Webradio-applicatie met Windows Phone 7

Met kennis van WPF en Silverlight snel aan de slag

Michael Wolbert

Sinds de eerste publieke release van de Windows Phone 7 (WP7) SDK in februari 2010 is er veel veranderd en toegevoegd aan het framework: Panorama & Pivot controls, WP7 Bing Maps functionaliteit, *capability* permissies en WP7 project templates voor Visual Studio 2010 (VS2010). Kortom, te veel beweging op verschillende gebieden om in één artikel te kunnen beschrijven. We beperken ons tot het bouwen van een WebRadio-voorbeeldapplicatie.

Voor dit artikel zal ik aan de hand van een WebRadio voorbeeldapplicatie de onderwerpen belichten die van pas zullen komen bij het ontwikkelen van je eerste Silverlight WP7 applicatie: het configureren van permissies op WP7 functies, hoe gebruik te maken van de XAML design templates, databinding met Model-View-ViewModel pattern, navigatie van pagina naar pagina met het Navigation Framework, Geo Positioning System (GPS) module en locatie services en WP7 tasks.

Om de onderwerpen in de volgende secties te behandelen, heb ik een demo-applicatie gebouwd genaamd WebRadio. WebRadio bundelt bestaande digitale radiozenders die via het internet, 3G of Wifi, kunnen worden beluisterd. Voor deze applicatie zijn de volgende functionele vereisten gespecificeerd:

- Tonen van alle radiozenders. (naam, genres, logo);
- Tonen van alle radiozenders in het huidige land door middel van plaatsbepaling;
- Tonen van favoriete radiozenders;
- Markeren van een radiozender als favoriet;
- Tonen van details van een radiozender. (omschrijving, website)
- Luisteren naar een radiozender;
- Navigeren naar website van de radiozender.

De solution opzetten

De oktober-release van de SDK is een totaalpakket waar *VS2010 Express for WP7* en *Expression Blend 4 for WP7* in zijn gebundeld. Via de bekende webinstaller van Microsoft is de totale SDK gratis te downloaden net als de iPhone- en Android SDKs.
De eerste keuze die moet worden gemaakt is de Visual Studio template.

De beschikbare templates zijn:

- *Windows Phone Application*: een lege pagina met alle vrijheid om zelf controls toe te voegen;
- *Windows Phone Databound Application*: een pagina met een lijst welke bind aan een voorbeeld ViewModel in de template;
- *Windows Phone Panorama Application*: Panorama met drie secties die elk een databound lijst en een voorbeeld ViewModel bevat;
- *Windows Phone Pivot Application*: Pivot die een databound lijst en een voorbeeld ViewModel bevat.

De *WebRadio*-applicatie maakt gebruik van de *Panorama*-template waarbij de drie secties voor de drie overzichten van radiozenders kunnen worden gebruikt. De keuze voor de *Pivot*-template zou ook een logische redenering zijn, maar aangezien er, voor het mooie effect, ook transitie met de achtergrond in de applicatie komt, is de *Panorama*-template meer toepasselijk. De *Databound*-template zou niet direct in aanmerking komen, omdat er vanuit deze template nog een oplossing voor meerdere secties moet komen.

