Setup Wizard Guide

A Detailed, Screen-by-Screen Path from Delphi Source to Offline Language Packs

Version 1.0

Last changed: September 3, 2026

Windows • Delphi VCL and FireMonkey • Win32 and Win64

Table of Contents

1. Purpose, Audience, and Final Result

Welcome to the Translation Setup Wizard. This guide walks beside you through all eight steps, explains what each choice means, and tells you what to look for before you move on. Use it for a first VCL or FireMonkey translation, for a later update after source changes, or when you want to add another language.

You do not need to memorize the whole process. Follow the screens in order on your first run, and return to the individual step sections whenever you need a reminder. Each section includes the complete screen, a closer view of the important controls, and a plain-language explanation of what happens next.

During a normal run, the Wizard reads your selected Delphi project, creates or merges its development catalog, translates only eligible unresolved material, opens Localization Review, validates the result, exports the canonical source and translated packs, prepares the component integration kit, atomically installs one project-local dependency, builds Win32 Release with a temporary Search Path, and directly deploys packs to the folders you approved. It does not rewrite target Pascal, DPR, DPROJ, DFM, or FMX files.

Your finished application remains offline. Only the Studio contacts DeepL or Google, and it does so from the developer computer while translation work is under way. The application you ship reads validated local JSON packs and never receives the provider API key.

> **Start with a safe copy.** Keep a pristine backup and perform the first localization run on a separate test copy of the target project. This gives you an easy way back while you learn the workflow. The Wizard also creates its own timestamped safety ZIP before final processing.

Completion product	What it is	Where it is used
Development catalog	The full editable key/source/translation/context/review model.	Studio workspace under %LOCALAPPDATA%.
Canonical source pack	The validated English or other authored-source runtime pack.	Beside every target EXE under Localization\Languages.
Translated pack	The validated compact target-locale runtime pack.	Loaded by the VCL/FMX runtime manager.

Completion product	What it is	Where it is used
Development catalog	The full editable key/source/translation/context/review model.	Studio workspace under %LOCALAPPDATA%.
Localization Review package	HTML review, findings, glossary, layout proposals, and decisions.	Studio export\localization-review.
Component integration kit	Complete framework-appropriate runtime/component source, packs, manifest, README, deployment script, and report.	Studio export\component-integration.
Safety ZIP	Timestamped pre-processing backup of the selected project.	Reported in Wizard progress and completion report.

2. Before Opening the Wizard

A few minutes of preparation makes the Wizard much easier to use. Complete the checks below once, then keep this section nearby as your setup checklist. The current Beta source ZIP and matching guides are published together at https://delphitranslate.github.io/downloads.html.

2.1 Obtain and build the Studio

1. Download or clone the complete repository from https://github.com/delphitranslate/delphitranslate.github.io. Do not download isolated PAS, DPK, BPL, or JSON files.
2. Extract it to a short writable path, for example C:\DelphiProjects\Delphi App Translation.
3. Open DelphiAppTranslationStudio.dproj in RAD Studio 13 Florence and build Win32 Release for the simplest first run.
4. The verified RAD Studio environment is C:\Program Files (x86)\Embarcadero\Studio\37.0\bin\rsvars.bat and the Win32 compiler is dcc32.exe in the same folder.
5. Run bin\Win32\Release\DelphiAppTranslationStudio.exe from the repository copy you just built.

2.2 Prepare the target application

1. Create a pristine backup and a separate test copy. Build and briefly run the test copy before localization work begins.
2. Build the DAT runtime and design packages supplied with the Studio. In RAD Studio choose Component > Install Packages > Add and install the matching Win32 Release design BPL.
3. Open the target primary form in the Form Designer. Place one TDATVCLLanguageManager or TDATFMXLanguageManager and one matching language combo box.
4. In Object Inspector set ApplicationId to the exact Delphi project name, LanguagesFolder to Localization\Languages, SourceLanguage to the authored locale, and connect the combo box LanguageManager property.
5. Add and position any visible Language label or menu item in the designer. Choose File > Save All so the first scan sees every intended static string.
6. Choose File > Save All and close the running target application before Wizard final processing. The Wizard verifies this explicitly at Review and Authorize; the RAD Studio project may remain open because its files are read-only to the Studio.

> **Keep the setup visible in Delphi.** The supplied manager and selector are normal designer components. Place and configure them in the Form Designer and Object Inspector so another developer can see and maintain the setup later. The Wizard does not inject hidden runtime UI construction or silently rewrite the form.

2.3 Provider and network readiness

- DeepL is the recommended default. Use a DeepL API Free or API Pro key, not merely a consumer Translator subscription.
- Google Cloud Translation Basic v2 is supported when the API is enabled, billing/quota are valid, and the key restrictions permit this developer computer.
- A stored key is kept in Windows Credential Manager. A session-only key disappears when the Studio closes.
- Test the provider connection before final processing. Do not treat a timeout, 401/403, or 429 result as a successful setup.

3. Wizard Navigation, Keyboard, and Safety

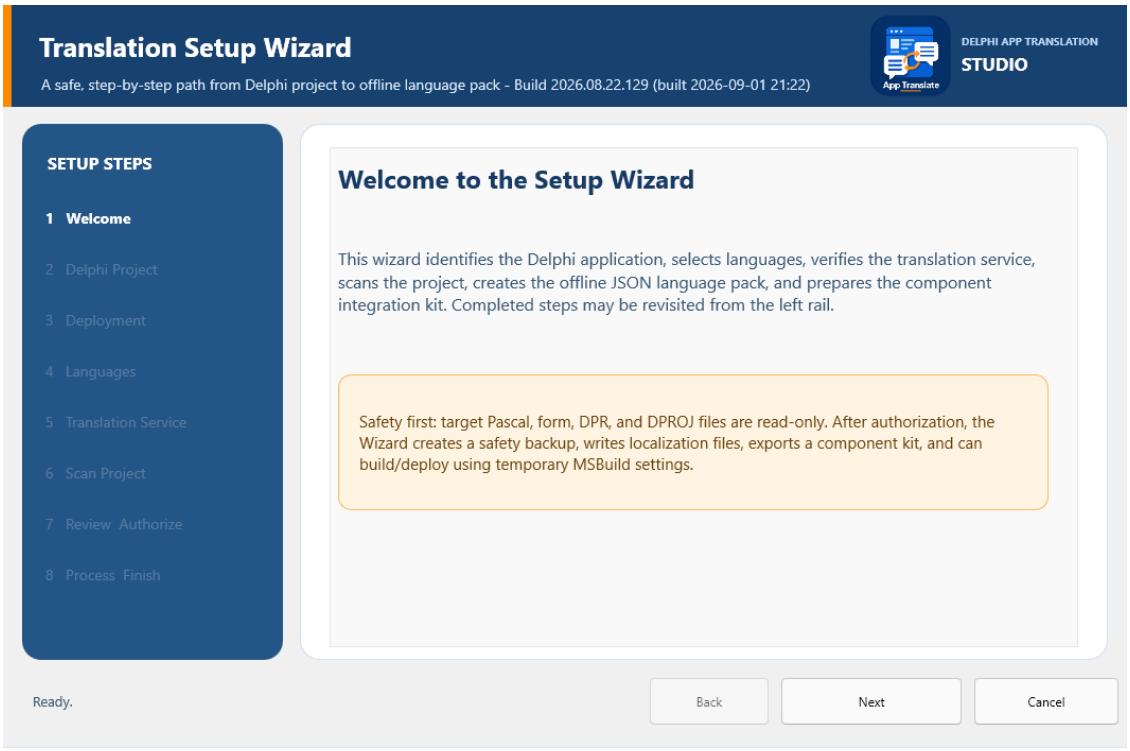

Figure 3-1. Complete Wizard frame and navigation controls.

Control or area	Purpose	Required operator action
Left step rail	Shows the eight-step sequence and the current step.	Select only a previously reached step; unreached steps remain disabled.
Next	Validates the current step and advances.	Press Enter when Next is the default button.
Back	Returns to the previous reached step.	Use before final processing; it is hidden or disabled when backward movement is unsafe or meaningless.
Cancel	Closes an ordinary Wizard session or a stopped run.	Press Esc before final processing. After successful completion use Finish instead.

Control or area	Purpose	Required operator action
Begin Final Processing	Replaces Next on Review and Authorize.	Select only after the saved-files/running-application check and authorization check are true.
Finish	Closes a successful completed run.	Press Enter after reading the completion status.
Footer status	Reports the last completed action and next requirement.	Read it whenever a button remains disabled.

