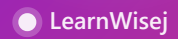


Translation Workflow: ResX Resource Manager and Other Tools

Lesson Guide · Lab / Exam Guide



Lesson Guide

Treating translations as team work: the ResX Resource Manager matrix over every .resx in the solution, adding a language column and filtering for untranslated entries, resource comments that give translators context, invariant strings, automatic translation as a draft only, export and import with a RESX-capable platform, and reviewing resource diffs like code. 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

Treating translations as team work: the ResX Resource Manager matrix over every .resx in the solution, adding a language column and filtering for untranslated entries, resource comments that give translators context, invariant strings, automatic translation as a draft only, export and import with a RESX-capable platform, and reviewing resource diffs like code.

1. The production problem

GlobalDesk now carries a neutral `Resources/Strings.resx`, a German `Strings.de.resx`, designer resources on the customer editor and a `Resources-de.resx` for Wisej system labels. Adding Italian with the Visual Studio editor means opening those files one tab at a time and comparing key lists by eye, and the person who speaks Italian is usually not the person who can open the solution. This module turns translation into a workflow with a visible gap list, a handoff and a review step.

2. Key concepts

Concept	Why it matters
Matrix view	Keys down the rows and languages across the columns, so a missing Italian value and a German term that drifted are both visible at once.
ResX Resource Manager	An open-source .NET localization tool, a Visual Studio extension and a standalone application, that shows a solution's RESX resources in one central grid.
Language column	Adding a target language creates the culture-specific file beside the neutral one and gives you an empty column to fill; the keys never change.
Untranslated filter	Lists only the entries with no value in the selected language, so "are we finished?" becomes a number instead of an opinion.
Resource comment	A note stored next to the value telling a translator whether <code>Open</code> is a verb, an adjective, a ticket state or a command.
Invariant string	A marker on entries that must never be translated, such as the product name and technical identifiers.
Export / import round trip	Columns sent out to a translator or a RESX-capable platform and returned as culture-specific <code>.resx</code> files the build treats like any other resource.

3. Lesson narrative

One grid instead of six files

The Visual Studio resource editor is perfectly good for one file. It stops being good the moment the question is comparative: does every key in the neutral file have a German value? A translation tool answers that by laying the resources out as a matrix, keys down the rows and each language as its own column. ResX Resource Manager is the usual choice: an open-source .NET tool that ships as a Visual Studio extension, which scans the active solution and gathers its RESX resources into one central grid, and as a standalone application.

It fits Wisej.NET because there is nothing to adapt. Wisej.NET keeps localized text in standard .NET RESX files, so a general-purpose RESX tool reads GlobalDesk the same way it reads any other .NET project; the Wisej StockViewer example calls out localization done with the designer together with ResX Manager. The documented feature set is the workflow itself: side-by-side language columns, untranslated entries highlighted, translating in the grid, filtering and sorting, automatic translation, import and export, code-reference detection, invariant strings, snapshots, scripting and spell checking. Code-reference detection earns its place quickly: knowing where a key is used is often the only way to decide what a one-word caption means.

What the tool is not is part of your application. It edits files; at run time GlobalDesk still performs ordinary .NET resource lookup through `Wisej.Resources.ResourceManager`, exactly as in Module 3. Never let a localization tool become a runtime dependency.

Adding Italian and turning "untranslated" into a number

Adding a language creates the culture-specific file next to the neutral one, `Strings.it.resx` beside `Strings.resx`, and gives you a new, empty column. Nothing is re-keyed; the neutral resource stays the single source of truth for which keys exist. The untranslated filter then turns a vague question into a list you can work through and hand over.

The safe order of work is worth writing on a card. Keep `Strings.resx` complete and reviewed in source control, open ResX Resource Manager, add the target language, translate or import values while preserving the keys, filter for the empty entries and clear them, build and run under the target culture, review the diff before committing because depending on its settings the tool may save your edits immediately, and finally test the user interface and not only the grid.

A full grid with a clipped button is still a bug, and Italian captions are typically longer than English ones. The opposite trap is quieter: a key with no Italian value does not blow up, it falls back to the neutral value, so the screen looks finished while the gap lives only in the grid.

Automatic translation is a draft, and the diff is the review

Automatic translation is genuinely useful for first coverage. It is a draft. Review terminology, placeholders, punctuation, accelerators, product names and context before any of it ships, because a translation can be technically correct and still be the wrong words in a user interface. The `Customer.LastOrder` entry from Module 3 makes the risk concrete: a draft that drops or reorders its two placeholders turns a formatting call into a failure.

The fix for ambiguity is context, and context lives in the resource comment, which travels with the entry through every round trip:

```
<data name="CustomerEditor.Open" xml:space="preserve">
  <value>Open</value>
  <comment>Button caption, verb: opens the selected customer.</comment>
</data>
<data name="App.ProductName" xml:space="preserve">
  <value>GlobalDesk</value>
  <comment>Product name, invariant: never translate.</comment>
</data>
```

Treat those files like code: review the diffs, keep keys stable, avoid mass renames while a round is in flight, and never let a machine draft overwrite an approved human translation without review. Because the tool may write as you type, the version control diff is the safety net.