Permissies configureren op applicatie niveau

Nadat de projecttemplate is gekozen is VS2010 klaar voor de start. Onder de *Properties* node van het project bevindt zich het bestand *WMAppManifest.xml* waar de complete applicatieconfiguratie is gedefinieerd. Denk hierbij aan auteur, omschrijving, icons en andere relevante metadata voor de applicatie. Daarnaast staat in de *capabilities*-node (zie code snippet 1) een opsomming van WP7 functies die geconsumeerd worden door de applicatie. Dit security-model is ook beschikbaar in de iPhone - en Android SDK. Wanneer gebruikers een applicatie via de MarketPlace willen installeren, zien ze eerst welke functies de applicatie wil consumeren van de WP7: het aanzetten van de GPS module of het uitlezen van de gebruikersgegevens. Mocht een applicatie functies gebruiken die de gebruiker niet vertrouwd in context van de applicatie omschrijving kan de gebruiker de installatie afbreken.
Tijdens ontwikkeling kun je alle *capabilities* aan laten staan; standaard krijg je het bestand zo in je solution. Op voorhand voorkomen dat een applicatie met alle *capabilities* in de MarketPlace terecht komt is natuurlijk een pluspunt; zet functies uit wanneer ze niet nodig zijn.

```
<Capabilities>
  <!-- gebruik van GPS module -->
  <Capability Name="ID_CAP_LOCATION"/>
  <!-- gebruik van internetverbinding voor streamen en ophalen logo -->
  <Capability Name="ID_CAP_NETWORKING"/>
</Capabilities>
```

CODESNIPPET 1: CAPABILITIES VOOR WEBRADIO.

XAML design templates

Begin september zijn op www.codeplex.com de eerste door Microsoft ontwikkelde XAML design templates voor WP7 opgedoken. Deze solution bevat out-of-the-box templates voor designs die helpen tijdens het samenstellen van de verschillende pagina's en controls die de eindgebruiker te zien krijgt. Via de emulator kan deze solution worden geopend zodat de verschillende controls kunnen worden getest. Wanneer een control template interessant is, kan de desbetreffende control via Expression Blend gekopieerd en geplakt worden naar de Expression Blend instantie van de eigen solution. Vanuit Expression Blend kan de template in de eigen solution verder worden bewerkt tot het gewenste eindresultaat. Codesnippet 2 is de originele template die gekopieerd is naar codesnippet 3, waarin een aantal wijzigingen zijn gedaan. Het visuele resultaat (zie figuur 1) is een toevoeging van een *Image* en een *CheckBox* met hartjes aan de bestaande template.

```
<DataTemplate x:Key="DoubleLineLargeGridListTemplate">
    <Grid Height="173" Margin="12,0,12,12">
        <Rectangle Width="173" HorizontalAlignment="Left" Fill="{StaticResource PhoneInactiveBrush}"/>
        <StackPanel Margin="107,-4,0,0" VerticalAlignment="Top" >
            <TextBlock Text="{Binding Line1}" Padding="0,3,0,0" FontFamily="{StaticResource PhoneFontFamilyLight}" FontSize="{StaticResource PhoneFontSizeExtraLarge}"
                       Foreground="{StaticResource PhoneForegroundBrush}" TextWrapping="Wrap" LineStackingStrategy="BlockLineHeight" LineHeight="43" Margin="0,0,0,5" />
            <TextBlock Text="{Binding Line2}" Opacity="0.65" FontSize="{StaticResource PhoneFontSizeSmall}" Foreground="{StaticResource PhoneForegroundBrush}"
                       FontFamily="{StaticResource PhoneFontFamilyNormal}" HorizontalAlignment="Left" VerticalAlignment="Top"/>
        </StackPanel>
    </Grid>
</DataTemplate>
```

CODE SNIPPET 2: XAML DATATEMPLATE VAN ELK LISTBOXITEM UIT DE DESIGN TEMPLATE SOLUTION.

