



TDATLanguageManager

Deep-Dive Feasibility and Implementation Plan

Design proposal only - no component implementation authorized

Last changed: August 25, 2026

Delphi VCL and FireMonkey - Windows Win32 and Win64

Purpose: determine whether component-first localization can replace automatic target-source rewriting without creating a new reliability problem.

Table of Contents

Appendix C. License Information	22
Table of Contents.....	i
1. Executive Finding.....	1
2. Decision Context.....	1
2.1 The trust problem.....	1
2.2 Required outcome.....	2
2.3 Non-goals.....	2
3. Current Integration Compared with Component Integration.....	2
4. Large-Project Model.....	3
4.1 Generation tracking.....	3
4.2 Complexity.....	4
5. RAD Studio 37 Lifecycle Evidence.....	4
5.1 FireMonkey.....	4
5.2 VCL.....	4
6. Proposed Architecture.....	5
6.1 Recommended unit and class separation.....	6
7. Object Inspector Contract.....	6
8. Public Methods and Events.....	7
9. Lifecycle Algorithms.....	8
9.1 Component loading.....	8
9.2 New form.....	8
9.3 Language change.....	8
9.4 Destruction.....	9
10. VCL Discovery Options.....	9
11. Control State and Application Semantics.....	10
12. Text the Component Cannot Discover Automatically.....	10
13. Language Selection UI.....	11
14. Design-Time Packaging.....	11

15. Safety, Failure, and Trust Model.....	12
16. Performance and Scale Qualification.....	12
17. Controlled Implementation Plan.....	13
Phase 0 - Baseline and branch protection.....	13
Phase 1 - Lifecycle spike.....	13
Phase 2 - Shared manager core.....	14
Phase 3 - FMX production manager.....	14
Phase 4 - VCL production manager.....	14
Phase 5 - Component state hardening.....	14
Phase 6 - Design-time packages.....	15
Phase 7 - Selector integration.....	15
Phase 8 - Studio component mode.....	15
Phase 9 - Real-project migration trials.....	15
Phase 10 - Release decision.....	16
18. Validation Matrix.....	16
19. Migration Without Premature Backtracking.....	17
20. Risk Register.....	17
21. Decision Alternatives.....	18
22. Go / No-Go Criteria.....	19
22.1 Go.....	19
22.2 No-go or redesign.....	19
23. Decisions Requested After Review.....	19
24. Final Recommendation.....	20
Appendix A. Current Source Reuse Map.....	20
Appendix B. Evidence Consulted.....	20

1. Executive Finding

Recommendation. Do not dismantle the working integration system yet. Build a controlled component-first proof of concept beside it. If the proof meets the lifecycle, no-flicker, coexistence, and removal tests in this plan, make component integration the recommended path and retain automatic source integration as an optional advanced path during transition.

A one-manager localization design is technically credible and is a better trust story than allowing the Studio to rewrite a valuable Delphi project. It is especially attractive for applications with dozens or hundreds of forms because the manager can localize forms lazily as they are instantiated. The project may contain 90 forms while only three or four are open; only those live forms need runtime work.

The proposed direction is not a zero-change mechanism. A Delphi executable must link the localization runtime somehow. Dropping a registered nonvisual component causes normal IDE-managed changes to the owner form or data module: a component field, a unit reference, and a persisted component block in the DFM or FMX resource. Those changes are visible, designer-owned, reversible, and materially less invasive than an external program rewriting DPR, DPROJ, Pascal, and form-resource files.

FireMonkey provides suitable lifecycle messages for a manager-only design. RAD Studio 37 exposes messages for form creation, before-show, activation, deactivation, release, and main-form changes. VCL does not expose an equivalent public before-show broadcast. It exposes Screen form collections, an active-form event, and a multicasting TApplicationEvents.OnIdle mechanism. A VCL manager can discover forms without one component per form, but its ability to translate before the first visible paint must be proven rather than assumed.

Most of the existing Studio remains useful. Scanning, catalogs, Google and DeepL translation, validation, JSON packs, preference storage, VCL/FMX property applicators, source-language packs, and state-preservation work are reusable. The principal change would be the target integration layer, not a restart of the whole application.

2. Decision Context

2.1 The trust problem

The present Studio can preview, back up, apply, restore, and completely reset its changes. Those controls reduce technical risk, but they do not fully solve perceived risk. A developer may still be unwilling to let an external program edit an application that has accumulated years of business logic, inherited forms, third-party controls, conditional project settings, and private build conventions.

Adoption depends on both actual safety and understandable safety. A nonvisual Delphi component placed by the developer through the IDE is familiar. Its properties can be inspected in the Object

Inspector, its presence is visible in the form resource, its unit reference is managed by Delphi, and removal follows the normal component workflow.

2.2 Required outcome

- One required manager component per application, not one component per form.
- Automatic localization of existing and later-created forms.
- Immediate language switching for all currently open forms.
- No Studio-authored edits to DPR, DPROJ, PAS, DFM, or FMX files in the recommended workflow.
- Normal designer visibility and Object Inspector configuration.
- Static linking of runtime units by default; no runtime BPL dependency for customers.
- Target application remains completely offline and contains no provider credentials.
- Support for Delphi Windows Win32 and Win64 VCL and FMX targets.
- No persistent idle CPU cost that becomes significant in 90-form applications.
- Predictable removal with no orphaned hooks or hidden project mutations.

