VCL2FMXConverter v7.0 Atlas

Contract System Overview

How conversion contracts, expectations and test fixtures work together

Delphi VCL to FireMonkey conversion

Last changed October 18, 2026

V7 Atlas documentation

Contents

Purpose....3
How It Fits Together....3
The Converter Does This....3
The Contracts Verify This....3
What A Contract Contains....3
Sample Input File....3
Expectation File....3
Example Contract Rule....4
What The Contract Runner Does....4
What The Contracts Do With Presented Data....4
Current Contract Families....5
Why This Matters....5
Practical Workflow....5
Inherited maintenance changes....5
V7 Atlas language support and operating safeguards....6
Choosing a language....6
Reading translated Help....6
Editing and adding translations....6
Translation review....6
Source and output checks....6
Text encoding....7

Purpose

The contract system is a safety net around the converter. It does not learn from project data or change rules dynamically. Instead, each contract is a small known problem case with an expected result.

[v7.0 current contract baseline] The current v7.0 suite contains 227 executable expectations. The suite includes dry-run preview accounting, report-card layout, include analysis, Windows messaging categories, DFM/FMXL fidelity checks, semantic structure checks, mapping assistance, graphics/GDI handling, and protected uses cleanup.

The converter is run against that input, and the contract runner checks whether the generated output, uses clauses, FMX files, and report items match what we said should happen.

How It Fits Together

The converter still does the real work. It reads a VCL project or unit, parses Pascal and DFM content, applies conversion rules, writes FMX output, and produces a conversion report.

The contracts sit outside that process as verification. They prove that known conversion rules continue to behave correctly after later changes.

The Converter Does This

1. Reads a VCL project or unit.
2. Parses Pascal, DFM, project files, include files, forms, uses clauses, and known risky patterns.
3. Rewrites VCL units, components, properties, and events into FMX equivalents where it safely can.
4. Copies or preserves things that should not be rewritten automatically.
5. Creates generated FMX Pascal, DFM, and FMX output.
6. Creates a conversion report showing what was converted, what needs review, and what may block reliable output.

The Contracts Verify This

Each contract says, in effect: Given this input file, the converter must produce this kind of output, must not produce that dangerous output, must add these units, must remove those units, and must report these issues.

The contracts are not part of the user's converted application. They are test fixtures that prove the converter behaves correctly.

What A Contract Contains

A typical contract has two parts.

Sample Input File

The sample input file contains the problem pattern being tested. For example:

```
contracts\components_and_events\19_form_centering_float_rounding.pas
```

Expectation File

The matching expectation file describes what must happen after conversion. For example:

```
contracts\components_and_events\19_form_centering_float_rounding.expected.json
```

The .pas file contains the source pattern. The .expected.json file contains the rules that the generated output and report must satisfy.

Example Contract Rule

The new form-centering contract came from a real FMX compile problem. The unsafe converted output looked like this:

```
AForm.left := (ScreenWidth - AForm.Width) / 2;
```

That fails because FMX Width is floating-point and Left is integer. The safe converted output should be:

```
AForm.Left := Round((ScreenWidth - AForm.Width) / 2);
```

The contract therefore checks both sides of the rule.

- The converted output must contain the rounded FMX-safe assignment.
- The converted output must not contain the unsafe unrounded assignment.

What The Contract Runner Does

When the contract suite runs, the runner performs the same conversion process against each fixture, then compares the result against the expectation file.

7. Finds every contract fixture.
8. Copies the fixture into a temporary test output area.
9. Runs the converter against that fixture.
10. Reads the generated Pascal, FMX, and report output.
11. Compares the output against the expectation JSON file.
12. Fails if anything required is missing.
13. Fails if anything forbidden appears.
14. Fails if a fixture has no expectation when coverage enforcement is enabled.

What The Contracts Do With Presented Data

If the presented data is a contract fixture, the contracts validate it directly. They check the generated output, generated uses clauses, generated FMX files, and generated report text.

A contract can check questions such as these:

- Did Vcl.Forms disappear?
- Did FMX.Forms appear?
- Did Winapi.Messages get removed when no longer needed?
- Did a false positive avoid producing a report?
- Did a real Windows message produce a manual-review report?
- Did GDI drawing become FMX canvas-style code?
- Did include files get copied and analyzed?
- Did unsafe output get blocked?
- Did generated code avoid known Delphi compile errors?

If the presented data is a real project, like Carillon, the contracts do not directly act on it. The converter acts on the real project.

When a real-world failure is found, we make a small generalized contract that represents that class of failure. That way the fix protects all users, not just one project.

Current Contract Families

The conversion contracts cover several important conversion areas.

- Include-file analysis, including missing, nested, recursive, outside-tree, conditional, and UTF-8 includes.
- Windows messaging, including SendMessage, PostMessage, Perform, WndProc, message declarations, WM_USER, system commands, and false positives.
- Uses-clause cleanup, including VCL and Winapi units, conditional protected uses blocks, and leftover Vcl.Themes.
- GDI and graphics conversion, including FMX canvas output and visual-review reporting.
- DFM and FMX generation, including string collections, TStringGrid event order, and UTF-8 output.
- Components and events, including form lifecycle, timers, media, image properties, and coordinate conversion.
- Encoding, reporting, data-aware conversion, LiveBindings, project files, and third-party mapping behavior.

Why This Matters

Before the contract suite was introduced, fixes could be made by intuition. That worked for some cases, but each fix risked breaking another case.

With contracts, each known problem becomes a permanent rule. The converter can keep improving while the contracts guard against regressions.

These contracts provide a stronger foundation by checking defined conversion behavior. They complement application testing.

Practical Workflow

The working loop is simple.

15. A real project exposes a converter failure or suspicious output.
16. We reduce that failure to a small generalized contract fixture.
17. We write expectations for the safe output and forbidden output.
18. We run the contract suite and confirm the failing behavior is captured.
19. We fix the converter.
20. We rerun contracts and regression guards.
21. The fixed behavior becomes part of the permanent conversion test suite.

In short, the contracts turn real-world conversion problems into repeatable proof. They do not make the converter project-specific. They make the converter safer for everyone.

Inherited maintenance changes

This maintenance update incorporates field-test feedback from real Delphi projects. The converter now recognizes additional standard VCL controls, including TBitBtn and TTrayIcon, so they are not incorrectly reported as unknown third-party components.

TBitBtn receives a conservative FMX TButton substitution note. TTrayIcon remains an explicit platform-specific manual-review item because FMX has no direct standard tray-icon component.

V7 retains the established contract cases. Current validation results are recorded in the V7 Application Inventory; historical build results are not evidence of a current release build.

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.