User Guide

Delphi App Translation Studio

Version 1.0

Last changed: September 3, 2026

Windows • Delphi VCL and FireMonkey • Win32 and Win64

Table of Contents

1. Welcome - How This Guide Will Help You

Welcome to Delphi App Translation Studio. This guide is here to help you take a Delphi application from its original language to a tested, offline multilingual build without making you guess what happens next. It follows the current Pascal source, FMX form definitions, package projects, scanner contracts, provider code, workspace code, and verified application behavior as of September 3, 2026. The same workflow supports VCL and FireMonkey applications.

You can read the guide from beginning to end for your first project, or jump directly to the screen or task you need. Along the way, you will learn how to install the Studio, prepare a safe project copy, connect a translation provider, review translations, build language packs, add the runtime components, set Delphi's compiler path, deploy the result, and test both left-to-right and right-to-left languages.

The Studio is currently distributed as source rather than through an installer. That gives you full visibility into what you are building, but it also means the first setup includes a few Delphi steps. They are all explained here. For your first run, work with a disposable copy of your application until it passes the verification checklist in Chapter 34.

> **Your project remains yours.** Scanning reads the Delphi project you select. Final Setup Wizard processing creates a safety ZIP, workspace files, runtime packs, review artifacts, and a component kit. In the recommended workflow, it does not rewrite your Pascal, DFM, FMX, DPR, or DPROJ files.

1.1 Translation Is Part of Development

Delphi App Translation Studio is a development tool, not a utility that is run against a finished executable. Use it while the Delphi application is under active development, when its forms, resources, runtime messages, and language behavior can still be reviewed and adjusted by the developer. Run the Studio again as the application changes so that new or revised text becomes part of the same controlled translation workflow.

Machine translation from DeepL or Google can provide a strong, useful result, but no automatic translation service can guarantee perfect wording in every language or every application context. A provider translates the text and context it is given; it cannot always know the developer's intended meaning, a specialized industry term, the purpose of a short caption, or how translated text will fit a particular control when the application is running.

The developer therefore remains an active part of localization. Review terminology, dynamic and assembled messages, custom or third-party controls, right-to-left and mixed-direction paragraphs, control sizing, wrapping, and the screens that matter most to users. A small, simple project may require little or no manual adjustment. A larger or more specialized application will usually need at least a few developer decisions or small corrections before release.

The goal of the Studio is to automate the translation work and give the best translation possible, preserve the developer's project, identify the remaining work clearly, and make professional multilingual development practical—not to remove the developer from the process.

1.2 Choose the path that fits your work

- First project or first language: begin with Chapter 2 and follow the Setup Wizard chapters in order.
- Returning to an existing translation: open Maintenance Studio and use Chapters 19-28 as your working reference.
- Preparing a Delphi project to compile: go directly to Chapters 5 and 6 for components, dependencies, and the compiler Search Path.
- Something is not working: start with Chapter 33, then use the path and file reference in Chapter 35.

1.3 What you will have when you finish

The Studio keeps development work separate from the files your finished application uses. The following items are created during the workflow; later chapters explain each one in context.

What is created	Why you need it	Where to find it
Development catalog	Editable source/translation records, locale facts, review state, context, checksums, and scan provenance.	%LOCALAPPDATA% \DelphiAppTranslationStudio\Workspaces\<ApplicationId>\Development
Runtime language pack	Compact, validated JSON read by the target application without Internet access.	%LOCALAPPDATA% \DelphiAppTranslationStudio\Workspaces\<ApplicationId>\Languages
Project glossary	Approved project-specific terminology applied before general machine-translation wording.	%LOCALAPPDATA% \DelphiAppTranslationStudio\Workspaces\<ApplicationId>\Glossaries
Localization Review package	HTML review, glossary candidates, proposals, audit findings, and supporting JSON.	export\localization-review\<ApplicationId>\<locale>

What is created	Why you need it	Where to find it
Component integration kit	Framework-appropriate runtime/component units, source pack, translated packs, manifest, deployment script, README, and completion report.	export\component-integration\<ApplicationId>
Safety backup	Timestamped ZIP made before final processing.	The backup path is displayed in Wizard progress and the completion report.

1.4 What this release supports

- Supported target frameworks: Delphi VCL and FireMonkey (FMX).
- Supported target platforms: Windows Win32 and Win64; Debug and Release configurations may be built locally.
- Supported online providers during development: DeepL API and Google Cloud Translation Basic v2.
- The deployed application is offline. It does not contain a provider API key and does not call DeepL or Google.
- macOS, iOS, Android, Linux, C++Builder, and runtime cloud translation are outside the present release scope.
- Dynamic strings assembled from live data may require an application-authored semantic key or explicit runtime application; a static scanner cannot infer every possible runtime sentence.

2. Before You Begin

A little preparation makes the rest of the process straightforward. Gather the items below before you open the Studio. If your Delphi project does not already build cleanly, fix that first; translation should not be asked to hide an unrelated compiler problem.

You will need	What to have ready
Windows	A current 32- or 64-bit Windows development system with permission to write the selected project copy, Local AppData, and the Studio export folder.

You will need	What to have ready
RAD Studio	RAD Studio 12 Athens or RAD Studio 13 Florence. The verified RAD Studio 13 toolchain is under C:\Program Files (x86)\Embarcadero\Studio\37.0\bin.
Delphi project	A saved VCL or FMX .dproj/.dpr that builds successfully before localization.
Source form format	Text DFM/FMX resources are preferred for auditability. Save all designer changes before scanning.
Internet	Required only while testing a provider connection or translating unresolved entries.
Provider account	A DeepL API Free/Pro key or Google Cloud Translation API key. DeepL is the recommended default. Obtaining a provider key is outside this guide; consult the provider's current account and API documentation.
Version control	Git or an equivalent recoverable baseline for both the Studio source and the disposable target copy. Git and GitHub are essential to this workflow because they provide the authoritative recoverable backup source.

Start safely. Keep a pristine backup and make a separate test copy of the Delphi application. Build that test copy successfully before you add DAT components or language packs. This gives you a clean point of comparison at every stage.

3. Get the Studio from GitHub

Start at https://delphitranslate.github.io/downloads.html, where the current Beta source ZIP and matching guides are kept together. The official repository is https://github.com/delphitranslate/delphitranslate.github.io. Download the whole repository so the Delphi projects, packages, runtime units, images, localization files, tests, tools, and documentation stay together in the folder structure they expect.

A single .pas file, .dproj, or component BPL is not enough. Those files rely on neighboring source and package files, and a partial download is the most common way to end up with missing-unit or missing-resource errors.

3.1 Download a ZIP

1. Open the repository URL in your browser.
2. Use the branch selector to choose the published beta or release branch specified on the repository page. If a signed release is available, prefer its release tag; otherwise use the public beta branch or main branch identified by the project owner.
3. Choose Code, then Download ZIP.
4. Save the ZIP to your normal Downloads folder. Extract it before opening anything; Delphi cannot build the project correctly from inside the compressed ZIP.
5. Extract the complete archive to a short, writable development path such as C:\DelphiProjects\Delphi App Translation.
6. Open the extracted root and make sure you can see DelphiAppTranslationStudio.dproj together with the source, packages, Localization, docs, tools, and images and icons folders. If those are present, you have the complete source tree.

3.2 Clone with Git

```
git clone https://github.com/delphitranslate/delphitranslate.github.io.git
```

Cloning is the easiest choice if you expect to receive updates. Keep your local work on a separate branch, review incoming changes before merging them, and never commit provider credentials or proprietary application catalogs.

3.3 Avoid these partial-install shortcuts

- Do not treat bin as an installer. Local executables and BPLs must match the active RAD Studio toolchain.
- Do not install a .dpk through Install Component. Build the package and add the Win32 design BPL through Component > Install Packages.
- Do not rely on an older source-distribution ZIP if its date predates the selected repository branch. The full branch archive is authoritative until a signed release package is published.

4. Build and Open the Studio

Once the repository is extracted, building the Studio is a normal Delphi project build. Win32 Release is the friendliest first choice because the design-time packages also use Win32. All examples that follow use RAD Studio 13 Florence.

1. Start RAD Studio 13 Florence.
2. Open DelphiAppTranslationStudio.dproj from the extracted repository root.

3. Choose Win32 or Win64 and Debug or Release. Win32 Release is the simplest first build and is also the platform used for Delphi design-time packages.
4. Choose Project > Build DelphiAppTranslationStudio and wait for a successful build message.
5. Run the executable from bin\<Platform>\<Configuration>, for example bin\Win32\Release\DelphiAppTranslationStudio.exe.

If you prefer a command-line build, initialize Delphi with C:\Program Files (x86)\Embarcadero\Studio\37.0\bin\rsvars.bat. The Win32 compiler is C:\Program Files (x86)\Embarcadero\Studio\37.0\bin\dcc32.exe. These paths point to Delphi itself. Studio-initiated target builds pass the project-local DAT Search Path temporarily; the developer adds the same stable dependency path once through RAD Studio Project Options for normal IDE builds.

> **Where did the build go?** The executable is placed in bin\<Platform>\<Configuration>, and compiled DCUs go under dcu. Keep the repository folder structure intact so packages, localization files, exports, and documentation continue to resolve correctly.

5. Prepare Your Delphi Application

The Studio calls the application being translated the target application. You only need to prepare it once: install the matching DAT design package, place a language manager and selector on the primary form, and save their properties in the Form Designer.

1. Create a pristine backup, then make a separate test copy of the application for localization work.
2. Open the test copy in RAD Studio and build every platform/configuration you intend to support. Correct pre-existing build errors before localization.
3. Build the matching DAT runtime and design packages from the Studio repository.
4. In RAD Studio choose Component > Install Packages > Add and select DATLanguageManagerVCLDesign.bpl or DATLanguageManagerFMXDesign.bpl from bin\packages\Win32\Release.
5. Open the target application's primary form in the Form Designer.
6. Place one TDATVCLLanguageManager or TDATFMXLanguageManager on the primary form. It is nonvisual, so it will appear in the designer's component tray.
7. Set ApplicationId to the exact Delphi project name, LanguagesFolder to Localization\Languages, and SourceLanguage to the source locale, normally en-US.
8. Place one TDATVCLLanguageComboBox or TDATFMXLanguageComboBox and set its LanguageManager property to the manager. A designer-authored connected Language menu is an alternative, but the user needs a visible selection mechanism.
9. Add any supporting Language label or menu captions in the designer, then choose File > Save All. Keeping these controls in the designer makes them easy to see and maintain in the Object Inspector.

10. Choose File > Save All and close the running target application before the Setup Wizard's final-processing step. The RAD Studio project may remain open because the Studio never writes its project files.

> **One manager is enough.** The primary form owns the application's single language manager. You do not need another manager on every form. The manager retranslates forms that are already open and applies the active language to forms opened later.

6. Give Delphi Access to the DAT Dependencies

The Wizard and Maintenance Studio now prepare this dependency automatically. They stage and verify the complete framework-specific ComponentSource set, then install it as one versioned unit under dependencies\DelphiAppTranslation\source in the target project. You do not create the folder or copy individual PAS files.

The dependency transaction stages and hash-checks the complete set, writes an integration manifest, and atomically promotes only dependencies\DelphiAppTranslation. Repeating Prepare / Update Dependencies replaces that one managed set cleanly instead of mixing files from different Studio builds. Target Pascal, form, DPR, and DPROJ files remain read-only.

