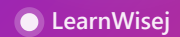


Right-to-Left Languages and Localization-Safe Layouts

Lesson Guide · Lab / Exam Guide



Lesson Guide

Right-to-left support in Wisej.NET: `RightToLeft` with `Inherit`, `No` and `Yes`, `RightToLeftLayout` mirroring a container's children, the `rightToLeft` application setting, layouts that survive text expansion, the theme's `rightToLeft` state, and the documented limits on `DataGridView`, `ListView` and `PropertyGrid`. What this means for a developer who knows basic C# and Wisej.NET controls and now has to ship the application in more than one language — and how to read this module.



Production focus

Right-to-left support in Wisej.NET: `RightToLeft` with `Inherit`, `No` and `Yes`, `RightToLeftLayout` mirroring a container's children, the `rightToLeft` application setting, layouts that survive text expansion, the theme's `rightToLeft` state, and the documented limits on `DataGridView`, `ListView` and `PropertyGrid`.

1. The production problem

GlobalDesk can already change language while it runs: the picker sets `Application.CurrentCulture`, the `Application.CultureChanged` handler reapplies every resource-backed caption, and the preview panel reformats its samples. Then a tester opens the dashboard under Arabic. Every word is translated and every one of them is in the wrong place: labels hug the left edge of their fields, the navigation column sits where an Arabic reader's eye arrives last, and the long German captions from the previous module already push one button outside a fixed-width panel.

Translation was the easy half. A localized screen also has a reading direction and a width it cannot control, and both are layout decisions you make while building the form, not repairs you make afterwards.

2. Key concepts

Concept	Why it matters
RightToLeft	A control property with the values <code>Inherit</code> , <code>No</code> and <code>Yes</code> that tells one control to adapt its own text and internal element direction.
RightToLeft.Inherit	The normal choice for a child control: it follows its parent, so the whole screen changes direction from a single decision at the top.
RightToLeftLayout	A different property that mirrors the horizontal position of a container's direct children when RTL is active, so the visual arrangement flips while your logical layout stays as designed.
The rightToLeft application setting	An entry in <code>Default.json</code> ; set to <code>"auto"</code> it makes Wisej.NET derive RTL behavior from the session's culture instead of from culture-name checks scattered through your screens.
Application.RightToLeft	The same decision expressed in code, for sessions that must be switched deliberately rather than by culture detection.
Text expansion	A translated caption is rarely the length of the English one, which is why fixed pixel positions and hand-measured panel widths break first in a localized build.
Documented RTL limits	Wisej.NET documents that <code>DataGridView</code> , <code>ListView</code> and <code>PropertyGrid</code> do not fully support RTL, so a test plan has to name them instead of assuming they mirror.

3. Lesson narrative

Two properties, two jobs

The first mistake almost everyone makes is treating right-to-left as a text-alignment setting. Arabic, Hebrew and the other right-to-left languages need the text to run the other way *and*, usually, the arrangement of the screen to be mirrored. Wisej.NET separates those two jobs into two properties, and knowing which one you need is most of this module.

`RightToLeft` belongs to a single control and changes what happens inside it: the direction of its text and of the elements it draws for itself. It takes `Inherit` , `No` or `Yes` . Leave it on `Inherit` almost everywhere, because a child then follows its parent and the whole GlobalDesk dashboard answers one decision made at the top. Setting `Yes` on individual controls gives you a screen where direction is configured in fifty places and nobody can say which one is wrong.

`RightToLeftLayout` is the other job. It mirrors the horizontal location of a container's direct children when RTL is active, so you keep designing the customer editor in the order that makes sense to you while Wisej.NET flips the visual arrangement. Note the words "direct children": setting it on the dashboard mirrors the panels the dashboard holds, and each panel decides for itself whether its own controls mirror. That is why the lab sets it on the dashboard and on the customer editor.