You stay in control until processing begins. Before you select Begin Final Processing, Cancel leaves the target unchanged. Once processing starts, the Wizard briefly blocks unsafe navigation while the active operation finishes or stops safely. After a successful run, Back and Cancel disappear because Finish is the only action you need.

4. Step 1 - Welcome

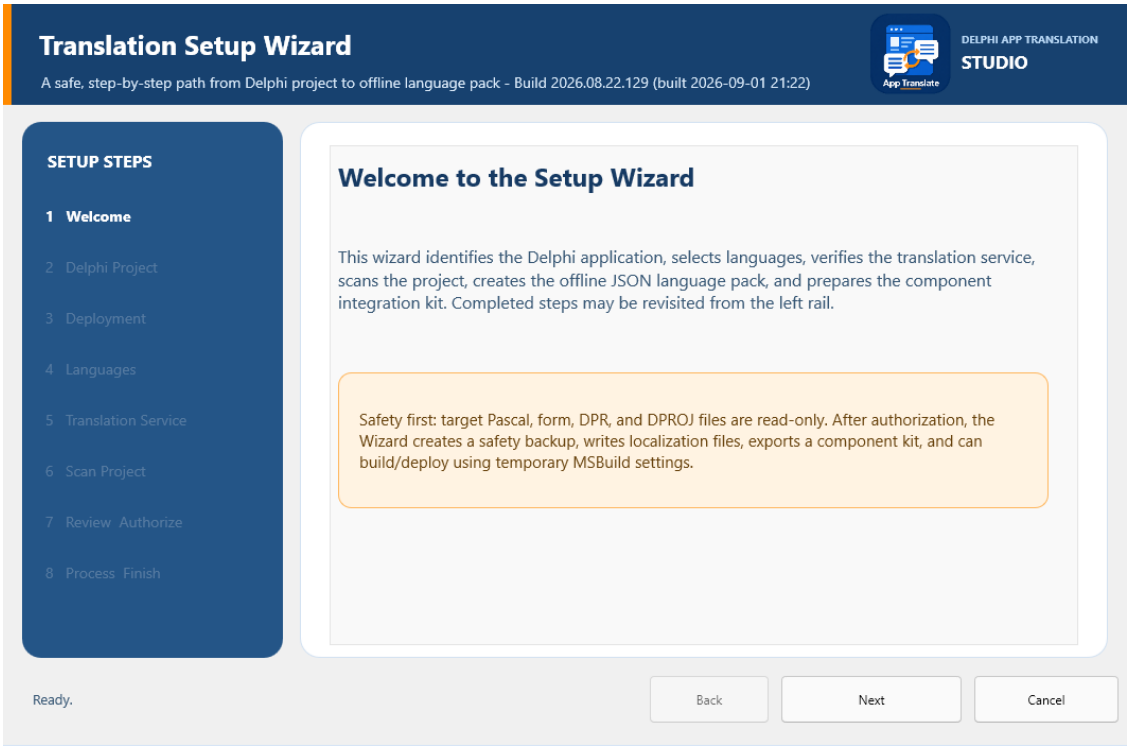

Figure 4-1. Step 1 - Welcome.

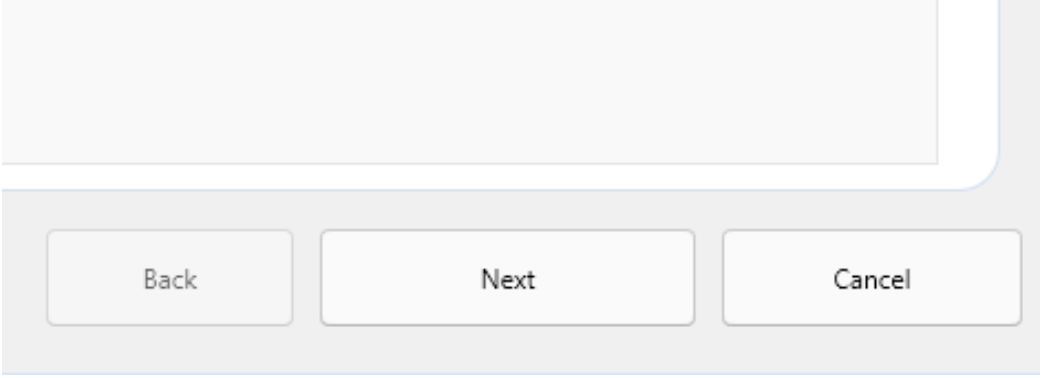

Figure 4-2. Welcome-page navigation buttons.

The Welcome page is a quiet starting point. Nothing has been scanned, translated, backed up, exported, or deployed yet, so you can read the overview without affecting a project.

The header carries the genuine App Translate product icon from the project Images folder and the Delphi App Translation Studio name. The icon is embedded in a designer-owned TImage so the same approved branding appears on every Wizard page without depending on an external file at runtime.

Read the safety notice, then press Enter or select Next. The notice explains that final processing creates a backup and Studio-owned artifacts while your target Pascal, form, DPR, and DPROJ files remain read-only.

Control or area	Purpose	Required operator action
Welcome explanation	Summarizes project identification, languages, provider verification, scanning, packs, and component integration.	Confirm that this is the intended guided workflow.
Safety notice	Defines the read-only target-source contract and required backup.	Do not continue if the selected target copy is not disposable or protected.
Next	Opens Delphi Project.	Default action; Enter should activate it.

5. Step 2 - Delphi Project

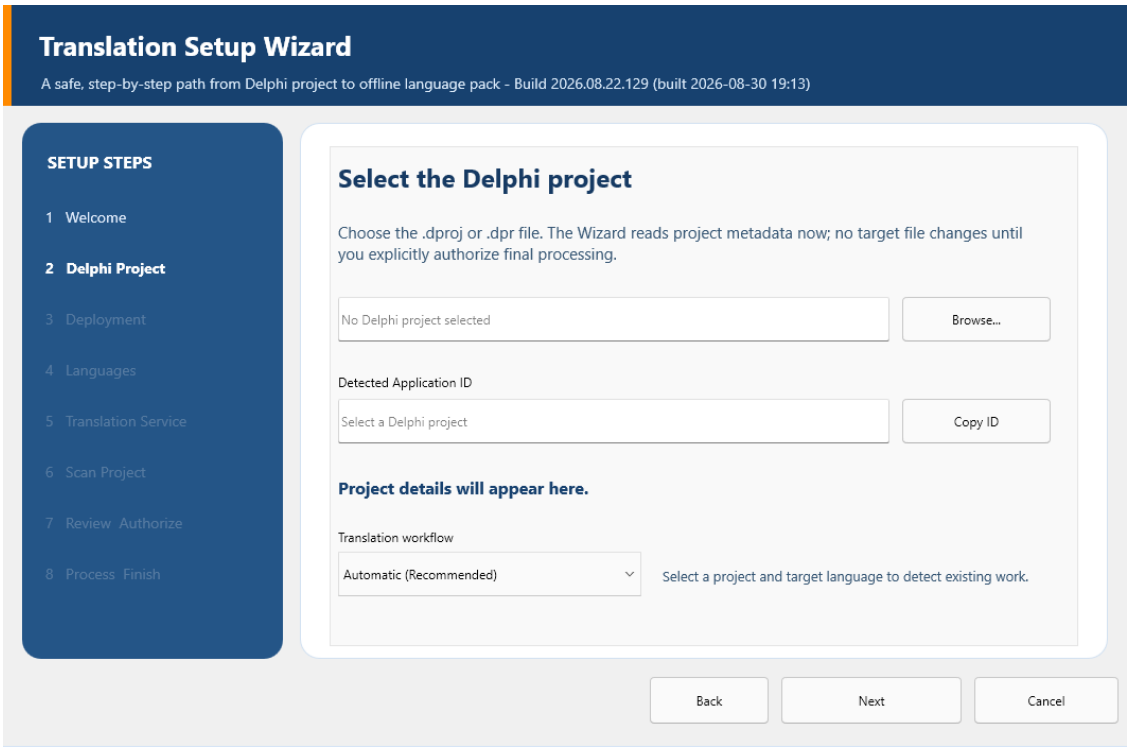

Figure 5-1. Step 2 - Delphi Project.

