

Silverlight 2 Tutorial

From ScottGu blog

Đào Hải Nam



2009

Silverlight 2 Tutorial

Được dịch từ blog **ScottGu** - <http://weblogs.asp.net/scottgu/default.aspx>

Người dịch: **Đào Hải Nam** – <http://www.daohainam.com>

Mục lục

Bài 1: Viết chương trình “Hello World” với Silverlight 2 và VS 2008	5
Có gì trong một ứng dụng Silverlight.....	6
Cách thêm vào các control và bắt các sự kiện	8
Bước tiếp theo	11
Bài 2: Sử dụng Layout management.....	12
Layout Management là gì?.....	12
Canvas Panel.....	12
Grid Panel	15
Dùng các layout panel để sắp xếp các thành phần trên trang Digg của chúng ta.....	17
Tự động thay đổi kích thước của ứng dụng	21
Bước tiếp theo	23
Bài 3: Kết nối mạng để lấy dữ liệu vào một DataGrid.....	24
Dùng mạng để lấy dữ liệu từ Digg.....	24
Truy cập vào các domain khác	24
Tập hàm API của Digg.com.....	24
Dùng System.Net.WebClient để thực hiện lời gọi không đồng bộ đến Digg REST Feed	25
Dùng LINQ để phân tích chuỗi XML trả về bởi Digg.com để đưa vào lớp Story.....	26
Hiển thị dữ liệu của Digg trong một DataGrid.....	27
Bước tiếp theo	29
Bài 4: Xây dựng giao diện dùng Style	30
Bước tiếp theo	32
Bài 5: Dùng ListBox và DataBinding để hiển thị dữ liệu	33
Bước tiếp theo	38
Bài 6: Dùng User Control để cho phép xem theo dạng Master/Detail.....	39
User Control là gì ?	39
Tạo User Control “StoryDetailsView”	40
Xây dựng một cửa sổ modal đơn giản bằng cách dùng một User Control:.....	41
Hiển thị control StoryDetailsView.....	43
Chuyển dữ liệu cho đối tượng StoryDetailsView	44
Hoàn chỉnh User Control Layout.....	46
Bước tiếp theo	48
Bài 7: Dùng các mẫu để tùy biến Look and Feel của control	49

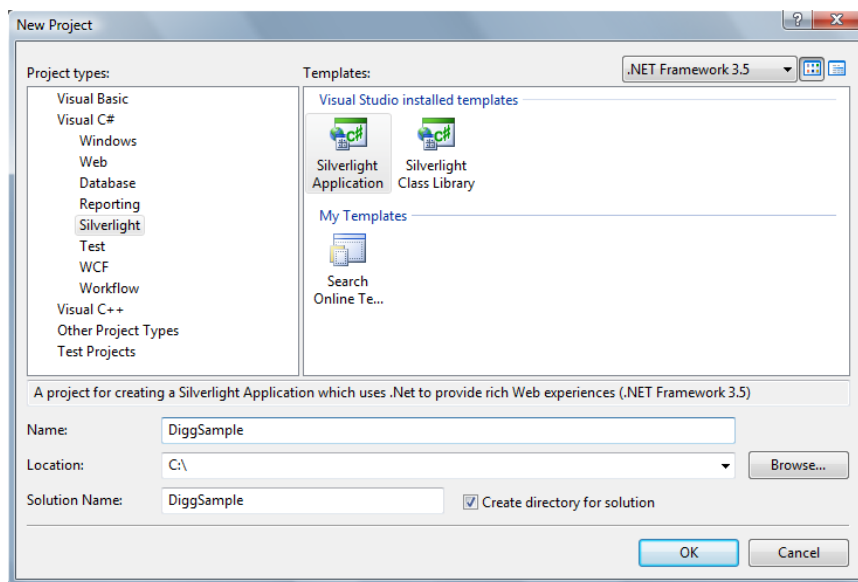
Tùy biến Look and Feel	49
Tùy biến nội dung của các control	49
Tùy biến các control dùng Control Template	52
Kết hợp nội dung bên trong Control Template	53
Đánh bóng lại chương trình Digg.....	56
Bước tiếp theo.....	59
Bài 8: Xây dựng phiên bản chạy trên desktop với WPF.....	60
Xây dựng phiên bản chạy trên desktop với WPF.....	60
Bước 1: Tạo ứng dụng WPF	60
Bước 2: Copy mã lệnh từ chương trình Digg vào ứng dụng WPF.....	61
Bước 3: Khắc phục một số vấn đề	62
Bước 4: Chạy chương trình Digg trên một cửa sổ desktop.....	62
Bước 5: Chạy chương trình.....	63
Tổng kết.....	64
Tham khảo:	66

Bài 1: Viết chương trình “Hello World” với Silverlight 2 và VS 2008

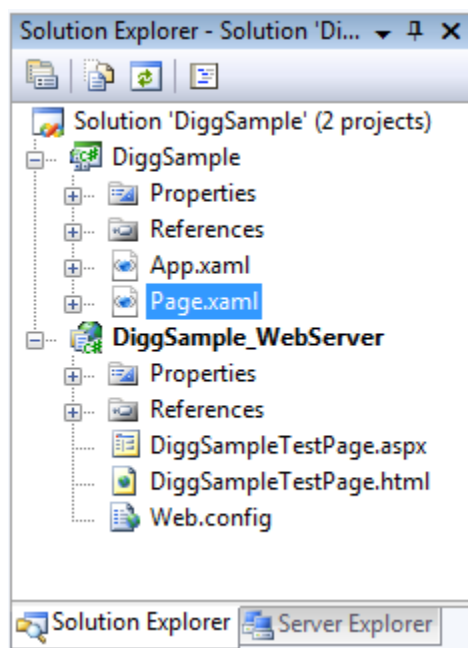
Bài này sẽ là bài đầu tiên trong loạt 8 bài được dịch lại từ Scottgu’s blog. Đây cũng là một loạt bài hướng dẫn khá hay về Silverlight 2, các bạn có thể xem danh sách 8 bài này trong bài viết [First Look at Silverlight 2](#).

Nếu bắt đầu từ đầu, lời khuyên là bạn nên đọc tất cả các bài viết này theo thứ tự, và hãy cố gắng tự mình làm lại các ví dụ có sẵn.

Chúng ta sẽ bắt đầu xây dựng chương trình Digg bằng cách chọn File->New trong VS2008 và dùng hộp thoại New Project để tạo một ứng dụng “Silverlight Application” (nhớ rằng bạn đã tải về và cài đặt [Silverlight Tools for VS 2008](#) thì mới có mục này).



Chúng ta sẽ đặt tên cho dự án này là “DiggSample”. Khi nhấn vào nút OK, Visual Studio sẽ mở một hộp thoại nữa cho phép lựa chọn việc tạo chỉ một ứng dụng Silverlight, hay tạo thêm cả một ứng dụng ASP.NET vào Solution để chứa ứng dụng Silverlight bên trong. Với ứng dụng này, chúng ta sẽ chọn để thêm một ứng dụng ASP.NET và đặt tên cho nó là “DiggSample_WebServer”. Khi chúng ta nhấn OK, VS sẽ tạo một solution chứa cả hai ứng dụng Silverlight và ASP.NET.

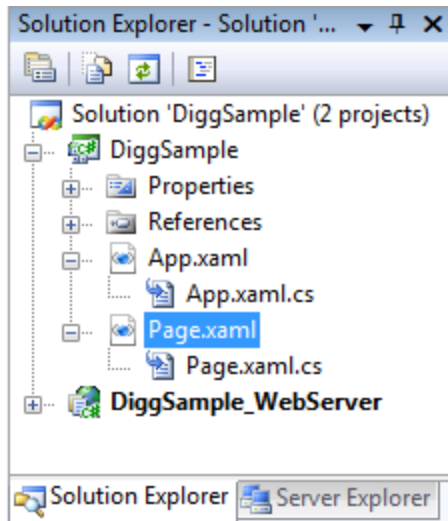


Khi thực hiện lệnh build, VS sẽ tự động sao chép ứng dụng Silverlight sang ứng dụng web của chúng ta. Ứng dụng web được tạo sẵn này sẽ có cả các trang ASP.NET và các trang HTML tĩnh, và chúng ta có thể dùng các trang này để test ứng dụng Silverlight.

Ghi chú: Các ứng dụng Silverlight có thể được dùng bên trong bất kỳ web server nào (bao gồm cả Apache trên Linux) và được chứa bên trong các trang HTML tĩnh hay bất kỳ dạng trang web động nào (bao gồm cả PHP, Java, Ruby, Python...). Với ứng dụng Digg, chúng ta sẽ không dùng mã server-side nào, chúng ta sẽ dùng tính năng “cross-domain networking” của Silverlight để truy cập tập hàm của dịch vụ Digg một cách trực tiếp. Sở dĩ tôi tạo cả ứng dụng ASP.NET là vì muốn tận dụng tính năng triển khai tự động và dùng thử nó trong web server có sẵn khi phát triển ứng dụng ASP.NET với VS.

Có gì trong một ứng dụng Silverlight

Mặc nhiên, một ứng dụng Silverlight mới được tạo sẽ chứa hai file có tên Page.xaml và App.xaml, cũng như các file code behind tương ứng (được viết bằng VB, C#, Ruby hoặc Python).

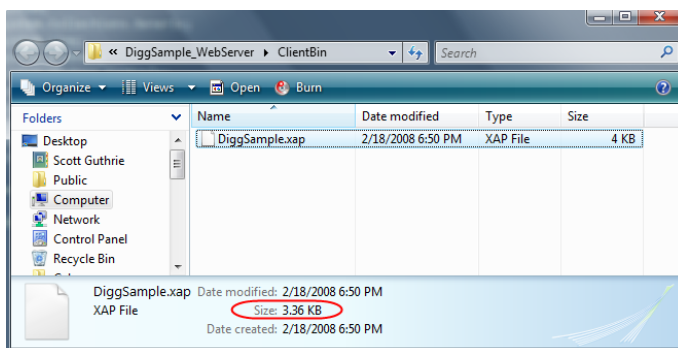


Các file XAML là các file ở dạng văn bản mà bạn dùng để khai báo các thành phần giao diện trong một ứng dụng Silverlight hay WPF. XAML cũng có thể được dùng cả trong việc khai báo các đối tượng .NET.

File App.xaml được dùng để khai báo các tài nguyên, kiểu như các đối tượng brush hay style mà sẽ được dùng trong suốt toàn bộ ứng dụng. Lớp Application (code-behind của file App.xaml) có thể được dùng để xử lý các sự kiện ở cấp độ ứng dụng, như Application_Startup, Application_Exit và Application_UnhandledException.

File Page.xaml mặc nhiên được dùng như thành phần giao diện mà nó sẽ được hiển thị khi ứng dụng được kích hoạt. Chúng ta có thể định nghĩa giao diện bên trong nó, và chúng ta cũng có thể xử lý các sự kiện gây ra bởi các thành phần giao diện này bên trong lớp code-behind của file Page.xaml.

Khi chúng ta build ứng dụng DiggSample, Visual Studio sẽ mặc nhiên biên dịch các lệnh và các thẻ XAML vào một file assembly .NET, rồi đóng gói nó cùng với các tài nguyên tĩnh (như các file hình ảnh hay văn bản mà chúng ta muốn đóng gói cùng) vào một file có tên DiggSample.xap:



Các file “.xap” (đọc là zap) dùng thuật toán nén zip để tối thiểu hóa kích thước file. Một ứng dụng “Hello world” được viết trong .NET Silverlight (dùng VB hay C#) chỉ có kích thước 4KB.

