Meertaligheid voor WPF

EEN GELOKALISEERDE APPLICATIE BOUWEN MET WINDOWS PRESENTATION FOUNDATION

Tijdens het ontwikkelen van Windows Forms en ASP.NET-applicaties is het redelijk eenvoudig meertaligheid in te bouwen. Maar hetzelfde geldt voor toepassingen die gebruikmaken van Windows Presentation Foundation (WPF) als gebruikersinterface. Dit artikel laat zien hoe je zowel de eigenschappen van je statische vensterobjecten als de gebruikersmeldingen meertalig maakt. Ook het omgaan met cultuurafhankelijke afbeeldingen komt aan de orde.

Een van de taken die ik in diverse projecten heb uitgevoerd, was het nadenken over en opzetten van mogelijkheden voor meertaligheid binnen applicaties. Dit ging bij Visual Studio 6.0 door met de VB 6 Resource Editor-add-in een .res-bestand te maken en uit te lezen met functies als LoadResPicture en LoadResString. Met Visual Studio .NET kunnen we voor zowel windows- als web-applicaties .resx-bestanden maken en deze met methoden uit de ResourceManager-class uitlezen.
Om het de windows forms-ontwikkelaar gemakkelijk te maken, bestaat er al sinds Visual Studio 2002 de mogelijkheid per venster een gelokaliseerd .resx-bestand per taal te maken door eerst de venstereigenschap Localizable op True te zetten. Vervolgens voorzie je via de eigenschap Language per taal de diverse vensterobjecten in de forms designer van een juiste waarde.
Voor web-applicaties bestaat er sinds Visual Studio 2005 de mogelijkheid via het menu Tools > Generate Local Resources per webpagina een .resx-bestand in de App_LocalResources-folder te plaatsen. Hierin worden naam/waarde-paren van alle text- en tooltip-eigenschappen van de vensterobjecten gezet. In de betreffende .aspx-pagina wordt automatisch code toegevoegd om de diverse eigenschappen weer te voorzien van de juiste waarden. De ontwikkelaar hoeft dan alleen dit .resx-bestand te kopiëren, van de juiste cultuurpostfixes te voorzien en de waarden te vertalen.

Wat te doen bij WPF

Voor WPF-applicaties is zo'n luxe ondersteuning (nog) niet in Visual Studio aanwezig. Dit artikel toont stap voor stap aan hoe je het project zo op kunt zetten dat er daarna steeds met een druk op de (F5) knop het nodige gebeurt om per specifieke cultuur één zogenaamde satelliet-assembly (dll) te maken. Zo'n assembly bevat naast gelokaliseerde eigenschappen van vensterobjecten ook vertaalde gebruikersmeldingen en cultuurafhankelijke afbeeldingen.

Projectopzet

We beginnen in Visual Studio 2008 met het maken van een nieuw WPF Application-project, genaamd WPFMultiCulti. Met wat eenvoudige XAML-code maken we een basisvenster om ergens in te kunnen loggen. Codevoorbeeld 1 toont de opmaakcode van dit window1.xaml-bestand.

Om na compilatie een taalneutrale hoofd-assembly (.exe) en een aparte satelliet-assembly (resources.dll) te krijgen, met daarin alle lokaliseerbare elementen, voegen we het element <UICulture> aan het .csproj-bestand toe. Dit gaat als volgt:

1. Selecteer in Solution Explorer de projectnaam.
2. Kies in het context-menu (rechtermuis) voor Unload Project.
3. Nog steeds staande op de projectnaam, kies je in het context- menu voor Edit WPFMultiCulti.csproj.
4. Voeg direct onder het eerste <PropertyGroup>-element het volgende toe:
   <UICulture>en-US</UICulture> (zie codevoorbeeld 2).
5. Weer op de projectnaam in Solution Explorer staand, kies je Reload Project in het context-menu om het project weer in te laden.