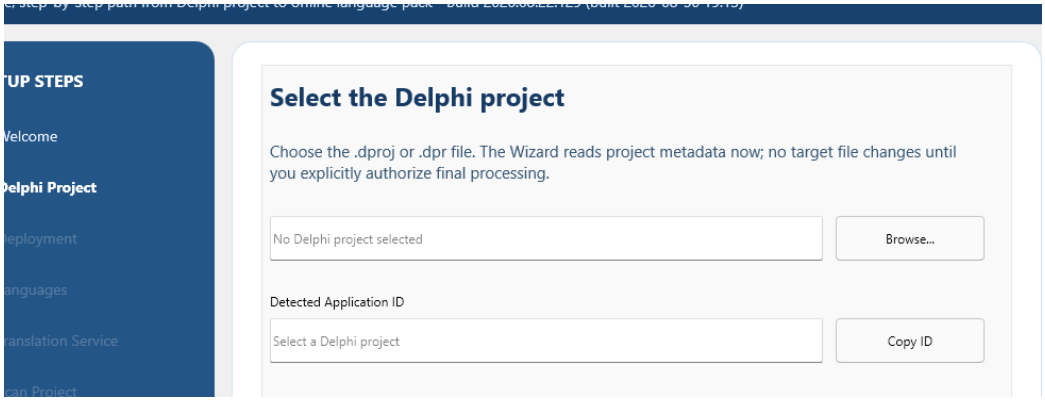

Figure 5-2. Project selection and detected identity.

This is where the Wizard learns which application you want to translate. Select Browse and choose the project's DPROJ file when one is available. A DPROJ gives the Wizard the clearest framework, platform, configuration, source, form, and output information. Detection is read-only; it does not compile or run the target.

Pause for a moment after detection and check the project name, framework, targets, forms, and source-unit count. ApplicationId is especially important: it is the stable identity shared by the catalog, runtime

packs, manager, workspace, and saved language preference. It must match the manager's ApplicationId property exactly.

Control or area	Purpose	Required operator action
Project path	Holds the selected DPR/DPROJ.	Verify the test copy path character by character.
Browse	Opens the project-file picker and reruns detection.	Choose the target DPROJ whenever available.
Application ID	Displays the detected stable application identity.	Copy it into the target manager ApplicationId property exactly.
Copy ID	Copies ApplicationId to the clipboard.	Use when configuring Object Inspector.
Framework / targets	Reports VCL or FMX and Win32/Win64 availability.	Stop if the framework is wrong or expected targets are missing.
Form resources / source units	Reports project discovery breadth.	A suspiciously low count normally means the wrong or incomplete project was selected.
Next	Accepts detected identity and opens Deployment.	Enabled only after a supported project is identified.

6. Step 3 - Deployment

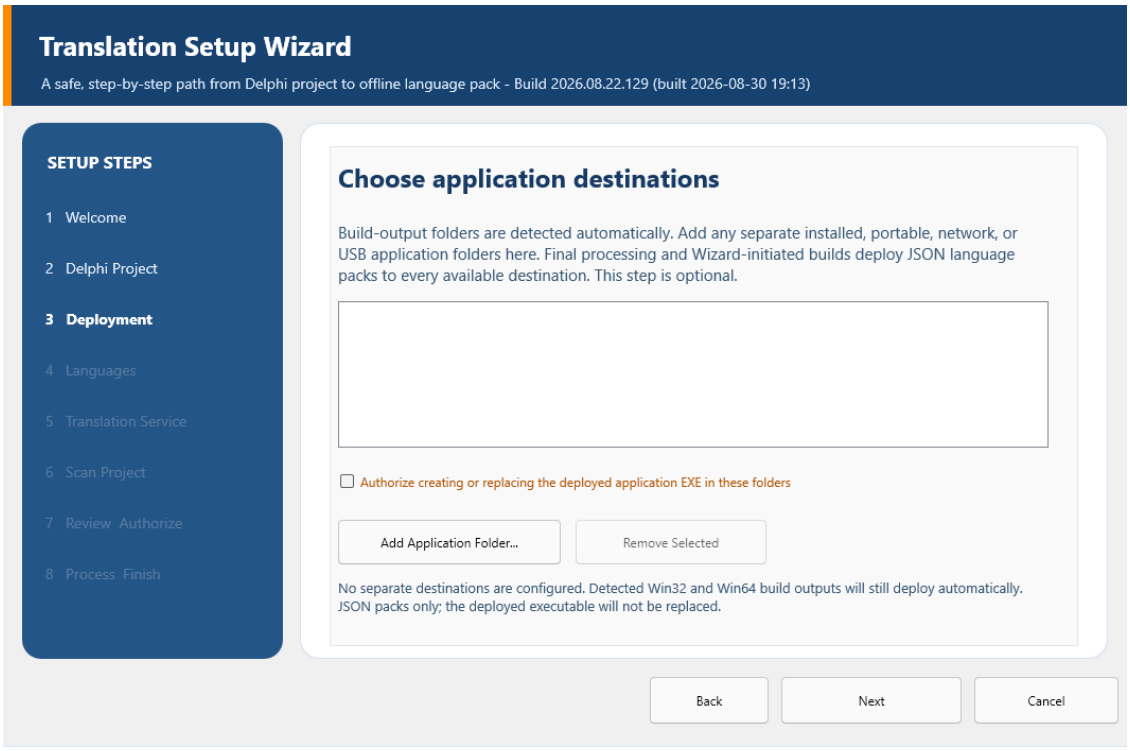

Figure 6-1. Step 3 - Deployment.

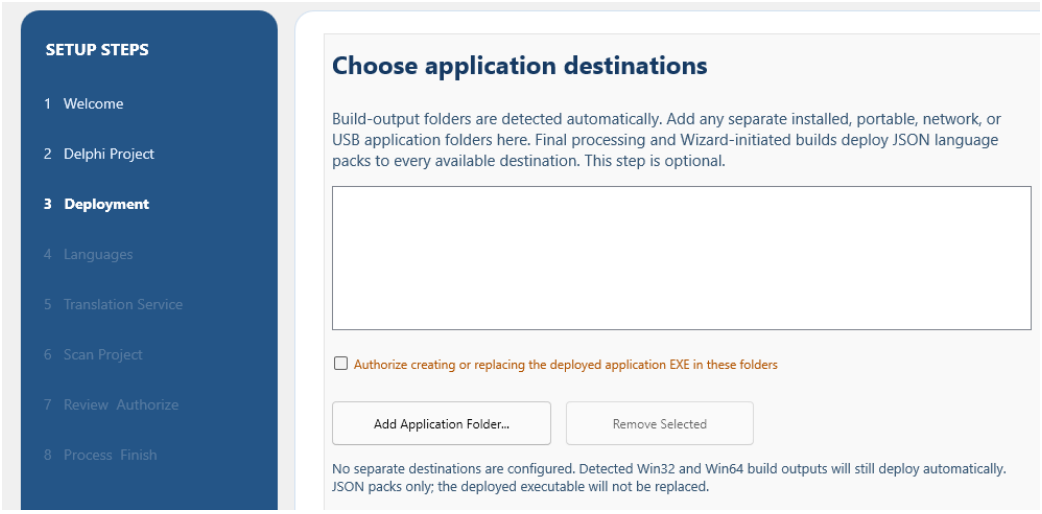

Figure 6-2. Optional destination controls.

Most developers can leave this page alone. The Wizard already detects normal Delphi build-output folders, so you only need to add a destination for a separate installed, portable, network, USB, or test-run copy of the application.

When you do add one, choose the folder that contains the executable - not its Localization or Languages child folder. If a removable or network destination is unavailable later, the Wizard reports and skips it without invalidating the language pack.

Control or area	Purpose	Required operator action
Detected outputs	Shows build-output folders inferred from the project.	Verify they correspond to the configurations you intend to run.
Destination list	Stores additional authorized application folders.	Leave empty when normal build outputs are sufficient.
Add Application Folder	Adds one executable-containing folder.	Use the full path, including drive letter.
Remove Selected	Removes an incorrect or obsolete optional destination.	Select the row first.
Deployment summary	Explains what will be deployed and when.	Read before continuing; no source file is modified.

