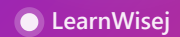


Localization Fundamentals: Culture, Language and Resources

Lesson Guide · Lab / Exam Guide



Lesson Guide

What localization actually covers beyond translation: culture names such as en, de-DE and fr-CA, .resx resource files read through ResourceManager, .NET culture fallback to the neutral resource, stable semantic keys instead of English sentences in code, and culture-sensitive formatting of dates, numbers and currency. 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

What localization actually covers beyond translation: culture names such as en, de-DE and fr-CA, .resx resource files read through ResourceManager, .NET culture fallback to the neutral resource, stable semantic keys instead of English sentences in code, and culture-sensitive formatting of dates, numbers and currency.

1. The production problem

GlobalDesk is a small service desk written for one office. Every heading, button caption and confirmation message is an English literal typed straight into the C#, and the dashboard prints amounts and dates the way a US machine happens to. When a German branch opens, none of that text has anywhere to go — and a developer who starts by hand-editing captions meets the second half of the problem: a German caption over an amount that still reads **1,850.75** is not a German screen.

2. Key concepts

Concept	Why it matters
localization	Adapting an application to a user's language <i>and</i> culture: translated text, correctly formatted dates, numbers and currency, room for longer words, and a layout that can mirror for right-to-left languages.
culture name	An identifier such as <code>en</code> , <code>de-DE</code> or <code>fr-CA</code> that names a language and, optionally, a region; the language part selects which resources are preferred, the whole name drives formatting conventions.
neutral resource	The complete fallback set of resources, held in a file such as <code>Resources/Strings.resx</code> with no culture in its name; every key the application asks for must exist here.
<code>Wisej.Resources.ResourceManager</code>	The object that takes a resource base name and an assembly and returns the value for a key in the best-matching culture, so your code never opens a file itself.
culture fallback	.NET's rule for finding a value: try the full culture, then the language-only culture, then the neutral resource, and use the first one that has the key.
semantic resource key	A stable name that describes meaning rather than wording, such as <code>CustomerEditor.Save</code> or <code>Validation.Required</code> , so the English text can be reworded without touching any other language.
<code>Application.CurrentCulture</code>	The culture of the current Wisej.NET session; two users signed in at the same time can hold two different values, which is why culture must never live in a static field.

3. Lesson narrative

Translation is only half the job

There are two different jobs hiding inside the word "localization", and beginners usually only see the first. The first job is **UI text localization**: the words the user reads, such as the GlobalDesk welcome heading, the Customers navigation caption, the Save button and the message shown after a customer is stored. Those words are authored by you, they live in resource files, and the language of the session decides which version is returned.

The second job is **culture-sensitive formatting**. A due date, a ticket count and an order total are not words at all; they are values. The same amount is written `$1,850.75` under `en-US` and `1.850,75 €` under `de-DE` , and neither of those strings belongs in a resource file or in a database column. You store the `decimal` and the `DateTime` , and you format them at the moment you put them on screen, passing the session's culture as the format provider.

That split is why the lab ends with a written note. Once you can point at something and say "this is data" or "this is presentation", the rest of the course has somewhere to hang. Build a string like `"€ " + amount` and you have quietly hard-coded a currency symbol for every country you will ever ship to.

Reading a culture name: en, de-DE, fr-CA

A culture name combines a language with regional conventions. A **language-only** (neutral) name is just the language: `en`, `de`, `fr`. A **language-region** name adds a country: `en-US`, `de-DE`, `fr-CA`, `it-IT`. In .NET these are `CultureInfo` values, and Wisej.NET exposes the one that applies to the current session as `Application.CurrentCulture`.

Two names can agree about language and disagree about everything else. A session under `en-GB` and a session under `en-US` both read your English resources, because both select the language `en`; they still write dates differently and use different currency symbols. The reverse also happens: `fr-CA` and `fr-FR` share French text but not number or currency conventions. So the language part of the name answers "which words?" and the full name answers "how are values written?".

For GlobalDesk this matters immediately. The culture preview panel you build in the lab shows today's date, a quantity and a currency amount side by side, so you can watch three values change while the captions above them stay in one language.

.resx files and ResourceManager

A `.resx` file is an XML resource file Visual Studio edits as a key/value grid. Wisej.NET does not need a translation format of its own: it uses the standard .NET model, so `.resx` files, `CultureInfo`, satellite assemblies and third-party RESX tooling all work unchanged. You add `Resources/Strings.resx` as the neutral file, and later `Strings.de.resx` beside it carrying the same keys with German values.

Your code never opens those files. It asks a `ResourceManager`, constructed from the resource base name and its assembly, for a key. Wrapping that in one tiny static helper gives you a single place to decide what a missing key looks like:

```
using Wisej.Resources;

public static class Texts
{
    private static readonly ResourceManager RS =
        new ResourceManager("GlobalDesk.Resources.Strings", typeof(Texts).Assembly);

    public static string Get(string key)
        ⇒ RS.GetString(key) ?? $"[{key}]";
}
```

Keys are the other half of the discipline. Use `CustomerEditor.Save`, `Navigation.Customers` or `Validation.Required`: names that describe meaning, not wording. Using the English sentence as the key means every reword breaks every translation file. And localize product text only — captions, menus, tooltips, validation and confirmation messages. Log identifiers, exception type names and database column names stay as they are.

