JPE Sims 4 Translation Suite

UI/UX Design Principles & Free Design Stack Specification

This document defines the UI/UX design principles, asset specifications, quality standards, and design-tooling expectations for the JPE Sims 4 Translation Suite. It also replaces all references to “Figma Make AI” with a fully free design stack built around:
• Banani – AI-driven UI generator used to turn text and JSON specs into first-pass screen layouts.
• Penpot – the primary, fully free/open-source design workspace and component library.
The goal is to keep the original intent of the SOP and UI/UX PRDs intact while ensuring the tooling is cost-free, vendor-agnostic, and friendly to iterative development.

1. Purpose & Scope

This document sits on top of the core JPE Sims 4 Translation Suite SOP and the Branding/UI/UX PRDs. It does not replace those; it clarifies how the UI should behave, which assets are required, what “high quality” means in practice, and how the free design stack (Banani + Penpot) fits into the pipeline.

The scope covers:
• Desktop application UI/UX (primary tuning and translation environment).
• Mobile (iPhone) companion application UI/UX (review, triage, light editing).
• Design assets and specifications (icons, layouts, branding, tokens).
• Integration of Banani and Penpot into the design workflow.

2. Free Design Stack Overview

2.1 Core Tools

The project standardizes on a free design stack with two main pieces:

1. Banani – AI UI Generator
• Role: Converts text prompts and structured JSON specs into initial, editable UI layouts.
• Use cases:
– Generate first-pass layouts for desktop screens (project browser, dual-pane editor, Problems pane, build history, settings, plugin manager).
– Generate first-pass layouts for iPhone screens (tabbed navigation, detail views, diagnostics list, build logs).
– Quickly explore alternative layouts, hierarchy, and flows before committing to the design system.
• Output:
– Canvas-based layouts that can be exported (e.g., as images, SVGs, or Figma-compatible files) and then reassembled or replicated in Penpot.

2. Penpot – Primary Design Workspace
• Role: The canonical design environment and source of truth for components, tokens, and final screen layouts.
• Use cases:
– Maintain the official design system (components, variants, typography, color tokens, spacing system).
– Refine and finalize screen layouts generated conceptually in Banani.
– Prepare assets and specs for frontend development (icons, spacing, type styles, export sizes).

• Output:
– SVG and PNG assets at required sizes.
– Layouts, components, and design tokens that mirror what is implemented in code.

2.2 Tool Responsibilities & Boundaries

Banani:
• Generates: First-pass UI layouts, multi-screen flows, and rough component placements from prompts.
• Does NOT own: The final design system, naming, or tokens.
• Acts as: A bootstrapper and exploratory engine.

Penpot:
• Owns: The final, approved UI language, including components, tokens, and layout rules.
• Mirrors: The structure defined in the UI/UX PRDs (screen lists, component inventories, interaction patterns).
• Acts as: The handoff bridge to development.

The JPE Sims 4 Translation Suite codebase:
• Treats Penpot (not Banani) as the design source of truth.
• Implements UI strictly against Penpot-defined components and tokens.

3. Core UI/UX Design Principles

3.1 JPE-First Clarity

The UI is built around the idea that Just Plain English (JPE) is the default lens for understanding mods. Every major screen should answer “What does this mod do?” before “What file is this?”

Principles:
• JPE views are primary; XML and JPE-XML views are secondary but always available.
• The main entities are gameplay concepts (interactions, traits, buffs, tests, autonomy, careers, etc.), not file extensions and tuning IDs.
• Panels and navigation reflect how modders think about content, not how files sit on disk.

3.2 One IR, Many Views

The Intermediate Representation (IR) is the single source of truth. UI views are alternate representations of that IR:

• JPE = readable explanation of what the IR encodes.
• JPE-XML = hybrid representation that exposes structure while staying language-like.
• XML = raw Sims 4 tuning representation.

UX Requirements:
• Selecting an entity in JPE highlights the corresponding XML snippet and vice versa.
• Switching views does not create drift. Edits made in one view reflect in all others once validated and applied.
• Validation logic is IR-driven, not per-view ad hoc logic.