Toevoegen x:Uid-attributen

Aan elk te vertalen element, dient het x:Uid-attribuut toegevoegd te worden. Dit gaat automatisch door in Visual Studio Command Prompt de regel uit codevoorbeeld 3 uit te voeren. Alle elementen die het x:Uid-attribuut bevatten zijn vanaf nu kandidaat om gelokaliseerd te worden. Om te controleren of er geen missende of dubbele x:Uid's zijn, kun je msbuild.exe daarna nog met de switch/target:checkuid uitvoeren.

```
<Window x:Class="WPFMultiCulti.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    SizeToContent="WidthAndHeight"
    FontSize="20"
    WindowStyle="ToolWindow">
    <StackPanel Margin="10">

      <Label   Name="userNameLabel">User Name</Label>
      <TextBox Name="userNameTextBox" />

      <Label   Name="passwordLabel">Password</Label>
      <TextBox Name="passwordTextBox" />

      <Button  Name="signInButton" IsDefault="True"
                   Margin="0, 5">Sign In</Button>

    </StackPanel>
</Window>
```

CODEVOORBEELD 1. XAML CODE VOOR EEN EENVOUDIG INLOGVENSTER

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="3.5"
DefaultTargets="Build" xmlns="...">
    <PropertyGroup>
      <UICulture>en-US</UICulture>
      <!-- rest van het .csproj
      bestand -->
```

CODEVOORBEELD 2. ELEMENT DAT NA COMPILEREN VOOR EEN APARTE RESOURCES.DLL ZORGT

```
msbuild.exe /target:updateuid WPFMulti-
Culti.csproj
```

CODEVOORBEELD 3. VOEG AUTOMATISCH X:UID ATTRIBUTEN TOE AAN VERTAALBARE ELEMENTEN

Satelliet-assembly aanmaken

Na het compileren van het project is er in de folder bin\Debug een folder en-US aangemaakt met daarin het bestand WPFMultiCulti.resources.dll. Deze satelliet-assembly bevat voor elk venster in het project de gecompileerde XAML (BAML, Binary Application Markup Language). In ons voorbeeld is window1.baml de gecompileerde versie van ons volledige window1.xaml-inlogvenster.
Let op: De onvolprezen tool .NET Reflector heeft een aparte add-in genaamd BamlViewer, waarmee je BAML kunt decompileren en dus de onderliggende XAML zichtbaar maakt.

Gebruik LocBaml-tool

Om uit de gemaakte WPFMulticulti.resources.dll een leesbaar (en dus vertaalbaar) kommagescheiden bestand te halen, is de LocBaml-tool ontworpen (zie om deze te downloaden onderaan dit artikel de referentie 'LocBaml Tool'). Deze tool gebruiken we straks ook om van vertaalde kommagescheiden bestanden weer cultuurafhankelijke satelliet-assembly-bestanden te maken.
Let op: om redenen die alleen bij Microsoft bekend zijn, moeten de hoofd-assembly (.exe) en de LocBaml.exe in dezelfde folder staan om alles goed te laten werken.
We plaatsen de LocBaml.exe in de projectfolder (dus in dezelfde folder als de .csproj-, .xaml- en .cs-bestanden). Vervolgens kopiëren we de hoofd-assembly (WPFMulitCulti.exe) uit de bin\Debug-folder (tijdelijk) ook naar de projectfolder en voeren in de Visual Studio Command Prompt de code van codevoorbeeld 4 uit.

De LocBaml-tool heeft nu in de projectfolder het bestand WPFMultiCulti_en-US.csv gecreëerd. Dit bestand bevat per XAML-venster alle door de ontwikkelaar gezette eigenschappen met de bijbehorende waarden. Gedetailleerde informatie over de andere velden in dit .csv-bestand vind je in de referentie 'How to Localize an Application'.

