



Dotnet France  
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

# Globalisation et Localisation d'applications Windows Mobile

# Sommaire

---

1	Introduction.....	3
2	Créer un programme pouvant utiliser plusieurs localisations .....	3
2.1	Globalisation d'un programme .....	3
2.2	Localisation.....	4
2.2.1	Notion de Resource Manager.....	4
3	Détection et modification de la culture active.....	5
3.1	Détection.....	5
3.2	Modification .....	6
4	Formats date, heure, nombres.....	7
4.1	date et heure.....	7
4.2	Nombres .....	8

## 1 Introduction

Tout d'abord, nous verrons dans ce chapitre ce en quoi consistent la globalisation et la localisation dans le cas de développement pour Windows Mobile.

Il est important de comprendre pour commencer ce qu'est une "culture" : il s'agit d'un ensemble d'éléments liés à la langue et aux personnes. Les cultures sont définies par la langue et le lieu sous la forme suivante : "fr-FR"(le Français en France), "fr-CA"(le Français au Canada), "en-CA"(l'Anglais au Canada), "en-US"(l'Anglais aux Etats-Unis), etc. A cela s'ajoutent les différences de vocabulaire telles que "voiture"(fr-FR) "car"(fr-CA) ou plus simplement "fichier"(fr) et "file"(en). **Nous pouvons remarquer que l'on utilise ici que fr ou en, cela est dû au fait que ces éléments-là "englobent" toute la langue : définir "fichier" pour fr le rend accessible aussi à fr-FR et fr-CA.** Cela évite des doublons et permet de gagner du temps lors de la localisation d'une application. Remarque : les noms de culture respectent la norme RFC 1766.

La Globalisation d'une application consiste à rendre(ou créer dès le début) une application neutre de tout langage et culture afin de la rendre traduisible sans devoir réécrire tout le code. En effet dans d'autres pays comme les Etats-Unis le séparateur décimal est le point et non la virgule comme chez nous, ce qui aura pour conséquence selon les paramètres du terminal Windows mobile de provoquer une erreur qui n'aura pas pu être détectée lors de la compilation.

La localisation d'une application consiste à rendre une application accessible à d'autres "cultures", c'est la suite logique de la globalisation. Cette fois on crée les changements spécifiques à chaque culture.

## 2 Créer un programme pouvant utiliser plusieurs localisations

### 2.1 Globalisation d'un programme

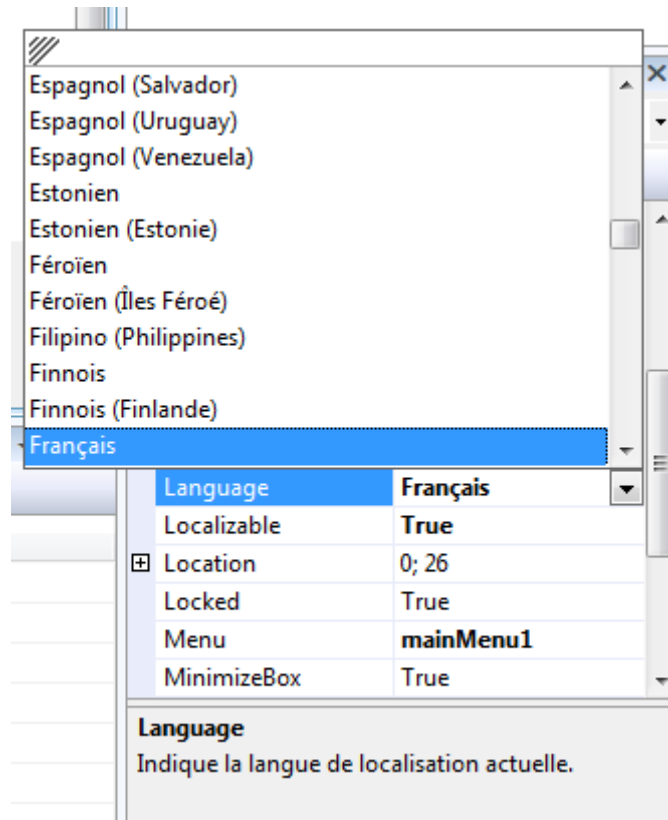
Nous allons suivre étape par étape la méthode à employer afin de rendre un projet ouvert à plusieurs localisations :

D'abord, nous utilisons une méthode un peu différente de celle de d'habitude.

Ensuite, le programme a jusqu'ici été créé en Français, je choisis donc Français dans la liste des langues des propriétés de chacune des Form que j'ai créé : la valeur « Localisable » de chaque Form passe automatiquement à « True »

Pour chaque nouvelle localisation un fichier resx est créé avec un nom tel que : « Form1.fr.resx ». Ces fichiers sont en fait des tableaux mettant en relation des variables avec du texte.

