VCL2FMXConverter v7.0 Atlas

Converter Runtime Flow

The operational path from startup to conversion reports

Delphi VCL to FireMonkey conversion

Last changed October 18, 2026

V7 Atlas documentation

Contents

Foreword....3
v7.0 Current Standing....3
System at a Glance....3
Startup Sequence....4
Mapping Load and Reference Surface....4
v7.0 Include Analysis Flow....4
v7.0 Windows Messaging Flow....5
Uses-Clause and Runtime Cleanup....5
DFM/FMXL and Encoding Flow....5
Contract Runner Relationship....5
End-to-End Runtime Flow....6
V7 Atlas language support and operating safeguards....7
- Choosing a language....7
- Reading translated Help....7
- Editing and adding translations....7
- Translation review....7
- Source and output checks....7
- Text encoding....8
- Implementation responsibilities....8

Foreword

This document describes VCL2FMXConverter v7.0 Atlas. Language support is the principal release feature. Existing conversion capabilities are retained, with additional input, output and encoding safeguards documented in this edition.

This document explains how the converter starts, loads mapping knowledge, scans a project, converts Pascal and DFM content, applies v7.0 include and Windows-message analysis, generates output, and reports the result.

Version 7.0 is contract driven. The converter still uses the same runtime pipeline for normal user conversions, but structural behavior is now guarded by executable conversion contracts before release.

v7.0 Current Standing

Area	Current v7.0 state
Release line	Version 7.0 Atlas
Contract expectations	227
Mapping packs	22 JSON packs carried forward from v4.1.8
Built-in component mappings	77
Mapping-pack component rules	493
Primary v7.0 additions	Contracts, include analysis, Windows messaging categories, protected uses cleanup, GDI reporting/conversion contracts, dry-run preview, semantic checks, and updated UI/reference surfaces.

System at a Glance

Phase	Runtime purpose	v7.0 notes
Startup	Create the main FMX form, initialize tabs, options, UI state, and reference browsers.	The Atlas UI exposes Dashboard, Project Scan, Component Map, Property Map, Event Map, Conversion Output, and Rules.
Input selection	Collect source folder, output folder, file scope, subfolder option, and rule toggles.	Normal conversion remains user-project based; contracts are not runtime data for user conversions.
Mapper load	Load built-in component mappings and optional JSON mapping packs.	22 mapping packs remain available; pack usage is reported.
Pascal analysis	Parse and rewrite source units, uses clauses, Winapi/VCL references, runtime helper needs, and report issues.	v7.0 analyzes include directives, nested includes, recursive includes, and include boundaries.
DFM/FMXL generation	Convert DFM form trees into FMX form output.	TMemo/TStrings collection handling, TStringGrid event ordering, and UTF-8 BOM FMX output remain preserved.
Specialized rewrites	Apply data-aware, LiveBindings, compatibility, WinAPI, GDI, and runtime-normalization rules.	Windows messaging remains category based, with false-positive protections and explicit manual-review reporting.

Phase	Runtime purpose	v7.0 notes
Project generation	Create the FMX project shell and copy safe companion files.	Source-local include files and source-root companions are copied without searching build-output folders for ambiguous duplicates.
Reporting	Write text/HTML reports and update the UI.	Reports include blocking status, manual-review status, issue details, mapping-pack usage, Windows-message categories, include findings, and GDI review notes.

Startup Sequence

The application starts by creating the main form and initializing the tabbed workspace. Form creation wires runtime events, prepares the source/target controls, initializes scope options, and loads live reference pages.

The UI reference tabs are not static screenshots. Component Map, Property Map, and Event Map are populated from the current mapper knowledge so they match the converter rule surface more closely than old manual tables.

The Rules tab exposes the four major specialized rule families used during a normal conversion pass. Those switches affect runtime conversion behavior; contract files do not.

Mapping Load and Reference Surface

The mapper loads built-in VCL-to-FMX class mappings first, then mapping packs when enabled. Mapping packs can convert, partially convert, detect only, preserve, or route third-party controls to manual review.