6.1 The recommended folder layout

```
<Target Project>\dependencies\DelphiAppTranslation\source
<Target Project>\dependencies\DelphiAppTranslation\deployment\Languages
```

1. Complete the Setup Wizard, or choose Prepare / Update Dependencies on Maintenance Studio > Integration.
2. The Studio creates a dated previous-dependency snapshot before replacing the managed dependency folder. It does not write the DPROJ.
3. The Studio stages and hash-checks ComponentSource, language packs, Apache license, manifest, and deployment target before promoting dependencies\DelphiAppTranslation.
4. The Studio performs the baseline Release build with a temporary compiler Search Path and directly deploys the current pack set. Later Studio-initiated builds repeat this safely; ordinary RAD Studio builds do not invoke the Studio.
5. Commit the managed dependency folder with the target project when the project's licensing/distribution policy permits vendoring Apache-2.0 source.

6.2 Canonical framework unit set

Used by	Units to copy
Every target	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 target	DAT.Runtime.VCL; DAT.Runtime.SplashTranslation.VCL; DAT.Runtime.TemplateRewrite.VCL; DAT.Components.VCL; DAT.Components.VCL.LanguageSelector
FireMonkey target	DAT.Runtime.FMX; DAT.Runtime.SplashTranslation.FMX; DAT.Runtime.TemplateRewrite.FMX; DAT.Components.FMX; DAT.Components.FMX.LanguageSelector

6.3 Tell Delphi where the units are

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

1. Open the target project in RAD Studio, then choose Project > Options > Building > Delphi Compiler.
2. At the top of Project Options, select All configurations and All platforms so the setting applies to every normal IDE build of this project.
3. Locate Search path. Keep every existing entry and add the exact relative path shown above as one additional entry, separated from other entries by a semicolon when necessary.
4. Save Project Options. Enter this path once per target project; do not add it again when the Studio refreshes the managed dependency folder.

The path is relative to the folder containing the target project file. It points to the PAS units automatically maintained under the target project's dependencies folder. Do not substitute the Studio repository path, the export\component-integration path, or the deployment\Languages path.

The Studio never writes the DPROJ. Studio-initiated builds pass the same dependency directory to MSBuild temporarily for that build process only. The persistent Project Options value above is the developer-owned setting used by ordinary RAD Studio builds.

What remains manual? RAD Studio package registration remains deliberate: choose Component > Install Packages > Add and select the exact design BPL shown by the Studio. The verified bundle is in <Kit>\DesignPackages\Win32\Release; the required core and framework runtime BPLs are beside it. Component placement, Object Inspector values, and the one developer-owned Project Options Search path remain deliberate IDE actions. Folder creation, dependency refresh, Studio builds, and pack deployment are automatic.

7. Understand the Files Under %LOCALAPPDATA%

The Studio keeps your working settings and translation workspace under %LOCALAPPDATA%, normally C:\Users\<WindowsUser>\AppData\Local. These are per-user files. Keeping them outside Program Files and outside your application lets the Studio save normal working state without asking Windows for permission to rewrite an installed program folder.

Most of the time, you will not need to open these files yourself. They become important when you are moving work to another computer, diagnosing a recovery, checking which catalog is active, or removing a remembered preference.

Path or file	Contents	Operational meaning
%LOCALAPPDATA%\DelphiAppTranslationStudio\provider-settings.json	Provider name, DeepL plan, remember-credential choice, request timeout, and batch size.	Contains no API key. Defaults are DeepL, API Free, remember enabled, 30 seconds, and 40 strings per request.
%LOCALAPPDATA%\DelphiAppTranslationStudio\language.ini	Studio interface language preference.	Lets the Studio reopen in the selected interface language.
%LOCALAPPDATA%\DelphiAppTranslationStudio\Workspaces\<ApplicationId>\Development\<ApplicationId>.<locale>.translation-project.json	Editable development catalog.	Authoritative scan/translation/review workspace for that application and locale.
...\Languages\<locale>.json	Validated runtime pack.	Copied into component kits and deployed beside target executables.

Path or file	Contents	Operational meaning
...\Languages\<locale>.layout-proposal.json	Reviewed layout decisions when present.	Accepted direction/fitting/relative layout rules can be embedded in the runtime pack.
...\Glossaries\<ApplicationId>.<locale>.glossary.json	Approved project-specific terms.	Overrides general machine wording for exact project terminology.
...\Deployment\deployment-destinations.json	Optional application destination folders.	Restores portable/network/USB deployment destinations on the next Wizard run.
%LOCALAPPDATA%\<Target ApplicationId>\language.ini	The target application's selected language.	Created by the DAT manager unless PreferenceLocation is customized.

7.1 Recovery files you may see

- A successful replacement may retain one .previous file containing the last valid text.
- A temporary sibling ending in .tmp is used during atomic publication and should not remain after a successful operation.
- An invalid current file may be quarantined with a .corrupt-<timestamp>-<identifier> name while the prior valid file is recovered.
- Do not delete recovery files while diagnosing a failed catalog, settings, pack, glossary, or preference load. Copy them first and compare timestamps.

7.2 Where your API keys are kept

When you choose Remember, the key is stored as a Windows Generic Credential, not in a JSON file. The credential names are DelphiAppTranslationStudio/Providers/DeepL and DelphiAppTranslationStudio/Providers/Google Cloud Translation. If Remember is clear, the key stays in memory only for the current Studio session, and saving that choice removes any stored credential for that provider.

You can confirm or remove a saved key in Windows Credential Manager. Treat the key like a password: never paste it into source code, catalogs, bug reports, screenshots, documentation, logs, deployment scripts, or Git.

8. Meet the Start Screen and Keyboard Controls

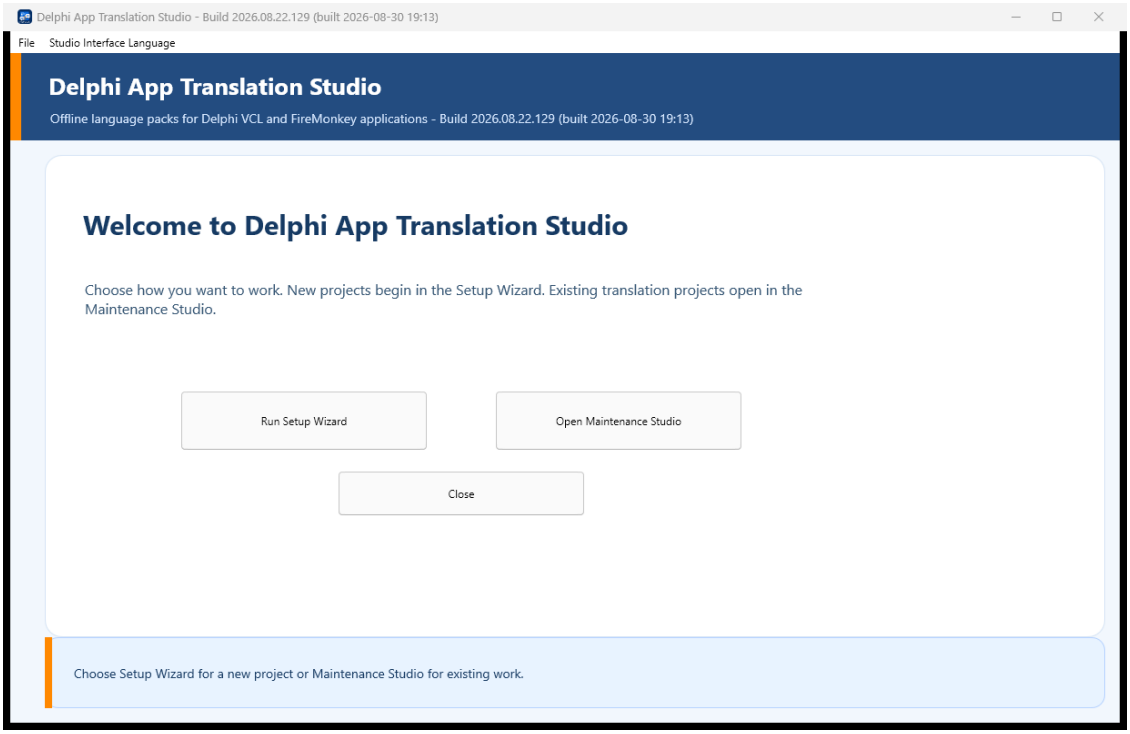

Figure 8-1. Current landing screen.

The start screen gives you two clear choices: a guided first-time setup or direct maintenance of work that already exists. Run Setup Wizard is the default button, so pressing Enter starts the guided path without reaching for the mouse. Use Tab and Shift+Tab to move between controls, Space to activate a focused button or check box, Alt+Down or F4 to open a combo box, the arrow keys to move through choices, Enter to accept, and Escape to close a list or cancel a dialog when it is safe to do so.

On the screen	How it helps you	When you will use it
Run Setup Wizard	Walks you through the complete eight-step setup.	Choose this for your first language or whenever you want one guided scan-to-kit pass.
Open Maintenance Studio	Opens the eight-page direct workflow.	Use for existing catalogs, manual review, glossary work, validation, export, integration, or provider settings.
Close	Closes the Studio.	Use when no operation is running.

On the screen	How it helps you	When you will use it
File > Exit	Normal application exit path.	Use from Maintenance Studio; an active operation observes safe cancellation boundaries.
Studio Interface Language	Changes the Studio's own UI language.	This does not change the source or target language of an opened Delphi project.

Figure 8-2. Landing-screen action buttons.

9. Move Through the Setup Wizard with Confidence

The Setup Wizard divides the job into eight manageable steps: Welcome, Delphi Project, Deployment, Languages, Translation Service, Scan Project, Review Authorize, and Process Finish. Until final processing begins, you can click any completed step in the left rail to review or correct an earlier choice.

On ordinary pages, Next is the default button, so Enter advances after the current page passes its checks. Back returns to the previous page without throwing away your entries. Cancel closes the Wizard before processing. Once processing starts, Cancel becomes Stop and waits for a safe stopping point rather than interrupting a file write. After a successful finish, Back and Cancel disappear so the only action left is Finish. If processing stops with a problem, Cancel remains available after you read the STOPPED message.

On the screen	How it helps you	When you will use it
Step rail	Shows progress and allows revisiting completed steps before authorization.	Select a completed step to correct a choice.
Back	Returns one step without discarding already entered values.	Available until final processing begins; hidden on the completed Finish page.

On the screen	How it helps you	When you will use it
Next	Validates the current step and advances.	The default Enter action on Steps 1-6.
Begin Final Processing	Starts the authorized single pass.	Appears on Review Authorize after required confirmations.
Cancel / Stop	Closes before work or requests cancellation during work.	Cancellation is honored at a safe boundary; it is not a forced thread termination.
Finish	Closes a successfully completed Wizard.	The only completion button after success.

