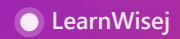


Browser Culture, Session Culture and Runtime Language Switching

Lesson Guide · Lab / Exam Guide



Lesson Guide

Where a session's culture comes from and what has to happen when it changes: the Default.json culture setting, automatic browser detection, the lang URL parameter, Application.CurrentCulture as per-session state, the Application.CultureChanged event, and reapplying resource-backed text so no half-translated screen survives the switch. 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

Where a session's culture comes from and what has to happen when it changes: the Default.json culture setting, automatic browser detection, the lang URL parameter, Application.CurrentCulture as per-session state, the Application.CultureChanged event, and reapplying resource-backed text so no half-translated screen survives the switch.

1. The production problem

After Module 3, GlobalDesk has a neutral resource file, a German one beside it, a **Texts** helper, a designer-localized customer editor and even one Wisej system label of its own. What it still has no answer for is the question a real user asks first: how do I get this in my language, and what happens to the screen when I ask? Everything so far was decided while the application was being built; this module is about the two moments when the decision is actually made — when a session starts, and when a user changes their mind halfway through it.

2. Key concepts

Concept	Why it matters
automatic browser detection	Wisej.NET can read the language the browser reports and start the session with it, so each user gets a sensible culture without one server-wide setting deciding for everybody.
<code>culture</code> in <code>Default.json</code>	The application setting that decides where a new session's culture comes from; it defaults to <code>auto</code> , and a fixed name such as <code>de-DE</code> pins every session to one locale instead.
<code>rightToLeft</code> in <code>Default.json</code>	Its neighbor setting, whose <code>auto</code> value follows whatever culture the session ends up with; Module 5 builds on it, but you meet it in the same file here.
<code>Application.CurrentCulture</code>	Gets or sets the culture of the <i>current Wisej.NET session</i> , which is exactly the scope a multi-user web application needs: one user's choice must never repaint another user's screen.
<code>lang</code> URL parameter	A supported way to ask for a language on the way in, for example <code>?lang=de</code> , which makes test links and language-specific entry URLs trivial to produce.
<code>Application.CultureChanged</code>	Raised when the current culture changes from the one the session started with; the single place to reapply resource-backed text and reformat anything already on screen.
designer-localized controls	Their localized values are applied when the controls are created, so a later culture change does not rewrite them — you recreate the container or reapply the values in code.

3. Lesson narrative

Where a session's culture comes from

Browsers announce the languages their user prefers on every request, and Wisej.NET can use that announcement to pick the culture for the session it is about to start. That is a much better default than a server setting, because the server has exactly one regional configuration while its users do not: two people opening GlobalDesk at the same second from browsers set to English and German each get their own session, each with its own culture, on the same process.

Detection is only a starting point, not a decision you are stuck with. A user's language may equally come from a profile row, a login preference or the picker you are about to build. The rule that matters is the same whichever source wins: resolve it once, at the start of the session, and load it into `Application.CurrentCulture` there. Code that reads resources or formats a value later should never have to ask again where the language came from.

Configuring the culture setting in Default.json

`Default.json` is where the application states its intent. The `culture` setting defaults to `auto`, which is the browser-detection behavior just described, and the neighboring `rightToLeft` setting behaves the

same way — `auto` follows the culture. In the lab you leave `culture` on `auto` so GlobalDesk starts from the browser, then prove it by changing your browser's preferred language and reloading.

A fixed value is not a mistake, it is a different intent. Pinning the setting to one culture name is the honest choice for a product that intentionally ships in a single locale, and it is the quickest way to reproduce a report about one language while you are debugging: every session lands in that culture whatever the tester's browser says. Write down which of the two you chose and why, because the next developer will read that setting as a statement about the product.

CurrentCulture is per session, never a static field

Changing the language at runtime is one assignment: set `Application.CurrentCulture` to the culture the user picked, built with `CultureInfo.GetCultureInfo` from a name such as `"en-US"`, `"de-DE"` or `"fr-CA"`. Because the property is scoped to the Wisej.NET session, that assignment changes one browser tab's world and nothing else. The language ComboBox in the lab does exactly this in its `SelectedIndexChanged` handler, storing the culture names as the items' values so the handler never parses a display caption.

The tempting shortcut is a `static CultureInfo` somewhere convenient, "so everything can see it". Static fields live in the server process, and a Wisej.NET server hosts every session in that one process, so the field would hold whichever value the most recent user selected. A colleague in another country would watch their screen change language because you tested German. If several classes need the active culture, read it from `Application.CurrentCulture` each time; that is already the per-session storage you were about to reinvent.

The lang parameter and the CultureChanged event

Two Wisej.NET features bracket the switch. On the way in, the `lang` URL parameter asks for a language before any of your code runs, so `?lang=de` opens GlobalDesk in German without touching the browser settings — ideal for a test link, a bug report or a language-specific entry URL you hand to a customer. On the way through, `Application.CultureChanged` is raised when the current culture changes from the session's initial culture, which is your hook for everything that is already on screen.

