Behaviors maken in Expression Blend 3

Krachtige editor voor compileerbare code

Rob Houweling

Met de komst van Silverlight 3 en Expression Blend 3 is er een aantal zeer interessante nieuwe features beschikbaar gekomen. In dit artikel passeren enkele nieuwe features, maar het richt zich vooral op mijn favoriete nieuwe feature: Behaviors.

Een Behavior betekent letterlijk vertaald 'gedrag' en dat is ook eigenlijk precies wat een Behavior is. Het is een stukje interactief gedrag wat je kunt koppelen aan een object. Een behavior bestaat uit een stuk code waarmee je dit interactieve gedrag creëert.
Voor het maken van een behavior in Visual Studio heb je de Expression Blend SDK nodig. Hierin bevindt zich de dll System.Windows.Interactivity.dll. Deze is nodig om de juiste basis classes te kunnen toepassen.
In het voorbeeld in dit artikel gaan we een Behavior maken waarin we geen gebruik maken van Visual Studio maar alleen van Expression Blend. Een andere nieuwe feature is namelijk dat we vanuit Expression Blend 3 nu ook de code-behind kunnen wijzigen. We hebben een krachtige editor, inclusief Intellisense tot onze beschikking dus het is niet meer noodzakelijk om Visual Studio te hebben geinstalleerd om compileerbare code te schrijven. Op deze manier kunnen ook designers en front-end developers code produceren.
Het heeft natuurlijk veel voordelen om de code direct in Expression Blend te kunnen invoeren, maar het kan ook problemen met zich meebrengen. Zorg daarom dat je, voordat je aan je project begint, goede duidelijke afspraken maakt met de betrokken teamleden zodat er geen misverstand over ontstaat wie welke werkzaamheden uitvoert en tot op welk 'niveau'.

Voorbereidend werk

Voor het maken van onze eerste behavior starten we Expression Blend 3 op. We maken een nieuw Silverlight 3 Application + Website project aan met de naam BehaviorTestApp.
Vervolgens voegen we een nieuw project van het type Silverlight 3 Control Library toe in de huidige Solution. Geef aan dit project de naam BehaviorLibrary. Als de BehaviorLibrary aangemaakt is, verwijder dan de MainControl.xaml en de bijbehorende code-behind uit de root van het project. Deze zullen we niet gebruiken.
Als je de projecten hebt aangemaakt is het tijd om de behavior aan te maken.

In dit voorbeeld gaan we een behavior maken die wanneer er met de linkermuistoets op een object wordt geklikt waaraan de behavior is toegekend de applicatie in volledig scherm weergeeft.
Om een behavior in Expression Blend aan te maken klik je met de rechtermuistoets op het project waarin je de behavior wilt aanmaken en selecteer je de optie 'New item'. In de pop-up die vervolgens verschijnt kies je voor de optie 'Behavior'. Type de volgende naam in: 'FullScreenBehavior'.
Zoals je zult zien maakt Expression Blend een cs file aan met daarin wat code en een hoop commentaar in de constructor. Onderin de cs file zul je nog wat uitgecommentarieerde code vinden. Verwijder deze ook.
Als je alles hebt verwijderd zoals aangegeven zal de code er nu als

```
15  namespace BehaviorLibrary
16  {
17      public class FullScreenBehavior : Behavior<DependencyObject>
18      {
19          public FullScreenBehavior() { }
20
21          protected override void OnAttached()
22          {
23              base.OnAttached();
24
25              // Insert code that you would want run when the Behavior is attached to an object.
26          }
27
28          protected override void OnDetaching()
29          {
30              base.OnDetaching();
31
32              // Insert code that you would want run when the Behavior is removed from an object.
33          }
34      }
35  }
```