2.3 Non-goals

- Intercepting every hard-coded string assignment in compiled Pascal code.
- Automatically translating data returned by databases, APIs, or users.
- Automatically resizing every form for text expansion.
- Guaranteeing full right-to-left layout mirroring.
- Supporting macOS, iOS, Android, or Linux in the initial component packages.
- Replacing human review of machine translation quality.

3. Current Integration Compared with Component Integration

Concern	Current transactional integration	Proposed component-first integration
Who changes the project	The Studio generates and applies approved changes.	The developer drops and configures a component in Delphi.
Files normally affected	DPR, DPROJ, main form source/resource, generated runtime units, packs.	Owner PAS and DFM/FMX through normal IDE streaming; runtime units linked by the component.
Language startup	Generated initialization and per-form ApplyTranslation calls.	Manager loads the preferred pack and discovers participating forms.
Dynamic forms	Require generated/manual ApplyTranslation wiring unless later discovered.	Manager discovers and translates new live forms automatically if lifecycle strategy succeeds.

Concern	Current transactional integration	Proposed component-first integration
Developer trust	Strong technical safeguards, but external rewrite remains visible risk.	Familiar component model, explicit Object Inspector ownership, easy removal.
Large projects	Every static CreateForm call may be edited; dynamic forms need attention.	One manager; work is proportional to live forms, not project form count.
Recovery	Backup, rollback, restore, complete reset.	Remove component and units through IDE; delete packs; Git remains the normal safety net.

Important limitation. Component-first does not mean no project change. It means the change is small, explicit, IDE-managed, designer-visible, and does not require the Studio to parse and rewrite the target project.

4. Large-Project Model

The number of form units in a project is not the main runtime cost. The relevant number is the count and complexity of instantiated forms. A 90-form business application commonly creates the main form and shared data modules at startup, then creates modal or modeless forms only when needed.

The manager should never instantiate forms merely to translate them. It should localize a form after its resource has loaded and before its first useful display whenever the framework permits. It should retain only a non-owning form reference and a generation number. Closed forms are removed through FreeNotification or framework release messages.

```
Project contains 90 forms
-> startup creates 3 forms
-> manager translates those 3 forms
-> user opens Reports later
-> manager translates Reports once
-> language changes
-> manager translates only currently live forms once
```

4.1 Generation tracking

Maintain a dictionary from form instance to the language generation last applied. Generation starts at one and increments after every successful language change. A form is translated only when it is absent from the dictionary or its recorded generation differs from the current generation.

This prevents repeated translation during idle discovery. It also makes immediate switching deterministic: increment generation, apply to open forms, then record the new generation for each successful form.

4.2 Complexity

- Form discovery cost: $O(\text{number of live forms})$.
- Form application cost: $O(\text{number of owned components times supported properties})$.
- Translation lookup: expected $O(1)$ dictionary lookup per stable key.
- Language switch cost: $O(\text{total supported properties across currently live forms})$.
- Closed and never-created forms impose no application cost.

5. RAD Studio 37 Lifecycle Evidence

5.1 FireMonkey

FMX.Forms defines TFormsCreatedMessage, TFormBeforeShownMessage, TFormActivateMessage, TFormDeactivateMessage, TFormReleasedMessage, TAfterCreateFormHandle, and TBeforeDestroyFormHandle. TCommonCustomForm sends the before-shown message from its show path and sends release/destruction messages when the form is torn down.

A FireMonkey manager can subscribe through System.Messaging.TMessageManager. Subscription is additive and does not require taking ownership of the form's OnCreate or OnShow event. The preferred initial application point is TFormBeforeShownMessage because the complete streamed form exists and the user has not yet seen it. TAfterCreateFormHandle is useful for registration but may be too early for application-specific FormCreate assignments. TFormReleasedMessage and FreeNotification provide cleanup.

FMX assessment. A single manager can plausibly cover normal, modal, modeless, dynamically created, and auto-created FireMonkey forms without a component on every form. This is the strongest part of the proposal.

5.2 VCL

Vcl.Forms.TScreen maintains Forms and CustomForms collections. Forms add themselves during TCustomForm initialization and remove themselves during destruction. TScreen exposes OnActiveFormChange, but AddForm and RemoveForm are private and there is no public form-added event comparable to FireMonkey's message stream.

Vcl.AppEvnts.TApplicationEvents provides a multicasting OnIdle implementation. A manager can use it to enumerate Screen.CustomForms and process only unknown generations without taking exclusive ownership of Application.OnIdle. This is safer than assigning Application.OnIdle directly, but first-visible-paint timing must be measured. An idle scan may occur after a form has briefly painted its source-language text.

TScreen.OnActiveFormChange can provide a faster signal, but it is a single event property rather than a multicaster. Saving, chaining, and restoring another component's handler is possible but fragile when several libraries compete for the same property. A low-level Windows CBT or call-window hook could see form creation or showing, but that would introduce exactly the kind of hidden, invasive mechanism the component design is intended to avoid.