3.3 Diagnostics-First Experience

Diagnostics are first-class citizens in the application:

• A dedicated “Problems” or “Diagnostics” pane is always accessible (and often visible by default).
• Every error or warning links directly to:
– The JPE representation (line/section).
– The XML node or tuning ID.
– The file in the project tree.
• Each diagnostic message must include:
– Human-readable description (“This interaction references a buff that does not exist.”).
– Machine-readable code (for filtering, tests, and automation).
– Suggested next step or quick action when possible (e.g., “Create stub buff with this ID,” “Open linked tuning,” “See docs…”).

3.4 Safe, Non-Destructive Workflows

The UI must make it extremely difficult to accidentally destroy or corrupt content:

• Original mods are treated as read-only by default; generated outputs live in clearly marked “output” or “build” directories.
• File badges indicate origin:
– “Original” for source files imported from existing mods.
– “Generated” for files created or rewritten by the toolchain.
• Destructive actions (delete, overwrite, purge) are:
– Explicitly labeled using clear language.
– Confirmed with context, not generic dialogs.
– Logged in a history or activity view and, where possible, undoable.

3.5 Multi-Surface Consistency (Desktop + iPhone)

The desktop application is the full control surface; the iPhone app is for review, triage, and light editing. Both share the same mental model:

• Shared concepts:
– Projects → Mods → Entities (interactions, traits, buffs, tests, etc.) → Diagnostics.
• Desktop:
– Multi-pane editor (JPE/JPE-XML/XML views side by side).
– Rich filters, diffing, and multi-file editing.
– Build configuration and history panels.
• iPhone:
– Tab-based or stacked navigation.
– JPE-focused reading and quick tweaks.
– Diagnostics browsing, tagging, and acknowledgement.
– Ability to trigger builds or reruns and inspect results.

Visual identity (colors, icons, typography, terminology) remains consistent between surfaces so switching devices feels like switching viewpoints, not switching products.

3.6 Power Without Hostility

The tools can support power users but must not intimidate newcomers:

• Onboarding flows explain core concepts with concrete examples (e.g., “Translate this interaction and see the JPE explanation next to the XML.”).
• Advanced features (custom rulesets, plugin authoring, JPE-XML schema editing) live in clearly labeled “Advanced” sections.
• Inline helpers (“Explain this,” “Show me the underlying XML,” “What does this enum mean?”) are surfaced in context instead of burying explanations in separate docs.

3.7 Accessibility & Ergonomics

The UI is expected to be usable during long sessions and across different devices:

• High-contrast default theme suitable for reading dense code and logs.
• Minimum hit targets:
– Mobile: at least 44 × 44 px for tappable elements.
– Desktop: comfortable click targets for frequently used controls.
• Keyboard shortcuts for core flows (open, build, run diagnostics, next/previous issue, toggle view).
• Adjustable text sizes without layout collapse, especially for the JPE and diagnostics text.

4. Asset Types & Specifications

4.1 Layouts & Screen Flows

Required layout families:

Desktop:
• Project Browser:
– Lists projects, mods, and key entities with search, filters, and status indicators.
• Dual-Pane Editor:
– Configurable panes for JPE, JPE-XML, and XML.
– Synchronized selection and navigation between panes.
• Diagnostics / Problems Pane:
– Grouped by severity, file, or entity type.
– Filter and search capabilities.
• Build History & Logs:
– Chronological list of builds, status, and quick links to diagnostics.
• Settings & Plugin Management:
– Clear separation between global preferences, project settings, and plugins.

Mobile (iPhone):
• Home / Projects Tab:
– Project list with status indicators.
• Detail View for Mods / Entities:
– JPE-first view with optional underlying structure previews.
• Diagnostics Tab:
– List of errors/warnings with quick filters.
• Build / Activity Tab:
– Recent builds and status.

Artboard baselines:
• Desktop:

– 1440 × 900 baseline, scalable up to 1920 × 1080.
• Mobile (iPhone):
– 390 × 844 baseline, with clear safe zones for OS chrome.

