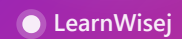


Fondamentaux de la localisation : culture, langue et ressources

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Ce que recouvre vraiment la localisation au-delà de la traduction : les noms de culture comme en, de-DE et fr-CA, les fichiers de ressources .resx lus via ResourceManager, le repli de culture .NET vers la ressource neutre, des clés sémantiques stables plutôt que des phrases anglaises dans le code, et la mise en forme des dates, des nombres et des montants selon la culture. Ce que cela signifie pour un développeur qui connaît les bases de C# et les contrôles Wisej.NET et doit maintenant livrer son application dans plusieurs langues — et comment aborder ce module.



Objectif en production

Ce que recouvre vraiment la localisation au-delà de la traduction : les noms de culture comme en, de-DE et fr-CA, les fichiers de ressources .resx lus via ResourceManager, le repli de culture .NET vers la ressource neutre, des clés sémantiques stables plutôt que des phrases anglaises dans le code, et la mise en forme des dates, des nombres et des montants selon la culture.

1. Le problème en production

GlobalDesk est un petit service desk écrit pour un seul bureau. Chaque titre, chaque libellé de bouton et chaque message de confirmation est un littéral anglais tapé directement dans le C#, et le tableau de bord affiche les montants et les dates comme le ferait une machine américaine. À l'ouverture d'une filiale allemande, aucun de ces textes n'a nulle part où aller — et le développeur qui commence par retoucher les libellés à la main découvre la seconde moitié du problème : un libellé allemand au-dessus d'un montant qui se lit toujours **1,850.75** ne fait pas un écran allemand.

2. Concepts clés

Concept	Pourquoi c'est important
localisation	Adapter une application à la langue et à la culture d'un utilisateur : textes traduits, dates, nombres et montants correctement mis en forme, place pour des mots plus longs, et une mise en page capable de se mettre en miroir pour les langues de droite à gauche.
nom de culture	Un identifiant comme <code>en</code> , <code>de-DE</code> ou <code>fr-CA</code> qui nomme une langue et, éventuellement, une région ; la partie langue sélectionne les ressources préférées, le nom complet pilote les conventions de mise en forme.
ressource neutre	Le jeu de ressources de repli complet, conservé dans un fichier comme <code>Resources/Strings.resx</code> sans culture dans son nom ; chaque clé demandée par l'application doit y exister.
<code>Wisej.Resources.ResourceManager</code>	L'objet qui prend un nom de base de ressource et un assembly et renvoie la valeur d'une clé dans la culture la mieux adaptée, de sorte que votre code n'ouvre jamais un fichier lui-même.
repli de culture	La règle .NET pour trouver une valeur : essayer la culture complète, puis la culture langue seule, puis la ressource neutre, et utiliser la première qui possède la clé.
clé de ressource sémantique	Un nom stable qui décrit le sens plutôt que la formulation, comme <code>CustomerEditor.Save</code> ou <code>Validation.Required</code> , pour que le texte anglais puisse être reformulé sans toucher à aucune autre langue.
<code>Application.CurrentCulture</code>	La culture de la session Wisej.NET en cours ; deux utilisateurs connectés en même temps peuvent avoir deux valeurs différentes, et c'est pourquoi la culture ne doit jamais vivre dans un champ statique.

3. Le fil de la leçon

La traduction ne fait que la moitié du travail

Deux travaux différents se cachent derrière le mot « localisation », et les débutants ne voient en général que le premier. Le premier travail est la **localisation du texte de l'interface** : les mots que l'utilisateur lit, comme le titre de bienvenue de GlobalDesk, le libellé de navigation Customers, le bouton Save et le message affiché après l'enregistrement d'un client. Ces mots, c'est vous qui les écrivez, ils vivent dans des fichiers de ressources, et c'est la langue de la session qui décide de la version renvoyée.

