- First-run and configuration flows that bridge from install to usable project:
  - Game folder detection and verification.
  - Workspace setup for mods.
- Update story:
  - Manual "Check for updates" from within app.
  - Support for in-place upgrades via updated installer.

2.2 Out of Scope (for this PRD)

- iOS app implementation details (covered by mobile/ios).

- Cloud backend specifics (sync, CI builds) beyond basic “future integration hooks.”
- Deep UI/UX layout decisions (covered in the UI/UX PRD).
- Cross-platform installers (macOS/Linux) — future PRDs.

---
3. Users & Personas
---

1. Everyday Modder (Non-Technical)
   - Wants: “Double-click installer, next-next-finish, open my Mods folder, make things English, don’t break my game.”
   - Fear: Messing up game files, complex setup, cryptic errors.

2. Power Modder / Tool Creator
   - Wants: Faster iteration, reliable builds, CLI or headless modes later.
   - Needs: Silent installs, predictable paths, logs, and diagnostics.

3. Tester / QA
   - Wants: Easy fresh installs, version switching, regression testing.
   - Needs: Clear build numbers, changelogs, install logs, safe rollback/uninstall.

---
4. Platform & System Requirements

4.1 OS & Hardware

- Supported OS:
  - Windows 10 (21H2+) 64-bit
  - Windows 11 64-bit
- Minimum Hardware:
  - 4 GB RAM (8 GB recommended)
  - 1.5 GB free disk space (app + core + language packs + logs)
  - x64 CPU with SSE4 or better
- Permissions:
  - Standard user install supported (per-user under %LOCALAPPDATA%).
  - Optional machine-wide install under %ProgramFiles% (requires elevation).

4.2 Dependencies

- The shipped EXE must:
  - Bundle all required runtimes (e.g., Python, VC++ redistributables) OR
  - Check and install prerequisites during the wizard if not bundled.
- No assumption of pre-installed dev tools (no Python, no Git, no IDE needed).

---
5. Core Concepts (adapted from SOP)

The Windows app must expose, through the packaged runtime, the core behaviors defined in the SOP: reading mod file types, translating to JPE/JPE-XML, generating XML tuning, and validating with rich diagnostics.

Required internal capabilities of the shipped app:

- Supported file types (v1):
  - XML tuning, STBL, .package, .ts4script/.py, JSON, CFG/INI.
- Core flows:

  - Import mod(s) → parse to IR → render JPE / JPE-XML → edit → compile → XML/STBL output.
- Diagnostics:
  - Structural & semantic validation, version-specific rules.
  - Human-readable errors + machine-readable reports.
- Safety:
  - Never modify original mod files in-place.
  - Always write outputs to a separate “build” directory.

---
6. Functional Requirements — Standalone Desktop App

6.1 Launch & Startup

- Application must start from:
  - Start Menu shortcut (“JPE Studio”).
  - Desktop shortcut (if selected).
  - Double-clicking a .jpe or .jpexml file (if associations enabled).
- On first run:
  - Show a First-Run Wizard (inside the app) with:
    1. Welcome + quick explanation.
    2. Sims 4 installation detection:
       - Try known paths (Origin/EA App/Steam).
       - If not found, prompt user to browse.
       - Validate folder structure to ensure it’s actually a Sims 4 install.
    3. Mods workspace setup:
       - Offer default workspace pointing to Documents\Electronic Arts\The Sims 4\Mods.
       - Option to choose/customize workspace directory for translated outputs.
    4. Telemetry/analytics opt-in (if any), with clear explanation.
  - Persist configuration in %APPDATA%\JPEStudio\config.json or similar.

6.2 Project & File Handling

- Open Project:
  - Open single mod folder.
  - Open multi-mod workspace (root folder).
- File Access Rules:
  - Read-only access to user’s Mods folder, unless explicitly saving new outputs.
  - All generated/compiled files written to a JPE_Build subfolder or configured output path.
- File Associations:
  - Optional (user-selectable in installer) associations:
    - .jpe → Open with JPE Studio.
    - .jpexml → Open with JPE Studio.
  - Must be reversible via installer “Modify” or via uninstaller.