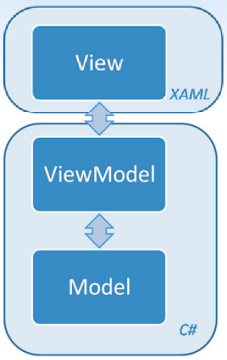
```
<DataTemplate x:Key="DoubleLineLargeGridListTemplate">
    <Grid Height="173" Margin="12,0,12,12">
        <Image Source="{Binding LogoUrl}"  Width="173" HorizontalAlignment="Left"/>
        <CheckBox IsChecked="{Binding IsFavourite, Mode=TwoWay}" Template="{StaticResource HeartCheckBoxControlTemplate}" Width="50" Height="50" HorizontalAlignment="Left"
                  Margin="8,8,0,0" VerticalAlignment="Top" d:LayoutOverrides="Width, Height"/>
        <StackPanel Margin="107,-4,0,0" VerticalAlignment="Top">
            <TextBlock Text="{Binding Name}" Padding="0,3,0,0" FontFamily="{StaticResource PhoneFontFamilyLight}" FontSize="{StaticResource PhoneFontSizeExtraLarge}"
                       Foreground="{StaticResource PhoneForegroundBrush}" TextWrapping="Wrap" LineStackingStrategy="BlockLineHeight" LineHeight="43" Margin="0,0,0,5" />
            <TextBlock Text="{Binding Tags, Converter={StaticResource TagConverter}}" Opacity="0.65" FontSize="{StaticResource PhoneFontSizeSmall}"
                       Foreground="{StaticResource PhoneForegroundBrush}" FontFamily="{StaticResource PhoneFontFamilyNormal}" HorizontalAlignment="Left" VerticalAlignment="Top"/>
        </StackPanel>
    </Grid>
</DataTemplate>
```

CODE SNIPPET 3: XAML DATATEMPLATEVAL ELK LISTBOXITEM UIT DE WEBRADIO.

FIGUUR 1: LINKS DE XAML TEMPLATE VOOR EEN LISTBOX, RECHTS HET RESULTAAT VAN DE OVERGENOMEN TEMPLATE.

Model-View-ViewModel

Met het Model-View-ViewModel pattern kunnen developers en designers samenwerken door middel van *seperation of concerns*. Kort door de bocht wil dat zeggen dat het model, dat verantwoordelijk is voor bedrijfslogica, niet direct communiceert met de *view* om de user interface te manipuleren. Andersom geldt dat ook voor de *view*, die niet direct het model kan manipuleren



(zie figuur 2). De *viewmodel* laag faciliteert de communicatie tussen beide. Als er bijvoorbeeld nieuwe data voor de *view* beschikbaar is vanuit het model, dan geeft het *viewmodel* een notificatie en de data. De *view* kent weinig logica in de code-behind behalve *EventHandlers* van controls, navigatie, *converters* en het afhandelen van animaties.

FIGUUR 2: COMMUNICATIE TUSSEN VIEW, VIEWMODEL EN MODEL.

Een belangrijk object bij MVVM implementaties is de collectie *ObservableCollection<T>*. Het *CollectionChanged* event op de *ObservableCollection<T>* notificeert de view wanneer er verandering optreedt in de collectie. Als er bijvoorbeeld een object uit de collectie wordt verwijderd, is dit direct terug te zien in het gebonden control. In *WebRadio* bevat de *ObservableCollection<WebRadioViewModel>* die direct kunnen worden gebonden aan de *ListBox ItemSource property* (zie codesnippet 4). De gebonden *viewmodels* worden met de *DataTemplate* (zie codesnippet 3) in de *ListBox* getoond.

```
<controls:PanoramaItem Header="Favourites">
    <Grid Margin="0,0,-12,0">
        <ListBox x:Name="FavouriteListBox" Margin="0,19,-12,0"
                 ItemTemplate="{StaticResource DoubleLineLargeGridListTemplate}"
                 ItemsSource="{Binding FavouriteItems}"
                 SelectionChanged="ListBox_SelectionChanged"/>
    </Grid>
</controls:PanoramaItem>
```