Le second travail est la **mise en forme dépendante de la culture**. Une date d'échéance, un nombre de tickets et un total de commande ne sont pas des mots : ce sont des valeurs. Le même montant s'écrit `$1,850.75` en `en-US` et `1.850,75 €` en `de-DE` , et aucune de ces chaînes n'a sa place dans un fichier de ressources ni dans une colonne de base de données. Vous stockez le `decimal` et le `DateTime` , et vous les mettez en forme au moment de les afficher, en passant la culture de la session comme fournisseur de format.

C'est à cause de cette séparation que le lab se termine par une note écrite. Dès que vous pouvez montrer quelque chose et dire « ceci est une donnée » ou « ceci est de la présentation », la suite du cours a un point d'accroche. Composez une chaîne comme "€ " + amount et vous avez codé en dur, sans y penser, un symbole monétaire pour tous les pays que vous servirez un jour.

Lire un nom de culture : en, de-DE, fr-CA

Un nom de culture associe une langue à des conventions régionales. Un nom **langue seule** (neutre) n'est que la langue : en, de, fr. Un nom **langue-région** ajoute un pays : en-US, de-DE, fr-CA, it-IT. Dans .NET, ce sont des valeurs CultureInfo, et Wisej.NET expose celle qui s'applique à la session en cours sous le nom Application.CurrentCulture.

Deux noms peuvent s'accorder sur la langue et diverger sur tout le reste. Une session en en-GB et une session en en-US lisent toutes deux vos ressources anglaises, parce que toutes deux sélectionnent la langue en ; elles écrivent pourtant les dates différemment et utilisent des symboles monétaires différents. L'inverse se produit aussi : fr-CA et fr-FR partagent le texte français, mais pas les conventions de nombres ni de monnaie. La partie langue du nom répond donc à « quels mots ? » et le nom complet à « comment les valeurs sont-elles écrites ? ».

Pour GlobalDesk, cela compte tout de suite. Le panneau d'aperçu de la culture que vous construisez dans le lab affiche côte à côte la date du jour, une quantité et un montant, pour que vous puissiez voir trois valeurs changer pendant que les libellés au-dessus d'elles restent dans une seule langue.

Les fichiers .resx et ResourceManager

Un fichier .resx est un fichier de ressources XML que Visual Studio édite sous forme de grille clé/valeur. Wisej.NET n'a pas besoin d'un format de traduction qui lui soit propre : il utilise le modèle .NET standard, si bien que les fichiers .resx, CultureInfo, les assemblés satellites et les outils RESX tiers fonctionnent sans rien changer. Vous ajoutez Resources/Strings.resx comme fichier neutre, et plus tard Strings.de.resx à côté, portant les mêmes clés avec des valeurs allemandes.

Votre code n'ouvre jamais ces fichiers. Il demande une clé à un ResourceManager, construit à partir du nom de base de la ressource et de son assembly. Encapsuler cela dans une petite classe utilitaire statique vous donne un endroit unique où décider de l'apparence d'une clé manquante :

```
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}]";
}
```

Les clés sont l'autre moitié de la discipline. Utilisez CustomerEditor.Save, Navigation.Customers ou Validation.Required : des noms qui décrivent le sens, pas la formulation. Prendre la phrase anglaise comme clé signifie que chaque reformulation casse tous les fichiers de traduction. Et ne localisez que le

texte du produit — libellés, menus, info-bulles, messages de validation et de confirmation. Les identifiants de journalisation, les noms de types d'exception et les noms de colonnes de base de données restent tels quels.

Le repli : trois tentatives, puis le fichier neutre

Quand une session qui tourne en `fr-CA` demande une clé, .NET n'abandonne pas si `Strings.fr-CA.resx` n'a rien pour elle. Il essaie d'abord la ressource la plus spécifique, puis la moins spécifique `fr`, puis la ressource neutre, et renvoie la première valeur trouvée. C'est pourquoi le fichier neutre doit être complet : c'est le dernier endroit où la recherche regarde, et une clé qui y manque est une clé qui manque partout.