Er staan twee methodes in de overgebleven code: OnAttached en OnDetaching.
Wat we als allereerste zullen aanpassen is het type object waaraan we de behavior willen koppelen. Alle objecten die afgeleid zijn van een DependencyObject zijn te gebruiken voor een behavior.
In ons geval willen we de MouseLeftButtonDown gaan afvangen, dus we zullen hiervoor de basis class moeten gebruiken die dit event toepast. In ons geval is dat de UIElement class. UIElement heeft als base type DependencyObject maar implementeert, in tegenstelling tot de DependencyObject, wel de diverse mouse-events.
Verander de class definitie naar:
public class FullScreenBehavior: Behavior<UIElement>
De volgende stap die we doen is het event koppelen. Dit doen we

in de OnAttached methode. De OnAttached methode wordt aangeroepen op het moment dat er een object, in ons geval een UI-Element, wordt 'gekoppeld'. Dus op dat moment hebben we pas de beschikking over ons object en kunnen we events koppelen, properties uitlezen of vullen, etc.
Type de volgende code onder de regel base.OnAttached():

```
this.AssociatedObject.MouseLeftButtonDown +=
```

Druk vervolgens op tab (voor de Visual Studio ontwikkelaars: 1 keer op tab drukken is voldoende). Als het goed is zie je dat Expression Blend, net zoals Visual Studio, automatisch de eventhandler voor je aanmaakt.
Binnen de eventhandler typen we de volgende code:

```
Application.Current.Host.Content.IsFullScreen = !Application.
Current.Host.Content.IsFullScreen;
```

Wat we hiermee bereiken is dat we een soort 'toggle button' maken. Als er op de knop wordt gedrukt en FullScreen is actief, dan gaan we terug naar normale modus. Als we in normale modus zitten en men drukt op de knop, schakelen we naar fullscreen modus.
Als laatste, om de behavior netjes af te ronden voegen we de onderstaande regel toe, welke de eventhandler weer ontkoppelt:

```
this.AssociatedObject.MouseLeftButtonDown -= AssociatedObject_
MouseLeftButtonDown;
```

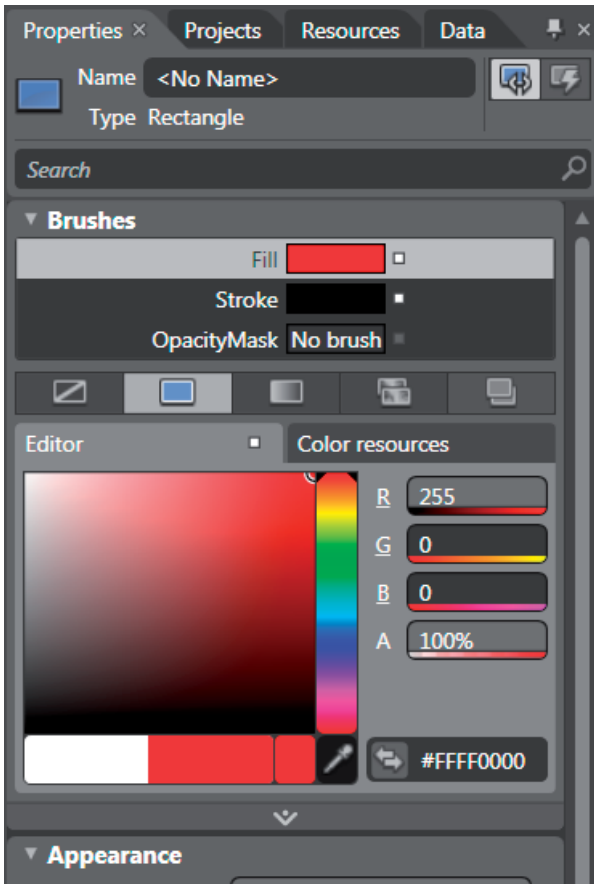
Als je de overgebleven commentaar regels ook verwijderd uit de OnAttached en OnDetaching methodes, dan zal je code er als volgt uitzien:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Interactivity;

namespace BehaviorLibrary
{
    public class FullScreenBehavior : Behavior<UIElement>
    {
        public FullScreenBehavior() { }

        protected override void OnAttached()
        {
            base.OnAttached();
            this.AssociatedObject.MouseLeftButtonDown +=
                new System.Windows.Input.MouseButtonEventHandler(AssociatedObject_MouseLeftButtonDown);
        }