```
LocBaml.exe /parse .\bin\Debug\en-US\WPF-
MultiCulti.resources.dll
/out:.\WPFMultiCulti_en-US.csv
```

CODEVOORBEELD 4. GENEREER EEN .CSV BESTAND MET ALLE LOKALISEERBARE EIGENSCHAPPEN

Van dit .csv-bestand kunnen we voor iedere cultuur die we willen ondersteunen een kopie maken. In ons voorbeeld maken we WPFMultiCulti_nl-NL.csv en WPFMultiCulti_de-DE.csv. Vervolgens veranderen we met een teksteditor (of in Visual Studio zelf) per relevante regel de tekst achter de laatste komma in de gewenste vertaling of setting. Hierna staan de gelokaliseerde bronbestanden klaar voor de volgende stap. Voor de in ons voorbeeld gebruikte vertalingen, zie tabel 1.

Gebruikersmeldingen vertalen

In een applicatie zijn niet alleen vertalingen nodig van statische eigenschappen van vensterobjecten, ook gebruikersmeldingen moeten we vertalen. We maken echter geen gebruik van de al in de Properties-folder aanwezige Resources.resx en onderliggende Resources.Designer.cs-bestanden. Dit omdat we anders in de problemen komen met de zogenaamde NeutralResourcesLanguage-instelling die we straks nog in het AssemblyInfo.cs bestand gaan zetten.
In de Solution Explorer van Visual Studio hernoemen we het bestand Resources.resx naar Resources.en-US.resx en openen het. Nu kunnen we via de Managed Resources Editor gebruikersmeldingen toevoegen en later ook afbeeldingen. Zorg wel dat de DropDown Box achter Access Modifier steeds op No code generation staat (zie afbeelding 1).
Als we bestand Resources.en-US.resx in de Solution Explorer van Visual Studio twee keer kopiëren en hernoemen naar de gewenste culturen, krijgen we er onder folder Properties twee bestanden bij, genaamd Resources.nl-NL.resx en Resources.de-DE.resx. We gebruiken de gegevens van tabel 2 om de drie .resx-bestanden van inhoud te voorzien.

Samenvoegen resources

Op dit moment hebben we per cultuur een .csv-bestand met vertaalde eigenschappen van vensterobjecten en een .resx-bestand met vertaalde gebruikersmeldingen. Deze gaan we samenvoegen tot één enkel resources.dll-bestand per taal en plaatsen deze in de

AFBEELDING 1. DE MANAGED RESOURCES EDITOR VAN VISUAL STUDIO IN ACTIE

daartoe bestemde folder. Dit hoeft alleen voor de Nederlandse en Duitse resources te gebeuren, samenvoegen van de Amerikaanse resources gebeurt automatisch door Visual Studio. Om dit proces te automatiseren maken we gebruik van de Post-build event command line-optie die zich onder tabblad Build Events van de WPFMultiCulti Properties bevindt (zie codevoorbeeld 5).

De eerste twee lange regels code maken van de eerder vertaalde .csv-bestanden met behulp van de LocBaml-tool .resources-bestanden die gelokaliseerde BAML bevatten. Als basis dient daarbij de oorspronkelijke BAML in het bestand WPFMultiCulti.g.en-US.resources. Deze actie levert de bestanden WPFMultiCulti.g.nl-NL.resources en WPFMultiCulti.g.de-DE.resources op in de folder obj\Debug (zie de "LocBaml"-pijl in afbeelding 2).

De twee laatste langere regels command line code maken gebruik van de Assembly Linker (AL.exe)-tool om de zojuist gegenereerde .resources-bestanden (gelokaliseerde BAML) samen te voegen met de door Visual Studio uit de .resx gemaakte .resources-bestanden (gebruikersmelding en later ook afbeeldingen). De uitkomst is per cultuur één resources.dll satelliet-assembly in de bijbehorende cultuurfolder, zie "Assembly Linker (AL.exe)" in afbeelding 2. Na een succesvolle build-actie staan in de bin\Debug-folder naast de en-US-folder nu ook de folders nl-NL en de-DE. In al deze folders is het samengevoegde cultuurafhankelijke bestand WPFMultiCulti.resources.dll aanwezig.

Nog een paar regels code

Om nu onze WPF-applicatie ook daadwerkelijk met de zojuist gegenereerde satelliet-assembly-bestanden te laten werken, is er nog wat code nodig die de gewenste cultuur activeert. Dit kan op twee manieren:

1. Als we via een setting de cultuur willen bepalen, gaan we eerst via de tab Settings in de WPFMultiCulti Properties een string genaamd culture aanmaken met als

en-US	nl-NL	de-DE
User name	Gebruikersnaam	Benutzername
Password	Wachtwoord	Kennwort
Sign in	Inloggen	Einloggen

TABEL 1. VERTALINGEN VOOR DE VISUELE OBJECTEN OP HET INLOGVENSTER

Resource Bestand	Inhoud veld Name	Inhoud veld Value
Resources.en-US.resx	msg	Is anybody home?
Resources.nl-NL.resx	msg	Is er iemand thuis?
Resources.de-DE.resx	msg	Ist jemand zu Hause?

TABEL 2. VERTALINGEN VOOR DE GEBRUIKERSMELDING