7. Step 4 - Languages

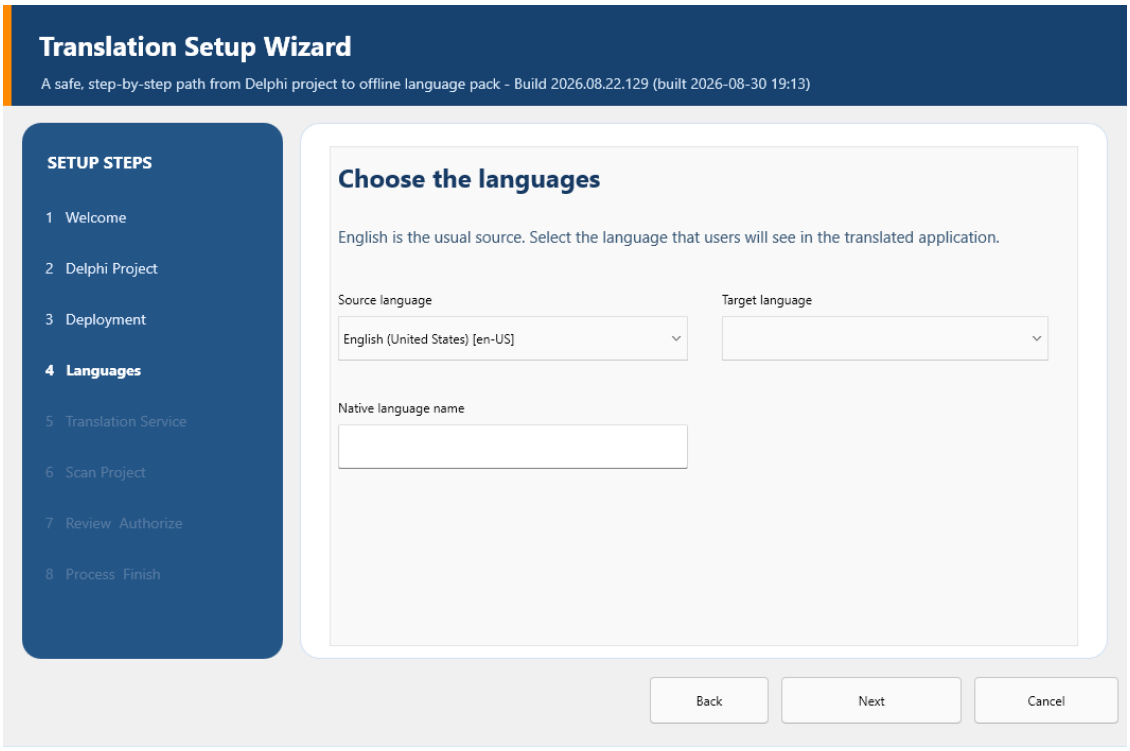

Figure 7-1. Step 4 - Languages.

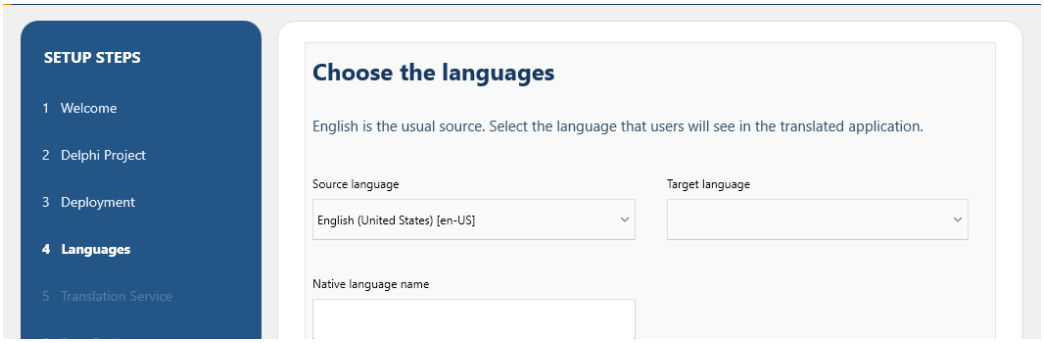

Figure 7-2. Source and target locale selection.

Choose the language already written in the forms and source as the Source language, then choose the new language you want to create as the Target language. The locale code also controls the pack name, native display name, reading direction, and regional formatting facts shown on the page.

Check the native name and direction before continuing. If you change the project or either language later, the Wizard clears the downstream scan state for this session. That protective reset keeps a catalog built for one application or locale from being exported as another.

Control or area	Purpose	Required operator action
Source language	Identifies the authored source locale.	Normally English (United States) [en-US] for an English Delphi application.
Target language	Selects the pack to create or update.	Choose one locale for this Wizard run.
Native name	Shows the end-user language label.	Verify spelling and script.
Direction	Reports LTR or RTL.	Arabic, Hebrew, and Urdu must report RTL.
Locale facts	Shows date/time, separators, currency, and related regional metadata.	Review for the intended region rather than only the base language.

8. Step 5 - Translation Service

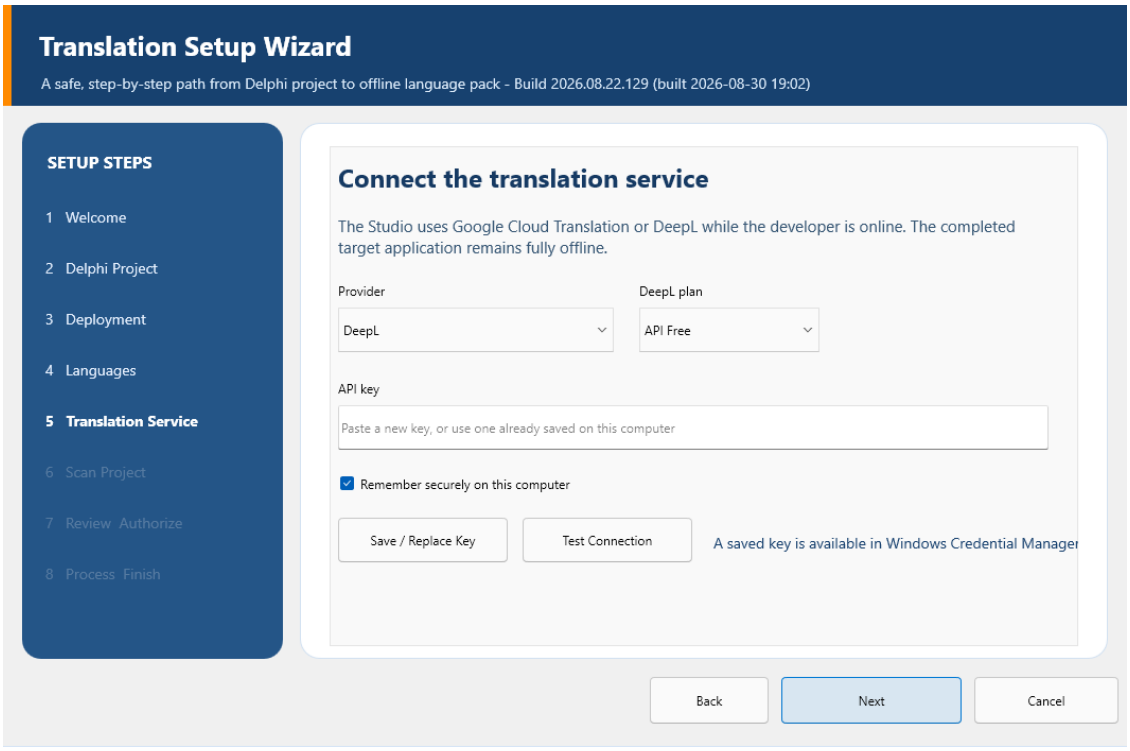

Figure 8-1. Step 5 - Translation Service.

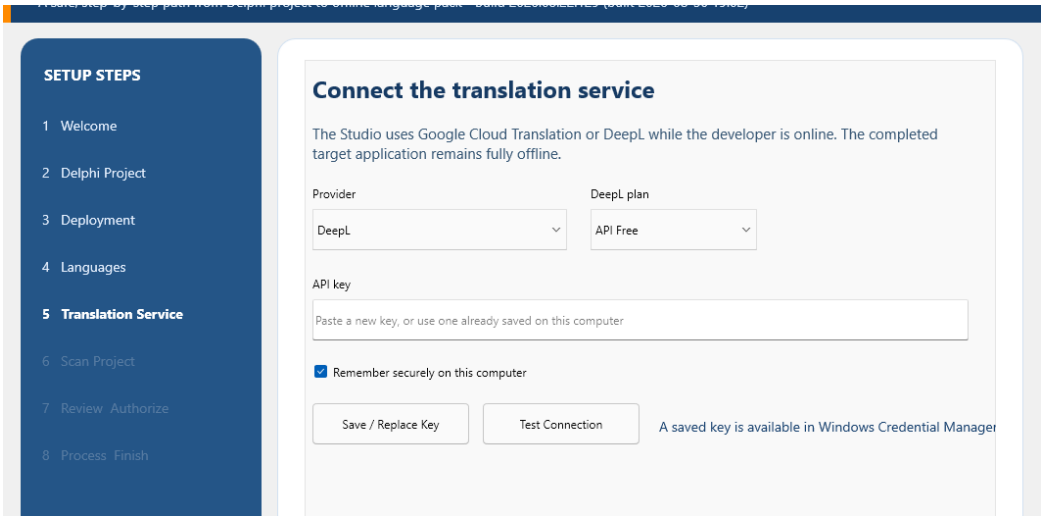

Figure 8-2. Provider, key, and connection controls.