Let the application setting decide

`Default.json` carries a `rightToLeft` setting beside the `culture` setting you configured in the previous module. With the value `"auto"`, Wisej.NET derives RTL behavior from the culture the session is running under, so adding Arabic to the picker is enough to make direction follow the language.

`Application.RightToLeft` exposes the same decision to code when a session has to be steered deliberately.

The alternative is the pattern to avoid: a culture-name test inside every screen, then a second list of prefixes when Hebrew arrives, then a screen somebody forgot. One application setting replaces all of it, and it is why your controls can stay on `Inherit`. Direction becomes session state, exactly like `Application.CurrentCulture`: one tester switching to Arabic must never mirror a colleague's screen.

Designing a layout that can mirror

Mirroring only looks right if the layout was flexible to begin with. A control at a hand-measured `Location` inside a fixed-size panel mirrors into a position that was never designed, and a caption that is 40% longer in German simply runs past its edge. The tools are the ordinary Wisej.NET layout ones: docking so a region owns an edge, anchoring so a control grows with its container, `AutoSize` where a label must be allowed to be as long as its translation, a container layout where a row of buttons must reflow, and sensible minimum sizes so nothing collapses.

There is one deliberate exception, and the lab asks you to write the comment that explains it. Some strings are not prose: a customer code, a serial number, the contents of a code editor. Those read left to right in every culture, and flipping them turns a correct identifier into a confusing one. That is what `RightToLeft.No` is for — used sparingly, always with a reason beside it.

```
private void ConfigureDirection()
{
    // Default.json carries rightToLeft "auto", so Application.RightToLeft
    // already follows the session culture. Children stay on Inherit.
    this.RightToLeft = RightToLeft.Inherit;
    this.RightToLeftLayout = true;

    // The customer code is a technical identifier, not prose:
    // it must keep reading left to right in every culture.
    txtCustomerCode.RightToLeft = RightToLeft.No;
}
```

Testing expansion and mirrored controls

Neither problem is visible in the code, so the module ends in a test list rather than a feature. Run every localized screen through at least five cases:

- Short neutral-language text, which is the state everything was designed in.
- A language with noticeably longer translations, such as German.
- A narrow browser window, which is where anchoring and minimum sizes earn their keep.
- An RTL culture such as Arabic, checking arrangement and not only alignment.
- Mixed content, such as an Arabic label beside a Latin product code.

A pseudo-localized resource file makes the expansion cases honest. Copy the neutral keys, then lengthen and bracket every value so a caption becomes something like `[[Save changes ~~~~]]`. Anything still reading as plain English was never a resource key, and anything that clips tells you which control needs anchoring. Wisej.NET themes also understand a `rightToLeft` state, so icons and one-sided styling can adapt without a second theme; custom theme rules and custom widgets are the parts that may still need mirroring written by hand.

The controls with documented limits

Finally, the honest part. Wisej.NET documents that `DataGridView`, `ListView` and `PropertyGrid` do not fully support right-to-left. GlobalDesk shows its contacts in a `DataGridView`, so the lab asks you to record what that grid actually does under Arabic rather than what you expected. Column order, header alignment and scrollbar side are the things to look at.

This is not a reason to avoid those controls; it is a reason to name them in the test plan and verify them on the exact version and configuration you ship. A localized layout should never clip a translated caption, push a control outside the visible area, depend on English word length, bake a directional arrow into an image with no RTL variant, or force an identifier into the wrong reading direction — each of those is cheaper to fix on the first screen than on the hundredth.



Localization layers habit

Decide where each string lives before you type it. Designer resources belong to one designed screen and are applied when its controls are created; shared application resources in `.resx` carry the text that code generates or several screens reuse; `Resources-[LANG].resx` overrides Wisej.NET system labels. Never store a formatted date, number or currency: keep the value and format it for the active culture at display time. And remember that culture is per session, so one user's language must never change another user's screen.