Ainsi, dans Form1.fr.resx, MessageAccueil correspondra à « Bienvenue ! » et dans Form1.en.resx MessageAccueil correspondra à « Hello ! ».

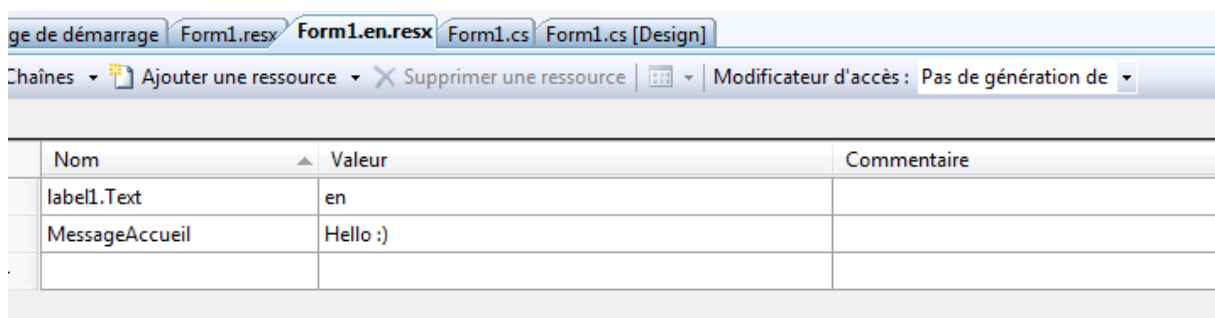


## 2.2 Localisation

Une fois la langue choisie il faut modifier tous les éléments de l'interface de sorte à ce qu'ils correspondent à la langue choisie. Soit en éditant directement au niveau du « design » des form soit en utilisant le Resource Manager. En pratique c'est en fait les deux outils qui seront utilisés (selon les cas).

### 2.2.1 Notion de Resource Manager

Le ResourceManager est ce qui fait la correspondance entre ce que nous devons afficher et ce qu'il y a dans les fichiers de langue (les resx).



Nom	Valeur	Commentaire
label1.Text	en	
MessageAccueil	Hello :)	

Comme nous pouvons le voir ici, nous sommes dans le resx « en », il y a à l'intérieur `label1.text` qui a été créé automatiquement et défini à « en » suite à l'édition via le designer, puis il y a `MessageAccueil` défini à « Hello :) » cette fois directement de façon manuelle. Si `label1.Text` n'avait pas été défini, il aurait pris comme valeur celle de `label1.Text` issue de `Form1.resx` (idem pour `MessageAccueil`).

Voici comment déclarer le ResourceManager :

```
//C#
private System.Resources.ResourceManager RM = null;

...

//      LeResourceManager      prend      en      paramètre      :
nom_du_namespace.nom_de_la_ressource_principale
RM = new System.Resources.ResourceManager("Localisation.Form1",
System.Reflection.Assembly.GetExecutingAssembly());
```

Les CultureInfo permettent de définir les cultures présentes au sein de l'application

Voici comment déclarer les CultureInfo :

```
//C#
using System.Globalization;
...

private CultureInfo EnglishCulture = new CultureInfo("en-US");
private CultureInfo FrenchCulture = new CultureInfo("fr-FR");
```

Voyons comment utiliser de façon effective les .resx :

```
//C#
//      LeResourceManager      prend      en      paramètre      :
nom_du_namespace.nom_de_la_ressource_principale
RM = new System.Resources.ResourceManager("Localisation.Form1",
typeof(Form1).Assembly);

// Affecte la valeur de la ressource MessageAccueil à la propriété
Text de la fenêtre principale
this.Text = RM.GetString("MessageAccueil");
```

## 3 Détection et modification de la culture active

### 3.1 Détection

Dans certains cas il peut être utile de détecter la culture active lors de l'exécution de l'application, que ce soit à des fins de débogage, d'affichage, ou de modification de morceaux de code selon la culture (à éviter toutefois).

Le morceau de code suivant affiche la culture active :

```
//C#
using System.Globalization;
//mise en situation : afficher dans label1 l'information désirée

//sous la forme "fr-FR"
this.label1.Text = CultureInfo.CurrentCulture.Name;
//sous la forme "français (France)" (dans la langue de la culture)
this.label1.Text = CultureInfo.CurrentCulture.NativeName;
//sous la forme "French (France)" (en anglais)
this.label1.Text = CultureInfo.CurrentCulture.EnglishName;
```

Exécution de code selon la culture détectée :

```
//C#
//ici petite "astuce" utilisée pour l'usage du séparateur décimal.
if (System.Globalization.CultureInfo.CurrentCulture.Name == "fr-FR")
{
    this.button1.Text = ",";
}
else
{
    this.button1.Text = ".";
}
```

### 3.2 Modification

Dans certains cas, il se peut qu'on veuille créer une instance de la classe `CultureInfo` pour utiliser une culture différente de celle sélectionnée dans les paramètres régionaux. Par exemple, pour autoriser l'utilisateur de notre application à choisir une culture spécifique sans porter attention aux paramètres régionaux.