VCL assessment. A single manager is feasible for discovery and immediate language changes. The unresolved release question is whether new VCL forms can always be translated before visible paint without event hijacking, a base-form call, or a lightweight per-form marker. This must be answered by a prototype and instrumentation.

6. Proposed Architecture

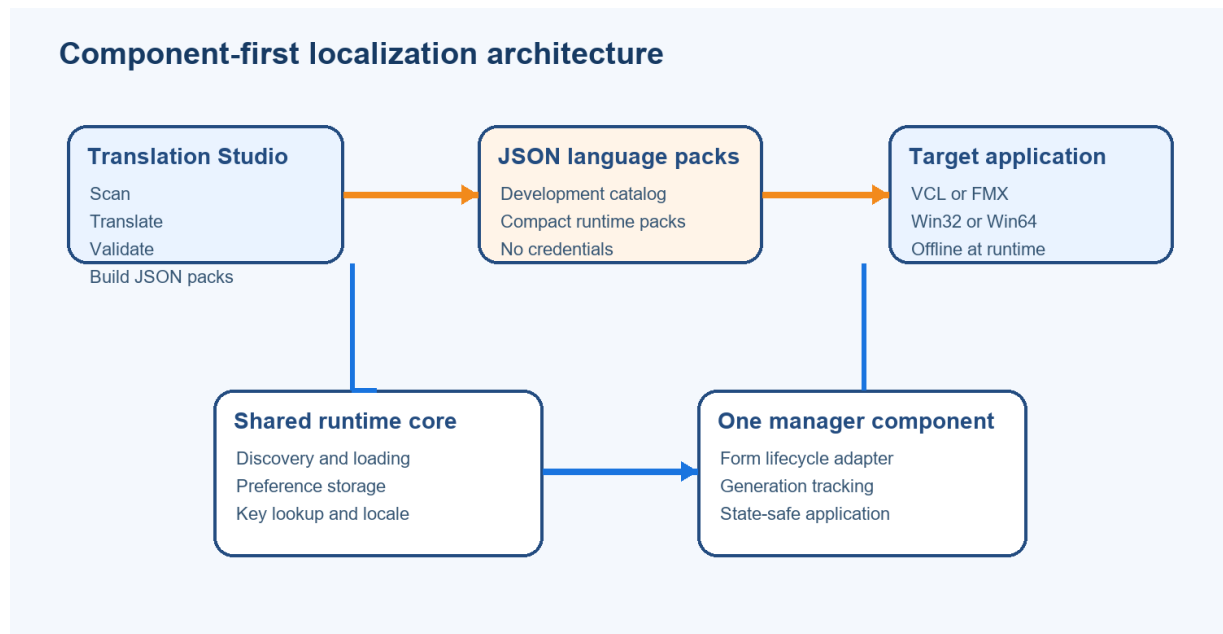


Figure 1. Proposed component-first boundary

The Studio remains responsible for authoring: project scan, development catalogs, Google/DeepL translation, review, validation, and compact runtime-pack creation. The target application receives only JSON packs and localization runtime units.

The existing TTranslationRuntime remains the core pack loader and lookup service. The component layer adds lifecycle management, framework-specific form discovery, language-change notification, state protection, and Object Inspector configuration.

6.1 Recommended unit and class separation

Layer	Proposed unit/class	Responsibility
Shared core	DAT.Components.Core / TDATCustomLanguageManager	Configuration, runtime ownership, generation tracking, events, error policy.
VCL bridge	DAT.Components.VCL / TDATVCLLanguageManager	VCL form discovery, VCL adapter dispatch, optional menu binding.
FMX bridge	DAT.Components.FMX / TDATFMXLanguageManager	FMX message subscriptions, FMX adapter dispatch, optional selector binding.
Design-time VCL	DAT.Design.VCL	Palette registration and VCL property/component editors.
Design-time FMX	DAT.Design.FMX	Palette registration and FMX property/component editors.
Existing runtime	DAT.Runtime.*	JSON discovery/loading, preferences, stable-key lookup, property application.

Using the same class name TDATLanguageManager in two simultaneously installed design-time packages risks class-registration and streaming ambiguity. Distinct public class names are safer. Documentation and palette grouping can still describe both as the DAT Language Manager family.

The shared base must not reference Vcl.Forms or FMX.Forms. Otherwise a VCL application could accidentally link FireMonkey units or vice versa. Framework types belong only in the derived bridge units.

7. Object Inspector Contract

Property	Suggested default	Purpose
ApplicationId	Owner project/app name	Reject packs intended for another application.
LanguagesFolder	Localization\Languages	Path relative to executable unless absolute.
SourceLanguage	en-US	Fallback and source-pack identity.
ActiveLanguage	SourceLanguage	Current language at design/runtime; read-only while loading.
AutoLoadPreferred	True	Read the saved per-user language at startup.
AutoTranslateOwner	True	Translate the owner form after component loading.
AutoTranslateNewForms	True	Discover and translate later-created forms.
ReapplyOpenForms	True	Instantly update live forms after selection.