Learning objectives

By the end of this module you'll be able to:

- ✓ Use Wisej.NET `RightToLeft` and `RightToLeftLayout` correctly.
- ✓ Explain the `rightToLeft="auto"` application setting.
- ✓ Build a layout that mirrors appropriately for RTL cultures.
- ✓ Test text expansion and mirrored controls.
- ✓ Identify Wisej controls whose RTL support has documented limitations.

Lab / Exam Guide

What you'll build in the hands-on lab, the suggested approach, and the required deliverables.



Hands-on lab

Make GlobalDesk survive a right-to-left culture. Add Arabic to the language picker and set `rightToLeft` to `auto` in `Default.json`, leave the child controls on `RightToLeft.Inherit`, and set `RightToLeftLayout` on the dashboard and the customer editor so their children mirror. Keep exactly one control left-to-right, the customer code field, and say in a comment why that identifier must not flip. Rebuild the fixed-position part of the layout with docking, anchoring and a container layout so a long translation cannot clip, add a pseudo-localized resource file whose values are lengthened and bracketed, and record what the contacts `DataGridView` actually does under RTL rather than assuming it mirrors. Finish with a five-case layout test list you can rerun on any screen.

Starting point

GlobalDesk after Module 4: the dashboard, the customer editor and the contacts grid all read their captions from resources, the language picker offers English, German and one language-region culture, and an `Application.CultureChanged` handler already calls `ApplyTextResources` and `UpdateCulturePreview`. Every screen is still built left to right, and the dashboard's navigation panel still uses the fixed `Location` and `Size` values from Module 1.

Suggested approach

- ✓ Add an Arabic entry to the language picker, set `rightToLeft` to `"auto"` beside the existing `culture` setting in `Default.json`, then run the application and switch to Arabic before changing anything else, so you can see exactly what auto detection does on its own.
- ✓ Set `RightToLeftLayout` to `true` on the dashboard and on the customer editor, and check that every child control is still on `RightToLeft.Inherit` — resist the urge to set `Yes` on individual controls to "fix" what the container should be deciding.
- ✓ Set `RightToLeft.No` on the customer code field only, and write the comment that says why a technical identifier must keep its reading direction while the prose around it flips.
- ✓ Replace the navigation panel's absolute positions with docking and anchoring, put the editor's button row in a container layout so it can reflow, and give the captions `AutoSize` and the panels sensible minimum sizes.
- ✓ Add a pseudo-localized resource file whose values are lengthened and bracketed, run the five test cases through it, note what the contacts `DataGridView` really does under Arabic, and turn the whole pass into a reusable test list.

Required deliverables

- ✓ Arabic entry in the language picker with `rightToLeft` set to `auto` in `Default.json`
- ✓ Dashboard and customer editor using `RightToLeftLayout` with their children left on `Inherit`
- ✓ One deliberately left-to-right control with a comment explaining the exception

- ✓ Layout rebuilt with docking, anchoring and a container layout so long translations cannot clip
- ✓ RTL test list covering short text, long text, a narrow window, an RTL culture and the DataGridView limitation

Student review questions

- ✓ A colleague sets `RightToLeft.Yes` on twenty controls and the Arabic screen looks correct, but the German screen is unchanged. What did they actually configure, and which single property would have done the job for the whole container?
- ✓ Your pseudo-localized pass shows three captions still in plain English. Nothing clipped and nothing threw. What does that tell you about those three captions, and which earlier module's work has a gap?
- ✓ The contacts grid does not mirror the way the rest of the dashboard does. Which of your options — accept it and document it, replace the control, or write mirroring by hand — can you defend to a reviewer, and what would you have to verify before promising any of them?



Localization layers habit

Decide where each string lives before you type it. Designer resources belong to one designed screen and are applied when its controls are created; shared application resources in .resx carry the text that code generates or several screens reuse; Resources-[LANG].resx overrides Wisej.NET system labels. Never store a formatted date, number or currency: keep the value and format it for the active culture at display time. And remember that culture is per session, so one user's language must never change another user's screen.