Voici les 4 surcharges du constructeur de la classe `CultureInfo` :

```
//C#
public CultureInfo(int culture);
public CultureInfo(string name);
public CultureInfo(int culture, bool useUserOverride);
public CultureInfo (string name, bool useUserOverride);
```

Nous allons créer un projet afin de permettre à l'utilisateur de changer à la volée la culture qu'il utilise au sein de l'application.

Dans cet exemple, une `ListBox` est créée à laquelle nous ajoutons 5 cultures que l'utilisateur peut choisir :

```
cultureList.Items.Add("Default");
cultureList.Items.Add("en-US");
cultureList.Items.Add("en-GB");
cultureList.Items.Add("fr-FR");
cultureList.Items.Add("sv-SE");
cultureList.Items.Add("de-DE");
cultureList.SelectedIndex = 0;
setCulture();
```

A noter qu'il y a une fonction appelée `setCulture` qui est appelée pour définir la culture actuellement sélectionnée. Le choix de l'utilisateur sera stocké dans une variable appelée `selectedCulture`, que nous devons déclarer comme une classe en utilisant l'écriture suivante :

```
private CultureInfo selectedCulture;
```

La fonction `setCulture` contient le code nécessaire pour changer la culture active lorsqu'il choisit une culture alternative. Celle-ci est détaillée ci-dessous :

```
//C#
private void setCulture ()
{
    if (cultureList.SelectedIndex > 0)
        selectedCulture = new CultureInfo (cultureList.Text);
    else
    {
        selectedCulture = CultureInfo.CurrentCulture;
    }
    currentCultureLabel.Text = selectedCulture.EnglishName;
}
```

Enfin, nous appelons la fonction `setCulture` à chaque fois que l'utilisateur choisit une langue dans la `comboBox`. Il faut double-cliquer sur la `comboBox` pour créer la procédure `cultureList_SelectedIndexChanged` et lui ajouter le code suivant :

```
setCulture ();
```

## 4 Formats date, heure, nombres

Nous allons voir dans cette partie comment afficher la date, l'heure ou des nombres aux formats définis par les cultures de façon automatique.

### 4.1 Date et heure

La portion de code qui suit va afficher la date en version "longue" : Vendredi 10 juillet 2009 en fr-FR et Friday, July 10, 2009 en en-US

```
//C#
using System.Globalization;
.
.
this.label1.Text =
DateTime.Now.ToString (CultureInfo.CurrentCulture.DateTimeFormat.LongDateP
attern);
```

Et ceci affiche le mois et le jour : 10 juillet en fr-FR et July 10 en en-US

```
//C#
using System.Globalization;
.
.
this.label1.Text =
DateTime.Now.ToString (CultureInfo.CurrentCulture.DateTimeFormat.MonthDayP
attern);
```

Ceci affiche l'heure en version "longue" : 14:15:09 en fr-FR et 02:15:09 PM en en-US

```
//C#
using System.Globalization;
.
.
this.labell1.Text =
DateTime.Now.ToString(CultureInfo.CurrentCulture.DateTimeFormat.LongTimeP
attern);
```

Enfin, voici ce qui affiche le mois et le jour : ven., 10 juil. 2009 11:26:20 GMT en fr-FR et Fri., 10 jul. 2009 11:26:20 GMT en en-US

```
//C#
using System.Globalization;
.
.
this.labell1.Text =
DateTime.Now.ToString(CultureInfo.CurrentCulture.DateTimeFormat.RFC1123Pa
ttern);
```

## 4.2 Nombres

```
//C#
double myVal = 10321.5;
string myNumericVal = myVal.ToString("n");
MessageBox.Show(myNumericVal);
```

Cela donnera avec une culture Française : 10 321,50.

Et avec une culture anglaise : 10,321.50.

Nous pouvons également « forcer » une culture de façon ponctuelle :

```
//C#
CultureInfo uk = new CultureInfo("en-GB");
double myVal = 10321.5;
string myNumericVal = myVal.ToString("n", uk);
MessageBox.Show(myNumericVal);
```

Le même principe est applicable pour les devises :

```
//C#
CultureInfo fr = new CultureInfo("fr-FR", false);
double myVal = 10321.5;
string myNumericVal = myVal.ToString("c", fr);
MessageBox.Show(myNumericVal);
```

## 5 Conclusion

Comme nous avons pu le voir il est possible de façon relativement simple de rendre une application multiculturelle. En effet, Visual studio met à notre disposition tous les outils nécessaires.

Ce qui est important de comprendre est la séparation du processus en deux étapes : Globalisation puis Localisation. Après cela il devient aisé de créer une culture différente de celle de base et ainsi rendre l'application accessible au plus grand nombre.