Property	Suggested default	Purpose
TranslateHiddenForms	False	Prefer lazy before-show application.
PreserveControlState	True	Protect selection and edit state during reapplication.
PreferenceLocation	LocalAppData	Choose LocalAppData, executable folder, or custom path.
PreferenceFileName	language.ini	Persist selected language.
MissingPackBehavior	UseSourceLanguage	Fallback instead of crashing when a pack is absent.
ErrorBehavior	NotifyAndContinue	Report per-form errors without taking down the application.
ExcludedForms	Empty	Form names intentionally omitted from automatic localization.
EnableDiagnostics	False	Optional timing and coverage events; no end-user logging by default.

Object Inspector rule. All durable configuration should be published and designer-editable. Internal dictionaries, message-subscription identifiers, active packs, and generation counters remain runtime-only implementation state.

8. Public Methods and Events

Member	Kind	Contract
Initialize	Method	Load configuration and preferred language; safe to call more than once.
SelectLanguage(Code)	Method	Validate/load pack, persist preference, increment generation, reapply live forms.
ApplyToForm(Form)	Method	Explicit escape hatch for special form factories or tests.
ApplyToOpenForms	Method	Reapply current pack to all eligible live forms.
AvailableLanguages	Method	Return validated pack descriptors; caller owns returned list under existing contract.
Translate(Key, Fallback)	Method	Translate resourcestring and dynamic application text explicitly.
OnLanguageChanging	Event	Cancelable notification before active pack changes.
OnLanguageChanged	Event	Notification after successful pack load and form reapplication.
OnFormTranslated	Event	Form, translated-property count, elapsed time, and generation.
OnTranslationError	Event	Form/key/error context and handled flag.

Member	Kind	Contract
OnMissingTranslation	Event	Optional diagnostics only; avoid per-key overhead unless enabled.

9. Lifecycle Algorithms

9.1 Component loading

1. During construction, initialize internal collections only. Do not access Screen, Application, or files while csDesigning or csLoading is set.
2. In Loaded, resolve relative paths, validate that only one active manager exists for the framework, and subscribe to framework lifecycle notifications.
3. Load the saved language. Missing or malformed non-source packs invoke the configured error policy and fall back to the source language.
4. Translate the owner form only after its streamed components exist. Do not invoke user events while the component is still loading.
5. Enumerate currently existing forms once so forms created before the manager are not missed.

9.2 New form

1. Receive the framework signal or discover the form in the live-form collection.
2. Reject nil, destroying, design-time, excluded, popup-internal, or already-current-generation forms.
3. Wait until component streaming is complete. For FMX, prefer the before-shown message. For VCL, use the validated prototype strategy.
4. Set a reentrancy guard and apply translations on the UI thread.
5. Record generation, property count, and optional timing; register FreeNotification where appropriate.

9.3 Language change

1. Validate and fully load the candidate pack before disturbing the active pack.
2. Raise OnLanguageChanging and honor cancellation.
3. Swap the active pack atomically on the UI thread and update pack-specific TFormatSettings.
4. Increment generation and translate a stable snapshot of eligible live forms.
5. Persist preference only after successful pack activation.
6. Raise OnLanguageChanged after reapplication completes; isolate individual form errors according to policy.

9.4 Destruction

- Unsubscribe every FMX message by its subscription identifier.
- Release or destroy any owned VCL TApplicationEvents bridge.
- Remove forms through Notification and release events; never free a form owned by the application.
- Clear the singleton registration only if it still points to this component.
- Do not write preferences or touch UI from destruction paths.

10. VCL Discovery Options

Option	Advantages	Risks	Recommendation
TApplicationEvents. OnIdle inventory	Public, familiar, multicasting, no source rewrite, sees dynamic forms.	May run after first paint; requires zero-cost generation skip; direct Application.OnIdle assignments elsewhere can bypass multicaster.	Primary prototype.
Screen.OnActiveFormChange chaining	Fast signal for user-facing forms.	Single event slot, ordering conflicts, may occur after show, misses nonactive forms.	Supplement only if coexistence is proven.
Common localized base form	Deterministic lifecycle and no polling.	Requires forms to inherit or be reparented; unsuitable for arbitrary existing projects.	Excellent opt-in for projects already using a base form.
One Apply call in central form factory	Deterministic and tiny change.	Only works when the project already centralizes form creation.	Documented advanced option.
Per-form localizer component	Deterministic and designer-visible.	Tedious for 90 forms and easy to omit.	Exceptional forms only.
Windows CBT/call-window hook	Could observe form creation/showing early.	Low-level, hidden, Windows-message fragile, difficult to coexist and debug.	Reject for v1.
Current generated ApplyTranslation edits	Already working and deterministic for known forms.	External source rewrite and dynamic-form coverage concerns.	Keep as optional fallback during transition.

Release rule. Do not advertise VCL manager-only localization as flicker-free until an instrumented test proves translation completes before visible paint for normal, modal, modeless, inherited, MDI, and dynamically created forms.

11. Control State and Application Semantics

Immediate localization changes presentation while the application is running. It must not be allowed to masquerade as user input. The current runtime already replaces indexed text in place, preserves ItemIndex, and temporarily suppresses OnChange for translated Items and Lines collections. The manager must retain and expand that discipline rather than merely call property setters more often.

Preservation should be control-specific and conservative. The manager should not attempt to snapshot every published property. It should protect only state known to be disturbed by translation and covered by tests.