```
:: Genereer BAML bevattende .resources satelliet bestanden voor
:: iedere cultuur gebaseerd op het Amerikaanse resource bestand
"$(ProjectDir)LocBaml.exe"
/generate "$(ProjectDir)obj\Debug\$(TargetName).g.en-US.resources"
/translation:"$(ProjectDir)$(TargetName)_nl-NL.csv"
/culture:nl-NL
/out:"$(ProjectDir)obj\Debug"

"$(ProjectDir)LocBaml.exe"
/generate "$(ProjectDir)obj\Debug\$(TargetName).g.en-US.resources"
/translation:"$(ProjectDir)$(TargetName)_de-DE.csv"
/culture:de-DE
/out:"$(ProjectDir)obj\Debug"

:: Voeg de beide resource bestanden per taal
:: samen tot de betreffende resources.dll
"C:\Program Files\Microsoft SDKs\Windows\v6.0A\Bin\al.exe"
/template:"$(TargetPath)"
/embed:"$(ProjectDir)obj\Debug\
$(TargetName).Properties.Resources.nl-NL.resources"
/embed:"$(ProjectDir)obj\Debug\$(TargetName).g.nl-NL.resources"
/culture:nl-NL
/out:"$(TargetDir)nl-NL\$(TargetName).resources.dll"

"C:\Program Files\Microsoft SDKs\Windows\v6.0A\Bin\al.exe"
/template:"$(TargetPath)"
/embed:"$(ProjectDir)obj\Debug\
$(TargetName).Properties.Resources.de-DE.resources"
/embed:"$(ProjectDir)obj\Debug\$(TargetName).g.de-DE.resources"
/culture:de-DE
/out:"$(TargetDir)de-DE\$(TargetName).resources.dll"
```

CODEVOORBEELD 5. POST-BUILD EVENT COMMAND LINES

AFBEELDING 2. HET PROCES VAN RESOURCES.DLL BESTANDEN MAKEN IN SCHEMAVORM

waarde nl-NL. Vervolgens moeten we in de constructor van het bestand App.xaml.cs de code uit codevoorbeeld 6 toevoegen om de gewenste UICulture te zetten.

2. Als we de taalinstellingen van Windows willen laten volgen, zetten we in de constructor uit codevoorbeeld 6 de CurrentUICulture gelijk aan de CurrentCulture. We hebben bij deze optie geen extra culture-setting nodig.

Voor ons voorbeeld kiezen we optie 1. Het omzetten van de cultuur gebeurt in bestand App.xaml.cs en kan niet in het inlogvenster zelf, omdat dan de cultuurspecifieke BAML-code al geladen is.

Om de gebruikersmelding te tonen zodra er op de knop wordt gedrukt, is nog de code uit codevoorbeeld 7 nodig. Dubbelklikken op de signInButton in het Design-venster van window1.xaml levert net als bij windows forms en webapplicaties een event-handler-methode op. Twee using statements voegen we nog toe: System.Resources en System.Reflection. Nu kunnen we bouwen en testen met de waarden nl-NL, en-US en de-DE.

Neutrale taal instellen

We stellen een neutrale cultuur vast. Dit om te zorgen dat werken met de applicatie mogelijk blijft na het kiezen van een bestaande UICulture waarvoor geen bijbehorend resources.dll-bestand aanwezig is. Hiervoor halen we in het bestand AssemblyInfo.cs (in de Properties-folder) de volgende regel van commentaar af: [assembly: NeutralResourcesLanguage("en-US", UltimateResourceFallbackLocation.Satellite)]. Dus wanneer we als gewenste UICulture bijvoorbeeld Russisch (ru-RU) kiezen, wordt teruggevallen op de Amerikaanse cultuur.

Deze instelling zorgt ervoor dat, indien nodig, binnen de WPFMultiCulti.resources.dll in folder en-US voor de gebruikersmeldingen (en straks de afbeeldingen) gezocht wordt naar WPFMultiCulti.Properties.Resources.en-US.resources. Zouden we gebruik hebben gemaakt van de Resources.resx en onderliggende Resources.Designer.cs-bestanden, dan had dat WPFMultiCulti.Properties.Resources.resources opgeleverd, dus zonder cultuuraanduiding. Hierdoor zou het fallback-mechanisme niet meer werken. Dat is de reden waarom we eerder kozen voor een Resources.en-US.resx-bestand, dus met specifieke cultuuraanduiding. Dit helaas met verlies van het bestand Resources.Designer.cs met daarin de gegenereerde onderliggende Resource-class.

Gelokaliseerde afbeeldingen

Als kers op de taart voegen we per cultuur een afbeelding van een vlag toe. In de Visual Studio Solution Explorer openen we hiervoor telkens een van onze .resx-bestanden en voegen de vlag van het betreffende land toe. Hiervoor kiezen we de optie Add Existing File (onder de Drop-Down Button Add Resource) en voegen we respectievelijk us.tif, nl.png en de.gif in. Geef na het invoegen elke afbeelding wel steeds dezelfde naam, in ons geval de naam flag.

Custom Markup Extension

We willen in de XAML van ons inlogvenster gebruikmaken van een WPF Image-vensterobject om de vlag te tonen. Het Source-attribuut van dit Image-object heeft formaat System.Windows.Media.Imaging.BitmapSource nodig, terwijl het Windows Forms resource-formaat voor afbeeldingen System.Drawing.Bitmap is. Formaatvertaling vindt niet automatisch plaats. We kiezen ervoor een zelfgemaakte markup-extension te gebruiken om deze formaatvertaling uit te voeren (codevoorbeeld 8). De class ResourcesExtension haalt, op basis van de property ImageName, de gewenste Bitmap uit het