CODESNIPPET 4: BINDING VAN DE FAVOURITEITEMS OP DE ITEMSOURCEVAN EEN LISTBOX IN MAINPAGE.XAML.

```
// collecties voor databinding
public ObservableCollection<WebRadioViewModel> Items { get; private set; }
public ObservableCollection<WebRadioViewModel> FavouriteItems { get; private set; }
public ObservableCollection<WebRadioViewModel> LocationItems { get; private set; }

public MainViewModel()
{
    // zwengel de locatie wrapper aan
    _locationHelper = LocationHelper.Create();

    // maak instanties voor de bindings van dit viewmodel
    this.Items = new ObservableCollection<WebRadioViewModel>();
    this.FavouriteItems = new ObservableCollection<WebRadioViewModel>();
    this.LocationItems = new ObservableCollection<WebRadioViewModel>();
}

public bool IsDataLoaded { get; private set; }
public void LoadData()
{
    // loop door model data
    foreach(WebRadio.Model.WebRadio webRadio in
        WebRadio.Model.WebRadio.GetTestData())
    {
        // maak viewmodel instantie aan
        WebRadioViewModel wrvm = new WebRadioViewModel(webRadio);
        // voeg deze toe aan alle items - filter voor alle radiozenders
        this.Items.Add(wrvm);

        // als radiozender favoriet is gemarkeerd voeg deze aan de
        // favorieten collectie toe
        if (webRadio.IsFavourite)
            this.FavouriteItems.Add(wrvm);

        // als radiozender gelijk is aan het land waarin het apparaat zich bevindt
        // voeg deze toe aan de locatie collectie
        if (webRadio.Country == this.CurrentCountry)
            this.LocationItems.Add(wrvm);

        // als er iets in de view wordt gewijzigd moet er een event optreden
        wrvm.PropertyChanged += new PropertyChangedEventHandler(wrvm_PropertyChanged);
    }

    // data is geladen
    this.IsDataLoaded = true;
}
```

CODESNIPPET 5: CONSTRUCTOR EN METHODE LOADDATA() UIT HET MAINVIEWMODEL.

De *DataContext* van de *MainPage.xaml* (view) is het *MainViewModel* die participeert in deze binding zodat de *ListBox* de inhoud van de *ObservableCollection<WebRadioViewModel>* kan tonen. In codesnippet 5 staat een deel van het *MainViewModel* dat als startpunt van de applicatie dient. In de methode *LoadData()* worden de radiozenders uit het model opgehaald en doorlopen om deze

toe te kennen aan een *viewmodel*. Op basis van filters worden de andere collecties met *viewmodels* gevuld. Aan het eind van de foreach-loop wordt het *PropertyChanged* event van het *viewmodel* geregistreerd aan een *EventHandler* om een notificatie in het *viewmodel* te krijgen.

De *viewmodels* implementeren de interface *INotifyPropertyChanged* om verandering in de gebonden *view* door te kunnen geven aan het *model* of aan andere *viewmodels* die zich kunnen registreren op het *PropertyChanged* event uit de implementatie van de interface. In codesnippet 7 wordt de binding van *IsFavourite* op de *CheckBox* (het hartje) gelegd: *viewmodel* naar *view*. Als de gebruiker op het hartje drukt, zorgt de binding op de *CheckBox IsChecked* property ervoor dat de *IsFavourite* (zie codesnippet 6) property wordt gezet en het *PropertyChanged* event wordt afgevuurd. In het *MainViewModel* is de *EventHandler* (zie codesnippet 7) van het *PropertyChanged* event gedefinieerd die verantwoordelijk is voor het afvangen van de gewijzigde property en de actie: het toevoegen of verwijderen van het *viewmodel* uit de favorieten collectie.

```
public bool IsFavourite
{
    get
    {
        // geef de waarde terug aan de view: data binding
        return this.Item.IsFavourite;
    }
    set
    {
        // update het model alleen als de inkomende waarde
        // niet hetzelfde is als het model
        if (value != this.Item.IsFavourite)
        {
            this.Item.IsFavourite = value;
            // informeer de (andere) view(s) dat de waarde is veranderd
            NotifyPropertyChanged("IsFavourite");
        }
    }
}

// implementatie uit INotifyPropertyChanged
public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged(string propertyName)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (null != handler)
    {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

CODESNIPPET 6: SNIPPET UIT WEBRADIOVIEWMODEL.