10. Wizard Step 1 - Welcome

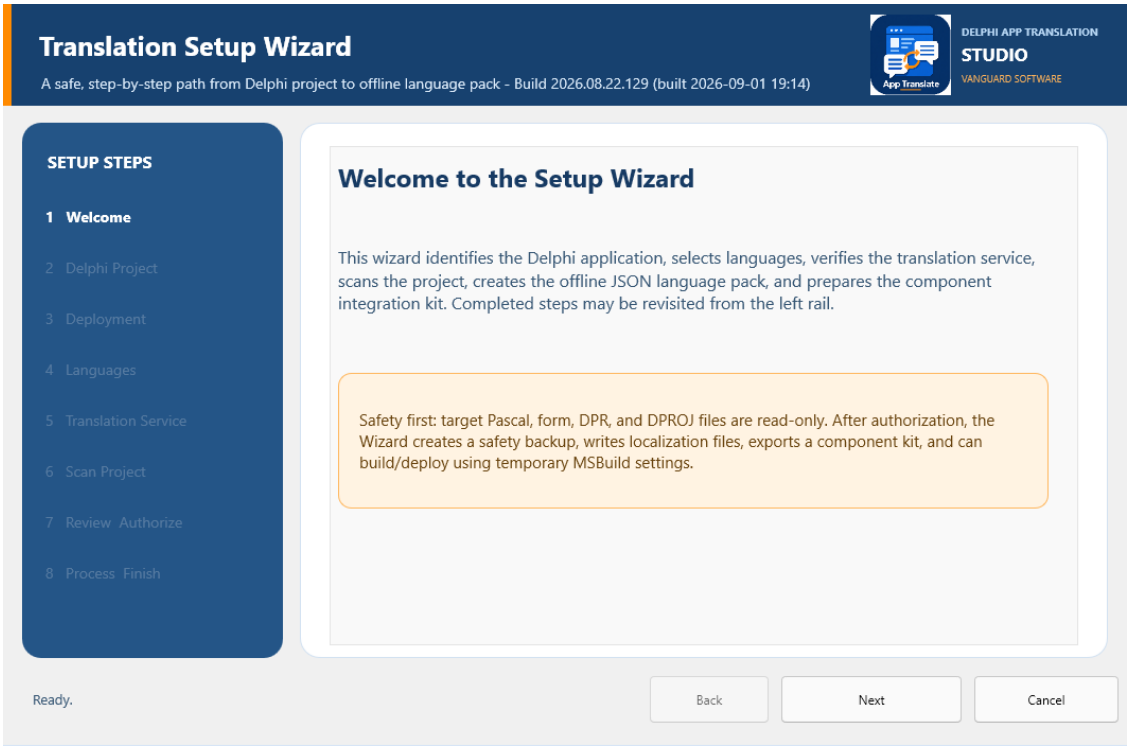

Figure 10-1. Welcome page.

This first page lets you know what the Wizard will do and, just as importantly, what it will not do. Nothing is changed here. Read the safety note, then press Enter or choose Next. Back is disabled because you are already at the beginning; Cancel returns you to the Studio.

11. Wizard Step 2 - Delphi Project

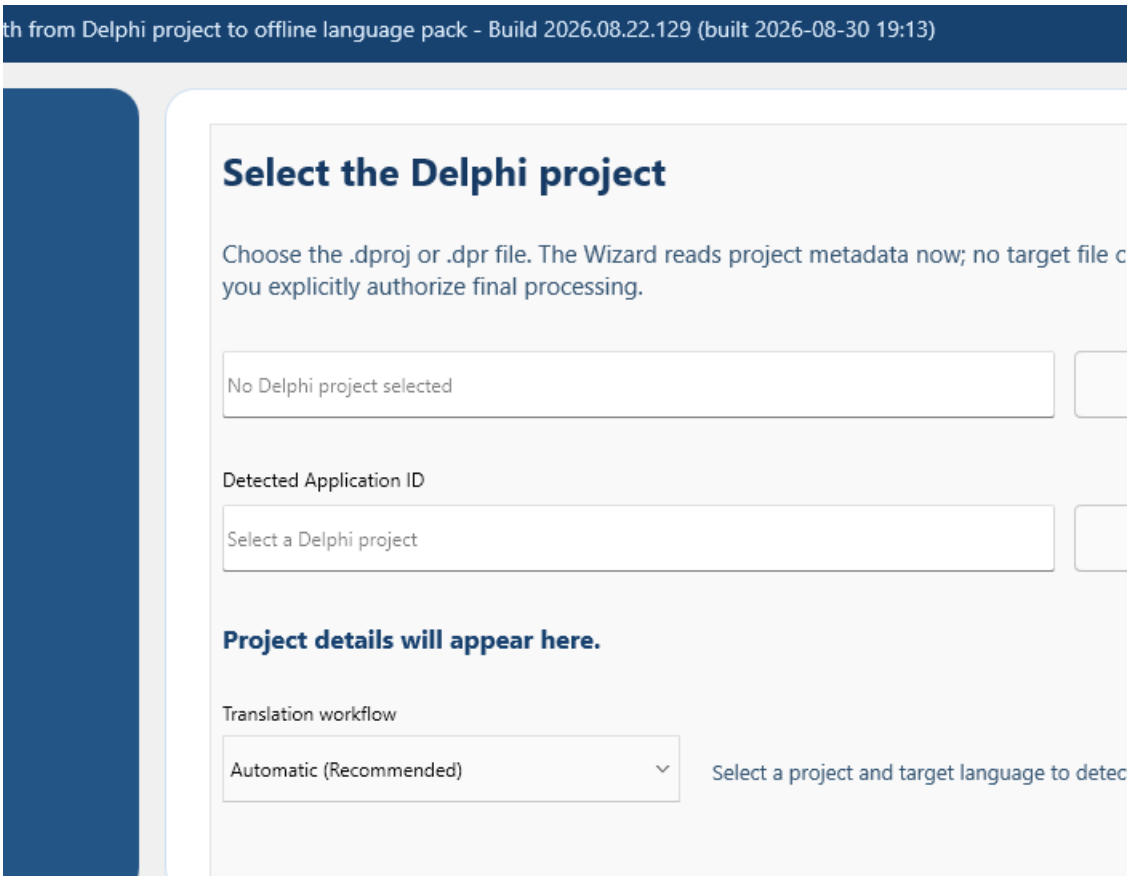

Figure 11-1. Delphi Project page.

On the screen	How it helps you	When you will use it
Project path	Shows the .dproj or .dpr you selected.	Use it as a final visual check that you opened the intended test copy.
Browse	Opens the file picker for your Delphi project.	Choose the saved disposable/test copy, not the production original.
Detected Application ID	Shows the identity shared by catalogs, packs, and the runtime manager.	Make sure it matches the manager component's ApplicationId.
Copy ID	Copies the detected ID to the Clipboard.	Paste it into the manager's ApplicationId property in the Object Inspector.

On the screen	How it helps you	When you will use it
Project details	Summarizes the framework, Windows targets, form resources, and source units the Studio found.	Pause here and confirm VCL or FMX and the expected project size before continuing.
Translation workflow	Lets the Studio recommend a new or update workflow, or lets you choose explicitly.	Choose Update Existing Translation when you want compatible reviewed and approved work to carry forward.

Keep the Application ID consistent. The catalog applicationId, runtime-pack applicationId, manager ApplicationId, workspace folder, and generated kit identity must all agree. If you change the ID, the Studio correctly treats it as a different localization workspace.

12. Wizard Step 3 - Deployment

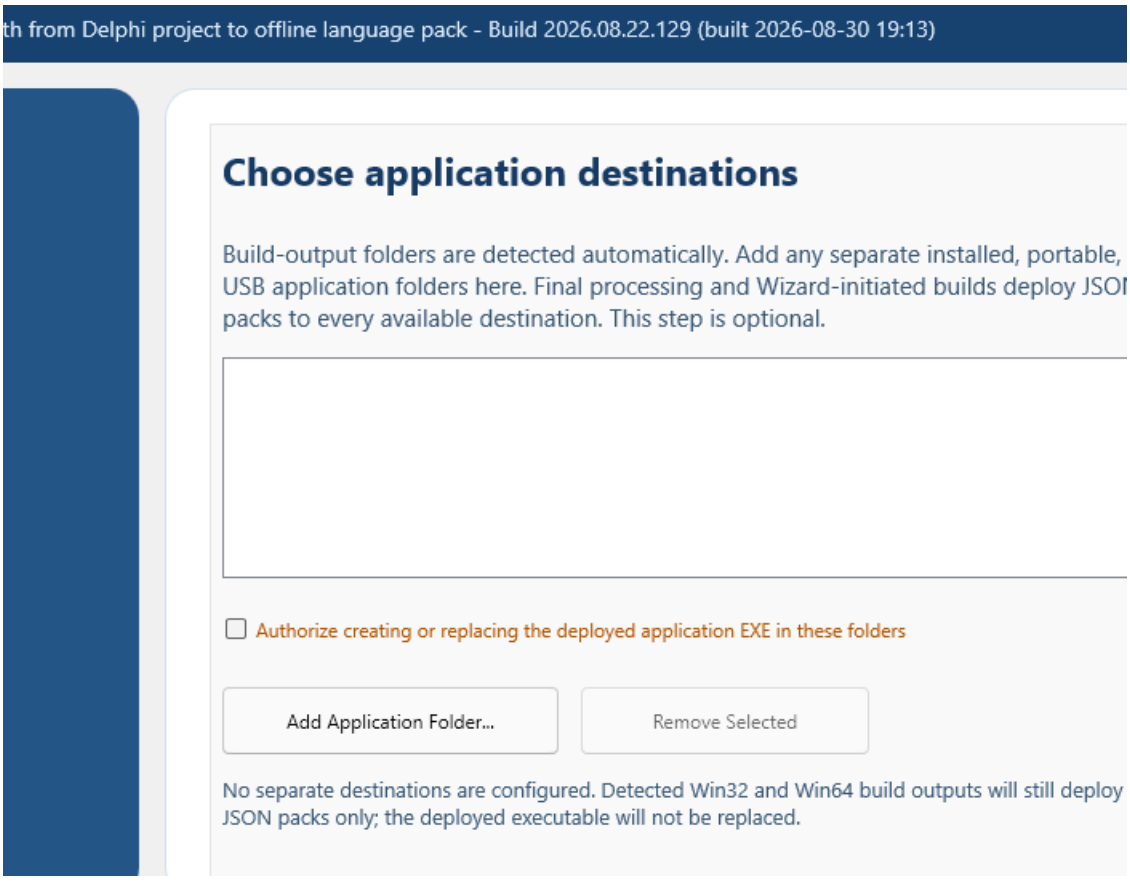

Figure 12-1. Deployment page.

On the screen	How it helps you	When you will use it
Destination list	Remembers extra installed, portable, network, or USB application folders.	Leave it empty if the normal detected build folders are all you need.
Add Application Folder	Adds one optional deployment destination.	Choose the folder that contains, or will contain, the deployed executable.
Remove Selected	Removes the selected destination from this list.	It does not delete the folder or anything inside it.
Authorize creating or replacing the deployed EXE	Allows the optional build/deploy step to place an executable in the listed destinations.	Leave it clear when you only want the JSON language packs copied.

You can safely leave this page empty for a normal local build. Final processing still looks for existing Delphi output folders and deploys language packs where appropriate. Add a separate destination only when you also need an installed, portable, network, or USB copy. If that location is unavailable, the Studio tells you instead of reporting a false success.