4.2 Iconography & Controls

Icon families:

Application & Navigation:
• App logo and wordmark.
• Primary navigation icons (Projects, Editor, Diagnostics, Builds, Settings).

Editor & View Controls:
• View toggles (JPE, XML, JPE-XML).
• Split-view controls (horizontal/vertical, link/unlink scrolling).
• Diff / compare icons.

File & Asset Types:
• XML tuning, .package, .ts4script/.py, STBL, JSON/config.

Status & Diagnostics:
• Error, warning, info.
• “Original” vs “Generated” badges.
• Synced vs offline indicators.

Specifications:
• Vector source: all icons authored as SVG.
• Size grid: 16, 20, and 24 px bases, with 2× and 3× raster exports where needed.
• Stroke style: consistent weight (e.g., 1.5–2 px at 24 px) for a cohesive icon set.
• Readability: icons must remain distinguishable at small sizes; avoid excessive detail.

4.3 Branding Assets

Brand assets include:

• Logo lockups:
– Horizontal and stacked variants.
– Light-on-dark and dark-on-light variants.
• Splash / Welcome Screen:
– Branded header with project name and brief description.
– Recent projects list and key actions (Open project, New project, Docs, Support).
• Application Icons:
– Windows ICO bundle (16, 32, 48, 64, 128, 256 px).
– macOS ICNS set (16 up to 1024 px).
– iOS app icon master (1024 × 1024) with derived sizes for all required slots.

Branding rules:
• Maintain consistent clear space around the logo.
• Avoid placing the logo over visually noisy backgrounds that reduce legibility.

4.4 Illustrations & Empty States

Illustrative assets should support key states without overwhelming the interface:

Target screens:
• Empty project list (“No projects yet – drop in Sims 4 mods to get started.”).
• No diagnostics (“No issues found – this build is clean.”).
• Build failed (“Build failed – review the diagnostics list and affected files.”).

Specifications:
• Formats: SVG where possible for crisp scaling; PNG @2× and @3× for raster environments.
• Recommended widths:
– Desktop: 400–600 px.
– Mobile: 240–320 px.
• Style: clean, minimal, and tool-focused. Illustration should clarify context, not distract from text.

4.5 Design Tokens & Theming

Design tokens must exist as a single, authoritative set defined in Penpot and mirrored into the codebase.

Token categories:

Colors:
• Backgrounds: surface, elevated, code/editor background.
• Text: primary, secondary, muted, inverted.
• Status: error, warning, success, info.
• Accents: selection, focus rings, active lines.

Typography:
• Headings: H1–H4 scale with consistent weights and sizes.
• Body: normal and small body text for content and labels.
• Monospace: for code, XML, IR previews, and logs.

Spacing & Layout:
• Base unit: 8 px spacing system.
• Tokens: 4, 8, 12, 16, 24, 32 px for padding, gaps, and margins.

Corners & Elevation:
• Corner radius tokens for inputs, panels, and dialogs.
• Subtle shadows for elevated surfaces, with limited depth levels to avoid visual noise.

All tokens are to be exported from Penpot (e.g., via JSON or clearly documented style definitions) and kept in sync with a frontend theme file so implementations never guess values from screenshots.

5. Definition of “High Quality” for This UI

5.1 Visual Quality

High visual quality means:

• No inconsistent paddings, misaligned grids, or visually “floating” elements.

• Hierarchy is clear at a glance: primary actions vs secondary actions, active panes vs background panes.
• Typography is disciplined: no random font sizes or weights; everything uses the defined scale.
• Icons and text labels work together; icons are not the sole carriers of meaning.

5.2 Interaction Quality

High interaction quality means:

• Critical flows are short and predictable:
– Open mod → Translate → Inspect JPE → Fix diagnostics → Rebuild → Export.
• Hover, focus, pressed, and disabled states are visually distinct and consistent.
• Long-running operations (e.g., big rebuilds) surface progress, remain cancellable where safe, and never silently hang.
• Keyboard navigation is first-class on desktop; common actions have shortcuts and visible hints where appropriate.