6.3 Translation & Build Flows

- Import:
  - Drag-and-drop .package / XML tuning / STBL / .ts4script/.py / JSON / CFG.
  - “Import from Sims 4 Mods folder” wizard:
    - Filter by file type.
    - Preview what will be imported.
- Translate to JPE/JPE-XML:
  - Per-file and bulk operations.
  - IR-based transformations (XML → IR → JPE and XML → IR → JPE-XML).
- Edit & Validate:
  - Use JPE and JPE-XML editors.
  - On save, run validation:

  - Structural (schema, types).
  - Semantic (ranges, enums, references).
- Compile Back to XML:
  - JPE/JPE-XML → IR → XML tuning/STBL.
  - Output into per-project build folder with clear naming.
- Diagnostics:
  - Problems pane wired to engine diagnostics.
  - Machine-readable logs stored to logs/ per project and global logs/ folder.

6.4 Offline Operation

- All above functionality must work offline by default.
- Any future cloud features must:
  - Fail gracefully when offline.
  - Not block core translation/editor usage.

---
7. Functional Requirements — Windows Installer Wizard

7.1 Installer Type & Distribution

- Format:
  - Primary: Single .EXE bootstrapper (preferred) which may wrap MSI or internal installer.
- Distribution expectations:
  - Downloadable from project site.
  - Can be run from user Downloads folder without extra steps.

7.2 Installer UX Flow

Screen 1 — Welcome
- App logo + name: “JPE Sims 4 Mod Translation Suite — JPE Studio for Windows”.
- Short description (“Translate and build Sims 4 mods in Just Plain English, locally and safely.”).
- Button: Next.

Screen 2 — License / EULA
- Show license text (scrollable).
- Require explicit acceptance.
- If not accepted, block forward progress and allow exit.

Screen 3 — Install Mode & Location
- Install for:
  - “Just me (recommended)” → %LOCALAPPDATA%\JPEStudio.
  - “All users (requires admin)” → %ProgramFiles%\JPEStudio.
- Folder browser to customize install path.
- Display disk space requirements and available space.

Screen 4 — Components Selection
- Checklist of components:
  - [x] JPE Studio core application.
  - [x] JPE Language + JPE-XML schemas and rulesets.
  - [ ] Additional language packs (future).
  - [ ] CLI utilities (future option).
  - [ ] Desktop shortcut.
  - [ ] File associations for .jpe and .jpexml.
- Real-time disk space estimate based on selection.

Screen 5 — Sims 4 Path Detection (Optional but recommended)
- Attempt automatic detection of Sims 4 install and Mods folder.
- If found:
  - Show detected paths with green check.
  - Allow override via browse.
- If not found:
  - Explain: user can configure later inside the app.
  - Provide optional browse flow to set it now.

Screen 6 — Telemetry / Analytics (Optional)
- Explain clearly:
  - What's collected (e.g., anonymous usage metrics, error stats).
  - What's NOT collected (no mods content or personal data).
- Default: opt-out (off) if there's any doubt on privacy.
- Toggle + link to privacy policy.

Screen 7 — Install Summary
- Show:
  - Destination folder.
  - Components selected.
  - Sims 4 path status (detected / not configured).
  - Telemetry choice.
- Buttons: Back, Install.

Screen 8 — Installation Progress
- Progress bar, with steps:
  - Extracting files.
  - Installing runtimes (if needed).
  - Registering app (shortcuts, associations, uninstaller).
- Logging:
  - Installer writes logs to %ProgramData%\JPEStudio\install.log or similar.

Screen 9 — Completion
- Show success or failure.
- On success:
  - Checkbox options:
    - [x] Launch JPE Studio now.
    - [ ] View release notes / user guide.
  - Button: Finish.
- On failure:
  - High-level human-friendly message.
  - Link/button to open log file location.

7.3 Modify, Repair, Uninstall

- Modify:
  - Change components (e.g., enable/disable file associations).
  - Change install path (with safe migration).
- Repair:
  - Reinstall any missing or corrupted files.
  - Validate checksums.
- Uninstall:
  - Remove binaries and registry entries.
  - Offer to:
    - Keep user data (workspaces, configs, logs).
    - Or remove all (hard cleanup).

