VCL2FMXConverter v7.0 Atlas

Translation Guide

Using the 35 languages and maintaining translation packs

Delphi VCL to FireMonkey conversion

Last changed October 18, 2026

V7 Atlas documentation

Contents

Installing the complete Windows package....3
Choosing a language and opening Help....4
Understanding what is translated....4
Preparing Translation Studio....5
Correcting an existing translation....5
Adding another language....6
Keeping terminology consistent....6
Exporting installing and testing a pack....7
Resolving problems and reporting corrections....7
Translation files and maintenance responsibilities....8

Use the included languages, correct wording, add another language and install a validated runtime pack. This guide also explains translated reports, online Help, glossary maintenance and the files involved.

The 35 included language packs are machine translations and have not been reviewed by native speakers. Native-speaker review and application testing remain part of translation maintenance.

This guide covers V7 Atlas. The installed Help contains the same instructions. The PDF and DOCX editions are in English.

Installing the complete Windows package

Download the Win32 or Win64 application ZIP and extract the entire ZIP into its own writable folder before starting VCL2FMXConverter.exe. Win64 needs 64-bit Windows. Keep the Win32 and Win64 editions in separate folders. Do not run from inside the ZIP or copy only the executable. RAD Studio is needed to build Delphi projects, not to run the downloaded converter.

Keep this layout together: VCL2FMXConverter.exe and WebView2Loader.dll in the same folder; Localization\Languages containing all 35 JSON language packs; mapping_packs containing the 22 supplied JSON packs; docs\Help containing the complete Help tree and guides; and the supplied licenses and README.txt. These are relative paths beneath the executable folder, not folders beside the ZIP.

Localization\Languages is required for interface and report translations. Preserve the English en-US.json pack as well as the other supplied languages. A missing or unreadable selected report language pack can stop a conversion with "The selected report language pack could not be loaded." Restore the complete folder from the same release ZIP and restart the converter.

WebView2Loader.dll must be directly beside VCL2FMXConverter.exe and match the executable architecture: x86 for Win32, x64 for Win64, even when running Win32 on 64-bit Windows. Use the DLL supplied in that edition of the release ZIP. The loader supports the embedded mapping browsers; it is not the WebView2 Runtime.

Install or update Microsoft Edge WebView2 Evergreen Runtime if it is absent. Obtain it from https://developer.microsoft.com/microsoft-edge/webview2/ under Download the WebView2 Runtime. The Evergreen Bootstrapper needs Internet access; use the appropriate Evergreen Standalone Installer for an offline computer. Having the Microsoft Edge browser installed is not a substitute for the Runtime. Do not download replacement DLLs from third-party DLL sites.

The embedded browsers use Edge when available and may fall back to an older browser engine if WebView2 is unavailable. A window that opens is therefore not proof of correct WebView2 deployment. If mapping views are blank or render incorrectly, check the loader location and architecture, then the Runtime installation. Close and reopen the converter after correcting the installation.

Keep mapping_packs with the application even when mapping-pack support is not currently enabled. Missing packs remove that optional conversion coverage. Preserve docs\Help and all its subfolders; missing Help files prevent Help from opening. Help opens in your default browser. Keep the converter running while using its local Help URL; English Help is available offline, while automatic Help translation needs Internet access.

Use a writable output folder separate from both the original VCL project and the converter installation. Keep an independent backup of the original project. The signed-in Windows account also needs access to its local application-data folder for language settings and browser data. Before replacing an installation, close the converter and back up customized language or mapping packs. Extract a fresh complete release rather than mixing files from different releases.

Before the first conversion, verify that the Language selector lists the supplied languages, the mapping views display, and Help opens. Review the conversion report, then compile and test the generated project in Delphi. Successful conversion does not remove the need to review unsupported components and manual changes.

Choosing a language and opening Help

Start Atlas and use the **Language** selector in the header. Each choice includes its native name and locale code, such as `fr-FR`. Select a language to update the interface. The selection is saved for the next launch. On first use, Atlas attempts to use a supported system language; English is the fallback.

The 35 bundled interface packs work offline. Select English at any time to return to the source wording. Language selection changes presentation; it does not change the conversion rules or translate the application you are converting.

1. Select the desired language in Atlas.
2. Click **Help**. The installed manual opens in your browser with that language requested.
3. Allow the online translation service time to load. Keep Atlas running while navigating Help.
4. After changing the converter language, click Help again to open it with the new selection. An already open Help tab does not automatically follow later changes in Atlas.

English Help remains readable offline. Opening `docs/Help/index.html` directly as a file also provides English Help. Use the Help button for automatic translation. Downloaded PDF and DOCX guides are English documents and are not translated by the Help widget.

Understanding what is translated

The interface pack supplies captions, instructions, built-in messages, reference-table explanations and other display text. Arabic, Persian, Hebrew, Pashto and Urdu also use right-to-left text presentation. Delphi identifiers, component names, paths, code examples and source evidence retain their original spelling where protected by the application.