5.3 Information Quality

High information quality means:

• Diagnostics are specific, contextual, and actionable.
• JPE text stays consistent in terminology and sentence patterns, avoiding contradictory phrasing for identical concepts.
• Complex constructs (tests, autonomy rules, loot, careers) come with inline JPE explanations and quick links to documentation.
• Users can quickly understand what changed between two builds or two versions of a mod via clear diffs and summaries.

5.4 Production Readiness

Production-ready design means the handoff from design to development is unambiguous:

• Every component has defined states and behaviors.
• Layout specs (margins, gaps, breakpoints) are either represented as tokens or explicitly documented.
• Assets exist at the right sizes and formats; developers never have to upscale or manually redraw assets.
• The same components, tokens, and patterns appear consistently across desktop and mobile implementations.

6. SOP Alignment

This document is aligned with the core SOP and UI/UX PRDs for the JPE Sims 4 Translation Suite. In particular:

• Purpose & Scope:
– The UI must support translation of Sims 4 mod file types into JPE and back to valid tuning, with strong diagnostics.
• Architecture:
– The IR sits at the center; UI views and editors are front-ends over that IR, not independent data silos.
• Engine & Diagnostics:

– All error reporting and exception handling is surfaced through structured diagnostics panels instead of raw stack traces.
• Multi-Platform Strategy:
– Desktop is the primary authoring environment; mobile is optimized for reading, triage, and light editing while on the go.

Any future changes to the SOP that affect UI/UX, diagnostics, or engine behavior must be mirrored here and in the Penpot design system.

7. Banani + Penpot Workflow Details

7.1 From PRD to Banani Prompts

Step 1: Extract screen and component requirements from the UI/UX PRDs.
• For each screen, define:
– Screen name and purpose.
– Must-have regions (e.g., project tree, editor panes, diagnostics list).
– Primary actions and navigation patterns.

Step 2: Create structured prompt/JSON descriptions for Banani.
• Describe the layout verbally (“A three-pane layout with tree on the left, editor in the center, diagnostics on the right, top bar with project switcher and build button.”).
• Provide hints about hierarchy (“Diagnostics pane is prominent and resizable, with filters across the top.”).

Step 3: Use Banani to generate first-pass layouts for:
• Desktop views (per screen).
• Mobile views (per tab/stack).

7.2 From Banani to Penpot

Banani-generated layouts are treated as sketches, not final sources of truth.

Workflow:
• Export or capture the Banani layouts (via supported exports or images).
• Recreate and refine the layouts in Penpot:
– Replace ad-hoc controls with Penpot components.
– Apply the official tokens for colors, typography, and spacing.
– Ensure responsiveness and variant behavior are defined where needed.
• Use Penpot frames as the canonical reference for implementation.

7.3 Design System Ownership in Penpot

Penpot holds the master design system:

• Component library:
– Buttons, inputs, dropdowns, tabs, panels, trees, tables.
– Diagnostics list items, badges, and tags.
– Layout shells (project browser, editor, diagnostics, build history).

• Tokens:
– Colors, typography, spacing, radii, shadows.

• Variants:
– All interactive states (default, hover, focus, pressed, disabled, error).

Developers must always implement from the Penpot system, not directly from Banani outputs or screenshots.

8. Implementation & Handoff Guidelines

To ensure smooth collaboration between design and development:

• Component naming:
– Match Penpot component names with code components (e.g., ProblemsPane, BuildHistoryList, EditorSplitView).
• Token mapping:
– Maintain a mapping file where Penpot token names map directly to frontend theme variables.
• Change management:
– Any token or component change in Penpot must be accompanied by a short change note so developers can keep the implementation in sync.
• Validation:
– As new UI features ship, periodically compare live UI against Penpot and this document to catch drift early.

This closes the loop between SOP, UX intent, free-tool design pipeline, and the actual shipped UI.

With Banani and Penpot replacing proprietary AI design tools, the JPE Sims 4 Translation Suite keeps a fully free and transparent UI/UX pipeline. The principles and specs in this document should be treated as the standard for all future UI/UX work on the suite, across both desktop and mobile surfaces.