7.4 Silent / CLI Install

- Support command-line flags:
  - /S or /silent to do a silent install.
  - INSTALLDIR=..., ALLUSERS=1, ASSOC_FILES=1, TELEMETRY=0 etc.
- On silent failure:
  - Non-zero exit code.
  - Log to known path (install.log).

---
8. Diagnostics & Error Handling (Installer + App)

8.1 Installer

- All errors standardized:
  - Human message (“Could not write to C:\Program Files\JPEStudio. You may need administrator rights.”).
  - Error code and context in install.log.
- Log format:
  - Machine-readable (JSON lines or structured key=value).
  - Timestamps, severity, component, message, stack if applicable.

8.2 App Startup

- If core engine or language packs fail to load:
  - Show dialog with:
    - Friendly description.
    - “View technical details” expand section (error code, log path).
- Write to app log: %APPDATA%\JPEStudio\logs\app-startup.log.

8.3 Runtime Diagnostics

- Build failures or translation errors:
  - Display in Problems pane.
  - Write machine-readable report in project logs/:
    - Including file path, line/column, rule ID, message, severity.

---
9. Non-Functional Requirements

9.1 Performance

- Installer:
  - Typical install under 2 minutes on a mid-range SSD.
- App:
  - Cold start under 10 seconds on baseline hardware.
  - Basic project with ~200 tuning XMLs:
    - Initial scan under ~15 seconds.
    - Single-file translations under ~1 second.

9.2 Security

- No elevation required for per-user installs.
- Only write to:
  - Install directory.
  - %APPDATA%\JPEStudio.
  - User-configured workspaces.
- No services or background daemons that persist after app exit.

- No network connections without explicit user permission (e.g., for updates or cloud).

9.3 Reliability

- Installer must handle:
  - Interrupted installation (resume or roll back cleanly).
  - Partial file extraction errors (disk full, permissions).
- Upgrades:
  - Preserve user settings, projects, and logs by default.

9.4 Accessibility

- Installer UI:
  - Fully keyboard navigable.
  - Screen reader friendly (proper labels, focus order).
- App:
  - High-contrast compatible.
  - Scalable UI fonts (handled in UI PRD).

9.5 Localization

- v1: English-only text for installer and app.
- Architecture:
  - All strings externalized to resource files to allow future translations.

---
10. Updates & Versioning

10.1 Version Schema

- App version (semantic): MAJOR.MINOR.PATCH.
- Internal:
  - engine_version,
  - jpe_version,
  - jpe_xml_version,
  - diagnostics_version embedded in About dialog and in logs.

10.2 Update Mechanism

- Inside JPE Studio:
  - “Check for updates” menu item.
  - Queries remote JSON (when online) for latest version and release notes.
- Update flow:
  - Download new installer EXE.
  - Launch installer in upgrade mode (in-place).
- Offline behavior:
  - Update check fails gracefully with simple message.

---
11. Data & File Layout (After Install)

11.1 Install Directory (Example)

- JPEStudio\JPEStudio.exe — main executable.
- JPEStudio\core\ — core engine, parsers, validators.
- JPEStudio\languages\ — JPE DSL, JPE-XML schemas, rulesets.

- JPEStudio\plugins\ — bundled plugins.
- JPEStudio\runtime\ — embedded Python/etc. (if used).
- JPEStudio\docs\ — offline docs and examples.

11.2 User Data

- %APPDATA%\JPEStudio\config\ — user-wide configuration.
- %APPDATA%\JPEStudio\logs\ — app-level logs.
- Per workspace:
  - <workspace>\mods\ — user's original mods.
  - <workspace>\jpe\ — JPE/JPE-XML sources.
  - <workspace>\build\ — compiled XML/STBL outputs.
  - <workspace>\logs\ — project logs, diagnostics exports.

---
12. Telemetry & Privacy (Optional Feature)

If telemetry is implemented:

- Consent:
  - Explicit opt-in at first run and in installer.
  - Toggleable later in Settings.
- Data collected (example):
  - App version, OS version.
  - Anonymous usage metrics (feature usage counts).
  - Error codes (without mod content).
- Never collected:
  - Raw mod files.
  - JPE source content.
  - Personal identifiers.