DeepL is the recommended first choice and the first-run default. If you previously saved a provider and plan, the Studio restores those choices. For security, it never puts a stored secret back into the API-key box, so a blank box beside a saved-key message is normal.

If this is a new or replacement key, paste it, choose whether to remember it securely, and select Save / Replace Key. Then select Test Connection. A successful test confirms that the key and endpoint work; it does not guarantee translation quality or enough remaining quota for a large catalog.

Control or area	Purpose	Required operator action
Provider	Selects DeepL or Google Cloud Translation.	Use DeepL unless project/account requirements call for Google.
DeepL plan	Selects API Free or API Pro endpoint.	Match the key's actual account plan; ignored for Google.
API key	Accepts a new or replacement secret and masks it.	Leave blank when the saved-key status says a usable credential exists.
Remember securely	Selects Windows Credential Manager instead of session-only memory.	Use only on a trusted developer computer.
Save / Replace Key	Stores provider settings and the entered secret according to the remember policy.	Use after pasting or rotating a key.

Control or area	Purpose	Required operator action
Test Connection	Performs one small provider request with timeout/cancellation.	Continue only after a success result.
Credential status	Reports saved/session availability without revealing the key.	Use it to distinguish an empty edit from a missing credential.

Your key is not stored in the project. Non-secret settings are in %LOCALAPPDATA% \DelphiAppTranslationStudio\provider-settings.json. A remembered key is stored as a Windows Generic Credential named DelphiAppTranslationStudio/Providers/DeepL or DelphiAppTranslationStudio/Providers/Google Cloud Translation. It is never copied into the target source, catalog, pack, kit, or guide.

9. Step 6 - Scan Project

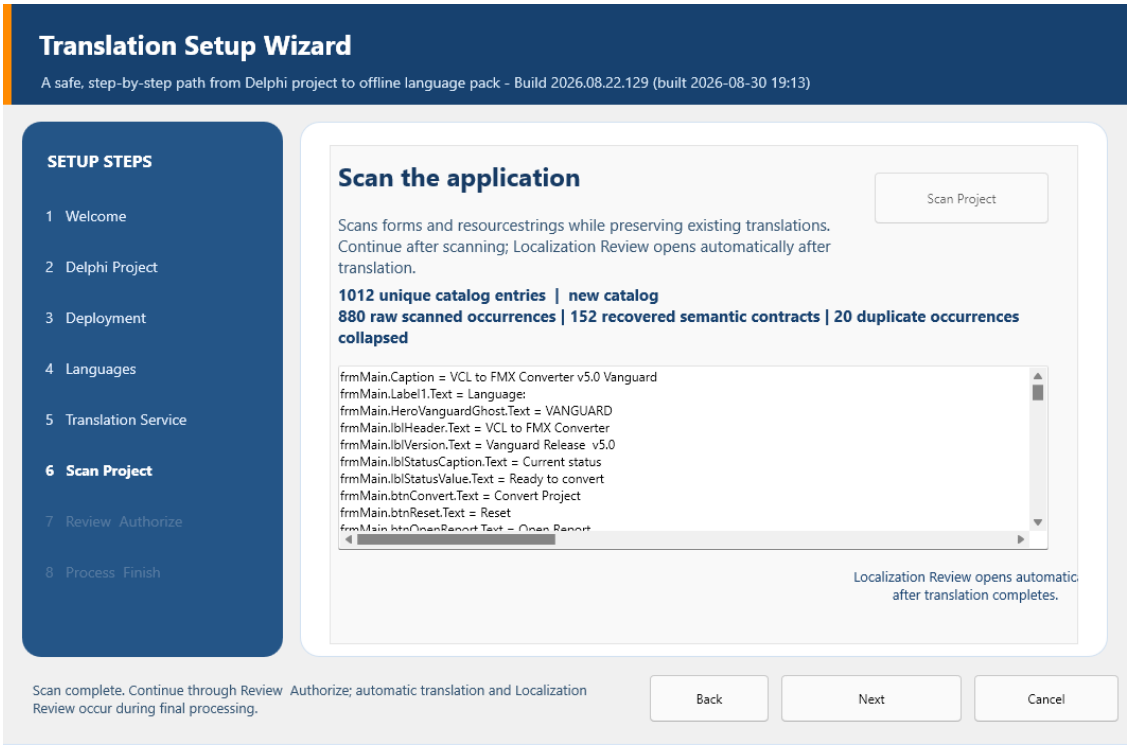

Figure 9-1. Step 6 - Scan Project.

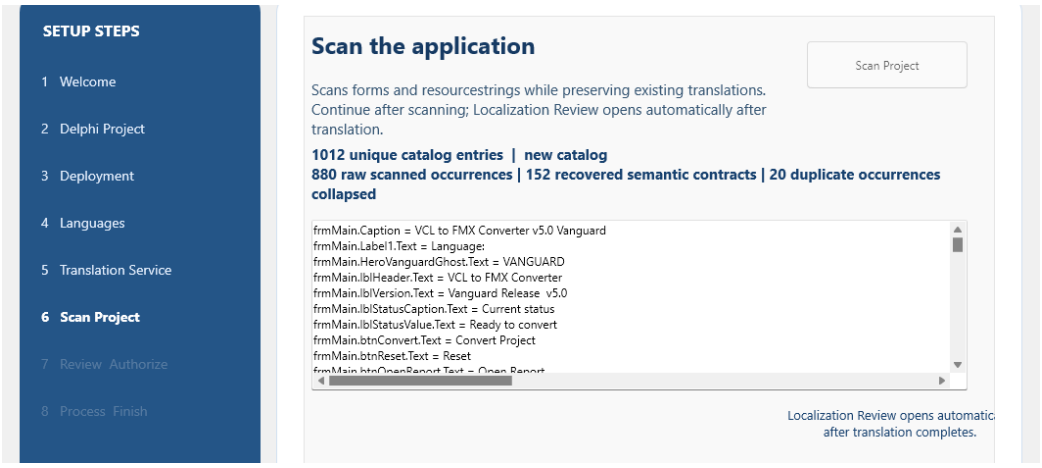

Figure 9-2. Canonical scan-count explanation.

Select Scan Project and let the Wizard inventory the text that belongs in the translation catalog. The Wizard and Maintenance Studio use the same canonical scanner, so the unique catalog-entry total should agree when both are looking at the same saved project snapshot.

The supporting numbers explain how the total was formed. Raw observations are direct discoveries, recovered semantic contracts restore known dynamic application text, and equivalent duplicate occurrences are represented once. The scanner reads saved text DFM/FMX resources, Pascal resourcestrings, eligible runtime assignments, and explicitly authorized project resource manifests; it does not sweep up arbitrary identifiers, user data, database values, or every string literal in source code.

Control or area	Purpose	Required operator action
Scan Project	Runs canonical project discovery and scan.	Use after every saved source/form text change.
Unique catalog entries	The stable-key records that enter catalog merge.	Use this as the Wizard/Maintenance comparison count.
Raw observations	Direct scanner observations before recovery/collapse.	Use to explain how the unique total was formed.
Recovered semantic contracts	Known dynamic/application text restored through stable semantic rules.	Review when runtime text coverage changes.

Control or area	Purpose	Required operator action
Duplicate occurrences collapsed	Equivalent repeated observations represented once.	A nonzero value is expected when identical ownership contracts repeat.
Scan list	Shows stable key and source text.	Inspect representative forms, components, properties, resourcestrings, and dynamic contracts.

