Usare il Behaviors SDK per Windows 8.1

#### di [Andrea Boschin](http://mvp.microsoft.com/en-us/mvp/Andrea%20Boschin-4000289) – Microsoft MVP

* 1. 

*Aprile, 2014*

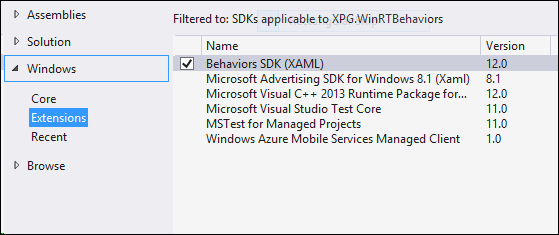
Rilasciati all'interno di Blend 3.0, nel 2009, come un mero strumento a supporto dei designer che li poteva usare per inserire sulla superficie di Blend non solo componenti di interfaccia, ma anche dei "frammenti di logica", i behavior hanno immediatamente assunto un ruolo primario nello sviluppo di applicazioni, grazie alla loro capacità di favorire la netta separazione tra logica di business e interfaccia utente. Questa capacità è rappresentata dal fatto che essi sono in grado di racchiudere e isolare porzioni di codice - con accesso diretto ai componenti di interfaccia ma pilotati dalla logica di business - inseribili facilmente direttamente nello XAML senza che sia necessaria alcuna logica accessoria.

Nati in particolare all'interno di Silverlight, essi sono immediatamente diventati parte del kit di sopravvivenza dello sviluppatore, specialmente di quello che intendeva implementare il pattern MVVM con facilità ed in seguito hanno visto il porting verso Windows Phone e finalmente anche in WinRT, con il rilascio dell'ultima versione 8.1. Conoscerli e usarli opportunamente può davvero far risparmiare un sacco di tempo. Vediamo come.

## Usare i behavior esistenti

Il primo passo nell'utilizzo dei behavior è quello di inserirli nel progetto di Visual Studio e cominciare ad usare quelli esistenti. In effetti la libreria - denominata "Behaviors SDK (XAML)" porta con se una serie di classi base da cui è possibile fare ereditare i propri behavior ma anche un set abbastanza nutrito di componenti già realizzati e pronti all'uso. Essi hanno la caratteristica di essere parametrizzabili e generici così da essere fortemente riutilizzabili e quindi sono anche una buona scuola per quando dovremo realizzarne di nostri.

Per inserirli nel progetto con Visual Studio 2013 è sufficiente selezionarli dalla libreria delle estensioni (vedi sotto) operando come se si dovesse aggiungere una referenza al progetto.

* 1. 

I Behavior sono una libreria di componenti nativi e managed, e come tali possono essere usati sia in progetti C#/VB.NET sia in C++. Non richiedono particolari cautele come avviene per altri tipi di componenti perciò, una volta inserita la referenza all'SDK sarà immediatamente possibile iniziare ad usarli. Non c'è dubbio che il modo migliore è quello di usare Blend, al cui interno vengono visualizzati nella scheda degli Assets, come dei piccoli componenti che possono essere trascinati sulla superficie del designer e agganciati ai vari elementi. Una volta collegati è possibile modificarne le proprietà usando la consueta finestra di blend.

Dal punto di vista dello sviluppatore un bahavior appare ne più ne meno che come una attached property che può essere valorizzata su qualunque elemento del DOM. La proprietà è dichiarata nel namespace "Microsoft.Xaml.Interactivity" e una volta inserito nel codice xaml essa appare come una collection:

* 1. XAML
  2. 1: <Page
  3. 2: x:Class="App1.MainPage" mc:Ignorable="d"
  4. 3: xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  5. 4: xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  6. 5: xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  7. 6: xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  8. 7: xmlns:i="using:Microsoft.Xaml.Interactivity">
  9. 8: <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  10. 9: <Rectangle x:Name="rect" Fill="Yellow" Width="100" Height="100">
  11. 10: <i:Interaction.Behaviors>
  12. 11: <!-- aggiungere qui in behaviors -->
  13. 12: </i:Interaction.Behaviors>
  14. 13: </Rectangle>
  15. 14: </Grid>
  16. 15: </Page>

A questo punto va referenziato un ulteriore namespace, chiamiamolo per brevità "core", il quale contiene i behavior predefiniti di cui parlavo poco fa. In particolare essi sono "EventTriggerBehavior" e "DataTriggerBehavior". I loro nomi dovrebbero essere autoesplicativi ma per completezza diciamo che il primo lo si può agganciare ad un singolo evento di un elemento mentre il secondo fa riferimento al un valore di una proprietà.