Ghi chú: Một số control trong bản Beta1 nếu được dùng sẽ phải được đóng góp kèm với ứng dụng, do vậy có thể sẽ làm tăng kích thước của file .xap. Chương trình Diggchi dùng các control trong bản Beta2 và phiên bản chính thức, do vậy kích thước tổng cộng sẽ chỉ khoảng 6-8KB.

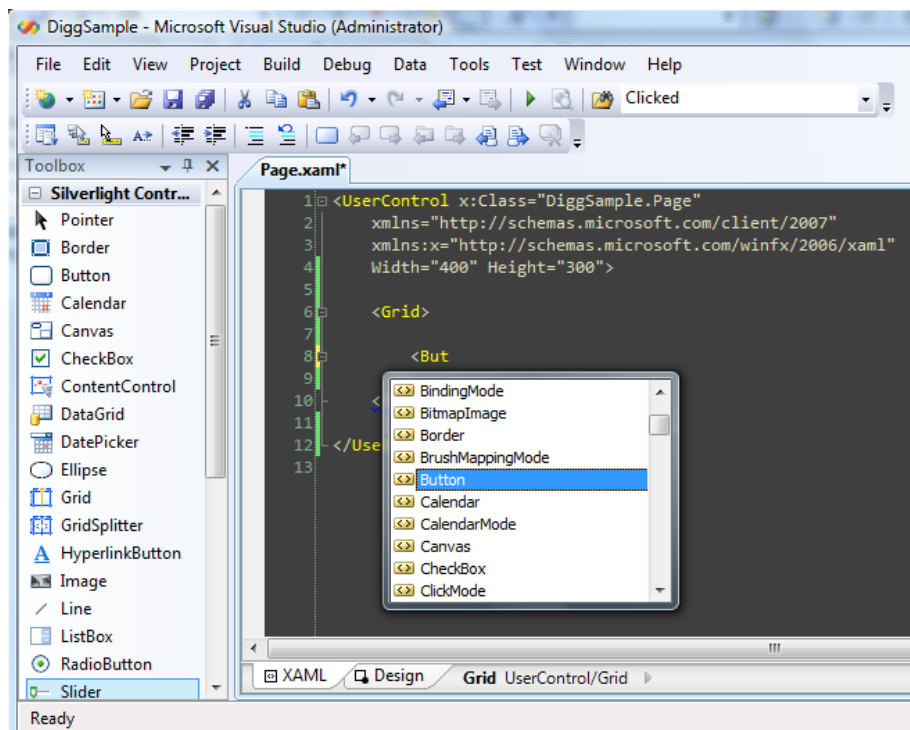
Để chạy một file Silverlight 2, bạn sẽ phải thêm thẻ vào trong một file HTML (không cần dùng Javascript) Silverlight sẽ tự động download file .xap, khởi tạo nó và chạy nó trên nền của trình duyệt.

Điều này cho phép việc chạy ứng dụng Silverlight không phụ thuộc trình duyệt (Safari, FireFox, IE, ...) và hệ điều hành (Windows, Mac, và Linux).

Các file HTML và ASP.NET để test cũng được thêm vào sẵn ngay khi tạo project, do vậy chúng ta chỉ việc đơn giản là nhấn F5 để build, run và test.

Cách thêm vào các control và bắt các sự kiện

Hiện tại chương trình Digg của chúng ta vẫn chưa có gì, do vậy khi chạy nó các bạn sẽ chỉ thấy một trang web trống. Chúng ta có thể thay đổi bằng cách mở file Page.xaml và thêm vào một số nội dung:

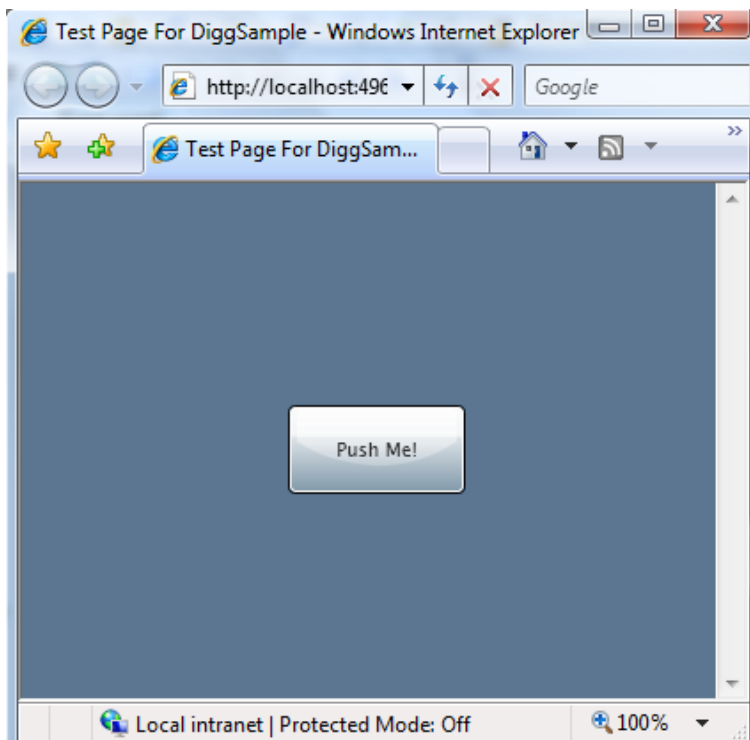


Chúng ta sẽ bắt đầu bằng việc thay đổi màu nền của grid và khai báo một nút bấm bên trong nó. Chúng

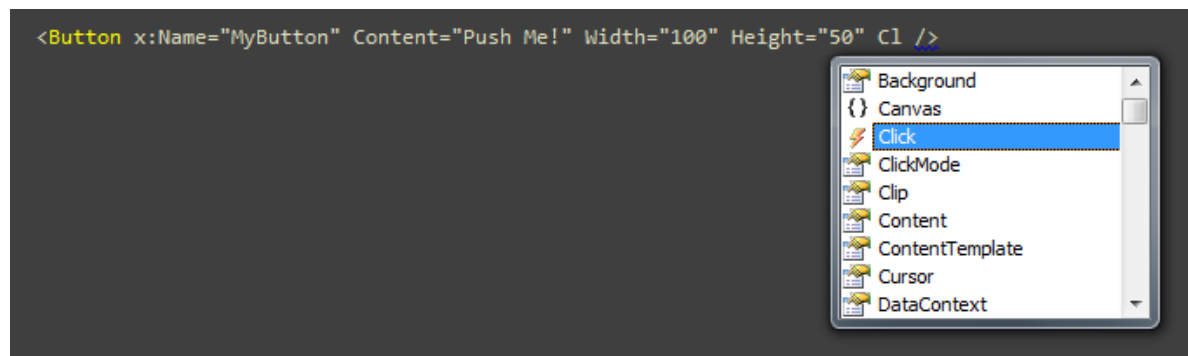
ta cũng sẽ đặt giá trị cho thuộc tính “x:Name” của nút bấm thành “MyButton” – điều này sẽ cho phép có thể tham chiếu đến nút bấm này khi lập trình, chúng ta cũng sẽ đặt giá trị cho các thuộc tính Content, Width và Height:

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="400" Height="300">
5
6   <Grid Background="#FF5C7590">
7
8       <Button x:Name="MyButton" Content="Push Me!" Width="100" Height="50"/>
9
10  </Grid>
11
12 </UserControl>
13
```

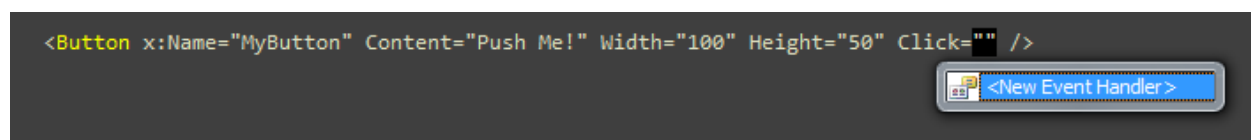
Khi chạy, chương trình của chúng ta sẽ hiển thị ở giữa trang với một nút bấm có dòng chữ “Push Me” giống như sau:



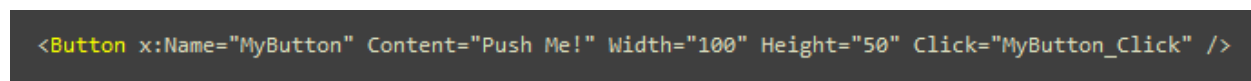
Để thêm một “hành vi” cho nút bấm, chúng ta có thể thêm một hàm xử lý sự kiện “Click” cho nó, để thêm vào, chúng ta sẽ gõ vào tên sự kiện như sau:



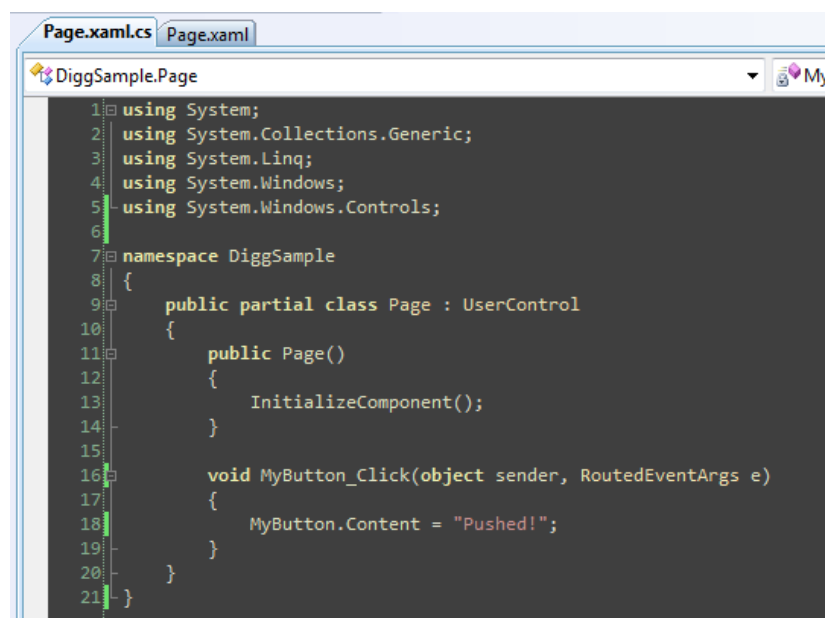
Ngay khi đó, VS sẽ nhắc chúng ta thêm hàm xử lý sự kiện vào lớp code-behind:



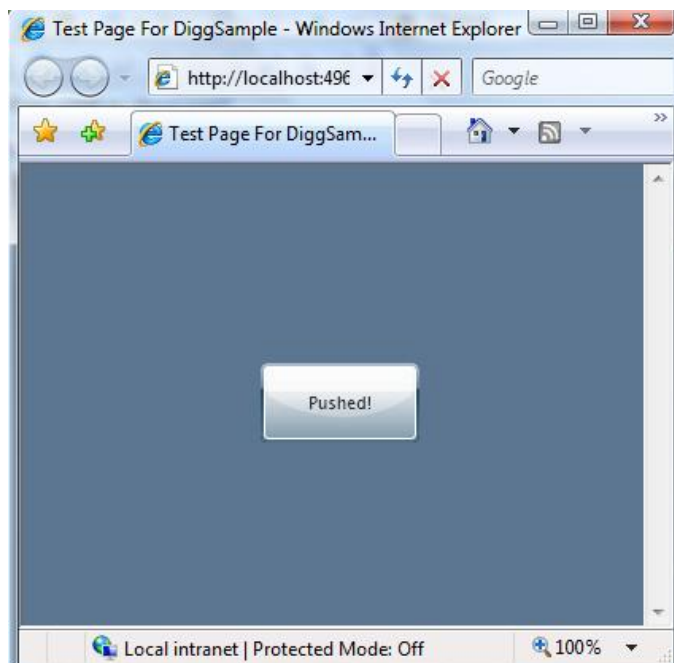
Chúng ta có thể gõ vào tên của một phương thức mới mà chúng ta muốn tạo, hoặc chỉ đơn giản nhấn Enter để dùng luôn tên mặc nhiên của nó:



VS sẽ tự thêm vào một hàm xử lý sự kiện trong file code-behind. Chúng ta có thể dùng hàm này để cập nhật nội dung của nút bấm mỗi khi nhấn chuột lên nó:



Sau khi đã thay đổi giống như trên, chúng ta có thể chạy lại ứng dụng và nhấn lên nút bấm lần nữa, bây giờ nội dung của nó sẽ được cập nhật lại thành "Pushed!":



Bước tiếp theo

Chúng ta vẫn còn một vài việc phải làm trước khi ứng dụng được hoàn thành... 😊
Bước kế tiếp là điều chỉnh lại cấu trúc tổng thể giao diện của ứng dụng, và sắp đặt thêm một số control bên trong nó. Để làm điều này, xin mời các bạn xem phần tiếp theo:

Bài 2: Sử dụng Layout management

Đây là phần hai của loạt 8 bài hướng dẫn cách xây dựng một chương trình Digg đơn giản sử dụng bản Beta1 của Silverlight 2. Các bài hướng dẫn này nên được đọc theo thứ tự, và giúp giải thích một số khái niệm nền tảng của Silverlight.

Layout Management là gì?

Silverlight và WPF hỗ trợ một hệ thống sắp xếp các thành phần giao diện rất mềm dẻo cho phép các nhà phát triển và người thiết kế dễ dàng đặt vị trí cho các control bên trong một giao diện. Hệ thống sắp xếp giao diện này cho phép đặt các control tại một tọa độ cố định, cũng như một mô hình đặt vị trí động, cho phép các layout và control có thể được đặt kích thước và hướng khi thay đổi kích thước cửa sổ trình duyệt. Các nhà phát triển dùng Silverlight và WPF dùng layout panel để đặt vị trí và kích thước của các control chứa bên trong. Silverlight Beta1 có 3 trong số những layout thường dùng nhất trong WPF:

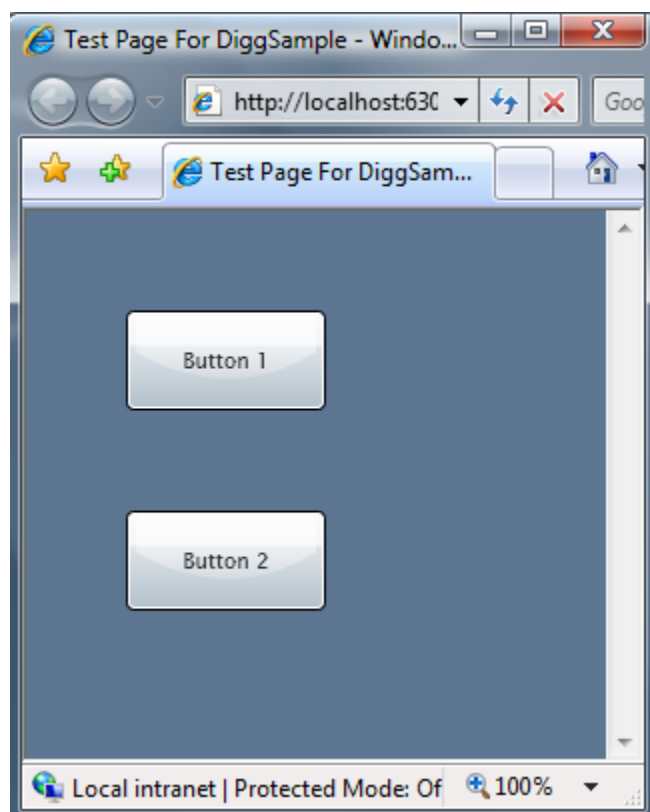
- Canvas
- StackPanel
- Grid

Canvas Panel

Canvas panel là loại layout panel cơ bản hỗ trợ việc đặt vị trí các control dùng tọa độ cụ thể. Bạn đặt vị trí các thành phần trong một Canvas dùng một đặc tính có trong XAML được gọi là “Attached Properties” (các thuộc tính đính kèm) cho phép bạn chỉ ra một vị trí tương đối của nó so với các thuộc tính Left, Top, Right hay Bottom của đối tượng Canvas chứa nó. Các thuộc tính gắn kèm này là một cách hữu dụng vì nó cho phép một panel cha mở rộng tập thuộc tính của một control chứa bên trong nó. Canvas, bằng cách định nghĩa một thuộc tính gắn kèm cho Top và Left, nó cho phép định nghĩa các thuộc tính Top và Left của một Button (hay bất kỳ đối tượng giao diện nào đã thêm vào Canvas), mà không cần phải thêm các thuộc tính này hoặc chỉnh sửa lại lớp Button. Chúng ta có thể thêm hai nút bấm vào một Canvas, và đặt vị trí cho nó là 50 pixel từ cạnh trái, 50 pixel và 150 pixel từ cạnh trên dùng XAML giống như dưới đây (các thuộc tính Canvas.Left and Canvas.Top là các ví dụ về thuộc tính gắn kèm):

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="500" Height="350">
5
6   <Canvas Background="#FF5C7590">
7
8       <Button Content="Button 1" Width="100" Height="50" Canvas.Left="50" Canvas.Top="50"/>
9
10      <Button Content="Button 2" Width="100" Height="50" Canvas.Left="50" Canvas.Top="150"/>
11
12   </Canvas>
13
14 </UserControl>
```

Và sau đây là kết quả hiển thị trên trình duyệt:



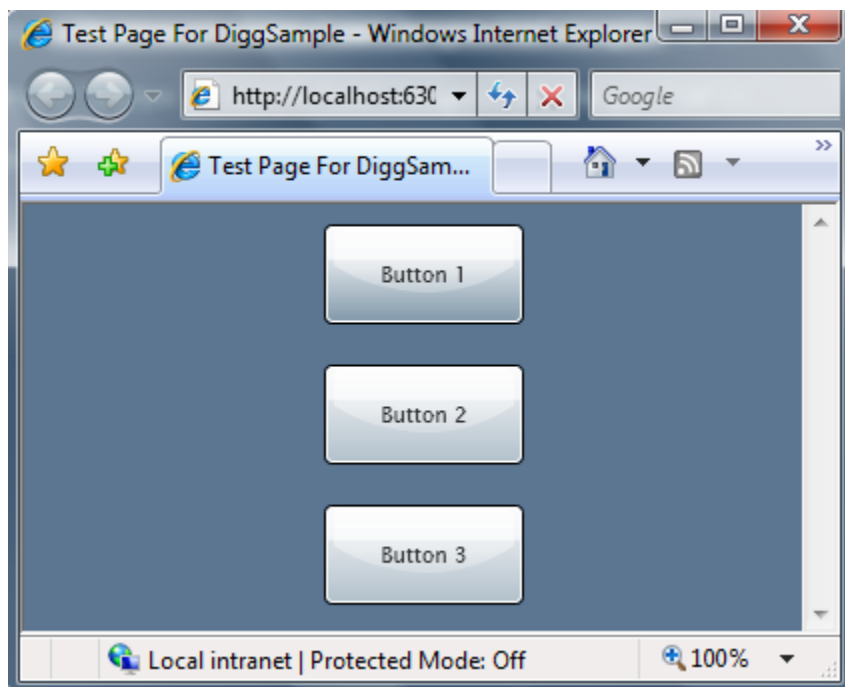
Canvas hữu dụng khi bạn không bao giờ thay đổi vị trí của các control bên trong, do vậy nó không thực sự mềm dẻo mỗi khi bạn thêm một control mới vào, hay mỗi khi bạn thay đổi vị trí hoặc kích thước, hoặc trong trường hợp bạn phải viết code để di chuyển các thành phần bên trong một Canvas. Một giải pháp tốt hơn trong những trường hợp như vậy là dùng một layout panel khác như StackPanel và Grid.

StackPanel

StackPanel là một dạng Panel đơn giản cho phép đặt các thành phần bên trong theo dòng hay cột, StackPanel hay được dùng khi bạn muốn sắp xếp chỉ trong một phần của toàn bộ trang. Lấy ví dụ, chúng ta có thể dùng StackPanel để đặt 3 nút bấm theo hàng dọc trên trang bằng cách dùng XAML như sau:

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="400" Height="350">
5
6   <StackPanel Background="#FF5C7590">
7
8       <Button Content="Button 1" Width="100" Height="50" Margin="10"/>
9
10      <Button Content="Button 2" Width="100" Height="50" Margin="10"/>
11
12      <Button Content="Button 3" Width="100" Height="50" Margin="10" />
13
14   </StackPanel>
15
16 </UserControl>
17
```

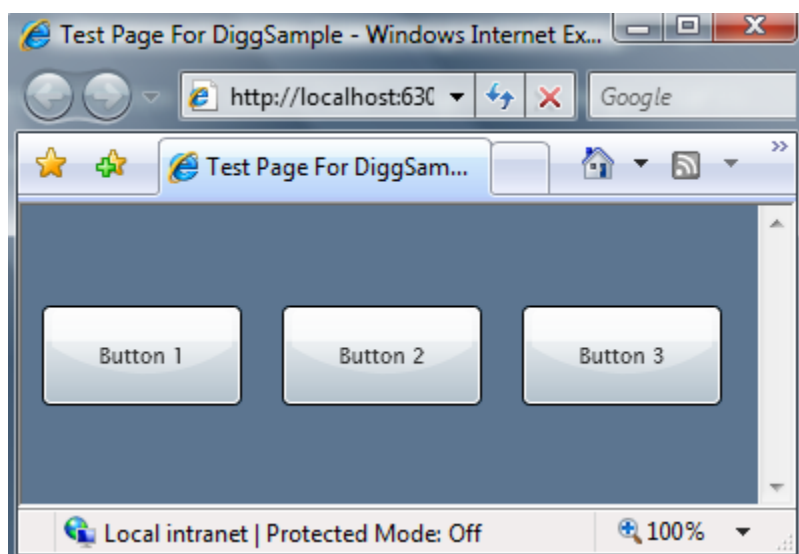
Khi chạy, StackPanel sẽ sắp xếp các nút bấm một cách tự động theo chiều dọc:



Hay chúng ta cũng có thể đặt thuộc tính "Orientation" của StackPanel thành "Horizontal" (chiều ngang) thay vì giá trị mặc nhiên là "Vertical":

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="400" Height="150">
5
6   <StackPanel Orientation="Horizontal" Background="#FF5C7590">
7
8       <Button Content="Button 1" Width="100" Height="50" Margin="10"/>
9
10      <Button Content="Button 2" Width="100" Height="50" Margin="10"/>
11
12      <Button Content="Button 3" Width="100" Height="50" Margin="10" />
13
14   </StackPanel>
15
16 </UserControl>
17
```

Điều này sẽ làm cho StackPanel tự động sắp xếp các nút bấm theo chiều ngang:

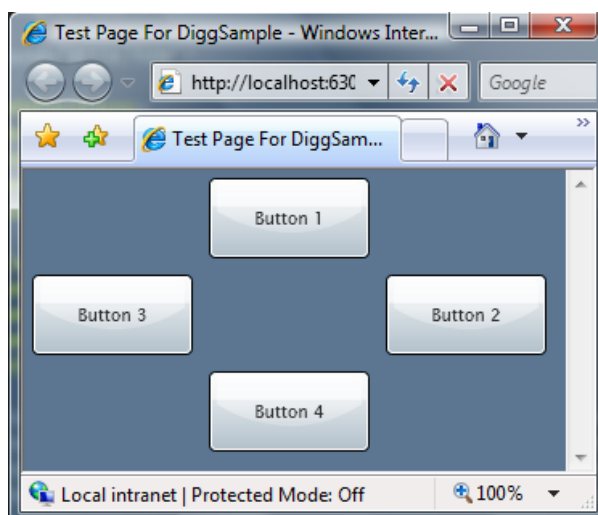


Grid Panel

Grid là layout panel mềm dẻo nhất, nó cho phép sắp xếp các control theo dạng bảng bao gồm nhiều dòng và nhiều cột. Về tính năng, nó tương tự như thẻ Table trong HTML. Không như table trong HTML, bạn sẽ không đặt các control trực tiếp trong cột hay dòng, thay vào đó bạn sẽ chỉ định vị trí dòng hoặc cột bằng cách khai báo ngay bên trong control <Grid> các thuộc tính <Grid.RowDefinitions> và <Grid.ColumnDefinitions>. Bạn có thể áp dụng cú pháp “Attached Property” của XAML lên trên các control bên trong Grid để chỉ ra nó sẽ phải nằm bên trong dòng hay cột nào. Lấy ví dụ, chúng ta có thể khai báo một Grid có 3 dòng và 3 cột, và sau đó chúng ta đặt 4 nút bấm vào bên trong dung XAML như sau:

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="400" Height="400">
5
6   <Grid Background="#FF5C7590">
7
8       <Grid.RowDefinitions>
9           <RowDefinition Height="60" />
10          <RowDefinition Height="60" />
11          <RowDefinition Height="60" />
12      </Grid.RowDefinitions>
13
14      <Grid.ColumnDefinitions>
15          <ColumnDefinition Width="110"/>
16          <ColumnDefinition Width="110"/>
17          <ColumnDefinition Width="110"/>
18      </Grid.ColumnDefinitions>
19
20      <Button Content="Button 1" Width="100" Height="50" Grid.Column="1" Grid.Row="0" />
21      <Button Content="Button 2" Width="100" Height="50" Grid.Column="2" Grid.Row="1" />
22      <Button Content="Button 3" Width="100" Height="50" Grid.Column="0" Grid.Row="1" />
23      <Button Content="Button 4" Width="100" Height="50" Grid.Column="1" Grid.Row="2" />
24
25  </Grid>
26
27 </UserControl>
28
```

Grid layout khi đó sẽ đặt vị trí các thành phần Button cho chúng ta giống như sau:

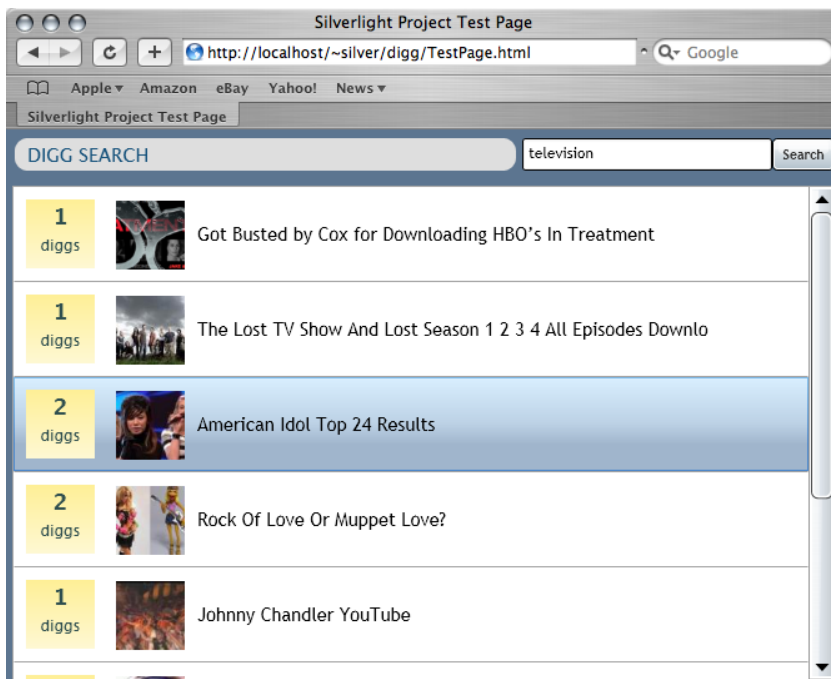


Ngoài khả năng cho phép đặt kích thước một cách tuyệt đối (ví dụ như `Height="60"`), `RowDefinition` and `ColumnDefinition` còn hỗ trợ chế độ đặt kích thước động (`Height="Auto"`), điều này cho phép đặt lại kích thước của Grid hay Row dựa trên kích thước của nội dung nó chứa bên trong (bạn có thể đặt kích thước tối đa hoặc tối thiểu), đây là một tính năng vô cùng hữu ích. Row và ColumnDefinitions của Grid cũng hỗ trợ một tính năng được gọi là “Proportional Sizing” – nó cho phép bạn đặt kích thước của

một Row hay một Column theo một tỷ lệ tương ứng so với một cái khác (ví dụ như bạn có thể đặt dòng thứ hai luôn cao gấp đôi dòng thứ nhất). Bạn sẽ thấy rằng Grid cung cấp rất nhiều sức mạnh và sự mềm dẻo – và có lẽ nó cũng là layout panel mà bạn sẽ sử dụng nhiều nhất.

Dùng các layout panel để sắp xếp các thành phần trên trang Digg của chúng ta

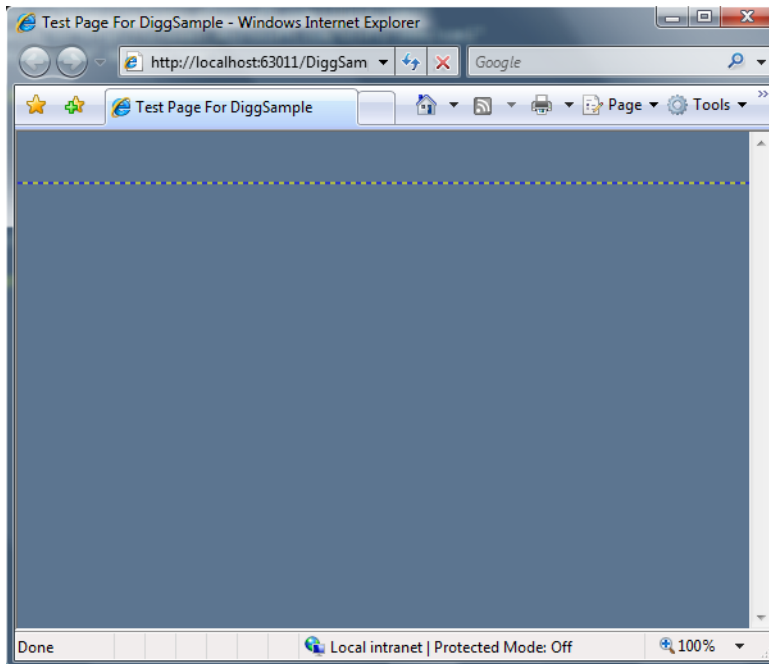
Chúng ta nhớ lại rằng mục đích khi xây dựng trang ví dụ Digg là tạo một trang trông giống như sau:



Để tạo dạng sắp xếp như vậy, bạn sẽ phải thêm một đối tượng Grid mà nó có 2 RowDefinitions bên trong. Dòng đầu tiên có chiều cao 40 pixel, dòng thứ hai chiếm toàn bộ phần còn lại (Height="*"):

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="600" Height="500">
5
6   <Grid Background="#FF5C7590" ShowGridLines="True">
7
8     <Grid.RowDefinitions>
9       <RowDefinition Height="40"/>
10      <RowDefinition Height="*/>
11    </Grid.RowDefinitions>
12
13  </Grid>
14
15 </UserControl>
16
```

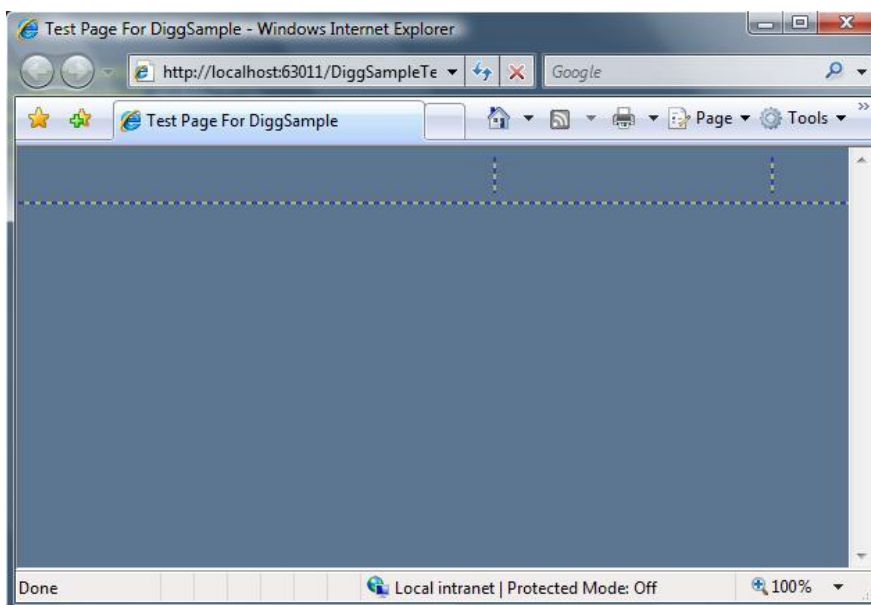
Mẹo: Bạn để ý rằng ở trên chúng ta đã đặt thuộc tính “ShowGridLines” của Grid là “True”. Điều này sẽ cho phép chúng ta dễ dàng thấy được các đường biên của dòng hoặc cột bên trong Grid khi chạy



Chúng ta sẽ nhúng một Grid khác như là một control con vào bên trong dòng đầu tiên của Grid gốc, và dùng nó để sắp xếp các control bên trong dòng đầu tiên (header). Chúng ta sẽ tạo ra 3 cột bên trong, một cho tiêu đề, một cho ô “Search” và một cho nút “Search”:

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="600" Height="500">
5
6   <Grid Background="#FF5C7590" ShowGridLines="True">
7
8     <Grid.RowDefinitions>
9       <RowDefinition Height="40"/>
10      <RowDefinition Height="*/>
11    </Grid.RowDefinitions>
12
13    <Grid Grid.Row="0" Margin="7" ShowGridLines="True">
14
15      <Grid.ColumnDefinitions>
16        <ColumnDefinition Width="*/>
17        <ColumnDefinition Width="200"/>
18        <ColumnDefinition Width="50"/>
19      </Grid.ColumnDefinitions>
20
21    </Grid>
22
23  </Grid>
24
25 </UserControl>
26
```

Một khi đã làm xong, chúng ta sẽ có được dạng sắp trang của trang Digg như sau:



***Ghi chú:** Thay vì dùng các Grid lồng nhau, chúng ta cũng có thể chỉ dùng một Grid với 3 cột 2 dòng và dùng tính năng ColSpan/RowSpan của Grid để ghép nội dung của nhiều cột làm một (giống table trong HTML). Tôi chọn cách dùng Grid lồng nhau vì tôi nghĩ nó sẽ đơn giản hơn.*

Giờ chúng ta đã sắp xếp xong, vấn đề còn lại là thêm các control vào đúng vị trí của nó. Với dòng header, chúng ta sẽ dùng control <Border> (với thuộc tính CornerRadius bằng 10 để có các góc bo tròn) và thêm một số văn bản bên trong để hiển thị tiêu đề. Chúng ta cũng sẽ dùng control <WatermarkedTextBox> trong cột thứ 2 để tạo ô tìm kiếm. Và đặt một <Button> để tạo nút “Search” trong cột thứ 3. Chúng ta cũng sẽ đặt một số văn bản trong dòng thứ 2 để sau này hiển thị kết quả tìm kiếm.

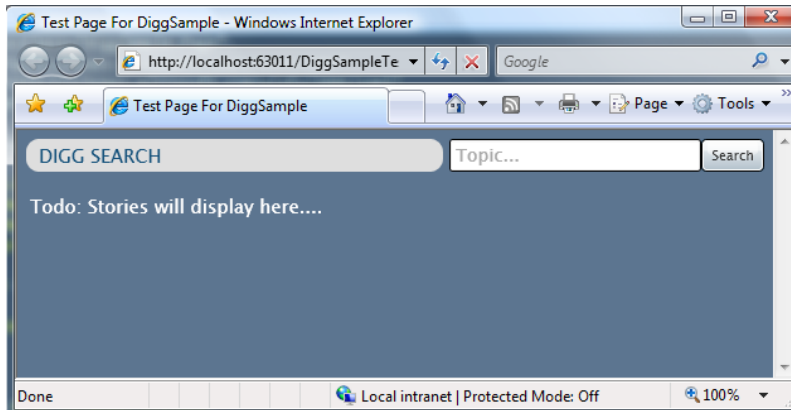
***Chi chú:** Ở sau đây tôi sẽ nhúng các thông tin định dạng (FontSize, Colors, Margins, ...) trực tiếp lên trên các điều khiển. Trong những bài sau tôi sẽ cho các bạn thấy cách dùng Styles để tách các thuộc tính trên vào một file riêng (như CSS) và cho phép dùng lại các thuộc tính này trong toàn bộ ứng dụng*

```

Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="600" Height="300">
5
6   <Grid Background="#FF5C7590">
7
8     <Grid.RowDefinitions>
9       <RowDefinition Height="40"/>
10      <RowDefinition Height="*"/>
11    </Grid.RowDefinitions>
12
13    <Grid Grid.Row="0" Margin="7">
14
15      <Grid.ColumnDefinitions>
16        <ColumnDefinition Width="*"/>
17        <ColumnDefinition Width="200"/>
18        <ColumnDefinition Width="50"/>
19      </Grid.ColumnDefinitions>
20
21      <Border Grid.Column="0" CornerRadius="10" Background="#FFDEDE" Margin="0,0,5,0">
22        <TextBlock Text="DIGG SEARCH" Foreground="#FF14517B" Margin="10,3,0,0" />
23      </Border>
24
25      <WatermarkedTextBox Grid.Column="1" FontSize="14" Watermark="Topic..." />
26
27      <Button Grid.Column="2" Content="Search" />
28    </Grid>
29
30    <TextBlock Grid.Row="1" Margin="10" Foreground="White">
31      Todo: Stories will display here....
32    </TextBlock>
33
34  </Grid>
35
36 </UserControl>
37
38

```

Và chương trình của chúng ta nếu chạy sẽ cho ra giao diện giống như sau:

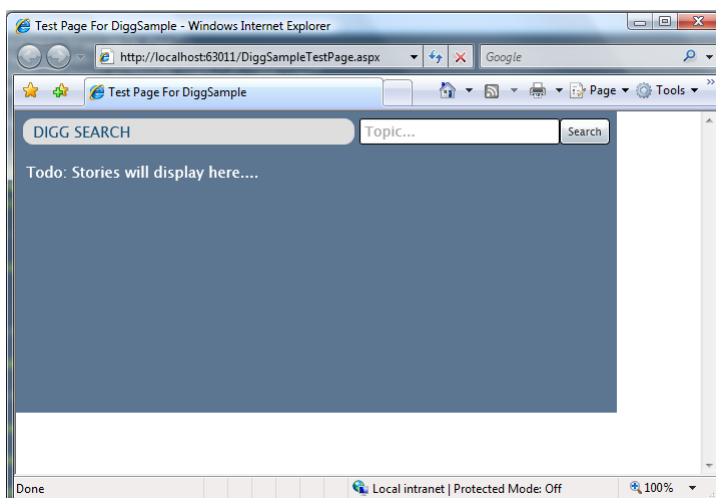


Tự động thay đổi kích thước của ứng dụng

Một điều các bạn có thể để ý thấy là trong đoạn XAML trên, control cấp cao nhất đã được đặt kích thước cố định:

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   Width="600" Height="300">
5
6   <Grid Background="#FF5C7590">
7
8       <Grid.RowDefinitions>
9           <RowDefinition Height="40"/>
10          <RowDefinition Height="*" />
11      </Grid.RowDefinitions>
12
```

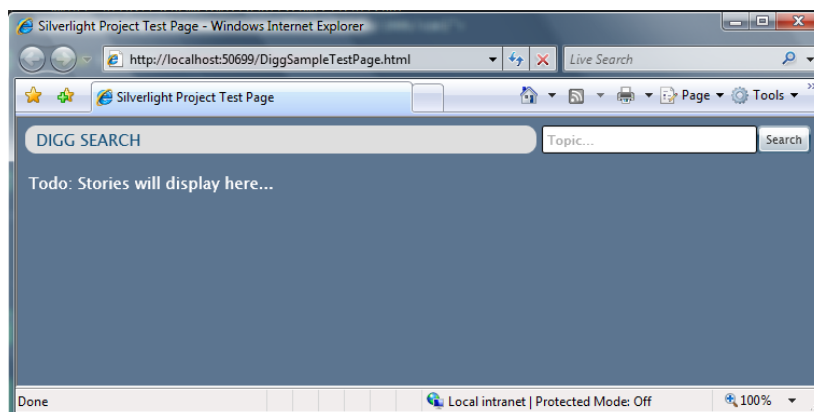
Nếu đặt theo cách này thì chương trình của chúng ta sẽ luôn có kích thước mà chúng ta đã đặt, khi đó nếu bạn mở rộng sửa sổ trình duyệt thì nó sẽ trở nên giống như sau:



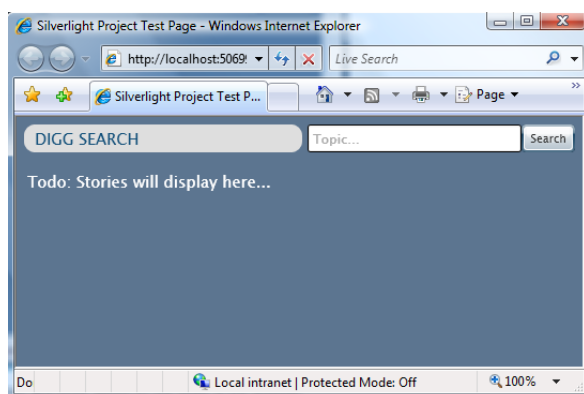
Trong khi việc đặt kích thước của một ứng dụng nhúng là cố định có thể hữu ích trong một số trường hợp, nhưng trong ứng dụng Digg của chúng ta, điều thực sự mong muốn lại là nó phải chiếm của sổ trình duyệt và thay đổi kích thước theo trình duyệt – giống như một trang HTML vậy. Một tin tốt lành là làm điều này vô cùng đơn giản. Bạn chỉ cần xóa đi hai thuộc tính Width và Height của control cấp cao nhất:

```
Page.xaml
1 <UserControl x:Class="DiggSample.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
4
```

Chương trình Silverlight của chúng ta nay sẽ tự động mở rộng (hoặc thu lại) để chiếm đầy phần trang web mà nó được nhúng bên trong. Vì file SilverlightTestPage.html mà chúng ta đang dùng để test chứa control Silverlight bên trong một thẻ với chiều rộng và chiều cao là 100%, do vậy chương trình Digg của chúng ta sẽ tự động chiếm đầy của sổ trình duyệt:



Bạn để ý sẽ thấy cách mà nội dung bên trong phần header của ứng dụng tự động thay đổi kích thước dựa trên chiều rộng của trình duyệt:



Khi thu hẹp lại cửa sổ trình duyệt, ô văn bản và nút tìm kiếm sẽ vẫn giữ nguyên kích thước, đó là vì các cột chứa nó trong Grid có kích thước cố định. Control <Border> chứa tiêu đề “Digg Search” sẽ tự động thay đổi là ví cột chứa nó đã được đặt Width=”*”. Chúng ta đã chẳng phải viết một dòng lệnh nào để thực hiện việc sắp xếp lại các thành phần, Grid và hệ quản lý layout sẽ làm việc đó cho chúng ta.

Bước tiếp theo

Giờ đây chúng ta đã có một cấu trúc trang cơ bản cho chương trình Digg, và cũng đã có một dòng header. Bước tiếp theo sẽ là thêm khả năng tìm kiếm vào chương trình – và làm cho nó có thể lấy được dữ liệu thực sự từ trang Digg.com mỗi khi người dùng thực hiện việc tìm kiếm trên một chủ đề nào đó. Để làm điều này, xin mời bạn đọc phần kế tiếp: Kết nối mạng để lấy dữ liệu vào một DataGrid.

Bài 3: Kết nối mạng để lấy dữ liệu vào một DataGrid.

Đây là phần ba của loạt 8 bài hướng dẫn cách xây dựng một chương trình Digg đơn giản sử dụng bản Beta1 của Silverlight 2. Các bài hướng dẫn này nên được đọc theo thứ tự, và giúp giải thích một số khái niệm nền tảng của Silverlight.

Dùng mạng để lấy dữ liệu từ Digg

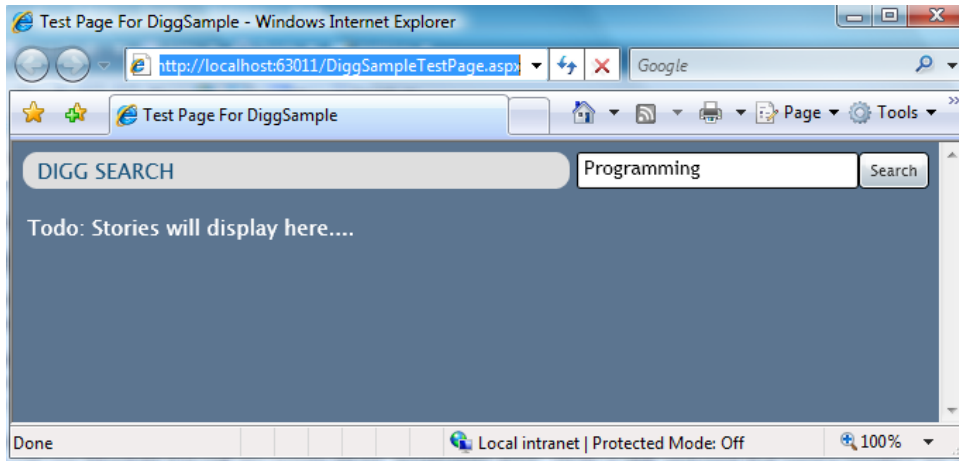
Silverlight cung cấp một tập API cho phép các chương trình có thể gọi các dịch vụ từ xa như REST, SOAP/WS*, RSS, JSON và XML HTTP. Silverlight 2 cũng bao gồm một tập API dùng cho socket (System.Net.Sockets) cho phép giao tiếp bằng các giao thức không dựa trên HTTP (chẳng hạn như các chat server...).

