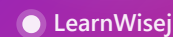


Culture du navigateur, culture de session et changement de langue à l'exécution

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



Guide de la leçon

D'où vient la culture d'une session et ce qui doit se produire quand elle change : le paramètre culture de Default.json, la détection automatique de la langue du navigateur, le paramètre d'URL lang, Application.CurrentCulture comme état propre à la session, l'événement Application.CultureChanged, et la réapplication des textes issus des ressources pour qu'aucun écran à moitié traduit ne survive au changement. 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

D'où vient la culture d'une session et ce qui doit se produire quand elle change : le paramètre culture de Default.json, la détection automatique de la langue du navigateur, le paramètre d'URL lang, Application.CurrentCulture comme état propre à la session, l'événement Application.CultureChanged, et la réapplication des textes issus des ressources pour qu'aucun écran à moitié traduit ne survive au changement.

1. Le problème en production

Après le Module 3, GlobalDesk dispose d'un fichier de ressources neutre, d'un fichier allemand à côté, d'une classe utilitaire `Texts`, d'un éditeur client localisé dans le designer et même d'un libellé système Wisej qui lui est propre. Ce à quoi il n'a toujours pas de réponse, c'est la question que pose d'abord un vrai utilisateur : comment obtenir tout cela dans ma langue, et qu'arrive-t-il à l'écran quand je le demande ? Tout ce qui précède a été décidé pendant la construction de l'application ; ce module traite des deux moments où la décision se prend vraiment — au démarrage d'une session, et lorsqu'un utilisateur change d'avis en cours de route.

2. Concepts clés

Concept	Pourquoi c'est important
détection automatique du navigateur	Wisej.NET peut lire la langue annoncée par le navigateur et démarrer la session avec elle, de sorte que chaque utilisateur obtient une culture sensée sans qu'un réglage unique côté serveur décide pour tout le monde.
<code>culture</code> dans <code>Default.json</code>	Le paramètre d'application qui décide d'où vient la culture d'une nouvelle session ; sa valeur par défaut est <code>auto</code> , et un nom fixe comme <code>de-DE</code> fige au contraire toutes les sessions sur une seule locale.
<code>rightToLeft</code> dans <code>Default.json</code>	Son paramètre voisin, dont la valeur <code>auto</code> suit la culture que la session finit par adopter ; le Module 5 s'appuie dessus, mais vous le rencontrez ici, dans le même fichier.
<code>Application.CurrentCulture</code>	Obtient ou définit la culture de la <i>session Wisej.NET en cours</i> , ce qui est exactement la portée dont a besoin une application web multi-utilisateur : le choix d'un utilisateur ne doit jamais redessiner l'écran d'un autre.
paramètre d'URL <code>lang</code>	Une façon prise en charge de demander une langue à l'entrée, par exemple <code>?lang=de</code> , qui rend triviale la production de liens de test et d'URL d'entrée propres à une langue.
<code>Application.CultureChanged</code>	Déclenché quand la culture courante s'écarte de celle avec laquelle la session a démarré ; l'endroit unique où réappliquer les textes issus des ressources et remettre en forme ce qui est déjà à l'écran.
contrôles localisés dans le designer	Leurs valeurs localisées sont appliquées à la création des contrôles : un changement de culture ultérieur ne les réécrit donc pas — vous recréez le conteneur ou réappliquez les valeurs dans le code.

3. Le fil de la leçon

D'où vient la culture d'une session

Les navigateurs annoncent à chaque requête les langues que leur utilisateur préfère, et Wisej.NET peut s'appuyer sur cette annonce pour choisir la culture de la session qu'il s'apprête à démarrer. C'est un bien meilleur défaut qu'un réglage serveur, car le serveur n'a qu'une seule configuration régionale alors que ses utilisateurs, eux, en ont plusieurs : deux personnes qui ouvrent GlobalDesk à la même seconde depuis des navigateurs réglés sur l'anglais et sur l'allemand obtiennent chacune sa session, chacune avec sa culture, sur le même processus.

La détection n'est qu'un point de départ, pas une décision dont vous êtes prisonnier. La langue d'un utilisateur peut tout aussi bien venir d'une ligne de profil, d'une préférence de connexion ou du sélecteur que vous allez construire. La règle qui compte reste la même quelle que soit la source gagnante : résolvez-la une seule fois, au démarrage de la session, et chargez-la à cet instant dans `Application.CurrentCulture`. Le code qui lit des ressources ou met une valeur en forme plus tard ne devrait jamais avoir à redemander d'où vient la langue.