Handing the strings to someone who never opens Visual Studio

Export and import exist for exactly this. You export the neutral column together with the target column, the translator works in that file, and the re-import matches the returned rows back to their keys, so keys and placeholders survive a person who has never seen a `.resx`. What comes back has to be the same key set.

When the languages or the reviewers multiply, the same shape moves to a cloud translation-management system. Platforms that support `.resx` include Crowdin, Lokalise, Phrase and POEditor, and the round trip is unchanged: upload or synchronize the neutral RESX, translate and review in the platform, export the

culture-specific `.resx` files, commit them, then build, run and verify in the real interface. What you buy is governance: translation memory, reviewer roles and centralized terminology.

Choosing between the designer editor, ResXManager and a cloud platform

The practical guide is short, and it is about the team rather than the tool:

- One developer, two languages: the Visual Studio RESX editor may already be enough.
- A developer team, several languages: ResX Resource Manager, an efficient in-IDE workflow.
- External translators: export and import, or a platform with RESX support.
- Many products, languages and reviewers: a cloud TMS is easier to govern.

Whatever you pick, the project's contract does not move: culture-specific `.resx` files beside the neutral file with identical keys, read through standard resource lookup. Take the lightest option that makes untranslated entries visible and the diff reviewable, and upgrade only when the round trip becomes the bottleneck.



Common pitfalls

Shipping a machine draft because the grid turned green. Losing a placeholder in a translated value. Renaming keys mid-round. Letting the tool save edits you never read in a diff. Translating the product name because nobody marked it invariant. Declaring the work done without running the application in the target culture.



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 ResX Resource Manager to review multiple RESX languages side by side.
- ✓ Add a language and identify untranslated entries.
- ✓ Explain safe use of automatic translation and source control.
- ✓ Export/import translation content for external translators or a TMS.
- ✓ Compare Visual Studio-only, ResXManager and cloud localization workflows.

Lab / Exam Guide

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



Hands-on lab

Run one honest translation round on GlobalDesk. Open the solution in ResX Resource Manager, add Italian as a new language column, and use the untranslated filter to list every key that still has no value. Add comments to the three keys whose meaning is ambiguous out of context, starting with the one-word Open caption, then translate or machine-draft the Italian values and mark the product name and the two technical identifiers as invariant so nobody translates them. Export the neutral and Italian columns, re-import them, and review the resulting .resx diff before you commit it as one reviewable change. Finish with a one-page workflow note comparing the Visual Studio editor, ResX Resource Manager and a cloud platform for a team whose translators are not developers.

Starting point

GlobalDesk after Module 5: `Resources/Strings.resx` and `Strings.de.resx` carry the same keys, the customer editor has German designer resources, `Resources-de.resx` overrides one Wisej system label, the language picker already offers English, German and Arabic, and the layout survives long strings. There is no Italian anywhere, no comment on a single entry, and no way to answer "how much is left?" except by scrolling two files side by side.

Suggested approach

- ✓ Before adding anything, open `Strings.resx` and `Strings.de.resx` in the Visual Studio editor and try to find the keys the German file is missing by eye. Time yourself; that number is the argument for the rest of the lab.
- ✓ Open the solution in ResX Resource Manager and add Italian. Confirm that `Strings.it.resx` appears beside the neutral file, that the key list is untouched, and that the new column is empty. Switch on the untranslated filter and write down the count.
- ✓ Comment before you translate. Give `CustomerEditor.Open` a comment saying whether it is a verb or a ticket state, then do the same for the two other keys whose meaning is not obvious from the value alone, and mark the product name and the two technical identifiers as invariant.
- ✓ Fill the Italian column, using automatic translation for first coverage if you like, then review every value: terminology, punctuation, and above all the placeholders in `Customer.LastOrder`. Re-run the untranslated filter until it returns nothing.
- ✓ Export the neutral and Italian columns, re-import the same file, and read the `git diff` of `Strings.it.resx` line by line before committing the round as one change. Then run GlobalDesk under `it-IT` and check the screens, not the grid.

Required deliverables

- ✓ `Strings.it.resx` produced through ResX Resource Manager with no untranslated key left
- ✓ Resource comments on at least three ambiguous keys so a translator has the context

- ✓ Invariant markers on the product name and the identifiers that must not be translated
- ✓ Export, re-import and a reviewed .resx diff committed as one change
- ✓ Workflow note comparing the Visual Studio editor, ResX Resource Manager and a cloud TMS

Student review questions

- ✓ Your Italian column is full and the application looks right in `it-IT`, but a colleague claims two keys were never translated. How could both statements be true at once, and which tool feature settles the argument?
- ✓ An Italian machine draft returned `Ultimo ordine: {1} il {0}` for a key whose neutral value is `Last order: {0} on {1}`. What breaks, when does it break, and what would have caught it before the commit?
- ✓ Your team has five languages, two of them translated by an agency that has never opened Visual Studio. Which of the three workflows would you adopt, and what would have to change before you moved to the next one up?



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.