Truy cập vào các domain khác

Các chương trình Silverlight 2 luôn có thể truy cập vào máy chủ mà nó đã được tải xuống. Silverlight 2 cũng có thể truy cập vào các máy chủ ở tên miền khác nếu (cross-domain) trên máy chủ đó có chứa một file (dạng XML) chỉ ra các máy trạm được phép tạo ra các lời gọi cross-domain đến nó. Silverlight 2 định nghĩa một định dạng file chứa các policy cho phép người quản trị máy chủ có thể kiểm soát được các lời gọi đến nó. Và định dạng mà nó dùng cũng chính là định dạng mà Flash dùng, do vậy bạn hoàn toàn có thể dùng Silverlight 2 để gọi đến các dịch vụ REST, SOAP/WS*, RSS, JSON hay XML sẵn có trên web nếu nó đã được cấu hình cho phép các chương trình Flash truy cập vào. Digg.com cung cấp một tập hàm API khá hay mà bạn có thể truy cập được thông qua HTTP. Bởi vì họ có một file cross-domain policy đặt trên server, do vậy có thể gọi đến chúng thông qua chương trình Digg của chúng ta (Bạn có thể xem file này tại đây: <http://digg.com/crossdomain.xml>), và nó không bắt buộc chúng ta phải gọi ngược lên server để truy cập vào các chức năng này.

Tập hàm API của Digg.com

Chúng ta cho phép người dùng gõ vào một chủ đề nào đó để tìm kiếm (ví dụ: “Programming”) và nhấn nút “Search” để lấy về nội dung khớp với từ khóa đó từ Digg.com.



Chúng ta có thể dùng tập hàm API [“List Stories REST API”](#) của Digg.com để làm điều này, nó nhận vào một tham số là chủ đề cần tìm trên URL (ví dụ: GET /stories/topic/programming), và trả ngược lại kết quả dạng XML của chủ đề được tìm kiếm. Bạn có thể nhấn [vào đây](#) để xem kết quả của phép tìm kiếm này.

Dùng System.Net.WebClient để thực hiện lời gọi không đồng bộ đến Digg REST Feed

Khi nút Search ở trên được nhấn, chúng ta sẽ bắt được sự kiện “Click”, lấy chuỗi chủ đề tìm kiếm trong ô văn bản có kiểu WaterMarkTextBox, khởi tạo lời gọi qua mạng đến Digg để nhận về chuỗi XML chứa kết quả tìm kiếm cho chủ đề trên.

Silverlight cũng bao gồm lớp WebClient bên trong namespace System.Net (giống cái có trong bộ .NET Framework đầy đủ). Chúng ta có thể dùng lớp này để tải về kết quả một cách không đồng bộ từ một URL. Ưu điểm của việc tải về không đồng bộ là chương trình chúng ta sẽ không bị khóa hay không phản hồi khi chúng đang chờ dữ liệu về từ máy chủ.

Tất cả những gì cần làm để thực hiện một lời gọi không đồng bộ với lớp WebClient là đăng ký một hàm xử lý sự kiện “DownloadStringCompleted” (hàm này sẽ được gọi khi nội dung yêu cầu đã được tải về), và gọi hàm WebClient.DownloadStringAsync(url) để bắt đầu việc download:

```

void SearchBtn_Click(object sender, RoutedEventArgs e)
{
    // Retrieve Topic to Search for from WaterMarkTextBox
    string topic = txtSearchTopic.Text;

    // Construct Digg REST URL
    string diggUrl = String.Format("http://services.digg.com/stories/topic/{0}?count=20&apikey=http%3A%2F%2Fscottgu.com", topic);

    // Initiate Async Network call to Digg
    WebClient diggService = new WebClient();
    diggService.DownloadStringCompleted += new DownloadStringCompletedEventHandler(DiggService_DownloadStoriesCompleted);
    diggService.DownloadStringAsync(new Uri(diggUrl));
}

void DiggService_DownloadStoriesCompleted(object sender, DownloadStringCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string result = e.Result;

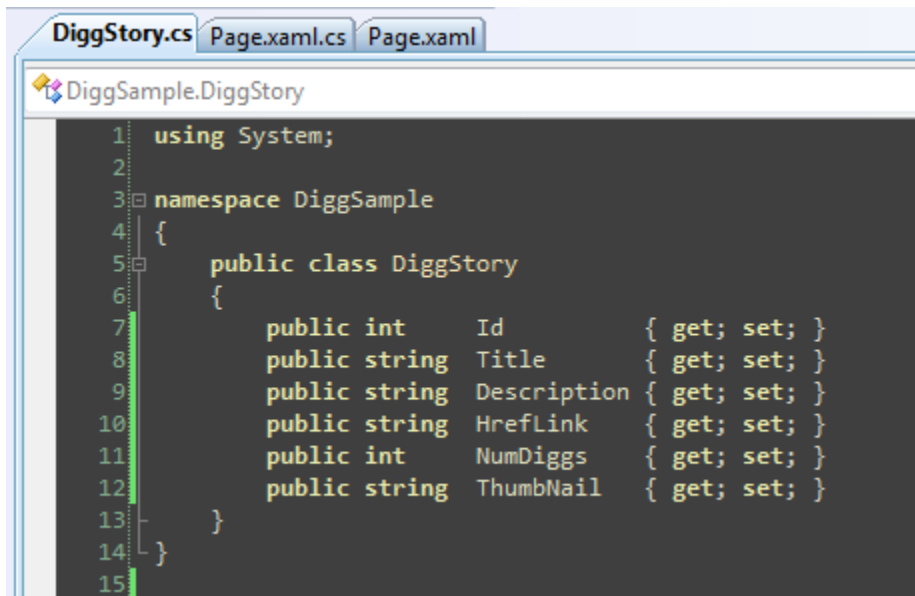
        // Todo: Do Something with the XML
    }
}

```

Với đoạn mã ở trên chúng ta có thể lấy về một chuỗi dạng XML chứa kết quả tìm kiếm của Digg.com.

Dùng LINQ để phân tích chuỗi XML trả về bởi Digg.com để đưa vào lớp Story

Bây giờ chúng ta đã có thể lấy về chuỗi XML kết quả, bước tiếp theo là phân tích và chuyển nó thành các đối tượng “DiggStory” để có thể được xử lý và gắn nối vào các control của chúng ta. Bước đầu tiên để làm điều này là định nghĩa một lớp “DiggStory” có chứa các thuộc tính khớp với nội dung của chuỗi XML trả về bởi Digg (chúng ta sẽ tận dụng ưu điểm của một tính năng mới trong C# là “automatic properties” để làm điều này).

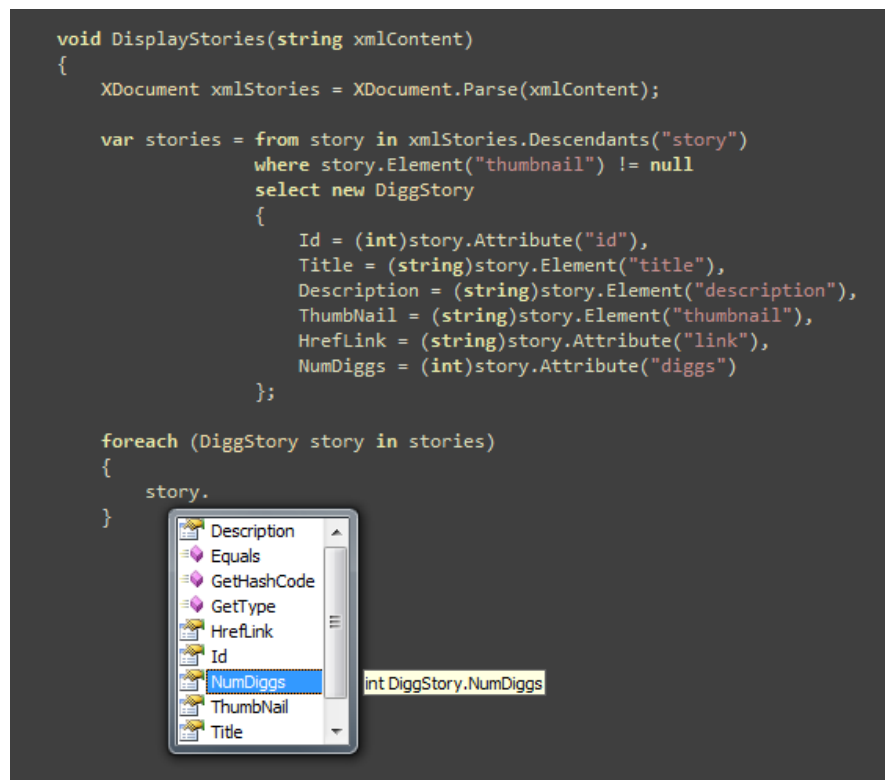


```

DiggStory.cs Page.xaml.cs Page.xaml
DiggSample.DiggStory
1 using System;
2
3 namespace DiggSample
4 {
5     public class DiggStory
6     {
7         public int Id { get; set; }
8         public string Title { get; set; }
9         public string Description { get; set; }
10        public string HrefLink { get; set; }
11        public int NumDiggs { get; set; }
12        public string ThumbNail { get; set; }
13    }
14 }
15

```

Sau đó có thể dùng LINQ (có sẵn trong Silverlight 2) và LINQ to XML (là một thư viện mở rộng mà chúng ta có thể gắn thêm vào chương trình Silverlight của chúng ta) để dễ dàng phân tích và lọc tài liệu XML được trả về từ Digg, và dịch nó thành một chuỗi các đối tượng DiggStory dùng đoạn mã dưới đây:



Chú ý rằng ở trên chúng ta đã có các đối tượng DiggStory thực sự.