---
13. Risks & Open Questions

13.1 Risks

- Bundling runtimes makes the installer heavier (download size).
- Automatic Sims 4 path detection could mis-detect non-standard setups.
- File associations might be blocked or require admin rights, depending on security policies.

13.2 Open Questions

- Preferred installer tech: NSIS, Inno Setup, WiX, or others?
- Do we need portable mode (no installer, just unzip and go)?
- How aggressive should auto-update be (background check vs manual only)?

---
14. Technical Design & Architecture

14.1 High-Level Architecture

The Windows build is a three-layer system:

- Core Engine (Python):
  - Package: jpe_sims4_core inside core/.

  - Responsibilities: IO, parsers, intermediate representation (IR), translators (JPE/JPE-XML), validators, diagnostics.
- Desktop UI (Python + Qt/PySide6):
  - Package: jpe_studio_desktop inside desktop/.
  - Responsibilities: project/workspace management, editors, views, wizards, wiring to core engine.
- Installer & Bootstrap:
  - Output of PyInstaller (one-folder build) wrapped by NSIS (or Inno Setup) installer.
  - Responsibilities: file copy, registry entries, shortcuts, file associations, uninstaller.

14.2 Technology Choices

- Language/runtime: Python 3.11 (minimum).
- UI toolkit: PySide6 (Qt for Python) for native-feeling Windows UI without external browsers.
- Packaging:
  - PyInstaller (one-folder mode) to build JPEStudio.exe + dependencies.
  - NSIS (or Inno Setup) to create the installer EXE with UI wizard.
- Configuration & data:
  - JSON configuration files in %APPDATA%\JPEStudio\config.
  - Project-level config files in each workspace.

14.3 Core Engine Modules

Core engine (core/jpe_sims4_core) is split into:

- io/
  - filesystem.py — safe file access, path normalization, sandboxing.
  - sims_paths.py — Sims 4 install + Mods folder detection helpers.
- parsers/
  - tuning_xml.py — XML tuning parsing to IR.
  - stbl.py — STBL reading/writing.
  - package_reader.py — extract resources from .package files (via existing libraries or custom code).
  - script_extractor.py — extract .py from .ts4script archives.
- ir/
  - model.py — definitions for IR nodes (objects, interactions, tuning properties).
  - visitors.py — transformation helpers.
- translators/
  - jpe_encoder.py — IR → JPE (plain-English DSL).
  - jpe_decoder.py — JPE → IR.
  - jpe_xml_encoder.py — IR → JPE-XML.
  - jpe_xml_decoder.py — JPE-XML → IR.
- validators/
  - schema_rules.py — structural checks (types, required fields, ranges).
  - semantic_rules.py — cross-file and version-specific rules.
- diagnostics/
  - errors.py — error/warning classes and codes.
  - reporter.py — machine-readable and human-readable output.
- versioning/
  - versions.py — engine_version, jpe_version, jpe_xml_version, diagnostics_version.

14.4 Desktop UI Modules

Desktop UI (desktop/jpe_studio_desktop) modules:

- app.py — application entry point; sets up Qt application, theming, global logging.
- main_window.py — main window, menus, toolbars, status bar.
- views/

  - project_explorer.py — workspace tree (mods, JPE sources, builds).
  - editor_jpe.py — JPE text editor with syntax highlighting.
  - editor_jpe_xml.py — JPE-XML editor with schema-aware hints.
  - diagnostics_panel.py — problems pane linked to diagnostics reporter.
  - logs_viewer.py — quick log viewer.
- wizards/
  - first_run_wizard.py — first-run flow (Sims 4 detection, workspace setup, telemetry).
  - import_wizard.py — mod import from Mods folder or arbitrary folder.
- services/
  - engine_bridge.py — high-level API that calls into jpe_sims4_core.
  - workspace_manager.py — load/save workspace configs, paths, recent projects.
  - settings_manager.py — global settings (telemetry, theme, fonts).
- integration/
  - file_associations.py — requests registration/unregistration via installer or helper.
  - update_checker.py — fetches remote JSON (when enabled) to check for updates.