The live conversion log is translated for display. Saved HTML and text reports use the language selected when the conversion starts. Selecting another language afterward does not rewrite reports already saved. JSON reports retain canonical English text and the stable report schema so tools can process them consistently.

Generated Pascal, FMX forms and project files are conversion output, not natural-language translations. A language pack changes displayed explanations; a mapping pack changes conversion mappings. Custom mapping-pack notes and external exception messages may remain in their original language.

Help uses a separate online translation service. Atlas serves the installed English pages through a local browser connection and passes the selected language to the translation widget. Help text is processed by that service. The converter's offline interface translation does not require that connection.

The included packs are machine translations and have not been reviewed by native speakers. Structural validation checks such things as placeholders and pack identity; it does not establish linguistic accuracy. Correcting an interface pack does not correct an online Help translation. Permanent Help wording changes belong in the English Help source; translation-service wording is a separate concern.

Preparing Translation Studio

Download the Delphi App Translation Studio (https://delphitranslate.github.io/) and its manuals and guides. Use its Maintenance Studio for the already integrated Atlas project. Atlas already contains its language manager and selector; ordinary translation maintenance does not require repeating component integration or rebuilding the converter.

Use a separate working copy of the matching V7 complete source package. It must contain `VCL2FMXConverter.dproj`, the source files, `Localization/Development`, `Localization/Languages` and the supplied translation metadata. Open the converter project, not the project you intend to convert. The executable-only package is sufficient to run Atlas, but is not the complete Studio editing workspace.

Development catalogs, named `VCL2FMXConverter.<locale>.translation-project.json`, contain editable entries, source keys and review information. The smaller `<locale>.json` runtime packs are what Atlas loads. Keep both kinds of file; a runtime pack is not a replacement for its development catalog.

Back up the working catalogs and installed packs before editing. Keep the original V7 English catalog and existing catalogs together so scanning can recover the project's runtime text contracts.

Manual editing needs no translation-provider account. Studio's **Translate Automatically** command uses its configured provider settings. Follow the Studio guide for the provider you select; those settings are separate from Atlas Help's online widget. Review exactly which text a provider operation will send.

Correcting an existing translation

1. Open **Maintenance Studio**. Under **Project**, open `VCL2FMXConverter.dproj` from the matching source workspace.
2. Under **Translate**, use **Open...** to open the existing development catalog for the language you want to correct. Check its target locale and native language name.
3. Choose **Scan Project** to compare the catalog with the saved forms and source. Review entries marked new, changed or obsolete. Do not discard runtime entries merely because they are not visible form captions.
4. Select the affected entry. Read its source text and context, edit **Translated text**, and choose **Apply Translation**. Use **Save** to save the catalog.
5. Use **Mark Reviewed** only after reviewing the translation's meaning. **Approve** records a separate final decision. Do not bulk-approve machine output just to clear a status.
6. Continue with validation, export and installation as described below.

Preserve format placeholders such as `%s`, `%d` and indexed variants, deliberate line breaks, HTML structure and protected Delphi identifiers. Translate the surrounding sentence rather than changing keys, component names or metadata.

For work outside Studio, **CSV Out** and **CSV In** provide an interchange workflow. Edit only the Translation column, then review the import's conflicts and protected-entry decisions. The development JSON catalog remains the complete record.

Adding another language

1. Save and close any catalog you are editing. Start a fresh Maintenance Studio session and open the same V7 converter source project.
2. Run **Scan Project** to create a fresh catalog from the converter and its supplied workspace metadata. Do not open an existing non-English catalog and simply change its locale; that can retain the old translations and catalog filename.
3. Under **Translate**, keep the source language English and choose the new target language. Verify its native name, direction and locale formats.
4. Choose **Save**. Check that the resulting development filename contains the new locale and does not overwrite an existing language.
5. Translate the entries manually, through CSV interchange, or with the Studio's configured automatic provider. Review dynamic messages and reference-table descriptions as well as form captions.
6. Validate, export the new runtime pack and install it beside the existing packs. Restart Atlas and select the new language.

The filename and pack metadata must describe the same locale. For example, a pack for locale `xx-YY` is named `xx-YY.json`; this is a placeholder, not a real language to select. Renaming another language's file does not create a translation.

A supported, valid additional interface pack is discovered at startup without recompiling Atlas. Additional Help languages are not automatic: the Help script has an explicit supported-language list. A maintainer must confirm service support, language-code mapping and right-to-left handling before extending that list. Otherwise Help falls back to English for an unsupported language.

Keeping terminology consistent

Use Studio's **Glossary** page to maintain preferred wording for a particular project and language. A glossary is useful when the same concept appears in a button, status summary, report and reference description.

