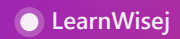


Localizing Wisej.NET UI in the Visual Studio Designer

Lesson Guide · Lab / Exam Guide



Lesson Guide

Designer localization on a top-level Wisej.NET container: the Localizable property, the designer Language selector, the properties that legitimately differ per language such as Text, Size, Location, colors and images, the resource files the designer generates, and the rule that designer resources are applied when the controls are created. 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

Designer localization on a top-level Wisej.NET container: the Localizable property, the designer Language selector, the properties that legitimately differ per language such as Text, Size, Location, colors and images, the resource files the designer generates, and the rule that designer resources are applied when the controls are created.

1. The production problem

Module 1 gave GlobalDesk a dashboard whose captions all come from `Strings.resx`, and that is the right home for text the code generates. The customer editor is different: it is a designed screen, arranged by eye in the Visual Studio designer. Routing each of its captions through a helper call means hand-editing the designer file, and it still does not solve the real problem, which is that a German *Kundennummer speichern* does not fit inside a button sized for *Save*. This module uses the localization workflow Visual Studio already offers, so a designed screen's captions and its geometry can differ per language with no hard-coded strings and no second layout file.

2. Key concepts

Concept	Why it matters
<code>Localizable</code>	A design-time property on a top-level container; <code>true</code> switches the designer into the localization workflow so values can be stored per culture.
Designer <code>Language</code> selector	Appears once <code>Localizable</code> is <code>true</code> ; the selected culture is the version you are editing, so each change is recorded against that culture only.
The neutral language	The default version every culture falls back to, and the only one where you add, remove or restructure controls; culture views hold overrides, not a second screen.
Localizable properties	Not only <code>Text</code> : <code>Size</code> , <code>Location</code> , colors and images can each carry a different value per culture, because translation changes geometry as well as words.
Generated resource files	The designer writes localized values into resource files instead of the C# designer code, keeping language-specific values out of the code you read and merge daily.
Applied at creation time	Designer-localized values are applied while the controls are created, so changing the session culture later does not rewrite a container that already exists.
Translation-tolerant layout	Docking, anchoring, <code>FlowLayoutPanel</code> , <code>TableLayoutPanel</code> , <code>FlexLayoutPanel</code> , <code>AutoSize</code> and padding let text grow on its own, so you need fewer geometry overrides.

3. Lesson narrative

Turning on Localizable

Designer localization is enabled on the container, not on the individual controls. Open a Wisej.NET `Form` , `Page` , `UserControl` or `Desktop` in design mode, select the container itself in the Properties window, and set `Localizable` to `true` . Every control the container owns is then covered by the workflow; you never set the property on a child `Button` or `Label` .

Do this after the neutral layout is finished, not before. In the lab you complete the English GlobalDesk `CustomerEditor` user control first: fields, group box, Save and Cancel buttons, docking. Turning the switch on halfway through means making structural decisions while a culture is selected, so a decision that belongs to every language gets recorded against one of them.

Editing through the Language selector

With `Localizable` on, the designer exposes a `Language` selection on the container. Leave it on the default entry to edit the neutral version. Pick a culture such as German and the design surface becomes that localized version of the screen: it still shows your layout, but every property you now change is recorded as an override for that culture alone.

The discipline is simple: change only what must differ. In the lab you switch to German, translate the captions, widen the two longest buttons and move one label, and touch nothing else. Then switch back to the default language before any structural change, because adding a control while German is selected is how a team ends up with a screen that has a control in one language and not in another.

More than Text: Size, Location, colors and images

Wisej.NET calls out `Size`, `Location`, `Text`, colors and images as properties that can differ per language, and that list is not decoration. German compound nouns run long, so a button sized for *Save clips Speichern*. French dialogs often need more vertical space. An illustration that reads clearly in one market can be meaningless in another, so an `Image` is a legitimate localized value too.

Mnemonic characters belong in the same category: the translated caption decides which accelerator letter is available and meaningful in that language, so never assume the English one carries over. And while colors are localizable, changing them per culture is rarer than changing geometry, so treat a per-language color as a decision you can explain rather than a reflex.

What the designer writes, and where it goes

When you edit a culture view, the designer does not append a branch to `InitializeComponent`. It stores the localized property values in generated resource files beside the container, one set per culture you have edited, keeping language-specific values out of the main C# designer code. That separation makes the workflow survivable: your designer code stays readable, the per-language values are data, and a reviewer can read a resource file without reading C#.

It also draws a boundary for the rest of the application. Designer resources belong to one designed screen, so fixed captions, button text, tooltips, static images and per-language geometry fit there. Anything generated in code, any message a service produces, any validation text and anything reused by more than one screen belongs in an application resource file such as `Strings.resx` read through `Wisej.Resources.ResourceManager` — exactly what Module 1 built. GlobalDesk keeps both.

Designer resources are applied when controls are created

This is the rule that surprises people. Designer-localized values are applied as the control tree is built. Setting `Application.CurrentCulture` afterwards changes the session's culture, and code that reads resources from that moment on picks up the new language, but it does not travel back and rewrite properties on controls that already exist. A user who switches language and sees half the editor still in English has met this rule, not a bug.