13. Wizard Step 4 - Languages

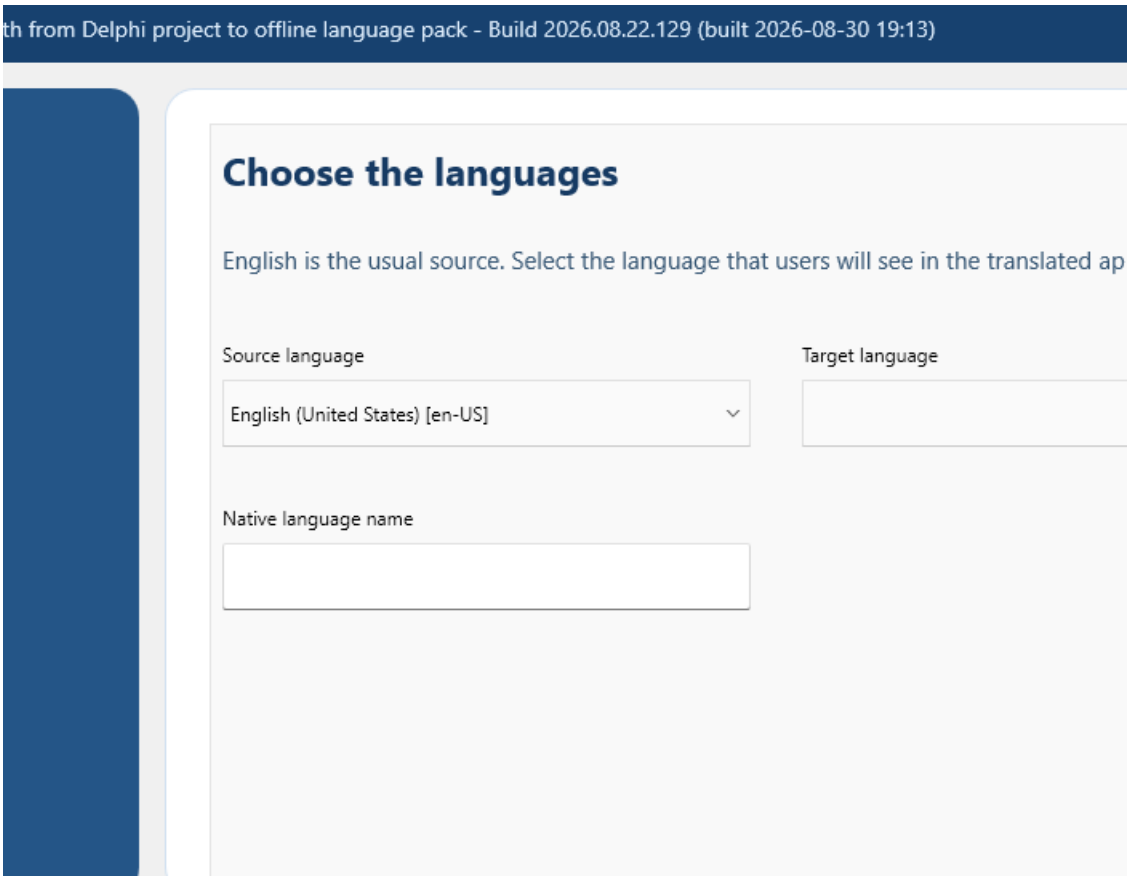

Figure 13-1. Languages page.

On the screen	How it helps you	When you will use it
Source language	Identifies the language already saved in your Delphi forms and source.	For most projects this is English (United States) [en-US]. Do not choose the language you want to create here.
Target language	Selects the locale you are creating or updating.	Choose it explicitly so the catalog receives the correct regional rules.
Native language name	Controls the language name your users will see.	The Studio fills it automatically; edit it only when your product prefers different wording.

On the screen	How it helps you	When you will use it
Direction	Comes from the locale's shared language facts.	Arabic, Hebrew, Persian, Urdu, Pashto, and other RTL locales automatically use the common right-to-left path.

A locale is more than a language name. Your selection also establishes date and time formats, decimal and thousands separators, currency facts, and reading direction. When you reopen a compatible catalog, its saved locale facts are restored rather than being overwritten by a default selection.

14. Wizard Step 5 - Translation Service

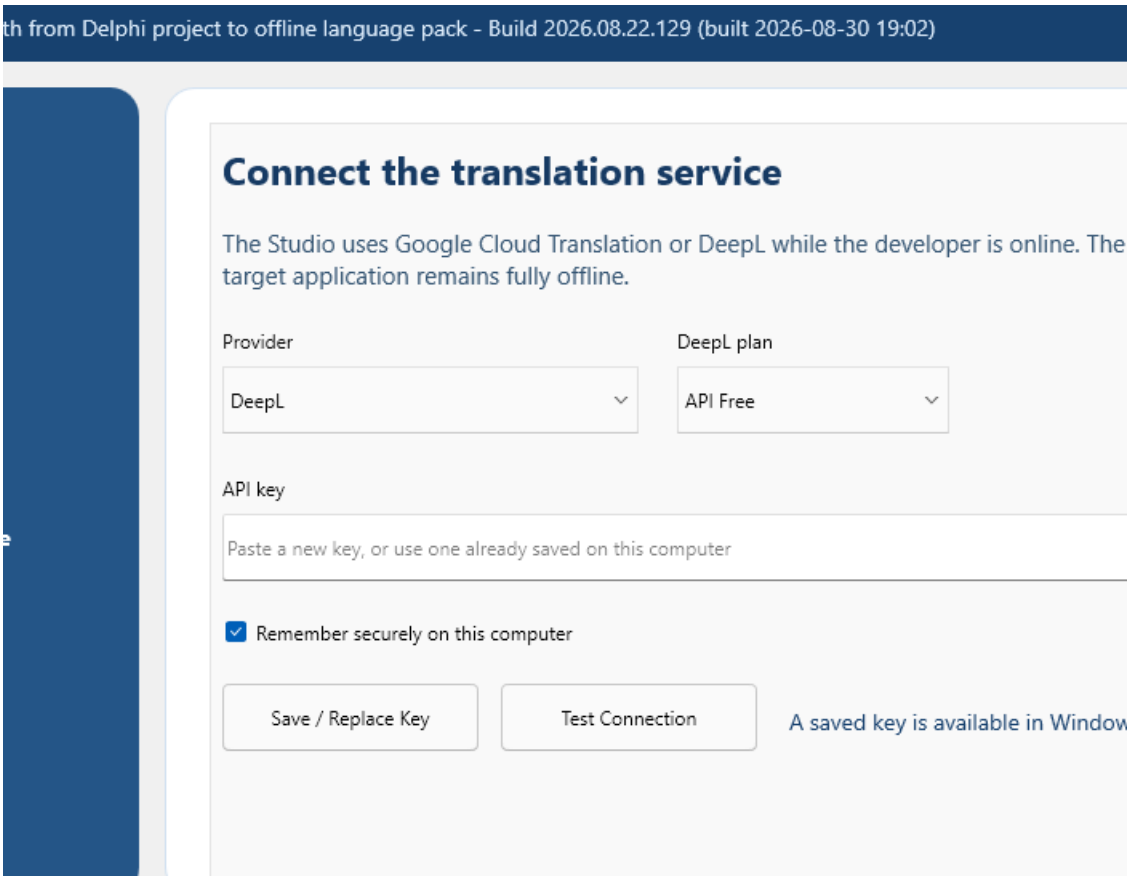

Figure 14-1. Translation Service page with DeepL selected.

On the screen	How it helps you	When you will use it
Provider	Chooses DeepL or Google Cloud Translation.	DeepL is selected by default and is the recommended starting point.
DeepL plan	Chooses the API Free or API Pro endpoint.	Match this to your DeepL account; Google ignores this setting.
API key	Accepts a new provider key and masks it on screen.	Leave it blank when the Studio reports that a saved key is already available.
Remember securely	Stores the key in Windows Credential Manager for this provider.	Clear it for session-only use. Saving the cleared choice also removes that provider's stored credential.
Save / Replace Key	Saves the pasted key according to your Remember choice.	Choose it whenever you add or rotate a key.
Test Connection	Makes a small, time-limited provider request.	Use it before a long translation run; you will receive either a result or a timeout, and the UI remains responsive.
Status text	Tells you whether a saved key exists and reports connection results.	It confirms availability without ever displaying the saved key.

Each provider keeps its own key. Switching providers does not copy or reuse the other provider's credential. You can safely have a DeepL key and a Google key stored at the same time.

15. Wizard Step 6 - Scan Project

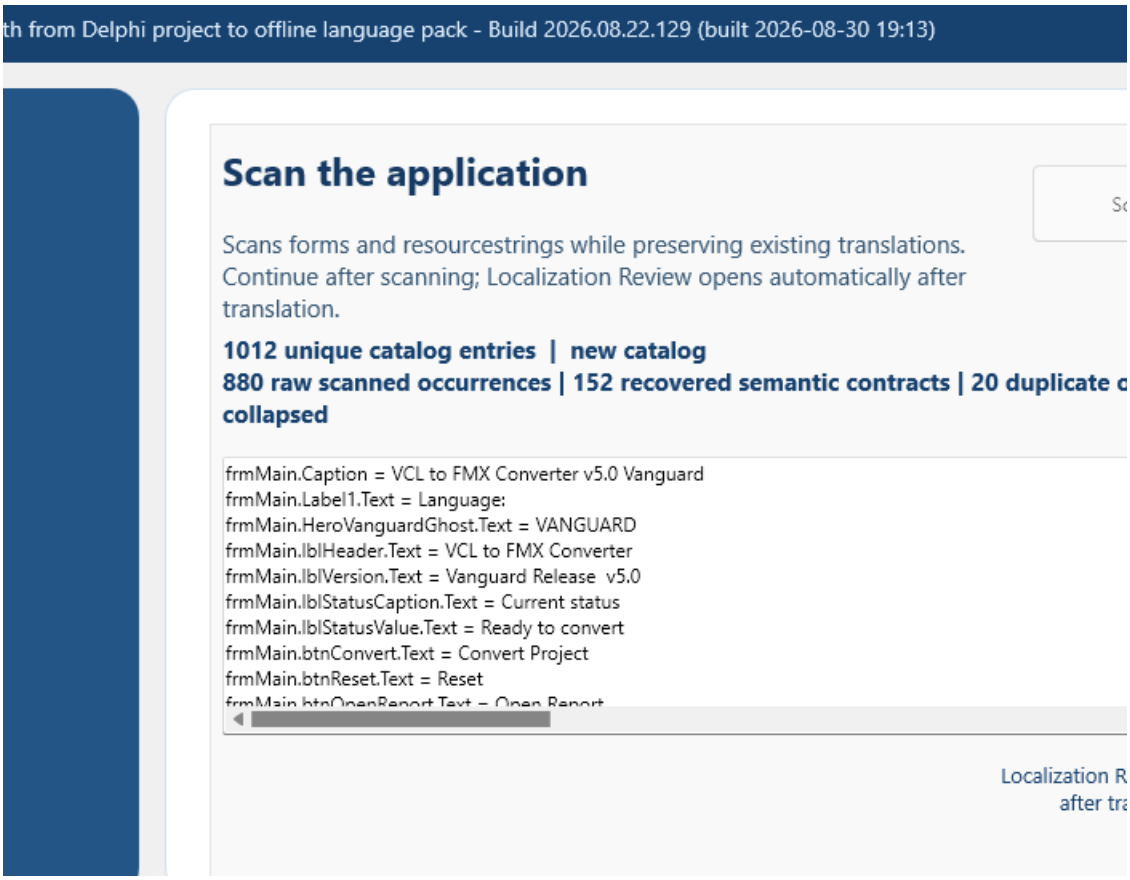

Figure 15-1. Scan page after the verified 1,012-entry scan.

This is where the Studio discovers the text your users can see. Scan Project reads saved forms and supported source/resource contracts, then reports both unique catalog entries and the raw observations that produced them. Equivalent duplicates are collapsed so you do not translate the same semantic entry repeatedly. Compatible semantic contracts recovered from earlier work remain identified, while conflicting duplicate keys stay visible for correction.

Treat the list as a confidence check rather than a second catalog editor. Look at the project, form, and source counts, then read a few familiar entries. If the result seems too small, first confirm that you selected the correct project copy and saved every form. Then check that expected source directories belong to the project and that external JSON prose is declared through dat-translatable-resources.json.

On the screen	How it helps you	When you will use it
Scan Project	Runs the inventory again.	Rerun it after you save changes to forms, UI source, or declared resources.
Summary	Separates unique entries, raw occurrences, recovered semantic contracts, and collapsed duplicates.	Use the same categories when you compare Wizard and Maintenance scans.
Results memo	Shows keys, source wording, and representative places where the wording was found.	Spot-check familiar entries here; the development catalog contains the complete record.
Next	Moves to the final review after a successful scan.	Translation and Localization Review happen later, during final processing.

