sop jpe sims4 translation suite

SOP — JPE Sims 4 Mod Translation Suite

1. Purpose
This Standard Operating Procedure (SOP) defines how to design, build, test, and ship a toolchain that:
- Reads all relevant The Sims 4 mod file types (XML tuning, STBL, .package, .ts4script/.py, JSON, cfg, etc.).
- Translates those mods into Just Plain English (JPE) so humans can understand the behavior.
- Allows authors to create and edit mods in JPE and an English-friendly XML fork (JPE-XML).
- Compiles all JPE and JPE-XML back into valid Sims 4-compatible XML tuning.
- Provides a desktop editor and an iPhone app using a shared engine.
- Produces rich diagnostics and error reports so that mod issues are easy to track and fix.

2. Project Structure
The project is organized into the following logical components:
- core/: Core translation engine, parsers, generators, validators, and diagnostics.
- languages/: JPE language specification, grammar, and JPE-XML schema.
- desktop/: Desktop GUI editor (JPE Studio) for Windows and optionally macOS/Linux.
- mobile/ios/: Native Swift/SwiftUI iPhone application for JPE editing and review.
- cloud/: Optional backend service for sync, cloud builds, and project history.
- plugins/: Extensible plugins for file-type adapters, version packs, and transformations.
- docs/: User-facing documentation, examples, and language references.

3. Environments
- Development: Local environment with access to sample mods, hot-reload where possible, and fake game data.
- Staging: Environment with anonymized real-world mods for regression testing and compatibility checks.
- Release: Signed builds, installer packages, and App Store distributions with version tagging and release notes.

4. Development Phases
Phase 0 — Research & Corpus
- Identify supported file types for v1: XML tuning, STBL, .package containers, .ts4script/.py, JSON/cfg/ini.
- Collect a small corpus of example mods to act as test fixtures.
- Document common tuning patterns: interactions, buffs, traits, loot actions, autonomy, careers, traits, and moodlets.

Phase 1 — Language and IR Design
- Design the JPE DSL: a plain-English, structured language that maps to an abstract syntax tree (AST).
- Design JPE-XML: an XML dialect with English tag and attribute names that map directly to Sims 4 tuning.
- Design an Intermediate Representation (IR) that can represent interactions, buffs, traits, tests, loot, enums, and localization.
- Define clear mapping rules:
  Existing XML → IR → JPE
  Existing XML → IR → JPE-XML
  JPE/JPE-XML → IR → XML tuning and STBL.

Phase 2 — Core Engine Implementation
- Implement parsers:
  XML → IR
  JPE → IR

JPE-XML → IR
- Implement generators:
  IR → XML tuning
  IR → JPE
  IR → JPE-XML
- Implement validators:
  Structural validation (schema and type correctness).
  Semantic validation (references, enums, ranges) where feasible.
  Version-specific rulesets for different Sims 4 patches.

Phase 3 — Desktop UI/UX
- Build JPE Studio desktop application:
  Project/file explorer for mods and resources.
  Dual-pane views (XML vs JPE vs JPE-XML).
  JPE editor with autocompletion and validation.
  Wizards for common mod patterns.
  Problems pane for diagnostics and quick navigation.

Phase 4 — iPhone Mobile App
- Implement a mobile client using SwiftUI that:
  Lets users browse projects and JPE files.
  Edit JPE/JPE-XML.
  View build errors and warnings.
  Request builds or sync with desktop/cloud.
- Support offline-first operation with queued sync.

Phase 5 — Packaging, Testing, and Release
- Create automated test suites for roundtrips (XML → JPE → XML) and validation.
- Include tests for diagnostics and error report generation.
- Provide installation instructions and safe defaults.
- Release in controlled rings: internal, closed testers, broader audience.

5. Operating Practices
- Versioning: All language, schema, engine, and diagnostics changes must bump versions (jpe_version, jpe_xml_version, engine_version, diagnostics_version).
- Backwards compatibility: Introduce migration paths when schemas change.
- Safety: Do not modify user save files. Do not overwrite original mods; write outputs into new folders.
- Legal/Ethical: Operate only on user-provided mods. Do not bundle or redistribute proprietary assets or game executables.

6. Tooling and Workflow
- Use a consistent SDLC flow: plan → design → implement → test → review → release.
- Keep configuration and project data in version control.
- Document JPE and JPE-XML thoroughly; treat language design as a first-class artifact.
- Maintain a shared fixture library for regression tests.

7. Diagnostics and Exception Handling
- All components must surface failures through a structured diagnostics layer, not raw stack traces.
- Builds should always produce human-friendly messages and machine-readable error reports when issues occur.
- The desktop and mobile applications must consume diagnostics to highlight problematic files and guide users to fixes.