You have two honest options: recreate the localized container so its resource values are applied under the new culture, or keep it and write code that reapplies the resource-backed properties yourself. For a screen this size recreating is simpler, which is why the lab adds a Recreate editor button that disposes the control and builds a new one.

```
private void btnRecreateEditor_Click(object sender, EventArgs e)
{
    // Designer resources are applied while the controls are created,
    // so the new culture only reaches this screen on a rebuild.
    this.hostPanel.Controls.Remove(this.customerEditor);
    this.customerEditor.Dispose();

    this.customerEditor = new CustomerEditor { Dock = DockStyle.Fill };
    this.hostPanel.Controls.Add(this.customerEditor);
}
```

Planning layouts that survive longer translations

Every geometry override you record is a value somebody has to maintain in every future language. The cheapest override is the one you never needed, so build the neutral layout out of things that adapt: dock a panel instead of pinning it, anchor a button to the right edge instead of fixing its `Location`, let a `FlowLayoutPanel`, `TableLayoutPanel` or `FlexLayoutPanel` place a row, switch `AutoSize` on where a caption should decide its own width, and leave padding so a longer word has somewhere to go.

Then prove it. A designer view is not the final answer: run GlobalDesk in the browser under the target culture. Check for clipped captions, buttons that overlap, labels that wrap into a neighbor, tab order that no longer follows the visual order, images, and every dialog the screen can open. The lab ends with a note listing which properties you overrode for German and which you deliberately left to the neutral layout, and that second list is the interesting one: it is the evidence your layout is doing the work instead of your overrides.



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:

- ✓ Enable localization on a Wisej.NET Form, Page, UserControl or Desktop.
- ✓ Use the designer Language selector to edit localized properties.
- ✓ Explain which properties commonly change per language: Text, Size, Location, Color and Images.
- ✓ Understand what the designer writes into `.resx` files.
- ✓ Plan layouts that survive longer translations.

Lab / Exam Guide

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



Hands-on lab

Localize the GlobalDesk customer editor through the Visual Studio designer. Finish the neutral English layout of the CustomerEditor user control first, then set Localizable to true and use the designer Language selector to add a German variant, changing only what must differ: the captions, the width of the two longest buttons and one label's Location. Run the application under de-DE and compare the real browser output with the designer view, then add a Recreate editor button that disposes and rebuilds the control so you can see for yourself why designer resources do not follow a runtime culture change. Finish with a lab note that lists the properties you overrode and the ones you deliberately left to the neutral layout.

Starting point

You open GlobalDesk from Module 1, where the dashboard page already reads every caption from `Resources/Strings.resx` through the `Texts` helper and formats its preview values with `Application.CurrentCulture`. In this lab you work on the `CustomerEditor` user control the dashboard's Customers button opens: the customer code, name, city and notes fields with their labels, a group box and the Save and Cancel buttons.

Suggested approach

- ✓ Finish the neutral English layout of `CustomerEditor` before you localize anything: dock the notes field, anchor Save and Cancel to the bottom-right, put the label and editor pairs in a container layout such as a `TableLayoutPanel` or `FlexLayoutPanel`, and set `AutoSize` on the labels that should size themselves.
- ✓ Select the user control itself in the designer, set `Localizable` to `true` in the Properties window, and confirm that the `Language` selection now appears; leave it on the default entry and rebuild once so the neutral version is the baseline.
- ✓ Switch the `Language` selection to German and translate the captions only — the labels, the group box header, Save and Cancel. Then widen the two buttons whose German text no longer fits by changing their `Size`, and move one label with its `Location` so the column still lines up. Switch back to the default language before touching anything structural.
- ✓ Run GlobalDesk under `de-DE` and check the browser, not the designer: look for clipped captions, overlapping controls, a label wrapping into its neighbor, tab order and the dialogs the editor can open. Record every difference you find between the designer view and the real page.
- ✓ Add a **Recreate editor** button on the host page that removes the current `CustomerEditor` from its container, disposes it, and adds a freshly constructed one. Change the culture, press the button, and watch the captions finally follow; then write the note listing each property you overrode for German and each one you left to the neutral layout.

Required deliverables

- ✓ CustomerEditor user control with Localizable set to true over a complete neutral English layout

- ✓ German designer language variant overriding the captions plus at least one Size and one Location
- ✓ Anchoring, docking or AutoSize applied so the German captions cannot clip
- ✓ Recreate editor button that disposes and rebuilds the container after a culture change
- ✓ Lab note listing the overridden properties and the ones left to the neutral layout

Student review questions

- ✓ A teammate adds a new phone field to `CustomerEditor` while German is selected in the `Language` list. What is the English user likely to see, and what is the rule that would have prevented it?
- ✓ The save confirmation the editor shows after a successful update still comes from `Strings.resx` rather than from the designer resources. Give two reasons why that is the right place for it, and name one caption on this screen that genuinely belongs in the designer instead.
- ✓ Your Recreate editor button works, but a colleague argues that the screen should simply reapply the localized properties instead of rebuilding. What would that code have to know that the rebuild does not, and when would their approach be the better trade?



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.