16. Wizard Step 7 - Review and Authorize

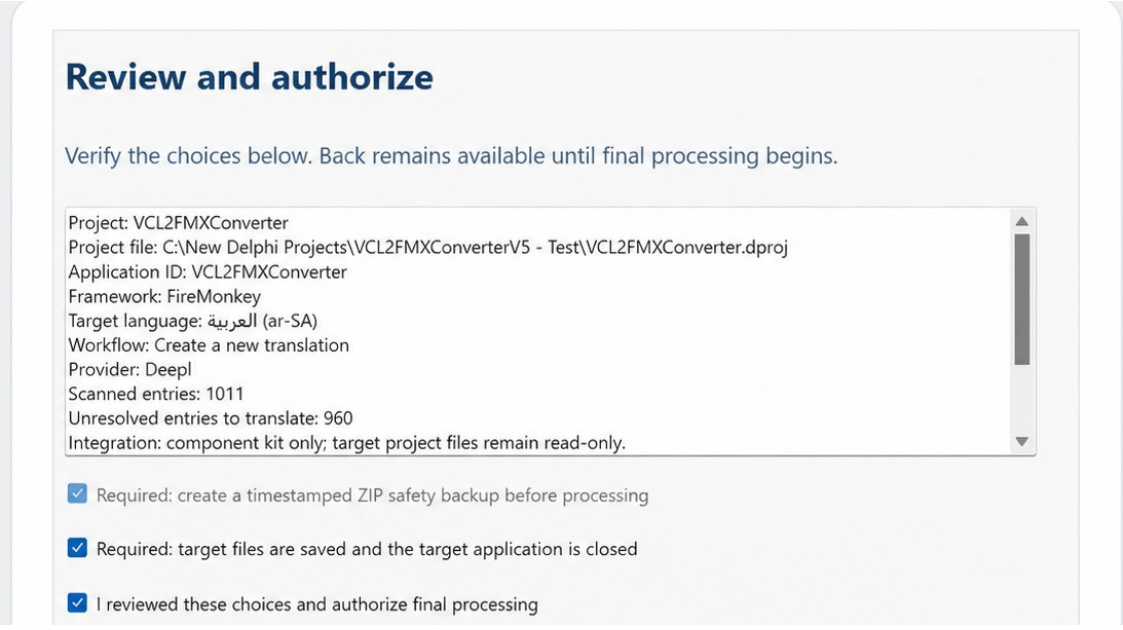

Figure 16-1. Review and Authorize content area; the current Wizard uses this same confirmation contract in Step 7.

On the screen	How it helps you	When you will use it
Review memo	Brings your project, application ID, framework, locale, workflow, provider, counts, integration mode, and destinations together in one place.	Read it from top to bottom; it is your chance to catch a wrong project or locale before work begins.
Required ZIP backup	Confirms that the Wizard will create its pre-processing safety archive.	It is required and selected by design.
Target files saved; application closed	Confirms the scanner sees the saved project and the executable is not locking a rebuild.	Choose File > Save All, close the running target application, then select this confirmation.
Authorize final processing	Records your explicit approval for the controlled processing pass.	Select it only after the summary is correct.
Begin Final Processing	Starts the backup, translation, review, validation, export, kit generation, configured builds/deployment, and completion report.	After you choose it, Back and the step rail are disabled; Stop is still honored at safe boundaries.

17. Localization Review Window

After machine translation, the Wizard pauses and opens Localization Review so you can look at wording and layout before the runtime pack is finalized. This is where human judgment matters most: short button captions and product terms can be technically translated yet still sound wrong in context. When you close the review window, the same Wizard pass continues with validation, export, component-kit generation, and deployment.

Tab	Purpose	Primary actions
Audit	Summarizes readiness, structural issues, untranslated/uncertain entries, and package artifacts.	Generate Package; Open Package.

Tab	Purpose	Primary actions
Project Glossary	Creates, edits, deletes, and saves exact source-to-target terminology for this application/locale during this review pass.	New Term; Add Term; Delete Term; Save Project Glossary. For later corrections without rerunning the Wizard, use Maintenance Studio > Glossary (Chapter 23).
Suggestions	Reviews terminology suggestions and confidence.	Use Suggestion; Approve High Confidence; Reject Suggestion.
Layout Proposals	Reviews direction, text fitting, columns, and geometry proposals.	Save Decision; Accept Safe All; Reset Pending. Absolute geometry remains an individual visual decision.

Give short UI text special attention. A provider can produce grammatically correct wording that is still wrong for a button, tab, or status label. Check the source context, glossary, placeholders, and visual fit before you approve it.

18. Wizard Step 8 - Process and Finish

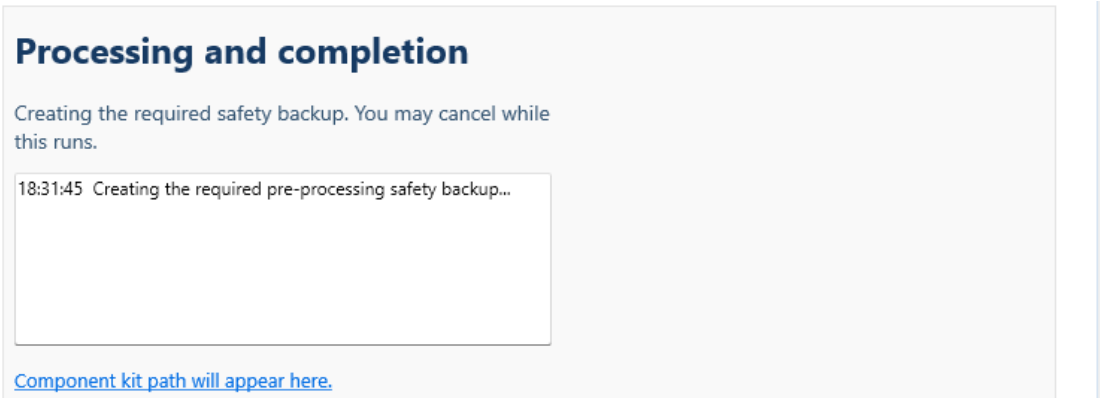

Figure 18-1. Processing status area. The final build has a simplified Finish page; obsolete command and kit-path controls are not part of the current workflow.

The progress memo lets you follow the work without having to interpret a console window. It records the safety backup, catalog work, translation, return from review, validation, runtime-pack export, component-kit generation, detected-output deployment, optional destination deployment, and completion report. If something cannot continue safely, the final diagnostic begins with STOPPED and Finish remains unavailable until you have a result you can close or retry.

After success, the page confirms that Pascal, DPR, DPROJ, DFM, and FMX files were not edited. It also identifies the previous-dependency snapshot and the managed project-local dependency. Back and Cancel disappear; choose Finish when you are ready to return to the Studio. Win32 Release, existing target outputs, and available destinations have already been built or deployed automatically.

On the screen	How it helps you	When you will use it
Progress memo	Shows each operation and diagnostic with a time stamp.	If processing stops, begin with the last few lines; they normally contain the cause and next action.
Deploy to App Folder	Copies packs to a folder you choose now.	Use it for a one-time destination that was not saved in Step 3.
Rebuild before deploying	Requests another selected build before destination deployment.	Normally leave clear because final processing already compiled available target outputs.
Platform / Configuration	Chooses Win32/Win64 and Release/Debug for an optional rebuild.	Choose only combinations supported by the target project.
Deploy to Application Folders	Deploys the built application and packs to configured destinations, subject to executable authorization.	Nothing else copies the executable to separate Step 3 destinations.
Finish	Closes the successfully completed Wizard.	Choose it even when you did not request the optional blue-card deployment.
Cancel after STOPPED	Closes an unsuccessful Wizard after you have read the diagnostic.	It disappears after success because Finish is then the correct action.

19. Maintenance Studio Overview

Maintenance Studio is your workspace after the guided setup. Its eight pages - Project, Scan, Translate, Glossary, Validation, Export, Integration, and Provider Settings - let you go directly to the part of the translation you want to maintain. Moving through the left rail does not rerun completed work. Keep an eye on the status line at the bottom; it confirms the last action and often tells you exactly what to do next.

A persistent Cancel button is available throughout Maintenance Studio. It is the normal way to leave the Maintenance workflow; pressing Esc invokes the same action. If a scan or provider request is active, the Studio requests cancellation and waits for that operation to reach a safe boundary before closing. If a glossary has unsaved changes, you are offered Save, Discard, or Cancel so closing cannot silently lose terminology work.

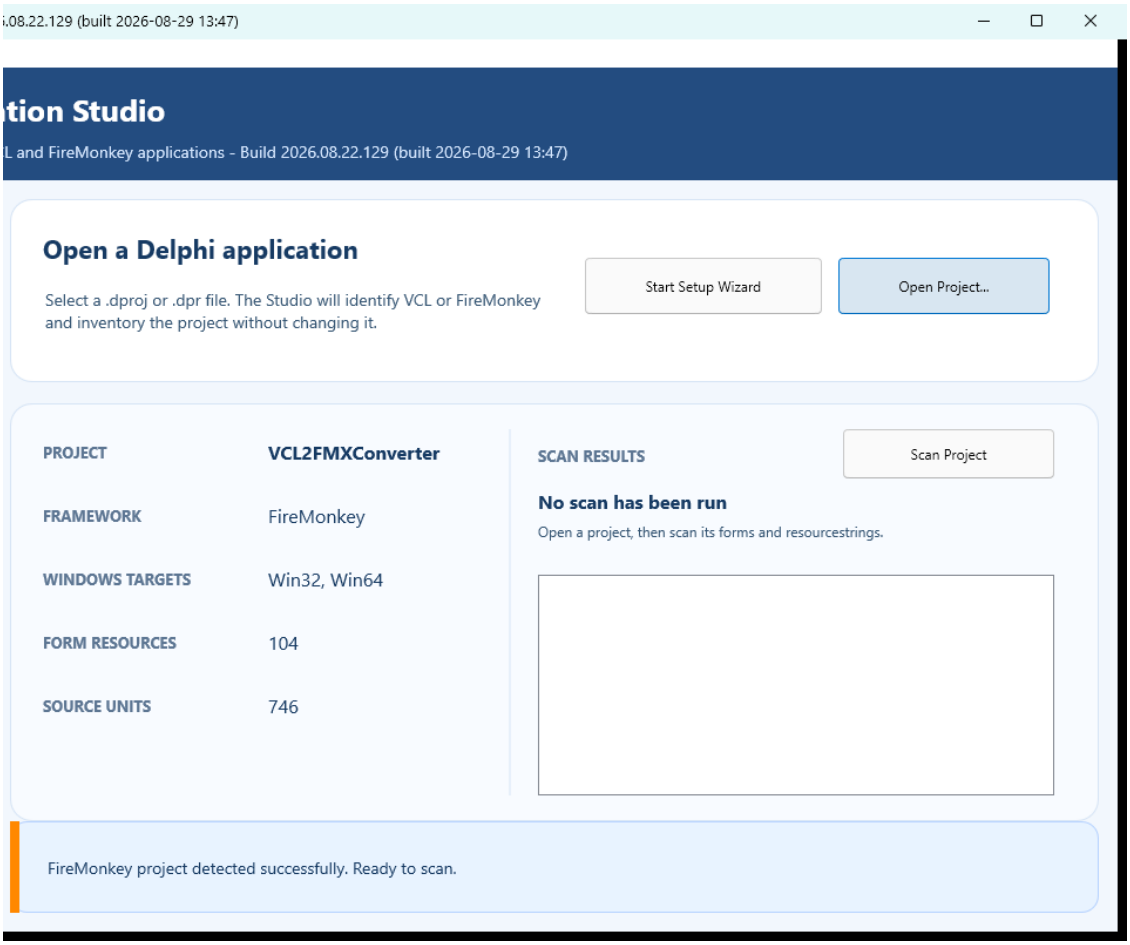

Figure 19-1. Maintenance Studio project workspace with a Delphi project loaded.