        protected override void OnDetaching()
        {
            base.OnDetaching();
            this.AssociatedObject.MouseLeftButtonDown -= AssociatedObject_MouseLeftButtonDown;
        }

        private void AssociatedObject_MouseLeftButtonDown(object sender, System.Windows.Input.MouseButtonEventArgs e)
        {
            Application.Current.Host.Content.IsFullScreen = !Application.Current.Host.Content.IsFullScreen;
        }
    }
}
```

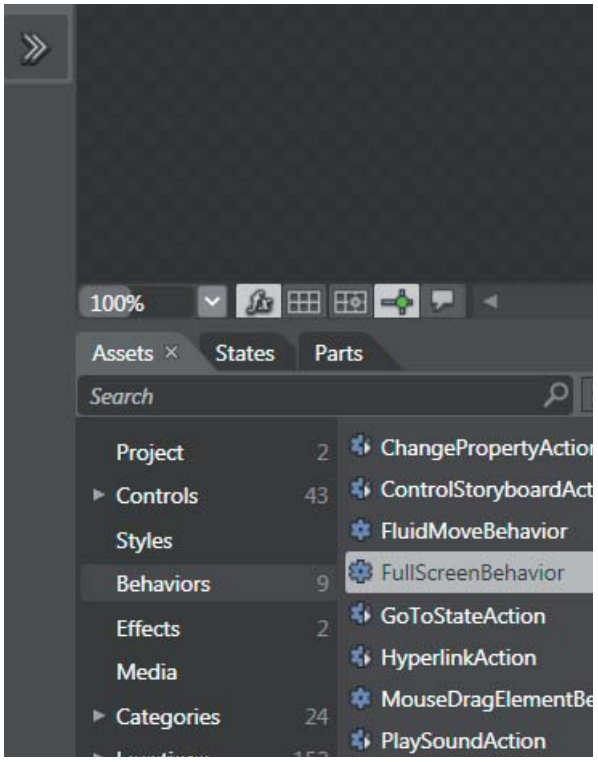
Behavior toepassen

Om nu de behavior toe te passen open je de MainPage.xaml van het BehaviorTestApp project. Teken op de LayoutRoot een Rectangle door gebruik te maken van de Rectangle tool:

Zorg dat je Rectangle een rode fill krijgt zodat hij beter opvalt. Je kunt dit doen in de Properties tab, in de Brushes panel aan de rechterzijde (zorg dat je de Rectangle geselecteerd hebt):



Nu willen we bereiken dat, indien er op de rode rechthoek wordt geklikt, de applicatie fullscreen wordt geopend.
In Expression Blend 3 is ook de Asset library, of eigenlijk de panel die de library weergeeft, sterk veranderd. Er zijn nu categorieen beschikbaar waaronder de Assets zijn onderverdeeld.
Tussen de categorieen zie je Behaviors staan. Selecteer deze categorie en zoals je zult zien staat de door ons aangemaakte FullScreenBehavior er ook tussen:

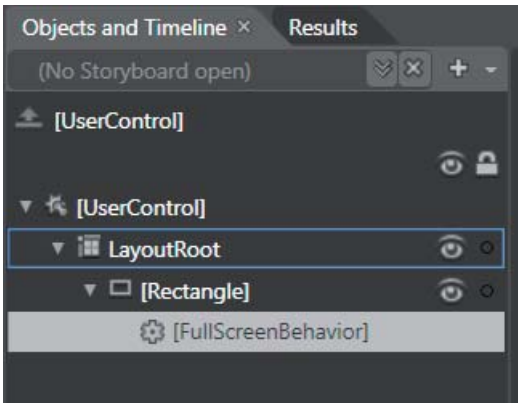


Zorg nu dat de MainPage.xaml in split-screen wordt weergegeven door middel van View => Active Document View => Split View of door op het Split icoon te klikken:

Sleep nu de behavior vanuit de Asset panel op de aangemaakte Rectangle.
Je zult zien dat in de XAML de volgende code wordt toegevoegd binnen de Rectangle tag:

```
<i:Interaction.Behaviors>
  <BehaviorLibrary:FullScreenBehavior/>
</i:Interaction.Behaviors>
```

Als je het Objects and Timeline panel opent, zie je dat er een object is toegevoegd onder de Rectangle, namelijk onze behavior:



Als laatste stap voordat we de behavior gaan testen moeten we een reference van onze library naar ons testproject maken. Dit gaat op soortgelijke manier als in Visual Studio. Door middel van met de rechtermuistoets te klikken op het project "BehaviorTestApp" => Add Project Reference => BehaviorLibrary kun je de reference toevoegen.
Druk nu op F5 om het project te builden en te runnen.
Klik nu ergens op het witte vlak. Je zult zien dat er niets gebeurt.
Klik nu vervolgens op de rode rechthoek en je ziet dat de applicatie naar fullscreen modus schakelt. Als je in fullscreen modus nogmaals op de rode rechthoek klikt zie je dat je terugkeert naar de normale "windowed" modus.
Het is natuurlijk ook mogelijk om deze Behaviors met Visual Studio te maken. Er zijn echter op moment van schrijven nog geen templates beschikbaar waarmee dit mogelijk is.
Andere soorten behaviors
Naast de behaviors uit het voorbeeld zijn in Expression Blend 3 en Silverlight 3 nog andere mogelijkheden die wat anders van opzet zijn namelijk de TriggerAction<T> en TargetedTriggerAction<T>.
De twee laatstgenoemde opties zijn beter inzetbaar voor de wat meer complexe behaviors.
Wanneer je een Behavior<T> gebruikt, is de Behavior class verantwoordelijk voor het attachen en detachen van de item events.
TriggerAction<T>
Bij het gebruik van een TriggerAction<T> kun je zelf bepalen welk event de Action die uitgevoerd moet worden zal aanroepen (triggeren). Dus in tegenstelling tot een Behavior die geschreven is voor bijvoorbeeld een MouseLeftButtonDown kun je een TriggerAction<T> laten uitvoeren door alle beschikbare events. Wil je bijvoorbeeld een DoubleClick Behavior maken is het logischer om een Behavior<T> class te gebruiken omdat deze altijd en alleen op het MouseLeftButtonDown event getriggered zal worden.
Voor de Behavior die we in ons voorbeeld (FullScreenBehavior) hebben gebruikt zou je denken dat het wellicht slimmer zijn om een TriggerAction<T> class te gebruiken. Op die manier kan je door middel van meerdere events de applicatie in volledig scherm weergeven. De reden waarom ik voor een Behavior<T> class heb gekozen is omdat volledig scherm alleen maar vanuit een gebruiker geinitieerd event getriggered kan worden. Dus door middel een klik met de muis of het indrukken van een toets op het toetsenbord. Het is dus mogelijk om de behavior uit te breiden met de genoemde events indien je dat wenst.
TargetedTriggerAction<T>
De laatste die overblijft is de TargetedTriggerAction<T>. De TargetedTriggerAction<T> komt grotendeels overeen met de TriggerAction<T>. Het grote verschil met de TriggerAction<T> is dat het met de TargetedTriggerAction mogelijk is om een ander object te manipuleren dan degene waarmee de TargetedTriggerAction is gekoppeld.

Conclusie

Behaviors zijn zeer krachtige stukken code welke eenvoudig te programmeren zijn en zeer goed herbruikbaar zijn in meerdere projecten.
Het kost de programmeur weinig extra moeite een behavior te creeeren in plaats van de code die normaalgesproken gemaakt zou worden.
Expression Blend geeft daarnaast geweldige ondersteuning voor behaviors. Door middel van drag en drop en het instellen van wat properties kan een designer/front-end developer al interactiviteit aanbrengen in een applicatie zonder dat een ontwikkelaar zich daar, buiten het schrijven van de behavior, mee hoeft bezig te houden.
Microsoft heeft weer een extra mogelijkheid gebracht om de samenwerking tussen ontwikkelaars en designers/front-end developers te verbeteren.

Rob Houweling, is Silverlight Developer bij Amercom.
Hij is te bereiken op rob@amercom.nl