10. Step 7 - Review and Authorize

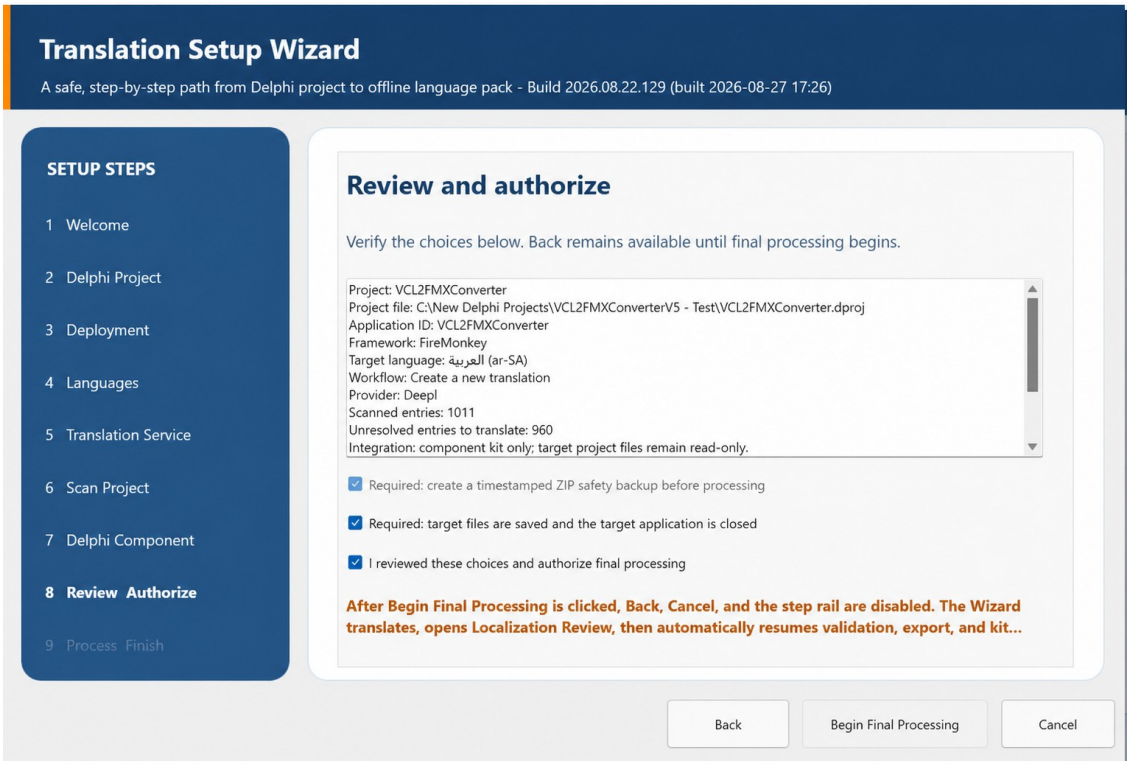

Figure 10-1. Step 7 - Review and Authorize content area.

This page gives you one last chance to confirm the whole job before anything time-consuming begins. Read the summary from top to bottom and make sure it names the exact test project, framework, source and target locales, provider, scan state, unresolved work, backup, and component workflow you intended.

Choose File > Save All in RAD Studio and close the running target application before you authorize the run. The project itself may remain open because the Wizard does not write its DPROJ or form files.

Control or area	Purpose	Required operator action
Review memo	Prints the complete proposed operation in plain text.	Read the path, application ID, locale, provider, counts, output, and backup line by line.
Target files saved; application closed	Confirms the target files are saved and the running target executable will not lock a rebuild.	Choose File > Save All, close the running target application, then select it. The RAD Studio project may remain open.
Safety backup	Requires the pre-processing ZIP.	Leave enabled; final processing will not start without it.
Authorization	Records that the developer reviewed the operation.	Select only after all summary values are correct.
Begin Final Processing	Starts backup, translation, review, validation, export, kit generation, and deployment.	This is not a generic Next button; it crosses the controlled execution boundary.

11. Localization Review During Final Processing

After provider translation, the Wizard opens Localization Review automatically. Take your time here: this is where you review machine output, glossary candidates, application-owned strings, layout findings, and language-specific proposals while all translated text is available.

When your decisions are saved, close Localization Review normally. You are not abandoning the run. Control returns to the waiting Wizard, which continues with validation, export, kit generation, deployment, and the completion report.

Review area	Engineering meaning	Developer decision
Findings	Information, warning, and high-risk localization issues.	Resolve high-risk issues before treating the language as release-ready.

Review area	Engineering meaning	Developer decision
Project glossary	Approved source/target terminology and matching rules.	Add, edit, delete, and save only product-correct terms.
Suggestions	Translation-memory or terminology proposals with confidence/context.	Accept, approve high-confidence, or reject explicitly.
Layout proposals	Conservative width/height/font/wrap/position changes scoped to locale and control.	Accept only after inspecting current versus proposed geometry and ownership.
Code-positioned controls	Controls whose geometry is intentionally assigned in Pascal.	Leave them alone unless the application design is changed deliberately.
Review package	HTML and JSON audit artifacts for later review.	Generate/open it when a printable or browser-based audit trail is useful.
Close	Returns accepted decisions to Wizard processing.	Close only after saving the decisions that should enter the pack.

Layout decision rule. Use natural word wrapping first, then intelligent hyphenation for long unbroken words, then a modest readable font reduction only when necessary. Never shift neighboring columns or controls merely to make one translated heading fit.

11.1 Maintain the glossary later without rerunning the Wizard

Localization Review is the right place to make terminology decisions while a Wizard run is already in progress. It is no longer the only way to maintain those decisions. After the run, open Maintenance Studio and select Glossary directly below Translate on the left rail. The standalone page opens the same authoritative per-project, per-language glossary and does not rerun setup, scanning, or provider translation.

The saved file is %LOCALAPPDATA% \DelphiAppTranslationStudio\Workspaces\<ApplicationId>\Glossaries\<ApplicationId>.<locale>.glossary .json. Select the target language, add or revise the source term and preferred translation, record optional semantic-concept and developer-note information, choose case sensitivity and approval deliberately, then choose Add / Update Term followed by Save Glossary.

Choose Apply and Deploy after saving terminology. Maintenance Studio loads the matching development catalog automatically, applies approved matches only to entries that are not already Reviewed or Approved, validates and saves the catalog, regenerates the affected external JSON runtime pack, refreshes the managed dependency pack, and deploys the complete verified pack set to existing build outputs and remembered destinations. It does not call a provider, rebuild or replace an executable, or edit Delphi source/forms/project files. A completed Wizard run for this application and target language is required once so the matching catalog and managed dependency exist.

Control or area	Purpose	Required operator action
Cancel Edits	Reloads the last saved glossary after a confirmation.	Use only to discard every pending change for the selected target language.
Apply and Deploy	Publishes saved approved terminology without rerunning the Wizard.	Read the resulting catalog/pack/destination report, then restart the target application so it reloads the changed external pack.
Maintenance Cancel / Esc	Requests the normal safe exit from Maintenance Studio.	Unsaved glossary work receives Save, Discard, or Cancel; active scans/provider requests first reach a safe cancellation boundary.