```
using System;
using System.Windows;
using System.Globalization;
using System.Threading;

namespace WPFMultiCulti
{
    public partial class App : Application
    {
      public App()
      {
        Thread.CurrentThread.CurrentUI-
Culture = new CultureInfo(
          WPFMultiCulti.Properties.Set-
tings.Default.culture);
      }
    }
}
```

CODEVOORBEELD 6. DE GEWENSTE CULTUUR ACTIVEREN

Adv

```
private void signInButton_Click(object
sender, RoutedEventArgs e)
{
    ResourceManager rm = new ResourceMana-
ger(
      "WPFMultiCulti.Properties.Resour-
ces",
      Assembly.GetExecutingAssembly());

    MessageBox.Show(rm.GetString("msg"));
}
```

CODEVOORBEELD 7. DE GELOKALISEERDE GEBRUIKERSMELDING TONEN

juiste WPFMultiCulti.resources.dll-bestand. Deze bitmap wordt vervolgens vertaald naar het BitmapSource-type, dat door een WPF Image-vensterobject getoond kan worden. We passen de XAML van ons inlogvenster aan zoals in codevoorbeeld 9 om gebruik te maken van onze markup-extension. Het woord flag achter het ImageName-gedeelte van onze markup-extension zet de property van het ResourcesExtension-object, zodat de juiste bitmap wordt opgehaald, omgezet en teruggegeven. Let op: je hoeft het woord Extension van ResourcesExtension bij gebruik in XAML niet uit te schrijven. Net als bij het gebruik van attributen boven classes en members waarbij je het woord Attribute er niet achter hoeft te zetten. Afbeelding 3 toont het resultaat van de diverse vertaalde gebruikersinterfaces, inclusief gebruikersmeldingen en afbeeldingen.

Conclusie

We hebben laten zien hoe je met ondersteuning van Visual Studio voor WPF- toepassingen per taal een cultuurafhankelijk satelliet-assembly-bestand kunt maken. Op deze manier is het mogelijk met een druk op de (F5) knop op basis van de besproken .csv- en .resx-bestanden telkens opnieuw de benodigde resources.dll-bestanden te laten genereren. Deze dll-bestanden bevatten zowel vertaalde eigenschappen van vensterobjecten als gelokaliseerde gebruikersmeldingen en cultuurafhankelijke afbeeldingen. .net