1. Open the V7 converter project and select the glossary's target language.
2. Enter the **Source term** and **Preferred translation**. Add a semantic concept or explanatory note when it helps distinguish meanings.
3. Choose **Add / Update Term**, then **Save Glossary**. Mark terminology approved only when that wording has actually been accepted.
4. Review the affected catalog entries. Use **Apply and Deploy** when you intend Studio to apply the terms and regenerate/deploy the corresponding pack to its detected application outputs. Confirm the reported destinations.
5. If you export or install manually, follow the next section. Restart the particular Atlas copy you are testing.

Saving a glossary alone stores the terminology; it does not update an already running converter. Keep the glossary with the source workspace so later translation work uses the same vocabulary.

Check context rather than applying a literal substitution everywhere. For example, a conversion mapping and a language translation are different operations. Preserve product names and Delphi API names. Keep rule-family names identical in their switches and summaries.

Studio glossary changes affect exported interface/report text. They do not configure the online Help translation service. If a Help term needs clarification, improve the English explanation and check the translated page separately.

Exporting installing and testing a pack

1. Save source files and the development catalog. Run a fresh **Scan Project** if source files were saved after the last scan.
2. Under **Validation**, choose **Run Validation**. Correct blocking errors and inspect warnings. Review status and structural validity are separate checks.
3. Under **Export**, choose **Export Runtime Pack**. Studio reports the output path. Keep any accepted layout settings associated with the export; replacing them with another language's geometry can cause clipping.
4. Close the Atlas copy you are updating. Back up its existing pack outside its language folder, then copy the exported `<locale>.json` into `Localization/Languages` beside that executable. Do not put development catalogs or backup JSON files in this runtime folder.
5. Keep `en-US.json` installed. Restart Atlas, select the language and check the result. Ordinary text-pack updates do not require recompiling the executable.

Test all seven pages, narrow and maximized windows, the display scaling you use, scrolling to the last paragraph, and switching back to English. Check longer captions, rule summaries, right-to-left text, live messages and reference-table notes.

Run a small sample conversion in a dedicated output folder. Inspect its newly generated HTML and text reports, and open Help from Atlas. Test Help separately because its translations do not come from the interface pack.

To undo an update, close Atlas and restore the backed-up pack, then restart. Keep customized catalogs and exported packs separately before replacing Atlas with a later version; an application update may replace bundled files.

Resolving problems and reporting corrections

A language is absent. Confirm you updated the folder beside the executable you actually launched, that the file is a runtime pack rather than a development catalog, and that its locale, application identity and framework are valid. Restart Atlas after adding files. Do not create duplicate packs for the same locale.

Some text remains English. First distinguish protected identifiers and source evidence from user-facing prose. Missing translations can fall back to English. Check the matching catalog entry, validate and export again. Custom mapping notes and external exception messages may require separate attention.

Text is clipped or overlaps. Record the selected locale, affected page, window size and Windows display scaling. Check the wording and accepted layout settings. Do not shorten a translation so much that it changes the instruction's meaning.

Help remains English. Open it from Atlas, keep Atlas running, and check Internet access and browser restrictions on the translation widget. A newly added interface language may need a Help mapping update. English remains available if translation cannot load.

A saved report uses the previous language. Existing reports retain the language captured for that run. Choose the language before starting the next conversion.

To report a correction, email churchesite@gmail.com or use the translation feedback instructions. Include the converter version, locale, screen or Help topic, current wording and proposed wording. Remove private paths and project details from screenshots and reports. An email template opens for review; it sends nothing automatically.

Translation files and maintenance responsibilities

`Localization/Languages` contains the installed runtime packs. They identify `applicationId` as `VCL2FMXConverter`, use the `FireMonkey` framework, and retain `en-US` as the source language. The selector discovers compatible packs; the advertised count of 35 describes the bundled set, not a hard-coded interface limit.

`Localization/Development` contains the editable catalogs. `Localization/Layout` records language layout proposals, while runtime packs contain accepted layout rules used by the application. Preserve these files and the project terminology metadata when maintaining translations.

The designer-owned language manager applies the selected pack to form controls, dynamic text and embedded reference content. It preserves control state, saves the user's selection, and falls back to source-language behavior when a pack is unavailable. Developers should keep control properties editable in the Form Designer.

The installed Help lives under `docs/Help`. Its editable Help Doc Creator project is `docs/guides/VCL2FMX User Help.hdc.json` in the source workspace. Keep that project, exported topics, navigation/search indexes and printable manual synchronized when changing English Help.

`docs/Help/assets/help-translation.js` maps converter locale codes to the online service. Maintainers adding Help support must check the supported list, aliases and direction handling, then run the Help translation checks. Adding a JSON interface pack alone does not alter this script or the website's language list.

For source maintenance, the existing language, form, report and Help checks under `tools` help verify coverage and behavior. These checks complement native-speaker review and visual testing; they do not replace either.