Fallback: three tries, then the neutral file

When a session running under `fr-CA` asks for a key, .NET does not give up if `Strings.fr-CA.resx` has nothing for it. It tries the most specific resource first, then the less specific `fr`, then the neutral resource,

and returns the first value it finds. That is why the neutral file must be complete: it is the last place the search looks, and a key missing there is a key missing everywhere.

Fallback is a safety net, not a translation strategy. It stops a half-finished German file from blanking a screen, but it also hides gaps: the user sees English where German was expected and may never report it. So keep fallback on and find the gaps deliberately — which is what the bracketed marker in the `Texts` helper is for. A caption that renders as `[Dashboard.Welcome]` is a bug you can see.

Preparing the GlobalDesk dashboard

Preparing a page for localization is mechanical once the decisions are made. Build the dashboard normally, then walk it and ask of every visible string: who reads this? The welcome heading, the Customers button and every preview label are read by users, so each gets a key in `Resources/Strings.resx` and a `Texts.Get` call in place of its literal. Twelve keys is a realistic start for one screen.

Then do the other half. The preview panel must not print text you assembled yourself; it formats real values, passing `Application.CurrentCulture` to `ToString` so the date pattern, the separators and the currency symbol are chosen for the session. That culture is per session, so two users can read two languages in the same server process — and a static culture field would let one of them change the other's screen.



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 the difference between UI text localization and culture-sensitive formatting.
- ✓ Identify neutral, language-only and language-region culture names such as `en`, `de-DE` and `fr-CA`.
- ✓ Explain how `.resx` files and `ResourceManager` participate in .NET localization.
- ✓ Describe .NET culture fallback in beginner terms.
- ✓ Build a small Wisej.NET page with all user-visible strings prepared for localization.

Lab / Exam Guide

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



Hands-on lab

Create the GlobalDesk dashboard page with a welcome heading, a Customers navigation button and a small culture preview panel. Add Resources/Strings.resx as the neutral resource with at least twelve stable semantic keys such as Navigation.Customers and CustomerEditor.Save, write a static Texts helper over Wisej.Resources.ResourceManager that returns a bracketed key marker when a key is missing, and replace every hard-coded caption on the page with a Texts.Get call. Then fill the preview panel from real values: format today's date, a quantity and a currency amount with Application.CurrentCulture instead of building the strings by hand, and finish with a short lab note that separates the stored value from its culture-sensitive presentation.

Starting point

Open the GlobalDesk Wisej.NET project in Visual Studio. It is a small service desk with a main page, a customer editor stub and a contacts grid, and right now every user-visible word in it is an English literal typed into the C#, with amounts and dates printed however the server happens to format them. In this lab you add the dashboard page and give the application its first localization layer: resource keys for the text, and culture-aware formatting for the values.

Suggested approach

- ✓ Add a `Dashboard` page and lay out a welcome heading, a `btnCustomers` navigation button and a small culture preview panel with three read-only labels for a date, a quantity and an amount. Build it in English first, exactly as you would normally.
- ✓ Add `Resources/Strings.resx` as the neutral resource and fill it with at least twelve semantic keys such as `Dashboard.Welcome`, `Navigation.Customers`, `CustomerEditor.Save` and `Preview.Amount`. Write the English values here and nowhere else.
- ✓ Add a static `Texts` class holding one `Wisej.Resources.ResourceManager` built from the resource base name and the helper's own assembly, and a `Get(string key)` method that returns the value or a bracketed `[key]` marker when the lookup finds nothing.
- ✓ Walk the page and replace every hard-coded caption with a `Texts.Get` call. Then deliberately ask for a key you never added and confirm the bracketed marker appears on screen instead of a blank label.
- ✓ Fill the preview panel from real values: call `ToString` on `DateTime.Today`, on a quantity and on a `decimal` amount, passing `Application.CurrentCulture` each time. Never concatenate a symbol or a separator yourself, then write the lab note.

Required deliverables

- ✓ GlobalDesk dashboard page whose captions all come from resource keys, with no hard-coded user-visible text
- ✓ Resources/Strings.resx neutral file holding at least twelve stable semantic keys

- ✓ Texts helper over Wisej.Resources.ResourceManager that marks a missing key instead of returning blank
- ✓ Culture preview panel formatting a date, a number and a currency with Application.CurrentCulture
- ✓ Lab note explaining which values are data and which are culture-sensitive presentation

Student review questions

- ✓ You change the English wording of the Save button from "Save" to "Save customer". Which files have to change if your key is `CustomerEditor.Save`, and which files would have had to change if the key had been the English sentence itself?
- ✓ Your `Texts` helper returns `[Preview.Amount]` for one label. Name two different mistakes that produce exactly that marker, and say how you would tell them apart without a debugger.
- ✓ A colleague suggests storing the order total in the database as the string "1.850,75 €" so German users always see it correctly. Explain what breaks, and which part of that string is data and which part is presentation.



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.