Control/state	Required behavior	Notes
Combo/list ItemIndex	Preserve exact semantic index; suppress selection event.	Already covered by VCL/FMX regression tests.
Editable text	Do not translate runtime user data.	Scanner/runtime must distinguish captions/prompts from user-entered Text.
Memo Lines	Translate only cataloged design-time content; preserve caret/selection when editable.	Current generic Lines support needs editable-memo tests.
Date/time controls	Do not replace Date/Time values; update display formatting only through explicit locale support.	The recent date-range bug demonstrates why state and text must be separated.
Grid headers	Translate header captions without clearing row data or changing current cell.	Requires explicit adapter coverage; not implied by generic Text RTTI.
Tabs/pages	Translate captions without changing active page.	Test TabIndex/ActivePage preservation.
Menus/actions	Translate visible text without breaking action links or shortcuts.	Accelerator validation remains necessary.

12. Text the Component Cannot Discover Automatically

The component can automatically apply cataloged designer properties because they have stable form/component/property keys. It cannot reliably intercept arbitrary assignments such as `lblStatus.Caption := 'Finished'` or `lblStatus.Text := 'Finished'`. Those statements execute after compilation and may overwrite a translation at any time.

Pascal resourcestring entries and dynamically created control text continue to require an explicit manager Translate call. This is not a component defect; it is the safe boundary between declarative form resources and application logic.

```
lblStatus.Text := DATLanguageManager.Translate(
  'Messages.ReportComplete',
```

```
SReportComplete);
```

- The Studio should continue scanning resourcestring declarations and labeling them Manual: Translate call required.
- The component must expose the same stable-key lookup as the generated integration unit.
- No heuristic should rewrite arbitrary literals or insert calls automatically.
- Application code that intentionally changes captions after localization remains the developer's responsibility.

13. Language Selection UI

The manager and the visible selector should be separate concerns. The manager owns language discovery and selection. The developer should remain in control of whether the application uses a menu, combo box, toolbar button, or settings page.

A first component release should not silently construct a complete menu bar. It may optionally populate the children of a designer-authored Language menu selected through an Object Inspector component-reference property. Dynamic child creation is justified because installed JSON packs can change after compilation, but it must be narrow, documented, and opt-in.

- Option A: developer-owned menu item assigned to LanguageMenu; manager creates only pack-derived children.
- Option B: dedicated TDAVCLLanguageComboBox and TDAFMXLanguageComboBox visual controls for applications that prefer a selector.
- Option C: developer handles AvailableLanguages and SelectLanguage manually for a fully custom UI.
- Do not require the Studio to insert File/Exit or redesign an application's menu structure.

14. Design-Time Packaging

Design-time registration and runtime code should be separated. The IDE package references DesignIDE and registers components on a DAT Localization palette page. Target applications reference only runtime units. Unless the developer intentionally builds with runtime packages, the compiled application should not require a DAT BPL on the customer's computer.

The design package must be built for the architecture and package interface expected by the installed RAD Studio IDE. Target runtime units must compile separately for Win32 and Win64. Source distributions should include package projects, runtime source, installation instructions, uninstall instructions, and version-compatibility notes.

Artifact	Contains	Customer deployment
DATRuntimeCore	Language pack, preference, manager core interfaces.	Statically linked by default.
DATRuntimeVCL	VCL applicator and VCL manager bridge.	Inside VCL executable.
DATRuntimeFMX	FMX applicator and FMX manager bridge.	Inside FMX executable.
DATDesignVCL	Register procedure and VCL design editors.	IDE only; never shipped to end user.
DATDesignFMX	Register procedure and FMX design editors.	IDE only; never shipped to end user.
JSON packs	Source and translated runtime text plus locale metadata.	Localization\Languages beside executable.

15. Safety, Failure, and Trust Model

- Runtime performs no Internet access. Google/DeepL credentials remain in the Studio's Windows Credential Manager entry and never enter component properties or packs.
- Malformed or wrong-application packs are rejected before activation.
- A failed non-source pack falls back to the source language according to policy; it should not prevent the application from starting by default.
- All UI property updates occur on the main UI thread.
- Reentrancy guards prevent a control event or language event from starting a nested translation pass.
- Form references are non-owning and removed on notification/release.
- The manager must not retain user data, form snapshots, provider responses, or API keys.
- Diagnostics should redact paths or sensitive application values when sent outside the local process.
- Multiple manager instances should produce a clear design-time warning and deterministic runtime failure/fallback rather than competing subscriptions.

Trust statement. In component-first mode, the Studio generates packs but does not alter target source. Delphi itself persists the developer-placed component. This should become the principal product message if the proof succeeds.

16. Performance and Scale Qualification

No production performance number should be promised until measured. Source-scan timing does not predict runtime application timing. The component proof should include both synthetic stress forms and real VCL/FMX projects.

The following are qualification targets, not current claims. They can be adjusted after baseline measurement, but the release must publish measured results and test-machine details.

Scenario	Qualification target	Failure signal
Idle application with no new forms	No repeated property traversal; negligible CPU use.	Continuous scans or measurable idle CPU growth.
Medium form	Translation completes before visible paint and without perceptible delay.	Source-language flash, blocked UI, event side effects.
Large form	Bounded main-thread pause with timing diagnostics available.	Multi-second unexplained freeze or data-state loss.
90 form definitions, 5 live	Cost tracks 5 live forms, not 90 units.	Instantiation or loading of unopened forms.
90 live forms language switch	One pass per form; progress/busy policy considered if measured duration is visible.	Repeated generations, reentrancy, excessive memory.
1,000 create/destroy cycles	Stable tracked-form count and memory use.	Stale pointers, dictionary growth, shutdown AV.