Hiển thị dữ liệu của Digg trong một DataGrid

Chúng ta sẽ dùng một control mới trong Silverlight là DataGrid để hiển thị dữ liệu Digg. Để cho phép điều này, chúng ta sẽ phải tham chiếu đến assembly chứa Silverlight Data control, và thay thế đoạn “Todo” trước đây với một khai báo DataGrid:

```

<Data:DataGrid x:Name="StoriesList" Grid.Row="1" Margin="5" AutoGenerateColumns="True">
</Data:DataGrid>

```

DataGrid cho phép bạn khai báo cụ thể các cột cùng với kiểu hiển thị của nó (để có thể kiểm soát một cách tối đa). Cách khác, bạn có thể đặt thuộc tính “AutoGenerateColumns” bằng True để yêu cầu DataGrid tự động tạo ra các cột dựa trên cấu trúc của các đối tượng của bạn. Chúng ta cũng sẽ cập nhật lại lớp code-behind để gắn thuộc tính ItemSource của DataGrid và chuỗi các đối tượng mà chúng ta đã lấy được khi gọi hàm của Digg mỗi khi nút Search được nhấn.

```

void DiggService_DownloadStoriesCompleted(object sender, DownloadStringCompletedEventArgs e)
{
    if (e.Error == null)
    {
        DisplayStories(e.Result);
    }
}

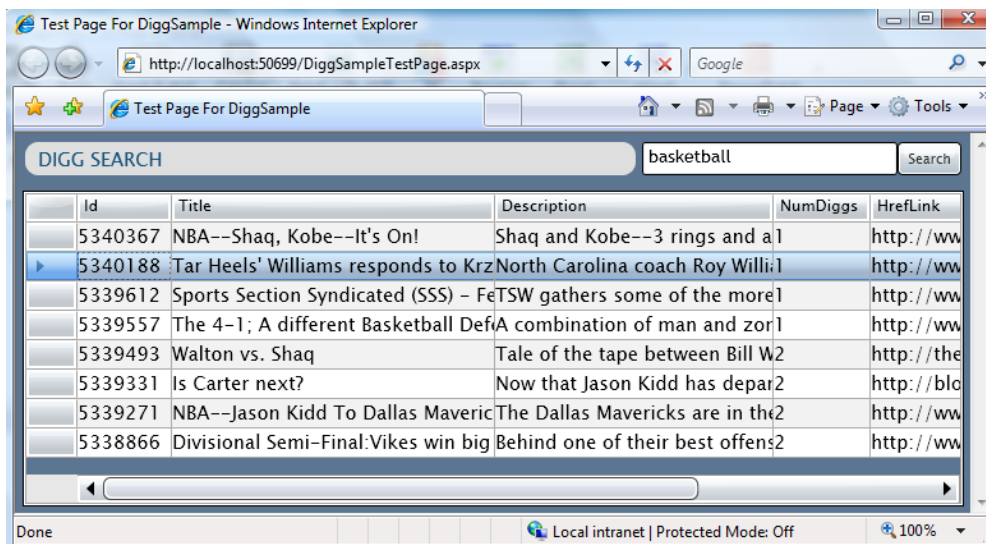
void DisplayStories(string xmlContent)
{
    XDocument xmlStories = XDocument.Parse(xmlContent);

    var stories = from story in xmlStories.Descendants("story")
                  where story.Element("thumbnail") != null
                  select new DiggStory
                  {
                      Id = (int)story.Attribute("id"),
                      Title = (string)story.Element("title"),
                      Description = (string)story.Element("description"),
                      ThumbNail = (string)story.Element("thumbnail"),
                      HrefLink = (string)story.Attribute("link"),
                      NumDiggs = (int)story.Attribute("diggs")
                  };

    StoriesList.ItemsSource = stories;
}

```

Bây giờ, mỗi khi chúng ta chạy chương trình Silverlight và thực hiện việc tìm kiếm, chúng ta sẽ thấy một danh sách các chủ đề lấy về từ Digg:



DataGrid trong Silverlight hỗ trợ tất cả các tính năng bạn mong muốn với một control chạy trên máy trạm: cho phép sửa chữa dữ liệu 2 chiều, chọn, cuộn, thay đổi chiều rộng cột... Nó cũng hỗ trợ auto-flow layout, cho phép tự động mở rộng hay thu hẹp để vừa với đối tượng chứa nó. DataGrid cũng hỗ trợ các mẫu cho phép tùy biến cả về cách định dạng cũng như cách chỉnh sửa dữ liệu. Tôi sẽ viết thêm các bài viết khác về cách dùng DataGrid.

Bước tiếp theo

Giờ chúng ta đã có thể lấy dữ liệu từ Digg.com và hiển thị nó lên cửa sổ ứng dụng. Bước kế tiếp chúng ta sẽ quay trở lại trang Page.xaml và bỏ đi các khai báo định dạng trực tiếp mà chúng ta đang dùng.

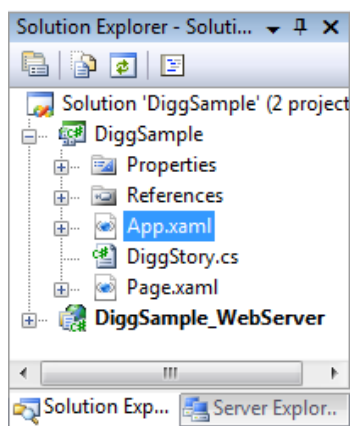
Để làm điều này, xin mời bạn đọc bài tiếp theo: Xây dựng giao diện dùng Style.

Bài 4: Xây dựng giao diện dùng Style

WPF và Silverlight hỗ trợ một cơ chế cho phép lưu giá trị thuộc tính của các control theo cách có thể dùng lại. Chúng ta có thể lưu giữ những khai báo đó trong các file riêng biệt, và dùng lại nó cho nhiều control và trang khác nhau trong ứng dụng, hoặc thậm chí có thể dùng nó trong những ứng dụng khác nhau. Khái niệm này tương tự như CSS khi bạn làm việc với HTML để thực hiện các phép tùy biến cơ bản.

Ngoài khả năng định nghĩa các cài đặt cho các thuộc tính cơ bản (như Color, Font, Size, Margins...), các kiểu định dạng trong WPF và Silverlight cũng có thể được dùng để định nghĩa và dùng lại các Control Template, cho phép bạn có thể tạo ra các control có hình thức và cấu trúc được tùy biến một cách tối đa (và hỗ trợ nhiều cách định dạng tiên tiến hơn so với CSS). Tôi sẽ nói thêm về Control Templates trong phần 7 của loạt bài này.

Đối với ứng dụng mẫu Digg, chúng ta sẽ định nghĩa các khai báo Style bên trong file App.xaml. Điều này cho phép chúng ta có thể dùng lại trong suốt toàn bộ các trang và control trong toàn ứng dụng.



Hãy bắt đầu bằng việc tạo dựng các kiểu định dạng cho control <Border> (và cả <TextBlock> chứa bên trong nó):

```
<Border Grid.Column="0" CornerRadius="10" Background="#FFDEDE" Margin="0,0,5,0">
  <TextBlock Text="DIGG SEARCH" Foreground="#FF14517B" Margin="10,3,0,0" />
</Border>
```



Chúng ta cũng có thể tạo hai thành phần Style bên trong file App.xaml để lưu giữ các cài đặt cho <Border> và <TextBlock> đã được khai báo trước đó:

```

App.xaml | Page.xaml.cs | Page.xaml
1 <Application xmlns="http://schemas.microsoft.com/client/2007"
2   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
3   xmlns:Data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
4   x:Class="DiggSample.App">
5
6   <Application.Resources>
7
8     <Style x:Key="TitleBorder" TargetType="Border">
9       <Setter Property="CornerRadius" Value="10"/>
10      <Setter Property="Background" Value="#FFDEDEDE"/>
11      <Setter Property="Margin" Value="0,0,5,0"/>
12      <Setter Property="Grid.Column" Value="0"/>
13    </Style>
14
15    <Style x:Key="TitleText" TargetType="TextBlock">
16      <Setter Property="Foreground" Value="#FF14517B"/>
17      <Setter Property="Margin" Value="10,3,0,0"/>
18    </Style>
19
20  </Application.Resources>
21
22 </Application>
23

```

Chú ý rằng chúng ta đã đặt cho mỗi Style một giá trị duy nhất cho “Key”. Chúng ta có thể cập nhật lại `<Border>` và `<TextBlock>` để tham chiếu đến các Style bằng cách dùng các Key. Chúng ta sẽ dùng một đặt tính trong XAML được gọi là “markup extensions” để làm điều này. “Markup extensions” được dùng khi có các giá trị không là literal (literal là các giá trị được đưa vào cụ thể như 1, 100, “abc” – khác với các giá trị chỉ biết được khi chạy chương trình như các giá trị được lưu trong các biến, kết quả trả về của hàm, hoặc dùng các biểu thức gắn nối dữ liệu...)

```

<Border Style="{StaticResource TitleBorder}"
  <TextBlock Text="DIGG SEARCH" Style="{StaticResource TitleText}" />
</Border>

```

Khi chúng ta cập nhật các điều khiển khác bên trong trang Page.xaml để dùng các style, chúng ta sẽ có một file trông như sau:

```

App.xaml Page.xaml.cs Page.xaml
1 <UserControl x:Class="Diggsample.Page"
2   xmlns:Data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
3   xmlns="http://schemas.microsoft.com/client/2007"
4   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
5
6   <Grid Style="{StaticResource TopGrid}">
7
8     <Grid.RowDefinitions>
9       <RowDefinition Height="40"/>
10      <RowDefinition Height="*" />
11    </Grid.RowDefinitions>
12
13    <Grid Style="{StaticResource Header}">
14
15      <Grid.ColumnDefinitions>
16        <ColumnDefinition Width="*" />
17        <ColumnDefinition Width="200"/>
18        <ColumnDefinition Width="50"/>
19      </Grid.ColumnDefinitions>
20
21      <Border Style="{StaticResource TitleBorder}">
22        <TextBlock Text="DIGG SEARCH" Style="{StaticResource TitleText}" />
23      </Border>
24
25      <TextBox x:Name="txtSearchTopic" Style="{StaticResource SearchBox}" />
26
27      <Button x:Name="btnSearch"
28        Content="Search"
29        Click="btnSearch_Click"
30        Style="{StaticResource SearchButton}" />
31
32    </Grid>
33
34    <Data:DataGrid x:Name="StoriesList" Style="{StaticResource StoriesList}" />
35
36  </Grid>
37
38 </UserControl>
39

```

Lưu giữ các cài đặt định dạng theo cách này cho phép người phát triển có thể tập trung hơn vào việc xây dựng các chức năng của chương trình, và cũng giúp chúng ta có thể dùng lại các style trong toàn bộ control hoặc các trang.

Ghi chú: Một điều bạn cần chú ý với bản Beta1 là báo cáo lỗi khi bạn gõ vào tên kiểu định dạng hay khai báo thuộc tính sai là không rõ ràng (nó sinh ra một exception nhưng lại không báo cho bạn biết bạn sai ở chỗ nào). Vấn đề này sẽ được sửa lại trong bản Beta2, hiện tại thì bạn nên chú ý kỹ vào những gì gõ vào nếu thấy một thông báo lỗi nạp một kiểu định dạng.

Bước tiếp theo

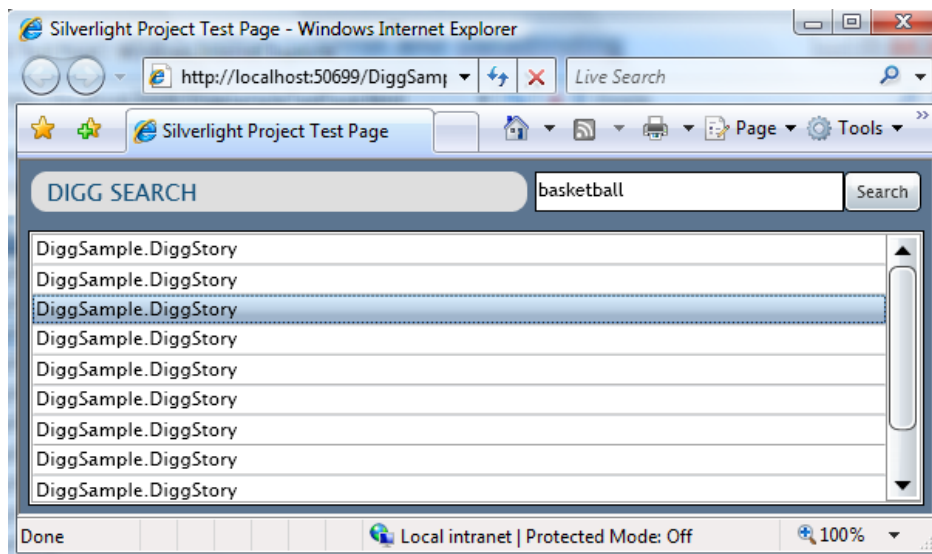
Bạn đã loại bỏ hoàn toàn các thẻ định dạng trong trang Page.xaml và dùng Style, giờ hãy đến bước tiếp theo để tùy chỉnh thêm hình thức của dữ liệu Story. Để làm điều này, hãy đọc tiếp bài: Dùng ListBox và DataBinding để hiển thị dữ liệu.