14.5 Process & IPC Model

Since UI and engine both run in Python within the same process:

- No external IPC needed in v1.
- engine_bridge.py simply imports jpe_sims4_core and calls functions.
- Long-running operations (bulk imports/builds) run in worker threads (Qt QThread or Python threading) to keep UI responsive.

Later versions could introduce a separate engine process, but v1 runs single-process for simplicity.

14.6 Configuration & Paths

- Global config file: %APPDATA%\JPEStudio\config\settings.json
  - Keys:
    - sims_install_path
    - mods_folder_path
    - default_workspace_root
    - telemetry_enabled
    - ui_theme
    - recent_workspaces[]
- Per-workspace file: <workspace>\.jpe_workspace.json
  - Keys:
    - workspace_name
    - mods_root
    - jpe_root
    - build_root
    - ruleset_version
- Engine uses sims_paths.py for Sims 4 auto-detection:
  - Try registry keys or known folders for EA App/Origin/Steam installs.
  - Fallback to manual user selection.

14.7 Logging Design

Logging subsystem (diagnostics/reporter.py + app-level logger):

- Global log file:
  - %APPDATA%\JPEStudio\logs\app.log (rotating, e.g., 5x 1MB files).
- Project logs:
  - <workspace>\logs\build_YYYYMMDD_HHMMSS.json (per build run).
- Log fields (JSON):

  - timestamp
  - component (engine/ui/installer_bridge)
  - level (INFO/WARN/ERROR)
  - message
  - context (file path, rule id, stack trace if any)

14.8 Installer Implementation Details

- Build chain:
  1. Run tests (unit + integration) for jpe_sims4_core and desktop modules.
  2. Build PyInstaller spec:
     - Entry point: desktop/app.py.
     - One-folder output: dist/JPEStudio/.
     - Include:
       - core/ and languages/ folders as datas.
       - plugins/ folder.
       - docs/ folder.
  3. Run PyInstaller to generate JPEStudio.exe and bundled runtime.
  4. Run NSIS (or Inno Setup) script:
     - Reads dist/JPEStudio/ as payload.
     - Defines installer wizard pages as per Section 7.
     - Registers uninstaller and file associations (if user selects).
- File associations implementation (NSIS example):
  - Writes HKCU\Software\Classes\.jpe and .jpexml default values pointing to ProgID “JPEStudio.file.jpe” etc.
  - ProgID specifies shell\open\command to JPEStudio.exe "%1".

14.9 Update Mechanism Technicals

- update_checker.py fetches a version manifest JSON when user triggers “Check for updates”:
  - Example manifest:
    {
      "latest_version": "1.2.0",
      "download_url": "https://example.com/JPEStudio_1.2.0_Setup.exe",
      "release_notes": "Bug fixes and improvements."
    }
- If newer version:
  - Prompt user to download.
  - On confirmation, download to temp file and launch installer in upgrade mode.
  - Current app exits (or instructs user to close) before running installer.

14.10 CI/CD Pipeline Overview

Recommended CI/CD steps:

- Trigger: push to main branch, tag build, or manual pipeline run.
- Stages:
  1. Lint & Unit Tests
     - Run flake8/ruff, pytest.
  2. Build Core Wheel
     - Build jpe_sims4_core wheel (for other consumers if needed).
  3. Build Desktop App
     - Install PySide6 and dependencies.
     - Run PyInstaller with spec to create dist/JPEStudio.
  4. Build Installer
     - Run NSIS/Inno Setup with installer script.
     - Produce JPEStudio_Setup_VERSION.exe.
  5. Sign Binaries (if code signing is available).

6. Publish Artifacts
   - Upload installer to release storage (e.g., website, internal server).
   - Publish version manifest JSON for update_checker.

14.11 Future-Ready Hooks

- Engine as a library:
  - jpe_sims4_core is fully importable as a library so future tools (CLI, other UIs, Codex workflows) can reuse it.
- Optional headless mode:
  - Future CLI entry point could call the same engine_bridge logic with no UI.
- Cross-platform:
  - Keeping UI and engine separated so a future macOS/Linux UI could re-use the same core engine.

End of Document