```
using System;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Resources;

namespace WPFMultiCulti
{
    [MarkupExtensionReturnType(typeof(BitmapSource))]
    public class ResourcesExtension : MarkupExtension
    {
      public string ImageName { get; set; }

      public override object ProvideValue(IServiceProvider sp)
      {
        BitmapSource bs = null;

        ResourceManager rm = new ResourceManager(
          "WPFMultiCulti.Properties.Resources",
          System.Reflection.Assembly.GetExecutingAssembly());

        Image img = rm.GetObject(ImageName) as Image;

        if (img != null)
        {
          // Zet een System.Drawing.Bitmap uit de resources.dll
          // om in een System.Windows.Media.Imaging.BitmapSource
          // om als Source voor een WPF Image te kunnen dienen.
          MemoryStream stream = new MemoryStream();

          img.Save(stream, ImageFormat.Png);

          PngBitmapDecoder decoder = new PngBitmapDecoder(
            stream,
            BitmapCreateOptions.PreservePixelFormat,
            BitmapCacheOption.Default
          );

          bs = decoder.Frames[0];
        }

        return bs;
      }
    }
}
```

CODEVOORBEELD 8. MARKUP EXTENSION VOOR HET OMZETTEN VAN BITMAP RESOURCES

```
<Window ...
    ...
    xmlns:twice="clr-
namespace:WPFMultiCulti"
    ...>

    <StackPanel ...>

      <Image Source="{twice:Resources
ImageName=flag}" Width="150" />

      <!-- Rest van de XAML voor het in-
logvenster -->
```

CODEVOORBEELD 9. HET GEBRUIK VAN EEN CUSTOM MARKUP EXTENSION

AFBEELDING 3. HETZELFDE WPF-INLOGVENSTER IN DRIE CULTUURAFHANKELIJKE VARIANTEN.

Referenties

WPF Globalization and Localization Overview: http://msdn.microsoft.com/en-us/library/ms788718.aspx

LocBaml Tool: http://msdn.microsoft.com/en-us/library/ms771568.aspx

How to Localize an Application: http://msdn.microsoft.com/en-us/library/ms746621.aspx

How to globalize and localize a WPF project: http://www.codeproject.com/KB/WPF/GlobalizeLocalizeAWPFPro.aspx

Jeroen Hartsuiker (jhartsuiker@twice.nl) is docent software development bij Twice IT Training te Driebergen (www.twice.nl).

Nieuw: Nederlandstalige website met alle Nederlandse en Engelse Microsoft Press-boeken

Bestel al uw Microsoft Press-boeken via www.microsoft-press.nl

UW ICT-PROJECT GEGARANDEERD OP TIJD OPGELEVERD!

SOMMIGEN BELOVEN HET. WIJ GARANDEREN HET!

De Caesar Groep is een ICT-dienstverlener die oplossingen biedt met rendement. Wij nemen daarbij de doelstellingen van de klant als uitgangspunt. In de vorm van TimeValue-projecten realiseren wij gegarandeerd op tijd opgeleverde ICT-oplossingen met korte doorlooptijden en aantoonbare waarde voor onze klanten. Indien wij toch te laat opleveren, leggen wij onszelf een aanzienlijke boete op. We betalen onze klant dan tot 50% terug! De technologie die wij gebruiken is onder andere gebaseerd op Oracle, Java, Progress en Microsoft.

Wat is de waarde van elk van uw ICT-projecten? Hoe kunt u ervoor zorgen dat uw ICT-project op tijd wordt opgeleverd? En waarom durft Caesar die opleverdatum keihard te garanderen?
Graag beantwoorden wij deze vragen in onze workshop TimeValue.
Voor meer informatie en aanmelding kijk op **www.timevalue.nl**

GRAAG NODIGEN WIJ U UIT VOOR EEN INSPIRERENDE TIMEVALUE WORKSHOP

Microsoft GOLD CERTIFIED Partner

TimeValue

Caesar Groep - Zonnebaan 9 - 3542 EA Utrecht - tel. 030 - 240 42 00 - www.caesar.nl - info@caesar.nl

ICT OPTIMA FORMA

CAESAR GROEP