20. Maintenance Page 1 - Project

On the screen	How it helps you	When you will use it
Start Setup Wizard	Returns you to the guided workflow and keeps current project context where possible.	Use it when the maintenance task has grown into a complete scan-to-kit update.

On the screen	How it helps you	When you will use it
Open Project	Selects a .dproj or .dpr and reads its project facts without editing it.	Use it at the beginning of a Maintenance session.
Project details	Shows the project name, framework, Windows targets, form count, and source-unit count.	Confirm these values before you compare scans or open a catalog.
Scan Project	Runs the same scanner used by the Setup Wizard.	Use it after opening the project or saving UI/source changes.
Scan results	Shows unique entries, elapsed time, category totals, and observations.	Wizard and Maintenance totals should agree when the project, saved source, and merge basis are the same.

21. Maintenance Page 2 - Scan

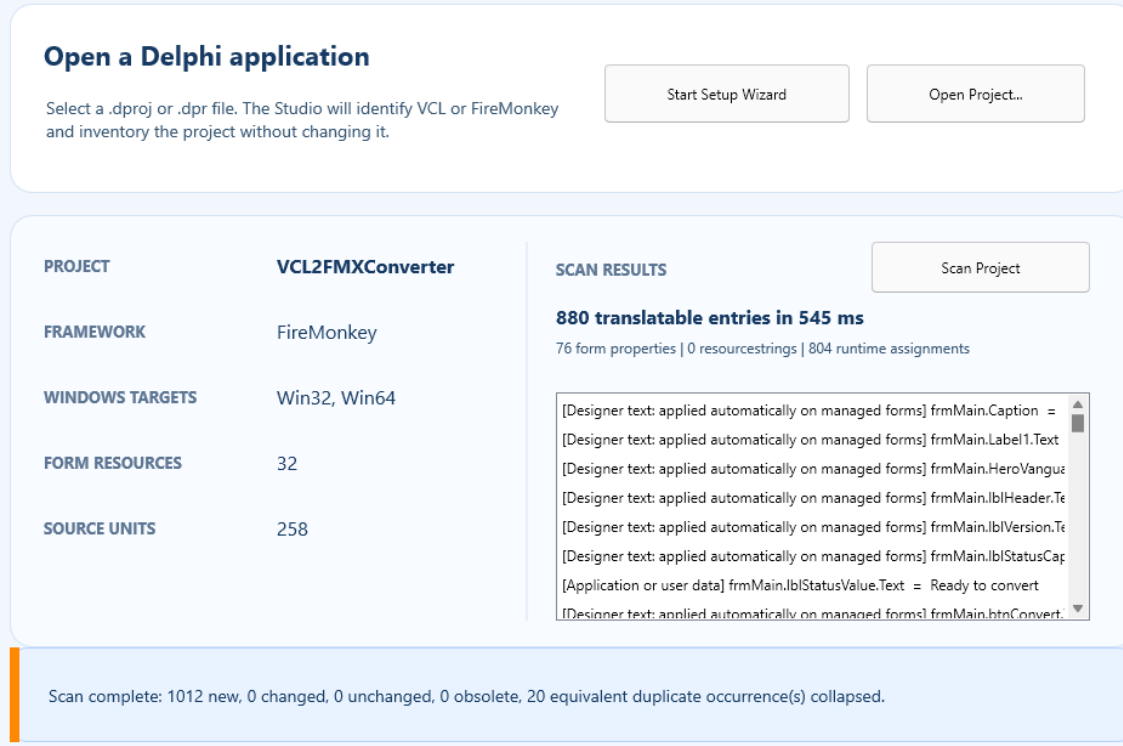

Figure 21-1. Maintenance scan showing the corrected unique/raw count contract.

Maintenance Studio and the Setup Wizard use the same scanner. When both are looking at the same saved project with the same catalog basis, their totals should agree. The large headline is the number of unique catalog entries; raw occurrences, recovered semantic contracts, and duplicates are shown separately so you can understand where that total came from.

Scanning does not edit the Delphi project. When you later save or merge the catalog, the Studio writes to its own workspace, not to your Pascal units or forms.

22. Maintenance Page 3 - Translate

This is the page you will spend the most time on when refining a language. It combines locale settings, the catalog entry list, source context, translated text, suggestions, review state, approval, CSV exchange, and automatic provider translation. Work one entry at a time when wording is delicate; use bulk actions only after you understand exactly which records they will affect.

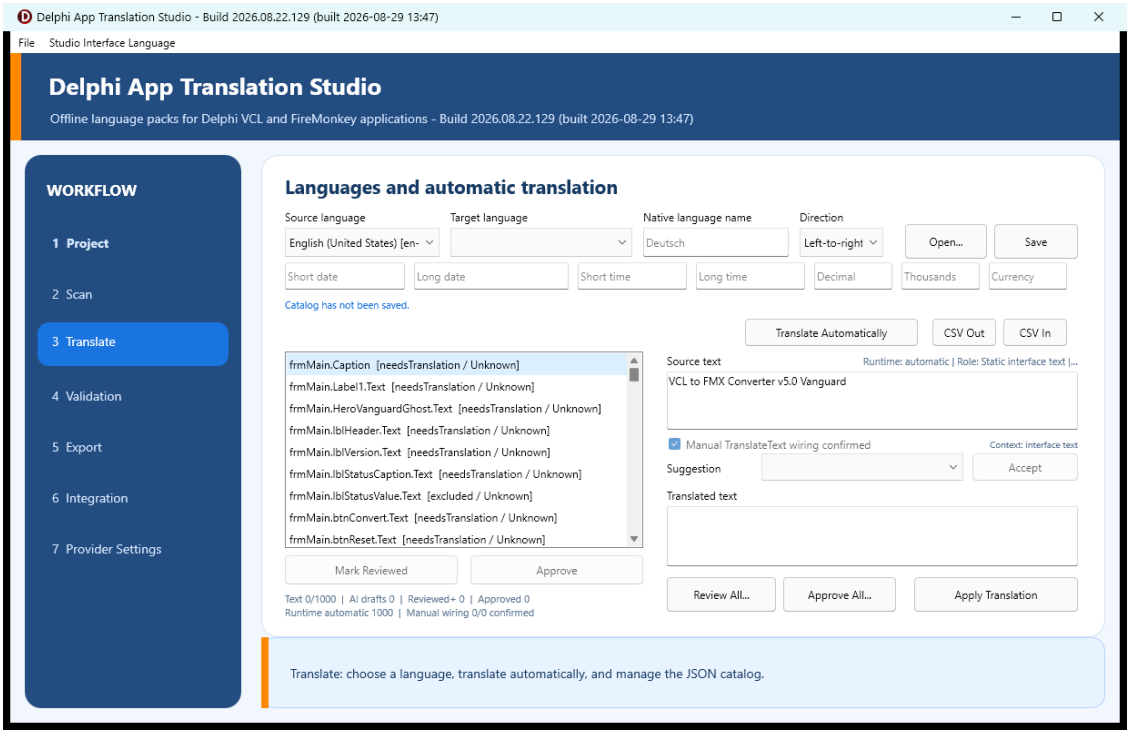

Figure 22-1. Translate page and catalog editor.

On the screen	How it helps you	When you will use it
Source language / Target language	Identifies the catalog's original and translated locales.	Choose the target before you create a new catalog.

On the screen	How it helps you	When you will use it
Native name and locale formats	Store display name, date/time, numeric separators, and currency facts.	Review them for regional correctness.
Open / Save	Loads or atomically saves a development catalog.	Open from the workspace or an explicit compatible catalog; save after manual changes.
CSV Out / CSV In	Exports/imports translator-facing interchange.	Preserve stable keys; validate after import.
Entry list	Shows each stable key and its translation/review state.	Select an entry to see its source context and edit its translation.
Source text	Shows the original wording and context without allowing accidental changes.	Read it before deciding between two plausible translations.
Suggestion / Accept	Offers wording from translation memory or terminology rules.	Accept it only after checking what this text means on the actual screen.
Translated text / Apply Translation	Edits and applies the selected translation.	Preserve placeholders, accelerators, line breaks, and protected tokens.
Mark Reviewed / Approve	Records linguistic review and approval for the selected entry.	Approval is a human decision, not inferred from structural validation.
Review All / Approve All	Bulk state actions.	Use only after a deliberate batch review; do not convert machine output into approval blindly.
Translate Automatically	Sends only unresolved eligible entries to your configured provider.	Existing unchanged work is preserved, and stable keys plus matching source text recover prior translations whenever possible.

23. Maintenance Page 4 - Glossary

Glossary is the direct terminology workspace requested for day-to-day development. It sits immediately below Translate on the left rail and lets you create, correct, approve, save, publish, and deploy project terminology without rerunning the Setup Wizard. A glossary belongs to one opened Delphi application and one target language. You may create and save it before a development catalog exists. Publishing requires that the Setup Wizard has completed at least once for that project and language, because the Studio must have a matching development catalog and managed dependency pack folder to update safely.

The authoritative file is %LOCALAPPDATA% \DelphiAppTranslationStudio\Workspaces\<ApplicationId>\Glossaries\<ApplicationId>.<locale>.glossary .json. The Studio displays the resolved path on the page. Saving is atomic: the new JSON is validated before it replaces the active file, and the normal .previous/.tmp/.corrupt recovery contract protects the last usable copy.

Runtime language packs are external JSON files beside the executable; glossary terms are not compiled into the EXE. Apply and Deploy updates those pack files directly. No target rebuild is required for a terminology-only correction, although you must restart a running target application or otherwise cause it to reload the updated pack before judging the result.

On the screen	How it helps you	When you will use it
Target language	Chooses the application/locale glossary to open or create.	Changing language first offers Save, Discard, or Cancel when the current glossary has unsaved changes.
Glossary file	Shows the exact authoritative workspace JSON path.	Use it to confirm the application identity and locale before editing.
Saved and pending terms	Lists each source term and preferred translation.	Select a row to load it into the editor; saved and unsaved terms are shown together until you save or cancel edits.
Source term	The exact wording in the authored application.	Enter the full source wording. Matching ignores surrounding whitespace; case is controlled separately.

On the screen	How it helps you	When you will use it
Preferred translation	The approved target-language wording.	Both source and target are required before Add / Update Term can accept the record.
Semantic concept	Optional meaning label such as close-window or play-media.	Use it to prefer the most context-appropriate term when identical source wording has more than one approved meaning.
Developer note	Optional reason or usage guidance for the term.	Explain product wording, capitalization, or domain restrictions for the next reviewer.
Case-sensitive match	Requires exact source capitalization.	Leave clear for normal case-insensitive UI matching; select only when capitalization changes meaning.
Approved terminology	Makes the term eligible to change unresolved catalog entries.	Clear it to keep a draft/reference term without applying it.
New	Clears the editor for another record.	It does not erase the glossary or save anything.
Add / Update Term	Adds a new term or updates the selected term in memory.	Choose Save Glossary afterward; the status line reports that the change is still pending.
Delete	Removes the selected term after confirmation.	The deletion remains pending until Save Glossary; Cancel Edits restores the saved file.
Cancel Edits	Discards every unsaved change to the selected language glossary after confirmation.	Use it to reload the last saved file. It does not close Maintenance Studio or cancel a scan/provider operation.