Le repli est un filet de sécurité, pas une stratégie de traduction. Il empêche un fichier allemand à moitié terminé de vider un écran, mais il masque aussi les manques : l'utilisateur voit de l'anglais là où il attendait de l'allemand et ne le signalera peut-être jamais. Gardez donc le repli actif et cherchez les manques délibérément — c'est à cela que sert le marqueur entre crochets de la classe utilitaire `Texts`. Un libellé qui s'affiche `[Dashboard.Welcome]` est un bug que vous pouvez voir.

Préparer le tableau de bord de GlobalDesk

Préparer une page pour la localisation est mécanique une fois les décisions prises. Construisez le tableau de bord normalement, puis parcourez-le et demandez-vous, pour chaque chaîne visible : qui lit ceci ? Le titre de bienvenue, le bouton Customers et chaque libellé d'aperçu sont lus par les utilisateurs : chacun reçoit donc une clé dans `Resources/Strings.resx` et un appel à `Texts.Get` à la place de son littéral. Douze clés sont un début réaliste pour un seul écran.

Faites ensuite l'autre moitié. Le panneau d'aperçu ne doit pas afficher un texte que vous avez assemblé vous-même ; il met en forme de vraies valeurs, en passant `Application.CurrentCulture` à `ToString` pour que le format de date, les séparateurs et le symbole monétaire soient choisis pour la session. Cette culture est propre à la session : deux utilisateurs peuvent donc lire deux langues dans le même processus serveur — et un champ de culture statique laisserait l'un d'eux changer l'écran de l'autre.



Réflexe des couches de localisation

Décidez où vit chaque chaîne avant de la taper. Les ressources du designer appartiennent à un écran conçu et sont appliquées à la création de ses contrôles ; les ressources d'application partagées dans les `.resx` portent le texte que le code génère ou que plusieurs écrans réutilisent ; `Resources-[LANG].resx` redéfinit les libellés système de Wisej.NET. Ne stockez jamais une date, un nombre ou un montant déjà mis en forme : gardez la valeur et mettez-la en forme pour la culture active au moment de l'affichage. Et souvenez-vous que la culture est propre à la session : la langue d'un utilisateur ne doit jamais changer l'écran d'un autre.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Expliquer la différence entre la localisation du texte de l'interface et la mise en forme dépendante de la culture.

- ✓ Reconnaître les noms de culture neutres, langue seule et langue-région comme en, de-DE et fr-CA.
- ✓ Expliquer comment les fichiers .resx et ResourceManager participent à la localisation .NET.
- ✓ Décrire le repli de culture .NET en termes accessibles aux débutants.
- ✓ Construire une petite page Wisej.NET dont toutes les chaînes visibles sont préparées pour la localisation.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Créez la page tableau de bord de GlobalDesk avec un titre de bienvenue, un bouton de navigation Customers et un petit panneau d'aperçu de la culture. Ajoutez Resources/Strings.resx comme ressource neutre avec au moins douze clés sémantiques stables telles que Navigation.Customers et CustomerEditor.Save, écrivez une classe utilitaire statique Texts au-dessus de Wisej.Resources.ResourceManager qui renvoie la clé entre crochets quand elle est absente, et remplacez chaque libellé codé en dur de la page par un appel à Texts.Get. Remplissez ensuite le panneau d'aperçu à partir de vraies valeurs : mettez en forme la date du jour, une quantité et un montant avec Application.CurrentCulture au lieu de composer les chaînes à la main, et terminez par une courte note de lab qui distingue la valeur stockée de sa présentation dépendante de la culture.

Point de départ

Ouvrez le projet Wisej.NET GlobalDesk dans Visual Studio. C'est un petit service desk avec une page principale, une ébauche d'éditeur client et une grille de contacts, et pour l'instant chaque mot visible par l'utilisateur y est un littéral anglais tapé dans le C#, les montants et les dates étant affichés comme le serveur se trouve les mettre en forme. Dans ce lab, vous ajoutez la page tableau de bord et donnez à l'application sa première couche de localisation : des clés de ressources pour le texte, et une mise en forme adaptée à la culture pour les valeurs.