17. Controlled Implementation Plan

Scope control. These phases are a proposed sequence only. Approval of this document does not authorize implementation. The current working integration remains intact until the proof passes its gates.

Phase 0 - Baseline and branch protection

- Tag or otherwise record the last known-good provider-only implementation and its Win32/Win64 test results.
- Freeze the current JSON schemas and runtime-pack contract for the proof.
- Create an isolated component proof branch/worktree; do not modify the real pilot applications initially.
- Record the exact current source mutations performed by automatic integration for later comparison.

Exit: reproducible baseline, clean status, no production behavior changed.

Phase 1 - Lifecycle spike

- Build minimal nonvisual FMX and VCL manager prototypes using existing runtime adapters.

- Instrument construction, Loaded, FormCreate, before-show, OnShow, activation, paint, and destruction order.
- Test auto-created, dynamic, modal, modeless, inherited, popup, MDI, and ownerless forms as applicable.
- Reject VCL strategies that overwrite exclusive events, require low-level Windows hooks, or visibly flash source text.

Exit: written evidence identifies a reliable manager-only lifecycle for each framework, or clearly records the VCL limitation.

Phase 2 - Shared manager core

- Factor configuration, runtime ownership, language generation, exclusion rules, error policy, and events into a framework-neutral base.
- Enforce main-thread access and reentrancy guards.
- Add non-owning form-generation tracking with deterministic removal.
- Keep VCL and FMX units out of the shared core uses list.

Exit: core unit tests pass without either visual framework linked.

Phase 3 - FMX production manager

- Subscribe to FMX before-show and release messages using stored subscription IDs.
- Translate owner, pre-existing forms, and later forms exactly once per generation.
- Verify no conflict with existing application events or FireMonkey services.
- Test source-language return, restart persistence, and immediate switching.

Exit: one FMX manager covers the complete FMX lifecycle test matrix with no per-form components.

Phase 4 - VCL production manager

- Implement only the lifecycle strategy proven in Phase 1.
- Use TApplicationEvents multicasting rather than assigning Application.OnIdle directly if idle inventory is selected.
- Provide an explicit ApplyToForm escape hatch and documented base-form/factory option for exceptional projects.
- Do not claim flicker-free automatic coverage until paint-order tests pass.

Exit: one VCL manager covers the agreed matrix, or the product clearly labels the minimal additional VCL integration requirement.

Phase 5 - Component state hardening

- Retain combo/list selection and OnChange suppression tests.

- Add editable memo, date/time, tab, grid-header, action/menu, and third-party-control tests.
- Introduce adapter-specific state guards only when a reproducible defect proves they are needed.
- Verify application-assigned runtime text is not silently mistaken for designer text.

Exit: language changes preserve application semantics across the supported control matrix.

Phase 6 - Design-time packages

- Create separate VCL and FMX design-time packages with distinct registered class names.
- Publish all durable configuration in the Object Inspector.
- Add package install, rebuild, upgrade, and uninstall instructions.
- Verify dropping and removing a component produces only normal Delphi designer changes.

Exit: clean IDE install/uninstall and Win32/Win64 target builds without runtime BPL dependency.

Phase 7 - Selector integration

- Implement manager APIs first; keep visual selection optional.
- Prototype binding to an existing designer-authored Language menu.
- Populate only pack-derived children and preserve developer-owned menu items.
- Offer custom UI methods so no menu structure is forced on the application.

Exit: language choices reflect installed packs without Studio-authored menu-file edits.

Phase 8 - Studio component mode

- Add a non-mutating Component Integration plan that generates packs and installation/configuration instructions.
- Do not remove automatic integration; label component mode Recommended and automatic mode Advanced during evaluation.
- Teach validation to distinguish pack readiness from component installation status.
- Update Complete Reset semantics so it never removes developer-placed components automatically.

Exit: Studio can complete scan-to-pack workflow without touching target source.

Phase 9 - Real-project migration trials

- Use disposable copies of Website Analytics and a representative real VCL project.
- Restore each target to a known pre-integration baseline before adding the component through the IDE.
- Compare translated coverage, startup, dynamic forms, language switching, and removal against current integration.
- Obtain developer visual review of every form category and record exceptions.

Exit: component mode matches or exceeds current coverage in one real FMX and one real VCL application.

Phase 10 - Release decision

- Run Debug and Release Win32/Win64 builds and all component/runtime/integration tests.
- Choose component-first hybrid, retain current integration only, or defer based on evidence.
- Regenerate User Guide, Engineering Guide, Help, source distribution, and release notes only after the decision.
- Keep migration reversible and do not remove proven integration code in the same commit that introduces the replacement.

Exit: explicit developer approval based on measured evidence, not architectural preference.