12. Step 8 - Process and Finish

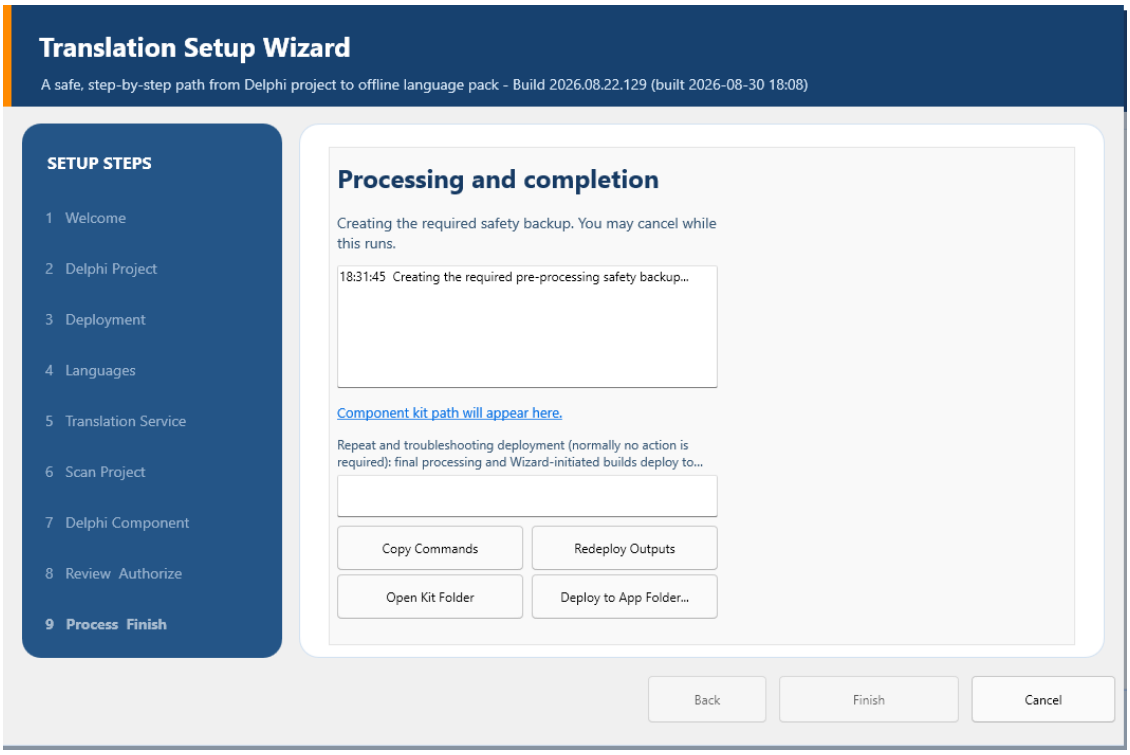

Figure 12-1. Step 8 - Process and Finish status area.

You can follow the run in the progress memo. It records the safety backup, catalog save, provider work, return from review, glossary application, validation, source and target pack export, kit generation, deployment, and completion report in the order they occur.

When the final line reports success, select Finish. If you see STOPPED instead, the Wizard has ended the operation safely. Read the last successful line and the reason that follows, keep the log and generated artifacts, correct that specific cause, and run the Wizard again. Do not treat partial output as a finished release.

Control or area	Purpose	Required operator action
Progress memo	Timestamped operation and diagnostic history.	Read from the last line upward when a run stops.
Optional application folder	Deploys packs to one new/temporary executable folder.	Use only when the folder was not already configured in Deployment.

Control or area	Purpose	Required operator action
Rebuild before deploying	Requests a selected build before optional deployment.	Normally leave clear; select only when another compile is actually required.
Platform / configuration	Selects a supported Win32/Win64 Debug/Release build.	Do not select unsupported targets.
Deploy to Application Folders	Deploys the just-built result and packs to configured destinations.	Use when optional destination deployment is part of this run.
Finish	Closes a successful completed Wizard.	Default action after reviewing success.
Cancel after STOPPED	Closes a stopped run after diagnostics are preserved.	Use only when Finish is unavailable because processing did not complete.

13. Component Kit and dependencies Folder

The Wizard places the verified kit under <Studio>\export\component-integration\<ApplicationId>, then automatically installs the compatible source set under <Target Project>\dependencies\DelphiAppTranslation\source. This project-local copy makes ordinary IDE and command-line builds independent of the Studio repository path.

The same transaction installs deployment\Languages, the Apache license, and machine-readable manifests. It stages and hash-checks the complete set before promotion and records the replaced managed dependency in a dated snapshot. It records and verifies the selected project-file hash without writing that file.

1. Authorize final processing after the required project ZIP backup has completed.
2. The Wizard generates the kit and copies the exact VCL or FMX source closure into a staging folder.
3. It verifies hashes and application/framework identity, then atomically promotes the managed dependency folder.
4. It verifies that the selected project-file hash is unchanged; a repeated run refreshes the same managed dependency folder rather than creating duplicates.
5. It performs the baseline Win32 Release build with a process-local Search Path and deploys the exact current pack set. Commit the managed dependency when repository policy permits.

Framework scope	Complete unit set
Shared by VCL and FMX	DAT.Core.AtomicFile; DAT.Core.Diagnostics; DAT.Runtime.LanguagePack; DAT.Runtime.Preference; DAT.Runtime.Manager; DAT.Runtime.LayoutOverrides; DAT.Runtime.SplashTranslation; DAT.Runtime.TemplateRewrite; DAT.Components.Core
VCL-specific additions	DAT.Runtime.VCL; DAT.Runtime.SplashTranslation.VCL; DAT.Runtime.TemplateRewrite.VCL; DAT.Components.VCL; DAT.Components.VCL.LanguageSelector
FMX-specific additions	DAT.Runtime.FMX; DAT.Runtime.SplashTranslation.FMX; DAT.Runtime.TemplateRewrite.FMX; DAT.Components.FMX; DAT.Components.FMX.LanguageSelector

14. Delphi Compiler Search Path

Exact compiler Search Path to enter $(PROJECTDIR)\dependencies\DelphiAppTranslation\source

1. Open the target project in RAD Studio and choose Project > Options > Building > Delphi Compiler.
2. Select All configurations and All platforms at the top of Project Options.
3. Locate Search path, preserve every existing entry, and add the exact relative path shown above as one additional entry. Use a semicolon between entries when necessary.
4. Save Project Options. This is a one-time setting for the target project; later Wizard or Maintenance runs refresh the same dependency folder and do not require another entry.

The path is relative to the target project file and points to the PAS dependency units prepared by the Studio. Do not use the Studio repository, the generated component-kit folder, or deployment\Languages as the compiler Search Path.

The Wizard never edits the DPROJ. Wizard-initiated builds temporarily prepend <Target>\dependencies\DelphiAppTranslation\source to DCC_UnitSearchPath for that process only. The Project Options value above is the developer-owned persistent setting for ordinary IDE builds.

One developer-owned Project Options entry. Add the dependency source path once. If preparation is repeated, the Studio refreshes the same folder and never adds or duplicates a DPROJ entry. The completion report gives the exact path and prior-dependency snapshot location.

Verified toolchain path. RAD Studio environment: C:\Program Files (x86)\Embarcadero\Studio\37.0\bin\rsvars.bat. Win32 compiler: C:\Program Files (x86)\Embarcadero\Studio\37.0\bin\dcc32.exe.

15. Files and Folders Created or Used

Path or file	Contents and reason
%LOCALAPPDATA%\DelphiAppTranslationStudio\provider-settings.json	Contains provider, plan, remember policy, timeout, and batch size, but never the API key. It restores non-secret Studio provider behavior.
%LOCALAPPDATA%\DelphiAppTranslationStudio\language.ini	Contains the Studio interface locale. It keeps the Studio UI language independent of target projects.
%LOCALAPPDATA%\DelphiAppTranslationStudio\Workspaces\<ApplicationId>\Development	Contains full development catalogs. It preserves stable keys, source, translations, status, context, and scan provenance.
...\Languages	Contains canonical source and translated runtime packs. It provides the workspace copy used for kit generation and deployment.
...\Glossaries\<ApplicationId>.<locale>.glossary.json	Contains the authoritative project/locale terminology JSON used by both Wizard review and Maintenance Studio > Glossary. It preserves approved product wording across runs and can be created without a Wizard run.
...\Deployment	Contains additional application destinations. It remembers authorized pack-deployment folders.

Path or file	Contents and reason
%LOCALAPPDATA% \<ApplicationId>\language.ini	Contains the target application's selected locale unless PreferenceLocation is customized. It restores the end user's language at startup.
<Studio>\export\localization-review\<ApplicationId>\<locale>	Contains HTML/JSON review artifacts. It creates a durable linguistic and layout audit trail.
<Studio>\export\component-integration\<ApplicationId>	Contains ComponentSource, Localization, LICENSE, manifests, README, deployment fallback, completion report, and DesignPackages\Win32\Release with the exact design/runtime BPL bundle.
<Target>\dependencies\DelphiAppTranslation	Contains the automatically managed source, deployment pack source/target, license, and integration manifests used by normal builds.
<Target EXE>\Localization\Languages	Contains deployed runtime packs. This is the default path resolved by LanguagesFolder.

Atomic recovery files. A .tmp file is an interrupted candidate, .previous is the last known-good version, and .corrupt-<timestamp> is quarantined invalid content. Do not delete recovery files until the active JSON has been validated and backed up.

16. Build and Deployment Procedure

1. Confirm the completion report names the installed dependency folder, exact developer-owned Search Path, unchanged project-file boundary, and previous-dependency snapshot.
2. The Wizard builds Win32 Release even when no output folder existed. It also refreshes supported configurations whose output folders show they are in use; build Win64 only when the target will ship it.
3. Confirm the build log reports automatic deployment of the canonical source pack and translated packs to each executable's Localization\Languages folder.
4. Run the executable from the output folder you just built, not an older copy elsewhere on disk.
5. Repeat pack deployment after any catalog, glossary, layout-decision, or source-pack change.