```
private void wrvm_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    if (e.PropertyName == "IsFavourite")
    {
        WebRadioViewModel wrvm = sender as WebRadioViewModel;
        if (wrvm.IsFavourite)
            this.FavouriteItems.Add(wrvm);
        else
            this.FavouriteItems.Remove(wrvm);
    }
}
```

CODE SNIPPET 7: TOEVOEGEN EN VERWIJDEREN VAN HET VERANDERDE WEBRADIOVIEWMODEL AAN DE COLLECTIE FAVOURITEITEMS.

Het gebruik van het MVVM pattern lijkt in *WebRadio* overkill vanwege de vrij simpele bindingen die plaatsvinden van *model* naar *view* waardoor er meer code beheerd moet worden (*model* en *viewmodel*). Op de lange termijn faciliteert het wel de evolutie van de applicatie door bijvoorbeeld snelle nieuwe *views* te kunnen ontwerpen en deze weer gemakkelijk te binden aan de bestaande *viewmodels*.

NavigationServices

Silverlight 3 introduceerde het Navigation Framework. Inmiddels is het framework verder ontwikkeld in Silverlight 4 en het is ook beschikbaar in de WP7 SDK. Met dit framework kan navigatie van pagina naar pagina (*view* naar *view*) door middel van een aantal methodes van het *NavigationService* object eenvoudig worden opgezet. Het *NavigationService* object is gedefinieerd in de baseclass *Page* uit de namespace *System.Windows.Controls*. De pagina's *MainPage* en *WebRadioDetailsPage* zijn afgeleid van de *PhoneApplicationPage* die op zijn beurt weer afgeleid is van *Page*. Bij navigatie komen de *viewmodels* van pas om de uiteindelijk bindingen te doen met de view nadat er succesvol is genavigeerd.
In *WebRadio* kan de gebruiker op een *ListBoxItem* drukken om meer details op te vragen van de radiozender. In het interactieontwerp (zie figuur 3) is te zien hoe dit event plaatsvindt en hoe de navigatie van de *MainPage* naar de *WebRadioDetailsPage* verloopt. Het gedrag van de hardware BackKey navigeert terug naar de *MainPage*.

FIGURE 3: GEBRUIKER DRUKT OP LISTBOXITEM.

Omdat de *MainPage* verantwoordelijk is voor de drie *ListBoxen*, bevat de code-behind van deze *Page* de *EventHandler* van het *SelectionChanged* event (zie codesnippet 8). Vanuit deze *EventHandler* vindt de navigatie plaats met een relatieve URI naar de *WebRadioDetailsPage* plus additionele parameters. Een *Page* heeft een methode *OnNavigatedTo()* die in de *WebRadioDetailsPage* wordt overschreven om de *DataContext* (*viewmodel*) te zetten. Dit gebeurt vanuit de selectie op de collectie *WebRadioViewModels* waarbij de naam gelijk is aan de naam uit de *NavigationContext.QueryString* (zie codesnippet 9).

```
private void ListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    // haal het viewmodel op uit het geselecteerde item van de ListBox
    WebRadioViewModel wrvm = (WebRadioViewModel)((ListBox)sender).SelectedItem;

    if(wrvm != null)
        // navigeer naar de pagina met als context de naam van de radiozender
        this.NavigationService.Navigate(new Uri("/WebRadioDetailsPage.xaml?WebRadio=" + wrvm.Name, UriKind.Relative));
}
```

CODE SNIPPET 8: LISTBOX EVENTHANDLER.

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    // maak string aan voor out parameter
    string name = "";
    // probeer de waarde uit de de additionele parameters van het navigatie request te halen
    if (NavigationContext.QueryString.TryGetValue("WebRadio", out name))
    {
        // haal de eerste radiozender die gelijk is aan de naam
        this.DataContext = App.ViewModel.Items.Where(w => w.Name == name).FirstOrDefault<WebRadioViewModel>();
        // als radiozender niet gevonden is throw exception
        if (this.DataContext == null) throw new NullReferenceException();
    }
    // WebRadio parameter niet gevonden in QueryString throw exception
    else throw new Exception("oeps..");

    // roep base aan
    base.OnNavigatedTo(e);
}
```

CODE SNIPPET 9: OVERSCHREVEN ONNAVIGATEDTO METHODE.