Configurer le paramètre culture dans Default.json

`Default.json` est l'endroit où l'application énonce son intention. Le paramètre `culture` vaut `auto` par défaut, c'est-à-dire le comportement de détection du navigateur que l'on vient de décrire, et le paramètre voisin `rightToLeft` fonctionne de la même façon — `auto` suit la culture. Dans le lab, vous laissez `culture` sur `auto` pour que GlobalDesk démarre depuis le navigateur, puis vous le prouvez en changeant la langue préférée de votre navigateur et en rechargeant.

Une valeur fixe n'est pas une erreur, c'est une intention différente. Figer le paramètre sur un nom de culture est le choix honnête pour un produit livré volontairement dans une seule locale, et c'est le moyen le plus rapide de reproduire un rapport portant sur une langue pendant que vous déboguez : toutes les sessions atterrissent dans cette culture, quoi qu'annonce le navigateur du testeur. Notez lequel des deux vous avez choisi et pourquoi, car le prochain développeur lira ce paramètre comme une déclaration sur le produit.

CurrentCulture est propre à la session, jamais un champ statique

Changer la langue à l'exécution tient en une affectation : définissez `Application.CurrentCulture` sur la culture choisie par l'utilisateur, construite avec `CultureInfo.GetCultureInfo` à partir d'un nom comme `"en-US"`, `"de-DE"` ou `"fr-CA"`. Comme la propriété a la portée de la session Wisej.NET, cette affectation change le monde d'un seul onglet de navigateur et rien d'autre. La ComboBox de langue du lab fait exactement cela dans son gestionnaire `SelectedIndexChanged`, en stockant les noms de culture comme valeurs des éléments pour que le gestionnaire n'ait jamais à analyser un libellé affiché.

Le raccourci tentant est un `static CultureInfo` placé à un endroit commode, « pour que tout le monde puisse le voir ». Les champs statiques vivent dans le processus serveur, et un serveur Wisej.NET héberge toutes les sessions dans ce processus unique : le champ contiendrait donc la valeur choisie par le dernier utilisateur en date. Un collègue dans un autre pays verrait son écran changer de langue parce que vous avez testé l'allemand. Si plusieurs classes ont besoin de la culture active, lisez-la à chaque fois depuis `Application.CurrentCulture` ; c'est déjà le stockage par session que vous étiez sur le point de réinventer.

Le paramètre lang et l'événement CultureChanged

Deux fonctionnalités de Wisej.NET encadrent le basculement. À l'entrée, le paramètre d'URL `lang` demande une langue avant que la moindre ligne de votre code ne s'exécute : `?lang=de` ouvre donc GlobalDesk en allemand sans toucher aux réglages du navigateur — idéal pour un lien de test, un rapport de bug ou une URL d'entrée propre à une langue que vous remettez à un client. En cours de route, `Application.CultureChanged` est déclenché quand la culture courante s'écarte de la culture initiale de la session, ce qui constitue votre point d'accroche pour tout ce qui est déjà à l'écran.

Abonnez-vous une seule fois, à la construction de la page, et faites le travail dans un seul gestionnaire pour que rien ne dépende du contrôle qui a déclenché le changement :

```

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
}

```

Redessiner ce que le basculement a laissé derrière lui

Un changement de culture ne remonte pas le temps jusqu'aux libellés que vous avez déjà affectés.

`ApplyTextResources` parcourt les contrôles dont le texte venait de la couche de ressources et redemande à `Texts.Get` chaque clé, de sorte que le titre, le bouton Customers et chaque message adoptent leurs nouvelles valeurs. `UpdateCulturePreview` en est le jumeau côté mise en forme : les données stockées n'ont pas changé du tout, il reprend donc la même date, la même quantité et le même montant décimal et les affiche avec la culture désormais active — c'est bien pourquoi vous gardez la valeur et jamais la chaîne mise en forme.

L'éditeur client localisé dans le designer est la partie qui piège les gens. Ses valeurs de propriétés localisées sont appliquées à la création de ses contrôles : il garde donc les libellés avec lesquels il est né, quel que soit le nombre de changements de culture qui suivent. Le remède est structurel, pas astucieux : libérez l'éditeur et ajoutez une nouvelle instance, fermez-le et rouvrez-le, ou naviguez vers une page nouvellement créée. Quel que soit votre choix, faites-le dans le même gestionnaire `CultureChanged`, puis regardez l'écran entier — un basculement qui laisse la moitié de la fenêtre dans la langue précédente est pire que pas de basculement du tout, car l'utilisateur se méfie désormais des deux moitiés.



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 détection automatique de la langue du navigateur par Wisej.NET.

- ✓ Configurer culture="auto" ou une culture fixe dans Default.json.
- ✓ Changer la culture de la session avec Application.CurrentCulture.
- ✓ Utiliser le paramètre d'URL lang et gérer Application.CultureChanged.
- ✓ Réappliquer l'interface issue des ressources après un changement de langue à l'exécution.