18. Validation Matrix

Area	Required cases	Pass condition
Platforms	VCL/FMX x Win32/Win64 x Debug/Release	Clean builds; target executable starts; no design package deployed.
Form lifecycle	Auto-created, dynamic, modal, modeless, inherited, hidden, recreated, MDI/popup where applicable	Correct language before user interaction; one application per generation; clean destruction.
Startup order	Manager on main form; manager on data module created first; forms existing before manager	All eligible forms eventually translated with documented timing and no exception.
Language switch	English-Spanish-French-English; repeated same language; missing/malformed pack	No restart; state preserved; fallback deterministic; preference correct.
Controls	Labels, buttons, menus, lists, combos, tabs, memos, prompts, dialogs, grids, actions	Supported properties translate; data and selections unchanged.
Scale	100 form definitions; 1/5/25/90 live; 10,000 catalog entries	No unopened form creation; measured duration and memory within accepted budget.
Coexistence	Existing TApplicationEvents, Screen active-form handler, timers, third-party controls	No handler loss, double dispatch, or shutdown errors.
Removal	Delete component in IDE; remove package; rebuild	Application returns to normal source-language behavior without hidden dependency.
Security	Wrong appld, path traversal attempt, corrupt JSON, unwritable preference path	Rejected/fallback safely; no credential or network dependency.

19. Migration Without Premature Backtracking

The component investigation should be additive. The existing Studio currently translates and integrates a real FMX pilot successfully. Removing proven integration code before a replacement survives real-project testing would create avoidable risk and would make comparison harder.

The component proof should reuse current runtime packs and adapters so both integration modes can be tested against identical language data. Only after coverage and lifecycle behavior match should the Studio's default workflow change.

1. Preserve the current main branch and release artifacts.
2. Develop components in isolation against samples.
3. Run parallel component and current-integration tests using the same packs.
4. Migrate disposable copies of real projects, never the only working source tree.
5. Make component mode the default only after explicit approval.
6. Retain automatic integration as an advanced fallback for at least one transition release.
7. Remove old code only in a later, separately reviewable cleanup after component adoption is proven.

Existing subsystem	Disposition during proof
Project scanner and catalog merge	Preserve unchanged.
Google/DeepL providers and credential storage	Preserve unchanged.
Validation, JSON/CSV, suggestions	Preserve unchanged.
Runtime pack and preference units	Reuse; extend only behind tests.
VCL/FMX applicators	Reuse and harden for component lifecycle.
Transactional integration	Keep available; do not make the proof depend on it.
Restore and Complete Reset	Preserve for existing integrated projects; redefine carefully for component mode.

20. Risk Register

Risk	Impact	Mitigation	Gate
VCL form translated after first paint	Visible language flash; poor quality.	Instrument paint order; prefer validated lifecycle; document base-form/factory fallback.	Release blocker for automatic VCL claim.
Application event collision	Lost handlers or broken third-party behavior.	Use additive message/multicaster APIs; coexistence tests; reject exclusive hijacking.	No handler loss in stress tests.

Risk	Impact	Mitigation	Gate
Runtime text overwritten after translation	Mixed-language UI.	Explicit Translate API and lifecycle documentation; optional reapply only at controlled events.	Known-boundary documentation and sample.
State changed during reapply	Wrong report/filter/data operation.	Control-specific preservation and event suppression tests.	Zero semantic-state regressions.
Design package installation burden	Lower adoption despite safer runtime.	Clear installer/build instructions; source and prebuilt package options where licensing permits.	Independent install/uninstall test.
Class-name/package conflict	Streaming or IDE registration errors.	Distinct VCL/FMX class names and package namespaces.	Both packages installed together.
Two managers installed	Duplicate translation and subscriptions.	Design-time warning; singleton validation; deterministic error policy.	Duplicate-manager test.
Frames/inherited components missed	Partial translation on large apps.	Recursive ownership/key tests against scanner output; explicit frame coverage matrix.	Catalog-to-runtime coverage audit.
Backtracking removes working path too early	Project loses proven integration.	Additive branch, side-by-side tests, delayed removal.	Separate approval for retirement.

21. Decision Alternatives

Alternative	Benefits	Costs	Assessment
Keep current integration only	Already working; deterministic startup and known safeguards.	External source-rewrite concern may limit adoption; dynamic forms require wiring.	Safe short-term baseline.
Component-first hybrid	Developer-controlled integration; one manager; current path remains fallback.	New package/lifecycle work; VCL timing proof required.	Recommended investigation.
Component-only immediately	Simple marketing message and no automated project edits.	Removes working fallback before proof; higher migration risk.	Not recommended.
Per-form component	Deterministic lifecycle.	Unacceptable manual work and omissions in 90-form projects.	Special cases only.
No runtime integration; instructions only	Studio never touches target.	Every developer must write and maintain integration manually.	Too little product value.

22. Go / No-Go Criteria

22.1 Go

- One manager translates every tested FMX form before display with additive lifecycle subscriptions.
- One manager translates VCL forms without visible source-language flash or destructive event ownership, or a clearly acceptable one-time base-form/factory integration covers the remaining gap.
- Win32 and Win64 Debug and Release targets build with runtime units statically linked.
- Existing Google/DeepL packs work unchanged.
- Language changes preserve selections, dates, tabs, grids, user text, and application events.
- A 90-form/large-catalog stress test shows cost proportional to live forms and stable memory.
- Removing the component through the IDE leaves no runtime dependency or hidden target modification.
- A real FMX and a real VCL application pass developer visual review.