WP7 tasks en launchers

Met tasks en launcher is het mogelijk om snel gebruik te maken van standaardfunctionaliteit die WP7 te bieden heeft. Deze voorgedefinieerde taken zijn te vinden in de *Microsoft.Phone.Tasks* namespace en zijn bijna allemaal afgeleid van de *ChooserBase<TtaskEventArgs>*. Generieke functionaliteit is in deze baseclass vastgelegd voor het tonen: *Show()* en het *Completed* event na afronding van de taak. De bestaande taken lopen uiteen van de *EmailComposeTask* met properties: To, CC, Subject en Body tot de *PhoneCallTask* met properties: DisplayName en PhoneNumber

Voor *WebRadio* is gekozen om gebruik te maken van de *WebBrowserTask-* en de *MediaPlayerLauncher* task, omdat er vanuit de specificatie naar voren komt dat de gebruiker respectievelijk vanuit de applicatie met een simpele druk op een hyperlink naar de website moet kunnen navigeren, en de radiozender moet kunnen afspelen. In het interactieontwerp (zie figuur 4) van de *WebRadioDetailsPage* moet de *view* reageren op het Click event van de *HyperlinkButton* en het Click event van de afspeel-Button, zodat de juiste taak wordt geactiveerd. Beide Buttons hebben een eigen *EventHandler* (zie codesnippet 10) in de code-behind van de *view* die respectievelijk de *WebBrowserTask* en *MediaPlayerLauncher* starten.

FIGUUR 4: INTERACTION DESIGN WEBRADIODETAILSPAGE.

Geo Positioning Systems en locatie services

Het gebruik van plaatsbepaling in mobiele applicaties is in de afgelopen jaren sterk toegenomen. Met Windows Mobile kon dat maar met de WP7 GPS mogelijkheden is er een uniforme manier om vanuit code de hardware module te benaderen. Wat in tijden van een Windows Mobile applicatie resulteerde in een scala van uiteenlopende implementaties om verschillende hardware modules te ondersteunen, biedt WP7 nu dus één manier om dit te doen.

De *WebRadio* applicatie heeft een sectie waarin de radiozenders uit het land van de huidige locatie moeten worden getoond. Bovenaan deze sectie zal de naam van het land moeten komen te staan. Vanuit een *property* van het *MainViewModel* zal de naam worden gebonden op de *MainPage*, waarin de sectie is gedefinieerd. Voor deze plaatsbepaling wordt gebruik gemaakt van de objecten *GeoCoordinateWatcher* en *CivicAddressResolver* (zie codesnippet 11). Deze objecten zijn gedefinieerd in de singleton instantie van de *LocationHelper* class die wordt geinstantieerd in het *MainViewModel*. Het *ResolveAddressCompleted* event treedt op wanneer de plaatsbepaling is afgerond. De *EventHandler* zet het resultaat in de properties van het *LocationHelper* object die op zijn beurt is gebonden aan de *CurrentCountry* property van het *MainViewModel*.

```
public partial class WebRadioDetailsPage : PhoneApplicationPage
{
    private MediaPlayerLauncher mediaPlayerLauncher;
    private WebBrowserTask browserLauncher;

    public WebRadioDetailsPage()
    {
        InitializeComponent();

        // maak nieuwe instanties aan van de taken
        this.mediaPlayerLauncher = new MediaPlayerLauncher();
        this.browserLauncher = new WebBrowserTask();
    }

    private void PlayButton_Click(object sender, RoutedEventArgs e)
    {
        // zet de locatie van de media vanuit de DataContext: ViewModel
        this.mediaPlayerLauncher.Media = new Uri(((WebRadioViewModel)DataContext).StreamUrl, UriKind.Absolute);
        // zet het type van de locatie
        this.mediaPlayerLauncher.Location = MediaLocationType.Data;
        // zet welke controls er beschikbaar zijn na het openen van de taak
        this.mediaPlayerLauncher.Controls = MediaPlaybackControls.Pause | MediaPlaybackControls.Stop;
        // toon de mediaspeler
        this.mediaPlayerLauncher.Show();
    }

    private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
    {
        // zet de locatie van de browser vanuit de DataContext: ViewModel
        this.browserLauncher.URL = ((WebRadioViewModel)DataContext).WebUrl;
        // toon de browser
        this.browserLauncher.Show();
    }

    protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)...
    protected override void OnBackKeyPress(System.ComponentModel.CancelEventArgs e)...
}
```