On the screen	How it helps you	When you will use it
Save Glossary	Atomically writes the complete glossary JSON.	Save before leaving the page or applying terms. Closing, opening another project, or changing glossary language also protects dirty work with a prompt.
Apply and Deploy	Publishes approved matching terms through the matching development catalog and current runtime pack, then deploys the verified pack set.	The Studio saves dirty work, loads the matching catalog automatically, skips protected Reviewed and Approved entries, validates the result, atomically refreshes the canonical and managed dependency packs, and deploys the complete pack set to every detected existing build output and remembered destination. It reports every updated, unavailable, or failed destination.

1. Open Maintenance Studio, open the Delphi project, and select Glossary on the left rail.
2. Choose the target language. An existing glossary loads automatically; otherwise a new in-memory glossary is ready immediately and no Wizard run is required.
3. Choose New, enter the required source term and preferred translation, add optional concept/note information, set Case-sensitive and Approved deliberately, then choose Add / Update Term.
4. Repeat for additional terms and choose Save Glossary. To revise a term later, select it in the list, edit the fields, choose Add / Update Term, and save again.
5. Choose Apply and Deploy. The matching catalog is loaded automatically; it does not have to be open on Translate. Read the completion report and confirm that the intended existing Win32/Win64 Debug/Release output folders and remembered application destinations were updated.

Safe terminology-only publication. Apply and Deploy does not call DeepL or Google, rerun the Wizard, rescan source, approve entries, rebuild an executable, replace an executable, or edit Pascal/DFM/FMX/DPR/DPROJ. It updates only eligible unprotected catalog records, validates and exports the affected external JSON pack, refreshes the managed dependency copy, and atomically deploys the complete pack set to destinations that already exist.

24. Maintenance Page 5 - Validation

Validation is the Studio's preflight check. It catches structural problems that would make a runtime pack unsafe or incomplete and separates them from items that need your linguistic judgment. Double-click an entry-specific issue to return directly to that record on the Translate page.

On the screen	How it helps you	When you will use it
Run Validation	Checks required translations, placeholders, accelerator keys, source changes, duplicate keys, locale/catalog metadata, and runtime structural safety.	Run after every provider pass, CSV import, glossary correction, or manual edit.
Summary	Separates errors, warnings, and informational findings.	Errors block runtime export; warnings require judgment.
Issue list	Lists key-specific and catalog-wide findings.	Double-click a key-specific item to edit it on Translate.

> **Validation cannot replace a human reader.** The Studio can prove that a pack is structurally safe, but it cannot decide whether every sentence sounds natural or uses the right product language. Reviewed and Approved therefore remain separate human decisions.

25. Maintenance Page 6 - Export

Export turns the development catalog into the compact JSON pack your application reads at runtime. The page shows current readiness and the exact output location. Export Runtime Pack remains disabled until the catalog passes the required structural checks, so an unsafe pack cannot be published by accident.

On the screen	How it helps you	When you will use it
Export summary	Reports structural readiness and separate linguistic-state counts.	Confirm the selected application ID and locale before export.
Output path	Shows the exact workspace runtime JSON path and can select it in File Explorer.	Use to inspect or manually copy a pack.

On the screen	How it helps you	When you will use it
Export Runtime Pack	Writes a compact checksum-backed offline pack atomically.	Run after validation; rerun after accepted layout proposals or translation changes.

26. Maintenance Page 7 - Integration

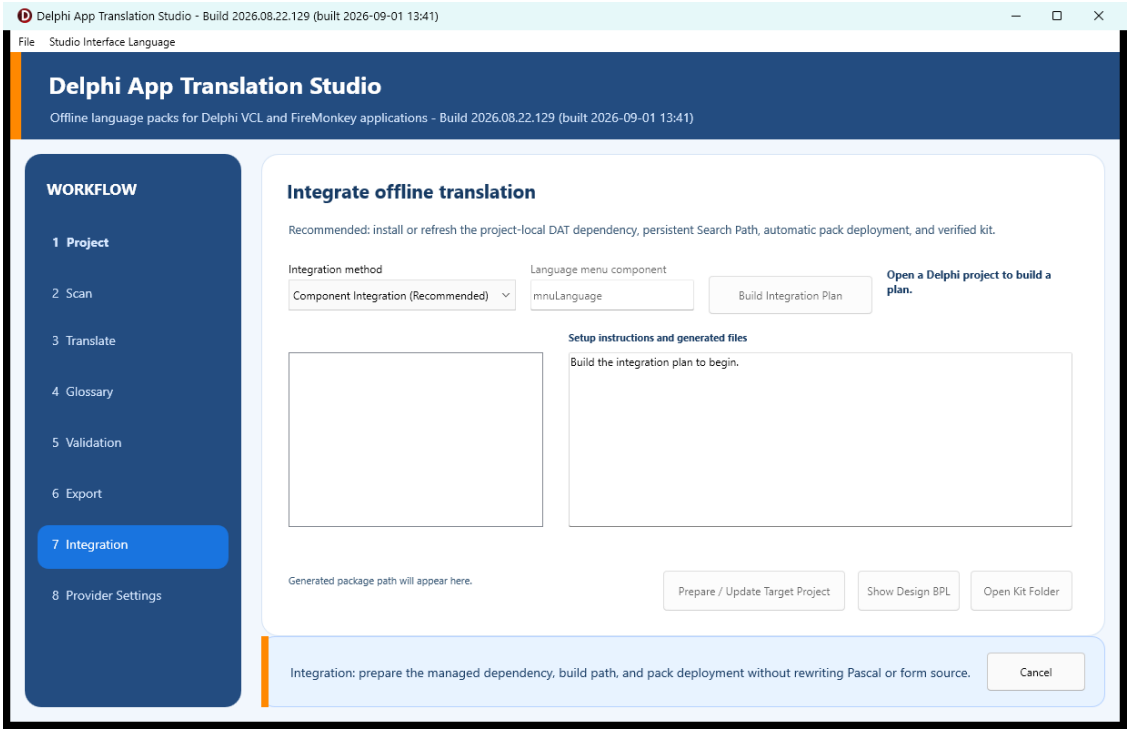

Figure 26-1. Integration page before generating a kit.

On the screen	How it helps you	When you will use it
Integration method	Chooses the normal component kit or the advanced source-integration path.	Use Component Integration unless you have a specific, reviewed reason to modify source automatically.
Language menu component	Names an application-authored menu for advanced source integration.	Not required for the connected component selector path.

On the screen	How it helps you	When you will use it
Build Integration Plan	Previews the framework, packs, generated files, and intended actions without changing your application.	Always read the plan before you generate or authorize anything.
Plan list / exact text	Shows each generated or proposed file and its content/diff.	Select each important item before generating or applying.
Prepare / Update Dependencies	Publishes the verified kit, snapshots the previous managed dependency, installs the complete project-local dependency, builds Release with a temporary Search Path, and deploys packs without editing project files.	This is repeatable and is the recommended action for normal projects.
Show Design BPL	Selects <Kit>\DesignPackages\Win32\Release\<framework design package>; its required runtime BPLs are beside it.	Use RAD Studio Component > Install Packages > Add. The Studio does not register IDE packages automatically.
Open Kit Folder	Opens the generated kit.	Use to inspect ComponentSource, DesignPackages, README, license, manifests, and completion report; copying is not required.
Advanced Preview / Authorize / Apply / Restore / Complete Reset	Supports explicitly authorized source integration with preview and recovery.	Advanced only; protect the target with Git and a backup first.

27. Maintenance Page 8 - Provider Settings

Use this page when you need to change providers, rotate a key, adjust the timeout, or tune the number of strings sent in one request. Provider settings affect translation performed by the Studio; they are never placed in the application you deploy.

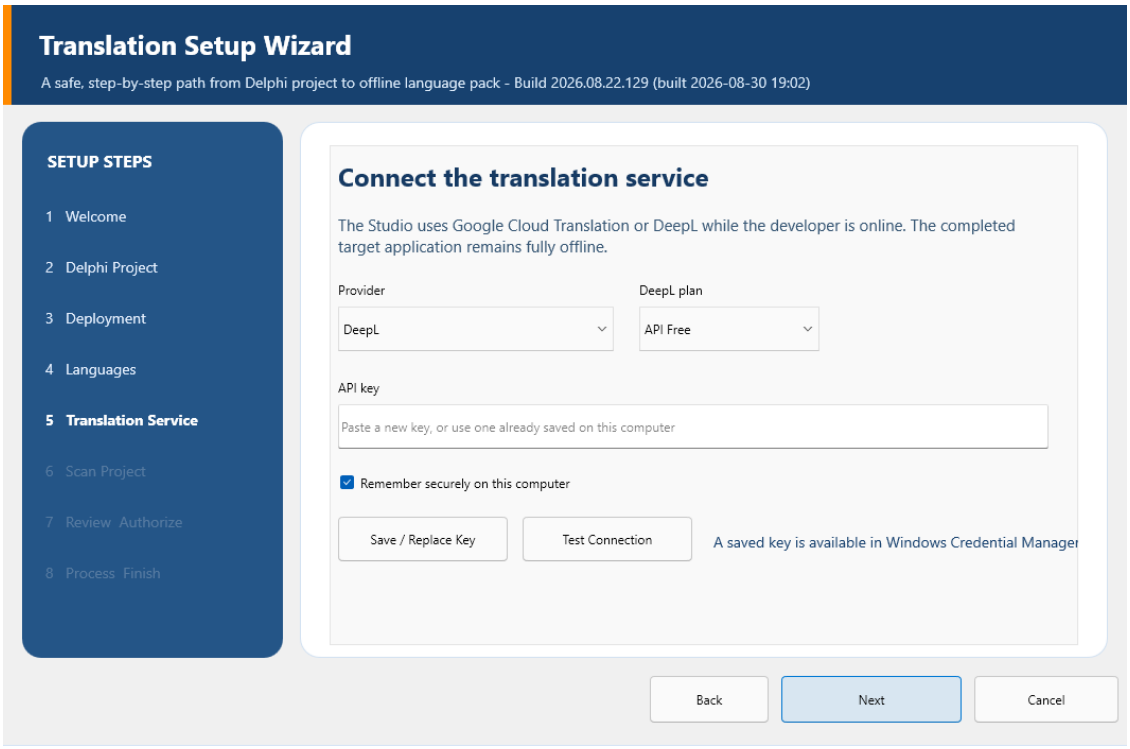

Figure 27-1. Provider controls; the Maintenance page additionally exposes timeout, batch size, and Remove Key.

On the screen	How it helps you	When you will use it
Provider	DeepL or Google Cloud Translation.	DeepL is default and recommended.
DeepL API plan	Free or Pro endpoint.	Choose the plan matching the DeepL key.
Timeout (seconds)	Limits how long one provider request may wait; values below 5 are raised to 5.	Keep the 30-second default unless a known slow connection needs more time.
Strings per request	Translation batch size; minimum 1.	Default 40 balances request overhead and response size.
API key	Masked new/replacement key.	A stored key is never displayed back into the field.
Remember securely	Credential Manager versus session-only behavior.	Use a dedicated provider key on a trusted developer computer.

On the screen	How it helps you	When you will use it
Replace / Save Key	Persists settings and handles the selected provider credential.	Use after any provider, plan, key, timeout, or batch change.
Test Connection	Checks the selected provider without freezing the Studio indefinitely.	A failure explains whether to look at authentication, quota, endpoint, network, or timeout settings.
Remove Key	Deletes only the selected provider's Generic Credential.	Use before transferring the computer or rotating a compromised key.

28. DeepL and Google Setup

You only need one provider. DeepL is the recommended default, while Google Cloud Translation is available when it better fits your account, language coverage, or quota. In either case, create a dedicated API key for the Studio rather than reusing a broad key from another application.