22.2 No-go or redesign

- VCL requires a low-level Windows hook or repeated visible repaint to appear automatic.
- The component must overwrite application-wide events used by other libraries.
- Manager discovery causes persistent idle CPU or stale-form references.
- Coverage is materially worse than current integration for frames, inherited forms, menus, or dynamic forms.
- Design-time installation becomes more difficult than the risk it removes.
- The component cannot be removed cleanly or introduces mandatory customer BPL deployment.

23. Decisions Requested After Review

No answer is required until this plan has been printed and reviewed. If the investigation proceeds, the following decisions should be made explicitly before implementation begins.

1. Approve or reject an isolated component proof of concept.
2. Confirm that installing a Delphi design-time package is acceptable for the recommended workflow.
3. Confirm that one manager component on the main form or main data module is acceptable.
4. Confirm that automatic source integration should remain available as an advanced fallback during transition.
5. Set the VCL quality bar: no visible source-language paint should be the default requirement.
6. Choose whether an optional designer-authored menu binding belongs in the proof or a later phase.

7. Choose the first real VCL pilot application after sample validation.

24. Final Recommendation

The component idea is not a retreat from the product's central value. It is a proposed replacement for the least trusted part of the workflow: target-source integration. Approximately the entire authoring side of the Studio can remain intact.

The recommended direction is a component-first hybrid, developed additively and evaluated against the working implementation. FireMonkey appears well suited to a one-manager design. VCL deserves a focused lifecycle proof before any promise is made. If that proof succeeds, the Studio can offer a much stronger adoption message: generate and validate language packs in the Studio, then enable them through one Delphi-managed component rather than allowing the Studio to rewrite the application.

No production code should be removed, migrated, or rewritten until the proof meets the go criteria and the developer explicitly approves the change in direction.

Appendix A. Current Source Reuse Map

Current unit area	Use in component direction
DAT.Runtime.LanguagePack	Preserve; JSON loading, discovery, indexed strings.
DAT.Runtime.Manager	Preserve and place behind component ownership; may gain observable state/events.
DAT.Runtime.Preference	Preserve; expose storage choice through component properties.
DAT.Runtime.VCL / FMX	Preserve; add lifecycle-safe application and broader state tests.
DAT.Scan.*	Preserve; no runtime component dependency.
DAT.Provider.*	Preserve in Studio only; never link into target runtime.
DAT.Validation.*	Preserve; add component-mode readiness checks later.
DAT.Integration.*	Keep during proof; later become optional advanced integration.

Appendix B. Evidence Consulted

- Current Delphi App Translation Studio runtime, scanner, integration, validation, tests, project files, and engineering notes at main commit d6a6578 plus the approved selection-preservation work.

- RAD Studio 37 source: C:\Program Files (x86)\Embarcadero\Studio\37.0\source\fm\FMX.Forms.pas.
- RAD Studio 37 source: C:\Program Files (x86)\Embarcadero\Studio\37.0\source\fm\FMX.ApplicationEvents.pas.
- RAD Studio 37 source: C:\Program Files (x86)\Embarcadero\Studio\37.0\source\vcl\Vcl.Forms.pas.
- RAD Studio 37 source: C:\Program Files (x86)\Embarcadero\Studio\37.0\source\vcl\Vcl.AppEvnts.pas.
- Observed Website Analytics FMX pilot behavior, including provider translation, pack deployment, immediate switching, preference restart, and date-range state correction.

Evidence boundary. This is a design and feasibility plan, not proof that the proposed components work. Lifecycle conclusions marked unresolved require compiled prototypes and visual tests before implementation approval can become a product commitment.

Appendix C. License Information

Unless a particular file or third-party component states otherwise, Delphi App Translation Studio and its project documentation are licensed under the Apache License, Version 2.0. The license permits broad use while preserving attribution, notices, and the stated warranty limitations.

License grant

Subject to the complete license terms, you may use, reproduce, modify, prepare derivative works from, publicly display, publicly perform, sublicense, and distribute the licensed material. The Apache License 2.0 also includes an express patent license from contributors for their applicable contributions.

Important conditions

- Give recipients a copy of the Apache License 2.0 when distributing covered material.
- State significant changes made to modified files.
- Retain applicable copyright, patent, trademark, attribution, and NOTICE information.
- Do not use project or contributor names and trademarks as an endorsement except as the license permits.

Warranty and liability

The licensed material is provided on an AS IS basis, without warranties or conditions of any kind. The complete Apache License 2.0 controls the warranty, liability, contribution, redistribution, and patent provisions.

Your applications and generated output

Using the Studio does not transfer ownership of the Delphi application being translated. The application developer remains responsible for the license and distribution terms of the target application, translated content, generated language packs, and any third-party components. Studio runtime or component source included with a project remains subject to the Apache License 2.0 unless its file states different terms. Third-party software retains its own license.

Full legal text

The complete controlling license is the LICENSE file in the repository and source distribution root. An official reference copy is available at <https://www.apache.org/licenses/LICENSE-2.0>. If this summary and the complete license differ, the complete Apache License 2.0 text controls.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this work except in compliance with the License. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an AS IS BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.