Mapping source	What it contributes
Built-in mapper	Standard VCL class mappings, property rows, event rows, explicit substitutions, and fallback research behavior.
JSON mapping packs	Third-party component identification, action policy, confidence, property/event rows, vendor notes, and manual-review reasons.
Live reference tabs	Current Component Map, Property Map, and Event Map views derived from mapper data at runtime.

v7.0 Include Analysis Flow

Pascal include directives are analyzed before conversion decisions are trusted. Supported forms include {$I FileName.inc}, {$INCLUDE FileName.inc}, (*$I FileName.inc*), and (*$INCLUDE FileName.inc*).

Include resolution is source-tree aware. Beside-source includes, source-subfolder includes, nested includes, missing includes, outside-tree includes, recursive include loops, conditional include locations, and UTF-8 include content are covered by contracts.

The include implementation remains analysis-first. Include contents can influence detection and reporting, while original include directives remain in generated Pascal output. Include files are copied to the matching output location when safe.

v7.0 Windows Messaging Flow

Windows messaging detection is category based rather than a handwritten list of every constant. The converter looks for active SendMessage, PostMessage, DispatchMessage, PeekMessage, GetMessage, Control.Perform, WM/CM/CN/common-control message families, TWM/TCM records, message declarations, WndProc overrides, WM_USER offsets, and system-command handlers.

Category	Runtime result
Safe conversion	Only applies where an FMX replacement is reliable and covered by a contract.
FMX helper replacement	Used for simple control-message families when the generated helper logic is safer than preserving Winapi calls.
System.Messaging / bridge	Reserved for message-bridge cases that need an FMX event/message model.
Platform-specific preservation	Kept only when active generated code still requires the Windows unit and the report explains why.
Manual review	Default for WndProc, message handlers, WM_USER, system command side effects, message pumps, and ambiguous behavior.
False positive	Comments, strings, user-defined methods, and unrelated identifiers are ignored under contract.

Uses-Clause and Runtime Cleanup

V7 retains protected and conditional uses cleanup. The former leftover Vcl.Themes case is covered by contract expectations, along with conditional VCL/Winapi units and no-op false-positive paths.

The converter should add Winapi.Messages, Winapi.Windows, System.Messaging, or FMXMessageBridge only when generated active code still requires those units. It should not inject Windows units for false positives.

DFM/FMXL and Encoding Flow

The DFM parser continues to preserve multi-line string collections such as Lines.Strings, Items.Strings, SQL.Strings, and Params.Strings. TStringGrid event properties are emitted before generated column child objects so Delphi can open the FMX form.

Generated FMX files are written as UTF-8 with BOM so accented French text and other international characters load correctly in Delphi.

Contract Runner Relationship

A conversion contract consists of a small sample Delphi project, form, or source file together with an expectation file that defines the converter output, report items, uses-clause behavior, and required validation behavior.

The contract runner is the test utility that loads each contract, runs the converter against its sample input, and verifies that the actual result matches the expectation file. It does not run during normal user conversions.

[v7.0 contract baseline] The current v7.0 suite contains 227 executable expectations.

Contract area	Expectation count
colors	15
comments_and_boundaries	20
components_and_events	19
data_aware	8
dfm_pairs	7
encoding	4
graphics	7
include_analysis	11
livebindings	6
project_files	5
project_integration	5
reporting	4
third_party	3
uses_clause	21
winapi_messages	40

End-to-End Runtime Flow

1. User selects a source folder and output folder.

2. Converter initializes context, options, loaded mapping packs, issue lists, and output paths.

3. Built-in and third-party mapping rules are loaded.

4. Source files are discovered while build-output folders are avoided for ambiguous companion-file discovery.

5. Pascal files are analyzed, include directives are resolved, and rewrite rules are applied.

6. DFM files are parsed and emitted as FMX with string-collection and event-order protections.

7. Project files, companion include files, and safe source-root companions are copied or generated.

8. Reports are written with conversion status, blocking/manual-review details, mapping-pack usage, include findings, Windows-message findings, and GDI/visual-review items.