Poniamo ad esempio di voler fare in modo che il rettangolo giallo dell'esempio precedente debba diventare rosso quando cliccato. Potremmo semplicemente aggiungere il seguente snippet all'interno della collezione dei behaviors:

* 1. XAML
  2. 1: <core:EventTriggerBehavior EventName="PointerPressed">
  3. 2: <core:ChangePropertyAction PropertyName="Fill" Value="Red" />
  4. 3: </core:EventTriggerBehavior>

In buona sostanza, stiamo dicendo che al verificarsi di un evento PointerPressed venga eseguita una "action" che cambia il valore di una proprietà. Le action sono un secondo livello di componenti predefiniti di cui dirò tra poco. Questa action in particolare si occupa di variare la proprietà "Fill" impostandola a "Red". Mandando in esecuzione l'esempio si vedrà che l'effetto ottenuto è proprio quello desiderato, ma anche che, a differenza di quanto avviene per il VisualStateManager, il sistema non tiene traccia dello stato originario e quindi per riportarlo allo stato precedente dovremmo aggiungere l'operazione inversa:

* 1. XAML
  2. 1: <Rectangle x:Name="rect" Fill="Yellow" Width="100" Height="100">
  3. 2: <i:Interaction.Behaviors>
  4. 3: <core:EventTriggerBehavior EventName="PointerPressed">
  5. 4: <core:ChangePropertyAction PropertyName="Fill" Value="Red" />
  6. 5: </core:EventTriggerBehavior>
  7. 6: <core:EventTriggerBehavior EventName="PointerReleased">
  8. 7: <core:ChangePropertyAction PropertyName="Fill" Value="Yellow" />
  9. 8: </core:EventTriggerBehavior>
  10. 9: </i:Interaction.Behaviors>
  11. 10: </Rectangle>

Le action che si possono applicare sono molteplici e coprono un vasto numero di necessità:

**CallMethodAction**: L'action richiama un metodo dell'oggetto cui si riferisce. Assomiglia molto al collegamento di un EventHandler nel codebehind.

**ControlStoryBoardAction**: Consente di collegare una o più storyboard e avviarle, così come si fa con i trigger di WPF.

**GoToStateAction**: Consente di attivare uno stato del VisualStateManager.

**InvokeCommandAction**: Uno delle action più utilizzate, in ambito MVVM consente di attivare comandi in risposta a qualunque evento se usato in concomitanza con EventTriggerBehavior.

**NavigateToPage**: Attiva la navigazione verso una pagina dell'applicazione

**PlaySoundAction**: Esegue un suono

Una nota a parte la merita sicuramente l'InvokeCommandAction, la cui esistenza è indispensabile in scenari MVVM. Esso infatti sopperisce alla mancanza dei consueti "*Command*" e "*CommandParameter*" relativi la maggioranza degli eventi dei controlli XAML. Come in Silverlight, anche nelle Windows Store apps solamente ButtonBase implementa queste proprietà. E qui viene in aiuto questa action che associata ad un opportuno EventTriggerBehavior è in grado di riportare ad un Command del ViewModel qualunque iterazione dell'utente, aprendo numerose strade.

L'uso del EventTriggerBehavior è sicuramente il più consueto ma non va dimenticata la presenza del meno intuitivo DataTriggerBehavior. Esso ha lo scopo di legare l'esecuzione di una o più action al cambiare del valore di una proprietà. Pensiamo ad esempio ad avere una proprietà che indica una soglia di pericolosità, aggiornata in tempo reale (magari via SignalR). L'utilizzo di un DataTriggerBehavior è specifico per questi casi e ci consente di attivare una action quando il valore va oltre una determinata soglia:

* 1. XAML
  2. 1: <Rectangle Background="Green">
  3. 2: <Interactivity:Interaction.Behaviors>
  4. 3: <Core:DataTriggerBehavior Binding="{Binding Severity}" ComparisonCondition="GreaterThanOrEqual" Value="3">
  5. 4: <Core:ChangePropertyAction PropertyName="Background">
  6. 5: <Core:ChangePropertyAction.Value>
  7. 6: <SolidColorBrush>Red</SolidColorBrush>
  8. 7: </Core:ChangePropertyAction.Value>
  9. 8: </Core:ChangePropertyAction>
  10. 9: </Core:DataTriggerBehavior>
  11. 10: </Interactivity:Interaction.Behaviors>
  12. 11: </Rectangle>

Il codice qui sopra usa una condizione, indicata dalla proprietà “Comparison Condition” per definire quale sia il valore soglia che converte. il colore da Verde a Rosso.

I behavior come abbiamo visto sono uno strumento davvero insostituibile in molti casi. Ancor di più se, come vedremo nel prossimo articolo ne creiamo di nostri, con proprietà che ne parametrizzano l’azione.

#### di [Andrea Boschin](http://mvp.microsoft.com/en-us/mvp/Andrea%20Boschin-4000289) – Microsoft MVP