28.1 DeepL

1. Create or sign in to a DeepL developer account at https://www.deepl.com/pro-api.
2. Choose API Free or API Pro. The normal consumer translator subscription is not automatically an API plan.
3. Copy the authentication key from the account's API Keys area.
4. In the Studio select DeepL and the matching plan, paste the key, choose the Remember policy, and Save / Replace Key.
5. Choose Test Connection. If the endpoint plan is wrong, switch Free/Pro and test again.
6. Review character usage and billing/quota in the DeepL account. A new key does not erase provider usage limits tied to the account/plan.

28.2 Google Cloud Translation

1. Create or select a Google Cloud project.
2. Enable Cloud Translation API Basic v2 and configure billing/quota as required by Google.
3. Create a dedicated API key.
4. Restrict the key to the Cloud Translation API and apply appropriate application/network restrictions that still permit this developer computer.
5. In the Studio select Google Cloud Translation, paste the key, save it according to the Remember choice, and Test Connection.

6. Monitor quota and costs in Google Cloud. A 401/403 usually indicates key, API enablement, billing, or restriction problems; 429 indicates quota/rate limits.

29. Runtime Integration in VCL and FMX

At runtime, the DAT manager connects the language packs to your application's forms. VCL and FMX use different framework adapters, but you work with them in the same way: one manager on the primary form, one connected selector, and validated packs under Localization\Languages.

Responsibility	VCL	FireMonkey
Manager	TDATVCLLanguageManager	TDATFMXLanguageManager
Visible selector	TDATVCLLanguageComboBox	TDATFMXLanguageComboBox
Applicator	DAT.Runtime.VCL	DAT.Runtime.FMX
Later forms	Additive application notifications plus explicit ApplyToForm before Show/ShowModal when first-paint certainty is required.	Additive before-show lifecycle and open-form reapplication.
Direction	Bidirectional/layout state restored from designer baseline and reapplied per language.	Designer geometry snapshot restored before each LTR/RTL transformation.

When the application starts, the manager discovers validated packs in Localization\Languages. It checks the application ID, framework, version, locale, checksum, and source-pack compatibility before admitting a pack. It then loads the user's saved preference. If that preference is stale or invalid, the manager safely returns to the validated source language.

Choosing another language retranslates open forms immediately and remembers the selection. English is generated as a real source pack, so switching back restores the scanned English text without restarting. The runtime is designed to preserve writable control contents, focus, selections, list state, and live data while visible language text changes.

30. Build, Deploy, and Verify

A successful build is not the end of localization; it is the beginning of visual verification. Follow this sequence so you always know which executable, dependency set, and pack folder you are testing.

1. Confirm the matching design package is installed and the target primary form contains one manager plus a connected selector.
2. Confirm ApplicationId, LanguagesFolder, SourceLanguage, and any FormIdentityMappings in Object Inspector.
3. Confirm the Studio reports the managed dependency at <Target>\dependencies\DelphiAppTranslation and the dated transaction backup.
4. Confirm Project Options contains $(PROJECTDIR)\dependencies\DelphiAppTranslation\source exactly once in the Delphi Compiler Search path, and confirm the DPROJ remains unchanged by the Studio.
5. Confirm the automatic build log reports language-pack deployment to <Executable Folder>\Localization\Languages.
6. Clean and build Win32 Debug and Release. Build Win64 only when the application will ship it and the dependency/toolchain path is valid.
7. Run the executable from the actual output folder. If Windows has another copy elsewhere, close it so you do not accidentally test stale code or stale packs.
8. Test startup in English, every translated locale, switching between two non-English locales, switching RTL to LTR and back, switching back to English, closing/restarting, and a stale/missing preference.
9. Open every form, dialog, menu, tab, HTML/report page, grid, long paragraph, status area, dynamic message, and error path. Check alignment, clipping, wrapping, readable font size, accelerator keys, placeholders, mixed-language remnants, and input state.
10. Repeat the matrix for each shipped platform/configuration and record defects by locale, screen, source key, expected text, actual text, and screenshot.

31. RTL, Layout, HTML, and Dynamic Text

Translation changes more than words. A longer label may need space, Arabic or Hebrew changes reading direction, and generated HTML must preserve its markup while translating visible prose. The shared DAT contracts handle these concerns consistently, but every shipped screen still deserves a visual review.

- Direction is applied from an immutable designer baseline on every switch. It must not accumulate from the previous language.
- RTL languages right-align appropriate paragraphs and reverse visual flow where Windows/framework contracts require it; technical tokens such as WinAPI, identifiers, paths, and numbers remain protected/bidirectional as appropriate.
- Automatic fitting measures text, preserves a control that already fits, grows only into verified free space, lowers font size only to a readable floor, and wraps only as a final fallback.
- Relative, reversible layout proposals may be accepted safely in bulk. Absolute positions, absolute sizes, and grid widths require individual visual review.

- Localized HTML translates visible prose while preserving markup, comments, scripts, styles, protected nodes, product identifiers, and technical tokens. Direction and safe wrapping/padding are applied by the document contract.
- External JSON is not scanned merely because it contains strings. Project-owned operator prose is scanned only through a validated dat-translatable-resources.json manifest naming project-contained directories, file patterns, and exact string properties.
- Dynamic UI text should use stable semantic translation keys. Do not key a changing sentence only by its current English rendering when the application can identify its semantic role.

32. Security, Privacy, and Recovery

The Studio is designed so the online provider is part of development, not part of your deployed application. Keep the following habits in place and you can translate without putting credentials or customer information into the deliverable.

- Send only confirmed user-interface source strings to the chosen provider. Do not place secrets, customer data, personal data, or regulated data in translatable UI source.
- Use a dedicated restricted provider key; rotate it immediately if exposed.
- provider-settings.json contains no API key. Remembered keys are Generic Credentials; session keys exist only in process memory.
- Catalog, pack, glossary, settings, preference, and integration publication uses staging/validation/atomic replacement and recoverable previous files.
- Component kits are staged, validated, and published under the Studio export folder. The recommended path never copies provider code or keys into the target.
- Diagnostics should identify operation, artifact, recovery/fallback, and corrective context without logging provider credentials or raw authenticated responses.

33. Troubleshooting

When something looks wrong, resist the urge to make several changes at once. Start with the symptom below, check the most likely cause, make one correction, and rebuild or rerun the affected step. That keeps a small configuration issue from turning into a hard-to-explain project change.

What you see	What to check first	What to do next
Project will not compile: DAT unit not found	The managed dependency is missing, damaged, not present in the developer-owned Search path, or shadowed by an old DAT unit beside the DPR.	Run Prepare / Update Dependencies again, verify $(PROJECTDIR)\dependencies\DelphiAppTranslation\source in Project Options, and remove only confirmed stale duplicate DAT copies.
Design package will not install	Wrong RAD Studio version, wrong framework bundle, stale BPL, or DPK selected through the wrong dialog.	Use Show Design BPL, keep its adjacent runtime BPLs in place, then use Component > Install Packages > Add on the selected design BPL.
Language selector is empty	No valid packs, wrong folder, wrong applicationId/framework/checksum, missing source pack, or stale executable copy.	Deploy canonical English plus translated packs under the running EXE's Localization\Languages and rebuild/run the correct output.
Switching back to English leaves translated text	Missing/incompatible source pack or runtime text lacks a stable semantic key.	Regenerate the canonical English pack and catalog the dynamic contract.
Mixed languages after switching	Previous-language runtime text was not rebuilt or a form/content surface is outside manager refresh.	Add/repair semantic runtime keys and the form/content language-change refresh contract; do not add a replay timer.
Provider test never completes	Network/API call lacks a valid endpoint or timeout path.	Confirm plan/key/network and the bounded timeout; read the returned status instead of repeatedly clicking.
401 or 403	Invalid key, API disabled, billing restriction, endpoint-plan mismatch, or key restriction.	Correct provider setup and test again.
429	Quota or rate limit.	Wait, reduce batch size, or increase provider quota/plan.

What you see	What to check first	What to do next
Wizard and Maintenance counts differ	Different project/source state, old headline semantics, catalog merge basis, or scanner version.	Confirm identical project and current build; compare unique entries, raw occurrences, recovered contracts, and duplicates separately.
Export disabled	Validation has not passed.	Run Validation and correct all errors.
RTL text is left-aligned or controls overlap	Container/paragraph direction or baseline geometry contract is missing for that surface.	Record the exact screen/control; correct the universal contract, then test RTL->LTR->RTL rather than adding a locale-specific coordinate fix.
A current JSON file is rejected	Truncated/corrupt write or incompatible schema.	Preserve the .corrupt file, inspect .previous recovery, and regenerate from a valid source/catalog.
A remembered key appears missing	Provider selection changed, credential removed, or session-only mode used.	Select the correct provider and inspect Windows Credential Manager target DelphiAppTranslationStudio/Providers/<Provider>.
A saved glossary correction is absent at runtime	Save Glossary persisted the terminology JSON but did not publish it, the matching Wizard catalog/dependency does not exist yet, or the running application has not reloaded its pack.	Choose Glossary > Apply and Deploy, read the destination report, then restart the target from the reported output folder and select the corrected language.

34. Complete First-Time Checklist

Use this checklist before you call the first localized build complete. It is deliberately thorough; checking each item once is faster than trying to reconstruct a missing setup step later.

- [ ] Entire repository downloaded/extracted or cloned; folder structure intact.
- [ ] Studio builds and starts from the intended bin output.
- [ ] Pristine target backup exists; disposable test copy builds before localization.

- [ ] Correct VCL/FMX design BPL installed through Install Packages.
- [ ] One manager and one connected selector saved on the primary form.
- [ ] ApplicationId, LanguagesFolder, and SourceLanguage verified.
- [ ] Target files saved and running target application closed before final processing.
- [ ] DeepL/Google key saved with the intended security policy and connection tested.
- [ ] Source and target locales verified; scan counts reviewed.
- [ ] Required safety ZIP, saved-project/application-closed confirmation, and authorization reviewed.
- [ ] Localization Review completed; glossary and layout decisions saved.
- [ ] Any later terminology-only correction was published with Glossary > Apply and Deploy; the destination report names the intended outputs.
- [ ] Validation has no blocking errors; runtime packs exported.
- [ ] Prepare / Update Dependencies completed; dated previous-dependency snapshot recorded.
- [ ] Complete dependency installed under dependencies\DelphiAppTranslation; target DPROJ hash unchanged.
- [ ] Baseline Release build passed; canonical source and translated packs deployed automatically beside each executable.
- [ ] Full LTR, RTL, switch-back, restart, dynamic-text, HTML/report, and platform matrix passed.
- [ ] Intended files committed to Git; API keys and proprietary catalogs excluded.

35. Quick Path Reference

These are the paths you are most likely to need while building, troubleshooting, or moving the project to another computer.

Purpose	Path
Studio repository	C:\DelphiProjects\Delphi App Translation (example)
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
Studio user settings	%LOCALAPPDATA%\DelphiAppTranslationStudio
Per-project workspace	%LOCALAPPDATA% \DelphiAppTranslationStudio\Workspaces\<ApplicationId>
Target preference	%LOCALAPPDATA%\<ApplicationId>\language.ini
Generated component kit	<Studio>\export\component-integration\<ApplicationId>
Managed target dependency	<Target Project>\dependencies\DelphiAppTranslation
Verified BPL bundle	<Studio>\export\component-integration\<ApplicationId>\DesignPackages\Win32\Release
Deployed packs	<Target EXE Folder>\Localization\Languages
Guides	<Studio>\docs\guides and <Studio>\docs\pdf

36. 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.