Build	Minimum deployment check	Runtime check
Win32 Debug	EXE plus source/target packs in its output tree.	Detailed first-pass diagnostics and all language switching.
Win32 Release	Independent Release output receives identical pack set.	Release candidate behavior and startup preference.
Win64 Debug	Only when the application supports Win64.	Architecture-specific runtime and layout behavior.
Win64 Release	Only when it will ship.	Final Win64 release candidate.
Installed/ portable folder	Executable identity verified before copying packs.	Run in place, including removable/network availability rules.

17. Complete Runtime Verification

1. Start in canonical English/source language and inspect every form, dialog, menu, tab, grid, report, HTML page, message, and dynamic status area.
2. Switch to each translated LTR language. Confirm static controls, dynamic strings, templates, HTML/report content, accelerator keys, placeholders, and persisted data.
3. Switch between two non-English languages without returning to English. No previous-language text or image may remain.
4. For Arabic, Hebrew, and Urdu, confirm paragraph direction and control alignment are RTL while identifiers, file paths, numbers, WinAPI, FMX, and other protected LTR tokens remain readable.
5. Switch back to English. The current screen must repaint immediately; no blank content panel or delayed click is acceptable.
6. Close and restart in each representative locale. Confirm the preference loads quickly and an invalid/missing pack falls back safely to the canonical source pack.
7. Inspect long headings and paragraphs at normal DPI and any supported scaling. Use natural space wrapping, intelligent hyphenation, and only modest font fitting; reject clipping, overlap, premature wrapping, or shifted neighboring columns.
8. Repeat the complete matrix for each shipped platform/configuration and record any exception by stable key, form, component, locale, and build.

18. Updating, Resuming, and Adding Languages

A later scan merges into the same application/locale catalog. Matching stable key and source text preserve existing work; changed source becomes attention-required; new entries are added; missing entries become obsolete rather than disappearing silently.

The provider receives only eligible unresolved records. Reviewed, approved, excluded, obsolete, and unchanged translated records are not retranslated merely because the Wizard runs again.

1. Make source/form changes in the test project and choose Save All.
2. Run the Wizard with the same ApplicationId and target locale, then scan again.
3. Review new, changed, unchanged, recovered, and obsolete counts before authorization.
4. Translate unresolved work, complete Localization Review, validate, export, regenerate the kit, and redeploy. For terminology-only corrections afterward, use Maintenance Studio > Glossary > Apply and Deploy instead of rerunning the entire Wizard. It automatically loads the matching catalog, regenerates the affected external JSON pack, refreshes the managed dependency copy, and deploys the complete pack set without rebuilding the EXE.
5. Refresh the complete vendored dependency set only when runtime/component source changed; do not replace it for ordinary catalog-only updates.
6. For another language, select that target locale and repeat scan/translation/review/export. Deploy the canonical source pack and all translated packs together.

19. Failure Recovery and Troubleshooting

Symptom	Most likely cause	Required correction
Project is not recognized	Wrong/partial DPR/DPROJ or unsupported metadata.	Select the saved DPROJ from the complete test copy and verify it opens in RAD Studio.
Scan is zero or unexpectedly small	Wrong project, unsaved forms, binary/unreadable resources, or incomplete source closure.	Save All, verify text resources and detected counts, then rescan.
Wizard and Maintenance totals differ	Different project/source snapshot, old catalog merge basis, or comparing unique entries with raw observations.	Use the same project and compare the same unique-entry headline and supporting counts.

Symptom	Most likely cause	Required correction
Connection test never completes	Network/provider request did not honor timeout/cancellation or an old build is running.	Run the current build, verify timeout/settings, provider endpoint, firewall, and account; do not kill processing before diagnostics.
401/403	Invalid key, wrong DeepL plan, disabled Google API, billing, or restriction problem.	Correct account/endpoint/key restrictions and retest.
429	Quota or rate limit.	Wait for the provider window, reduce work/batch pressure, or correct quota/billing.
Selector is empty	Packs missing, invalid, wrong ApplicationId/framework/version, or wrong executable folder.	Inspect the actual running EXE folder and validate source/target pack headers.
Language switch is slow	Repeated disk discovery, unnecessary full-form work, synchronous provider/network logic, or lifecycle loop.	Profile runtime switching; it must use admitted in-memory packs and bounded open-form application only.
Mixed languages remain	Previous-language dynamic/template text was not restored before applying the new pack.	Verify source snapshot/dynamic reverse mapping and language-change refresh contracts.
English switch shows a blank page	Current page was cleared without immediate rebuild/repaint.	Verify the language-changed handler rebuilds current dynamic/HTML content synchronously.
RTL text is left aligned	A paragraph/control direction contract is absent or overridden.	Verify pack direction, framework applicator, and per-control ownership; preserve protected LTR tokens.

Symptom	Most likely cause	Required correction
Compiler cannot find DAT units	The managed dependency is missing, its developer-owned Search Path is absent, or an older DAT unit shadows the managed set.	Run Prepare / Update Dependencies again, verify $(PROJECTDIR)\dependencies\DelphiAppTranslation\source in Project Options, and remove only confirmed stale duplicate DAT source.
STOPPED in final log	Controlled backup, translation, review, validation, export, kit, build, or deployment failure.	Preserve the log/report, correct the specific last error, and rerun; do not treat partial output as complete.

20. Security and Privacy

- Provider API keys never belong in PAS/DFM/FMX/DPR/DPROJ, JSON catalogs, runtime packs, kits, logs, screenshots, documentation, email, issue trackers, or Git.
- Remembered keys are Windows Generic Credentials for the current user; session-only keys exist only in process memory.
- Provider-settings JSON contains no secret. Review it before source distribution only to confirm it contains non-secret configuration.
- The deployed target is offline and should not contain provider client code or credentials.
- Atomic persistence and required safety ZIP protect recoverability, but Git remains the authoritative engineering history for Studio and target projects.
- A component kit contains source and language packs. Review catalog/pack licensing and proprietary text before publishing it.

21. Printable Completion Checklist

- [ ] Complete repository downloaded or cloned; Studio builds and starts.
- [ ] Pristine target backup exists; disposable test copy builds before localization.
- [ ] Correct VCL/FMX Win32 Release design BPL installed through Install Packages.
- [ ] One manager and one connected selector saved on the primary form.
- [ ] ApplicationId, LanguagesFolder, and SourceLanguage verified in Object Inspector.
- [ ] Target project saved and closed before final processing.
- [ ] DeepL/Google key stored with intended policy and Test Connection passed.
- [ ] Project, source locale, target locale, native name, direction, and locale facts verified.

- [ ] Scan headline and supporting raw/recovered/duplicate counts reviewed.
- [ ] Review and Authorize summary, safety ZIP, saved-files/application-closed check, and authorization confirmed.
- [ ] Localization Review findings, glossary, suggestions, and layout decisions completed; later terminology-only corrections use Maintenance Studio > Glossary > Apply and Deploy and the destination report is reviewed.
- [ ] Progress ends successfully; completion report and component kit exist.
- [ ] Managed dependencies\DelphiAppTranslation folder installed automatically; target DPROJ hash unchanged.
- [ ] Delphi Search path includes the exact dependency source for all shipped targets.
- [ ] Canonical source pack and every target pack deployed beside each actual EXE.
- [ ] LTR, RTL, non-English-to-non-English, switch-back, restart, dynamic, HTML, and layout matrix passes.

22. Quick Reference

Purpose	Exact or relative path
Studio repository	C:\DelphiProjects\Delphi App Translation (example)
RAD environment	C:\Program Files (x86)\Embarcadero\Studio\37.0\bin\rsvars.bat
Win32 compiler	C:\Program Files (x86)\Embarcadero\Studio\37.0\bin\dcc32.exe
Studio settings	%LOCALAPPDATA%\DelphiAppTranslationStudio
Project workspace	%LOCALAPPDATA% \DelphiAppTranslationStudio\Workspaces\<ApplicationId>
Generated kit	<Studio>\export\component-integration\<ApplicationId>
Vendored dependencies	<Target>\dependencies\DelphiAppTranslation\source
Search path entry	.\dependencies\DelphiAppTranslation\source
Runtime packs	<Target EXE>\Localization\Languages
Target preference	%LOCALAPPDATA%\<ApplicationId>\language.ini

23. Appendix A - 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.