Guide du lab / de l'examen

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



Lab pratique

Donnez à GlobalDesk un changement de langue qui va jusqu'au bout. Mettez culture sur auto dans Default.json pour que la session démarre sur la langue du navigateur, ajoutez une ComboBox de langue proposant l'anglais, l'allemand et une culture langue-région, et définissez Application.CurrentCulture à partir de CultureInfo.GetCultureInfo dans son gestionnaire SelectedIndexChanged, sans jamais stocker le choix dans un champ statique. Gérez Application.CultureChanged pour appeler une méthode ApplyTextResources qui rafraîchit tous les libellés issus des ressources et une méthode UpdateCulturePreview qui remet en forme les exemples de date, de nombre et de montant, et recréez dans le même gestionnaire l'éditeur client localisé dans le designer. Vérifiez la même page via le paramètre d'URL lang, puis rédigez une note de lab décrivant ce qu'un champ de culture statique ferait subir à un deuxième utilisateur connecté au même moment.

Point de départ

Ouvrez GlobalDesk tel que vous l'avez laissé après le Module 3 : le tableau de bord lit chaque libellé via `Texts.Get`, `Resources/Strings.resx` et `Strings.de.resx` portent les mêmes clés, le panneau d'aperçu de la culture met en forme une date, un nombre et un montant, et l'éditeur client possède une variante allemande dans le designer. Rien à l'écran ne permet encore à un utilisateur de changer de langue, et `Application.CultureChanged` n'a aucun abonné.

Approche suggérée

- ✓ Ouvrez `Default.json`, vérifiez que `culture` vaut `auto`, et prouvez-le : passez la langue préférée de votre navigateur à l'allemand, rechargez GlobalDesk et notez quels libellés arrivent traduits. Écrivez en une ligne pourquoi vous figeriez plutôt le paramètre sur un nom de culture fixe.
- ✓ Ajoutez au tableau de bord une ComboBox `languageCombo` avec trois éléments dont les valeurs sont des noms de culture — `en-US`, `de-DE` et une culture langue-région comme `fr-CA` — et dont les libellés sont eux-mêmes des clés de ressources, pas des mots codés en dur.
- ✓ Dans `languageCombo_SelectedIndexChanged`, affectez `Application.CurrentCulture = CultureInfo.GetCultureInfo(name)` et rien d'autre. Résistez à toute tentation de mettre le choix en cache dans un champ `static` ; la session le stocke déjà.
- ✓ Abonnez-vous à `Application.CultureChanged` dans le constructeur de la page et écrivez les deux méthodes qu'il appelle : `ApplyTextResources` relit chaque libellé issu des ressources via `Texts.Get`, et `UpdateCulturePreview` remet en forme la même date, la même quantité et le même montant décimal stockés avec `Application.CurrentCulture`.
- ✓ Dans le même gestionnaire, libérez l'éditeur client localisé dans le designer et ajoutez une nouvelle instance à son conteneur hôte, puis changez de langue avec l'éditeur ouvert et vérifiez chaque libellé à l'écran. Terminez en ouvrant `?lang=de` dans une deuxième session de navigateur et en ajoutant votre note de lab sur l'état de culture statique.

Livrables attendus

- ✓ Default.json avec culture sur auto, accompagné d'une note sur les cas où une culture fixe est le bon choix
- ✓ Gestionnaire de la ComboBox de langue qui définit Application.CurrentCulture à partir de CultureInfo.GetCultureInfo
- ✓ Gestionnaire Application.CultureChanged qui appelle ApplyTextResources et UpdateCulturePreview
- ✓ Éditeur localisé dans le designer recréé après le changement pour qu'aucun libellé ne reste dans l'ancienne langue
- ✓ Vérification via le paramètre d'URL lang et note expliquant pourquoi la culture ne doit pas vivre dans un champ statique

Questions de révision

- ✓ Votre méthode `ApplyTextResources` rafraîchit parfaitement le tableau de bord, et pourtant l'éditeur client garde ses libellés anglais après un passage à l'allemand. Expliquez ce qui distingue ces libellés, et nommez deux stratégies qui corrigent le problème.
- ✓ Un collègue propose de mettre le `CultureInfo` choisi en cache dans un champ statique « pour éviter de le rechercher deux fois ». Décrivez ce que verrait réellement un deuxième utilisateur, connecté depuis un autre pays au même moment.
- ✓ L'aperçu de la culture affiche un montant. Après un passage de `en-US` à `de-DE`, qu'est-ce qui a changé en mémoire et qu'est-ce qui n'a changé qu'à l'écran — et laquelle de ces deux choses écririez-vous un jour en base de données ?



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.