Subscribe once, when the page is constructed, and do the work in one handler so nothing depends on which control triggered the change:

```
public MainPage()
{
    InitializeComponent();
    Application.CultureChanged += Application_CultureChanged;
    ApplyTextResources();
}

private void Application_CultureChanged(object sender, EventArgs e)
{
    ApplyTextResources(); // every resource-backed caption
    UpdateCulturePreview(); // date, number and currency samples
    RecreateCustomerEditor(); // designer-localized: rebuild it
}
```

Repainting what the switch left behind

A culture change does not travel backwards through captions you already assigned. `ApplyTextResources` walks the controls whose text came from the resource layer and asks `Texts.Get` for each key again, so the heading, the Customers button and every message read their new values. `UpdateCulturePreview` is its formatting twin: the stored data has not changed at all, so it takes the same date, the same quantity and the same decimal amount and renders them with the now-active culture, which is why you keep the value and never the formatted string.

The designer-localized customer editor is the part that catches people out. Its localized property values are applied when its controls are created, so it keeps the captions it was born with no matter how often the culture changes afterwards. The fix is structural, not clever: dispose the editor and add a fresh instance, close and reopen it, or navigate to a newly created page. Whatever you choose, do it in the same `CultureChanged` handler, and then look at the whole screen — a switch that leaves half the window in the previous language is worse than no switch at all, because the user now distrusts both halves.



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:

- ✓ Explain Wisej.NET automatic browser-language detection.
- ✓ Configure `culture="auto"` or a fixed culture in `Default.json`.
- ✓ Change the session culture with `Application.CurrentCulture`.
- ✓ Use the `lang` URL parameter and handle `Application.CultureChanged`.
- ✓ Reapply resource-backed UI after a runtime language change.

Lab / Exam Guide

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



Hands-on lab

Give GlobalDesk a language switch that finishes the job. Set culture to auto in Default.json so the session starts from the browser language, add a language ComboBox offering English, German and one language-region culture, and set Application.CurrentCulture from CultureInfo.GetCultureInfo in its SelectedIndexChanged handler without ever storing the choice in a static field. Handle Application.CultureChanged to call an ApplyTextResources method that refreshes every resource-backed caption and an UpdateCulturePreview method that reformats the date, number and currency samples, and recreate the designer-localized customer editor in the same handler. Verify the same page through the lang URL parameter, then write a lab note describing what a static culture field would do to a second user signed in at the same time.

Starting point

Open GlobalDesk as you left it after Module 3: the dashboard reads every caption from `Texts.Get`, `Resources/Strings.resx` and `Strings.de.resx` hold the same keys, the culture preview panel formats a date, a number and an amount, and the customer editor has a German designer variant. Nothing on screen lets a user change language yet, and `Application.CultureChanged` has no subscriber.

Suggested approach

- ✓ Open `Default.json`, confirm `culture` is `auto`, and prove it: change your browser's preferred language to German, reload GlobalDesk and note which captions arrive translated. Write down in one line why you would pin the setting to a fixed culture name instead.
- ✓ Add a `languageCombo` ComboBox to the dashboard with three items whose values are culture names — `en-US`, `de-DE` and one language-region culture such as `fr-CA` — and whose captions are themselves resource keys, not hard-coded words.
- ✓ In `languageCombo_SelectedIndexChanged`, assign `Application.CurrentCulture = CultureInfo.GetCultureInfo(name)` and nothing else. Resist every temptation to cache the choice in a `static` field; the session already stores it.
- ✓ Subscribe to `Application.CultureChanged` in the page constructor and write the two methods it calls: `ApplyTextResources` re-reads every resource-backed caption through `Texts.Get`, and `UpdateCulturePreview` reformats the same stored date, quantity and decimal amount with `Application.CurrentCulture`.
- ✓ In the same handler, dispose the designer-localized customer editor and add a fresh instance to its host container, then switch language with the editor open and check every caption on screen. Finish by opening `?lang=de` in a second browser session and adding your lab note about static culture state.

Required deliverables

- ✓ Default.json with culture set to auto, plus a note on when a fixed culture is the right choice

- ✓ Language ComboBox handler setting `Application.CurrentCulture` from `CultureInfo.GetCultureInfo`
- ✓ `Application.CultureChanged` handler calling `ApplyTextResources` and `UpdateCulturePreview`
- ✓ Designer-localized editor recreated after the switch so no caption stays in the old language
- ✓ Verification through the lang URL parameter and a note on why culture must not live in a static field

Student review questions

- ✓ Your `ApplyTextResources` method refreshes the dashboard perfectly, yet the customer editor keeps its English captions after a switch to German. Explain what is different about those captions, and name two strategies that fix it.
- ✓ A colleague suggests caching the picked `CultureInfo` in a static field "to avoid looking it up twice". Describe what a second user, signed in from another country at the same moment, would actually see.
- ✓ The culture preview shows a currency amount. After a switch from `en-US` to `de-DE`, what has changed in memory and what has only changed on screen — and which of the two would you ever write to the database?



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.