[v7.0 dry-run report note] Dry-run mode follows the same analysis path as a real conversion, but generated-file writes are listed as Dry-run notices. Those notices do not count as conversion issues. Total issues is reserved for warnings, grouped manual-review items, and errors.

9. The UI updates the dashboard/log/report actions for the operator.

V7 Atlas language support and operating safeguards

Last changed October 15, 2026

VCL2FMXConverter v7.0 Atlas retains the established conversion workflow and adds 35 selectable interface language packs, translated display messages and reports, and online translation of Help. Language packs do not add conversion rules or translate generated applications. V7 also includes input, output and encoding safeguards described in this edition; it is not strictly a language-only code change.

Choosing a language

Choose a language in the header. The 35 bundled interface packs work offline and the preference is saved. On first use the language manager can select a matching system language. Missing packs fall back to English. Display messages and reference descriptions are translated; technical identifiers retain their spelling. The language selected when conversion starts is used for saved text and HTML reports. Existing reports are not rewritten when the interface language changes. Canonical JSON reports retain their English schema and data.

Reading translated Help

Help opened from Atlas uses a local HTTP server and passes the selected language to the manual. Keep Atlas running while browsing that Help session. Automatic Help translation requires Internet access; English Help remains available offline. Help hosted as ordinary web pages uses its language selector and can inherit the website selection. Opening local HTML directly with file:// does not provide the same online translation behavior. PDF and DOCX guides are supplied in English.

Editing and adding translations

Use the Delphi App Translation Studio with the matching V7 source workspace. Open the translation project/catalog under Localization/Development, scan the application, edit translations and glossary entries, run Validation, then Export Runtime Pack. Install the locale JSON under Localization/Languages beside the executable and restart Atlas. Back up customized packs before replacement. For a new language, create a catalog for that target locale rather than relabeling an existing pack. Check placeholders, technical names, long labels and right-to-left layout. A new Help language also needs a supported language mapping in the separate Help selector. Editing an interface pack does not change the online Help translation service.

Translation review

The included language packs are machine-generated and have not been reviewed by native speakers. Translation problems can be corrected using the Delphi App Translation Studio or reported to the developer with the locale, current wording and proposed correction. For a complete procedure, see the V7 Translation Guide and the Help topic Using and maintaining translations.

Source and output checks

Select one original VCL project and a separate output folder. V7 resolves paths before conversion, checks project selection before output cleanup, rejects ambiguous or mismatched DPR/DPROJ selections, and blocks empty input and unreadable source files. Scanned directory links and reparse points, and hard-linked output files, are rejected. Dry-run preview may write reports and create the output directory, but does not write converted source or remove existing build/history folders. Report actions use only paths produced by the current run.

Text encoding

Input text detection supports UTF-8, UTF-16 and explicit supported Pascal code-page directives. Without a byte-order mark or directive, valid UTF-8 is preferred; other legacy bytes use Windows-1252. Convert other legacy encodings to UTF-8 in Delphi before conversion. Generated Pascal output is UTF-8, and active code-page directives are normalized to {$CODEPAGE UTF8}. This can change generated text without changing the intended conversion rule.

Implementation responsibilities

The application remains a designer-owned FireMonkey form with no new StyleBook migration. TDATFMXLanguageManager loads locale packs and preserves control state. DAT.Runtime.FMX manages translation, text fitting and RTL layout; technical path, URL and email fields retain LTR handling. MainForm captures the report language for the worker. Converter.Core.Engine protects source evidence and identifiers while translating text/HTML report presentation; the JSON renderer remains canonical. HelpServer serves the installed manual over loopback HTTP. The parsing, rewriting, FMX generation and report-renderer modules were compared with the v6.0.0 baseline and are unchanged apart from version headers; file safety, session handling and encoding have additional changes.

Validation includes the established conversion contracts plus V7TranslationContracts, V7ReportLanguageContracts, V7ReleaseSafetyContracts, V7FormLanguageContracts and V7HelpServerContracts. These tests run during engineering validation, not during an ordinary conversion. Current test results must be assessed separately; the presence of a test does not establish that it passed.