CODESNIPPET 10: EVENTHANDLERSVAN DE TAKEN IN CODE-BEGIND VAN WEBRADIODETAILSPAGE.XAML.

```
private LocationHelper(string defaultCountry, string defaultCity)
{
    // zet de default waardes mocht het resolven mislukken
    this.CurrentCountry = defaultCountry;
    this.CurrentCity = defaultCity;

    // maak instantie adres resolver
    CivicAddressResolver car = new CivicAddressResolver();
    // definieer EventHandler voor adres resolver
    car.ResolveAddressCompleted +=
        new EventHandler<ResolveAddressCompletedEventArgs>(car_ResolveAddressCompleted);

    // zet de GPS module
    using (GeoCoordinateWatcher gcw = new GeoCoordinateWatcher())
        // is er permissie op GPS module?
        if (gcw.Permission == GeoPositionPermission.Granted)
            // probeer de GPS module te starten met een timeout van 3 sec
            if (gcw.TryStart(false, new TimeSpan(0, 0, 3)))
                // gebruik de adres resolver met de huidige positie
                car.ResolveAddressAsync(gcw.Position.Location);
}

private void car_ResolveAddressCompleted(object sender, ResolveAddressCompletedEventArgs e)
{
    // errors?
    if (e.Error == null)
    {
        // address unknown?
        if (!e.Address.IsUnknown)
        {
            // zet de properties
            this.CurrentCountry = e.Address.CountryRegion;
            this.CurrentCity = e.Address.City;
        }
    }
}
```

CODE SNIPPET 11: LOCATIONHELPER VOOR ADRES BEPALING.

De *defaultCountry* is gespecificeerd in de constructor omdat de emulator op dit moment geen GPS simulatie heeft, wat ontwikkeling zonder fysieke WP7 lastig kan maken. Oplossing hiervoor is een simulatie class te schrijven waarbij de interface *IGeocodeWatcher* wordt geïmplementeerd met logica waarbij om de zoveel tijd het *PositionChanged* event optreedt met een nieuwe locatie.

Conclusie

Snel applicaties ontwikkelen voor WP7 is een feit; developers en designers met een WPF/Silverlight achtergrond kunnen zo aan de slag met de SDK waardoor de leercurve erg klein is. Gebruik de XAML design templates op zo'n manier dat de gebruiker zoveel mogelijk de indruk krijgt dat de applicatie en het WP7 besturingssysteem in elkaar overlopen. En tenslotte een boodschap van het oorspronkelijke Zune Metro thema: houd het simpel!

Referenties

Windows Phone 7 Developer Portal: http://create.msdn.com

Windows Phone 7 XAML design templates: http://wp7designtemplates.codeplex.com/

GPS simulatie voor Windows Phone 7: http://timheuer.com/blog/archive/2010/03/22/geo-location-services-in-windows-phone-7-developer-emulator.aspx

WebRadio achtergrond foto; 'Berlin radiotower' door Julia T. Sho: http://www.foreignlight.com/

Michael Wolbert, is solution developer bij Avanade Nederland. Michael is tijdens zijn studie Technische Informatica in aanraking gekomen met Silverlight en andere Microsoft technologieën die uiteindelijk zijn werk zijn geworden. Hij is te bereiken via michael.wolbert@avanade.com en http://michaelwolbert.nl