Approche suggérée

- ✓ Ajoutez une page `Dashboard` et disposez-y un titre de bienvenue, un bouton de navigation `btnCustomers` et un petit panneau d'aperçu de la culture avec trois labels en lecture seule pour une date, une quantité et un montant. Construisez-la d'abord en anglais, exactement comme vous le feriez d'habitude.
- ✓ Ajoutez `Resources/Strings.resx` comme ressource neutre et remplissez-le avec au moins douze clés sémantiques telles que `Dashboard.Welcome`, `Navigation.Customers`, `CustomerEditor.Save` et `Preview.Amount`. Écrivez les valeurs anglaises ici et nulle part ailleurs.
- ✓ Ajoutez une classe statique `Texts` qui détient un `Wisej.Resources.ResourceManager` construit à partir du nom de base de la ressource et de l'assembly de la classe elle-même, et une méthode `Get(string key)` qui renvoie la valeur ou un marqueur `[key]` entre crochets quand la recherche ne trouve rien.
- ✓ Parcourez la page et remplacez chaque libellé codé en dur par un appel à `Texts.Get`. Demandez ensuite délibérément une clé que vous n'avez jamais ajoutée et vérifiez que le marqueur entre crochets apparaît à l'écran au lieu d'un label vide.
- ✓ Remplissez le panneau d'aperçu à partir de vraies valeurs : appelez `ToString` sur `DateTime.Today`, sur une quantité et sur un montant `decimal`, en passant `Application.CurrentCulture` à chaque fois. Ne concaténez jamais vous-même un symbole ou un séparateur, puis rédigez la note de lab.

Livrables attendus

- ✓ Page tableau de bord de GlobalDesk dont tous les libellés viennent de clés de ressources, sans aucun texte visible codé en dur
- ✓ Fichier neutre Resources/Strings.resx contenant au moins douze clés sémantiques stables
- ✓ Classe utilitaire Texts au-dessus de Wisej.Resources.ResourceManager qui signale une clé manquante au lieu de renvoyer une chaîne vide
- ✓ Panneau d'aperçu de la culture qui met en forme une date, un nombre et un montant avec Application.CurrentCulture
- ✓ Note de lab expliquant quelles valeurs sont des données et lesquelles relèvent d'une présentation dépendante de la culture

Questions de révision

- ✓ Vous changez la formulation anglaise du bouton Save de "Save" à "Save customer". Quels fichiers doivent changer si votre clé est `CustomerEditor.Save`, et quels fichiers auraient dû changer si la clé avait été la phrase anglaise elle-même ?
- ✓ Votre classe utilitaire `Texts` renvoie `[Preview.Amount]` pour un label. Nommez deux erreurs différentes qui produisent exactement ce marqueur, et dites comment vous les distingueriez sans débogueur.
- ✓ Un collègue propose de stocker le total de commande en base de données sous la forme de la chaîne "1.850,75 €" pour que les utilisateurs allemands le voient toujours correctement. Expliquez ce qui casse, et quelle partie de cette chaîne est une donnée et quelle partie est de la présentation.



Réflexe des couches de localisation

Décidez où vit chaque chaîne avant de la taper. Les ressources du designer appartiennent à un écran conçu et sont appliquées à la création de ses contrôles ; les ressources d'application partagées dans les `.resx` portent le texte que le code génère ou que plusieurs écrans réutilisent ; `Resources-[LANG].resx` redéfinit les libellés système de Wisej.NET. Ne stockez jamais une date, un nombre ou un montant déjà mis en forme : gardez la valeur et mettez-la en forme pour la culture active au moment de l'affichage. Et souvenez-vous que la culture est propre à la session : la langue d'un utilisateur ne doit jamais changer l'écran d'un autre.