Bài 5: Dùng ListBox và DataBinding để hiển thị dữ liệu

Trong những bài trước chúng ta đã dùng DataGrid để hiển thị kết quả tìm kiếm. DataGrid có thể chạy tốt khi chúng ta muốn hiển thị nội dung theo dạng một cột. Tuy nhiên đối với ứng dụng Digg, chúng ta muốn chỉnh sửa thêm giao diện một chút để nó trông giống một List hơn là một DataGrid. Một tin tốt lành là điều này tương đối dễ dàng – và nó không đòi hỏi chúng ta phải thay đổi bất kỳ đoạn lệnh nào. Chúng ta sẽ bắt đầu bằng việc thay thế DataGrid bằng một <ListBox>. Chúng ta sẽ vẫn giữ lại tên cũ như trước đây (StoriesList):

```
<ListBox x:Name="StoriesList" Grid.Row="1">
</ListBox>
```

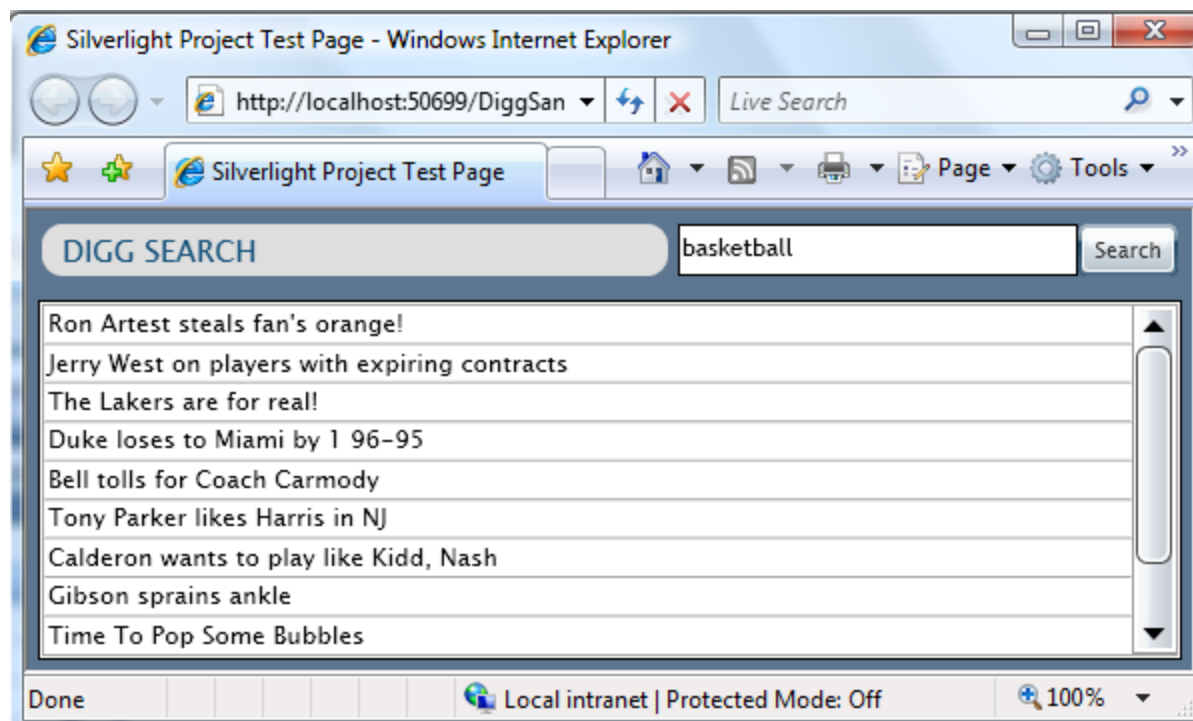
Khi chạy và thực hiện việc tìm kiếm, ListBox vừa thêm vào sẽ hiển thị kết quả như sau:



Bạn có thể sẽ tự hỏi – tại sao các dòng đều là “DiggSample.DiggStory”? Lý do là vì chúng ta đã gắn các đối tượng DiggStory vào ListBox (và mặc nhiên nó sẽ gọi các hàm ToString() trên các đối tượng này). Nếu chúng ta muốn hiển thị giá trị của thuộc tính Title của đối tượng DiggStory, chúng ta có thể đặt giá trị cho thuộc tính DisplayMemberPath của ListBox:

```
<ListBox x:Name="StoriesList" DisplayMemberPath="Title" Grid.Row="1" Margin="5">
</ListBox>
```

Khi chạy bạn sẽ thấy Title sẽ được hiển thị lên TextBox:

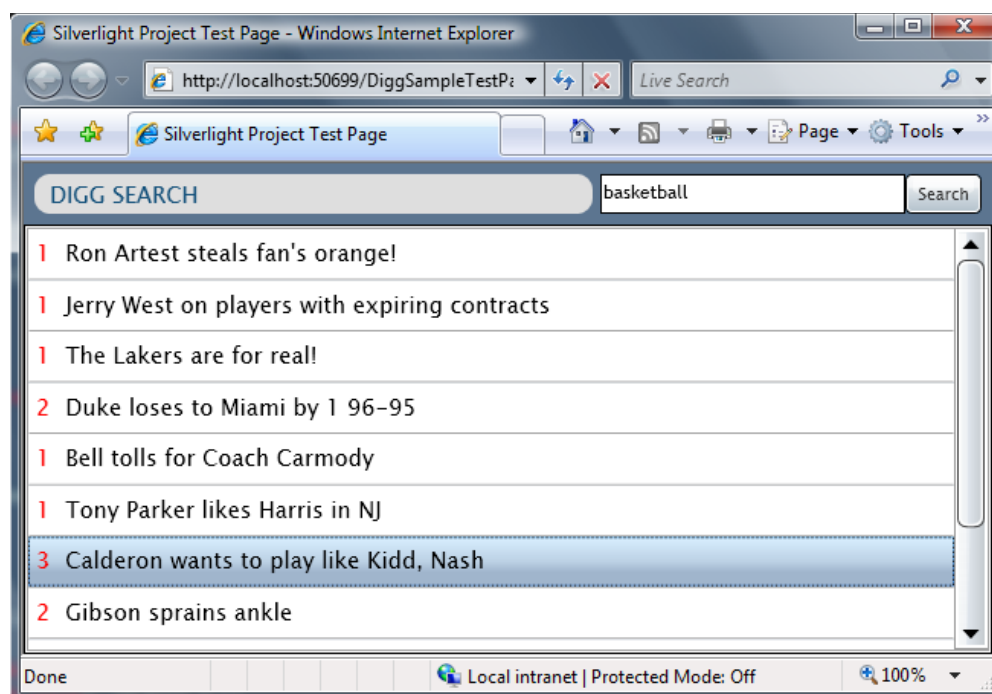


Nếu chúng ta muốn hiển thị nhiều hơn một giá trị, hay tùy biến lại dạng hiển thị của mỗi mục dữ liệu, chúng ta có thể viết lại ItemTemplate của ListBox để cung cấp một DataTemplate. Bên trong DataTemplate này, chúng ta sẽ tùy biến lại cách các đối tượng DiggStory hiển thị dữ liệu. Lấy ví dụ, chúng ta có thể hiển thị cả Title và NumDiggs bằng cách dùng DataTemplate giống như dưới đây:

```
<ListBox x:Name="StoriesList" Grid.Row="1" Margin="5">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding NumDiggs}" Margin="5" Foreground="Red" />
        <TextBlock Text="{Binding Title}" Margin="5"/>
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

Chúng ta có thể gắn bất kỳ thuộc tính public nào chúng ta muốn từ đối tượng DiggStory bên trong DataTemplate. Hãy để ý cách chúng ta dùng cú pháp {BindingPropertyName} để thực hiện điều này với hai control TextBlock.

Với DataTemplate được khai báo như trên, giờ ListBox của chúng ta sẽ hiển thị các thành phần như dưới đây:



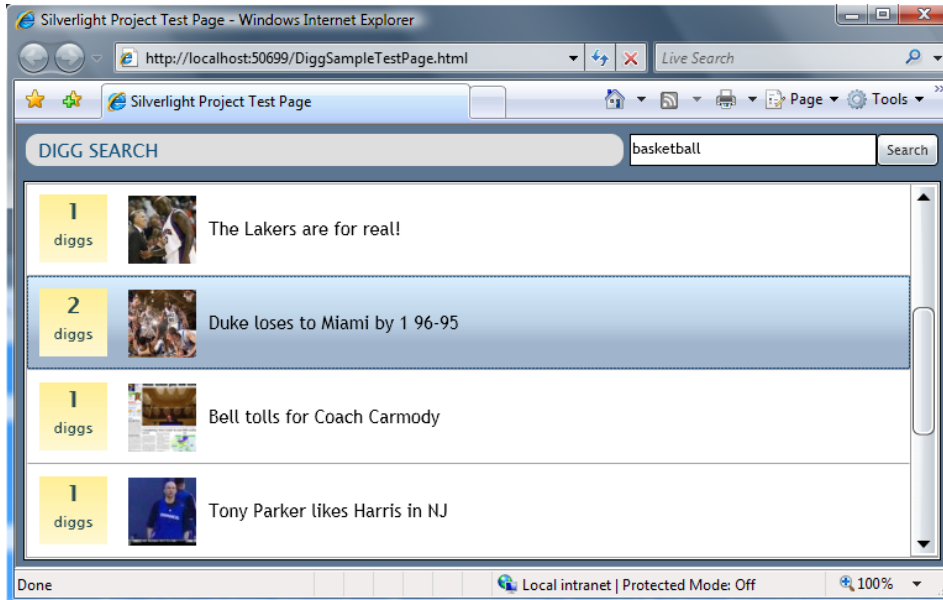
Thêm một bước nữa, chúng ta sẽ thay đổi DataTemplate như dưới đây. DataTemplate này dùng hai StackPanel – một để sắp các dòng theo chiều ngang, và một để sắp các khối văn bản theo chiều dọc.

```

<ListBox x:Name="StoriesList" Style="{StaticResource StoriesList}">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <!-- Yellow Digg Panel with NumDiggs-->
        <StackPanel Style="{StaticResource DiggPanel}" >
          <TextBlock Text="{Binding NumDiggs}" Style="{StaticResource NumDiggsBlock}" />
          <TextBlock Text="diggs" Style="{StaticResource NumDiggsSubBlock}" />
        </StackPanel>
        <!-- Story Thumbnail Preview -->
        <Image Source="{Binding Thumbnail}" Style="{StaticResource ThumbnailPreview}" />
        <!-- Story Title-->
        <TextBlock Text="{Binding Title}" Margin="5" Style="{StaticResource TitleBlock}"/>
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>

```

DataTemplate ở trên sẽ làm ListBox của chúng ta hiển thị các mục giống màn hình dưới đây:



Khi định nghĩa thêm các quy tắc Style sau đây trong file App.xaml (nhớ rằng chúng ta đang dùng LinearGradientBrush để tạo nền đồ màu vàng trên DiggPanel):

```
<Style x:Key="DiggPanel" TargetType="StackPanel">
  <Setter Property="Margin" Value="10"/>
  <Setter Property="Width" Value="55"/>
  <Setter Property="Height" Value="55"/>
  <Setter Property="Background">
    <Setter.Value>
      <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
        <GradientStop Color="#FFFFFF098"/>
        <GradientStop Color="#FFFFFF9D4" Offset="1"/>
      </LinearGradientBrush>
    </Setter.Value>
  </Setter>
</Style>

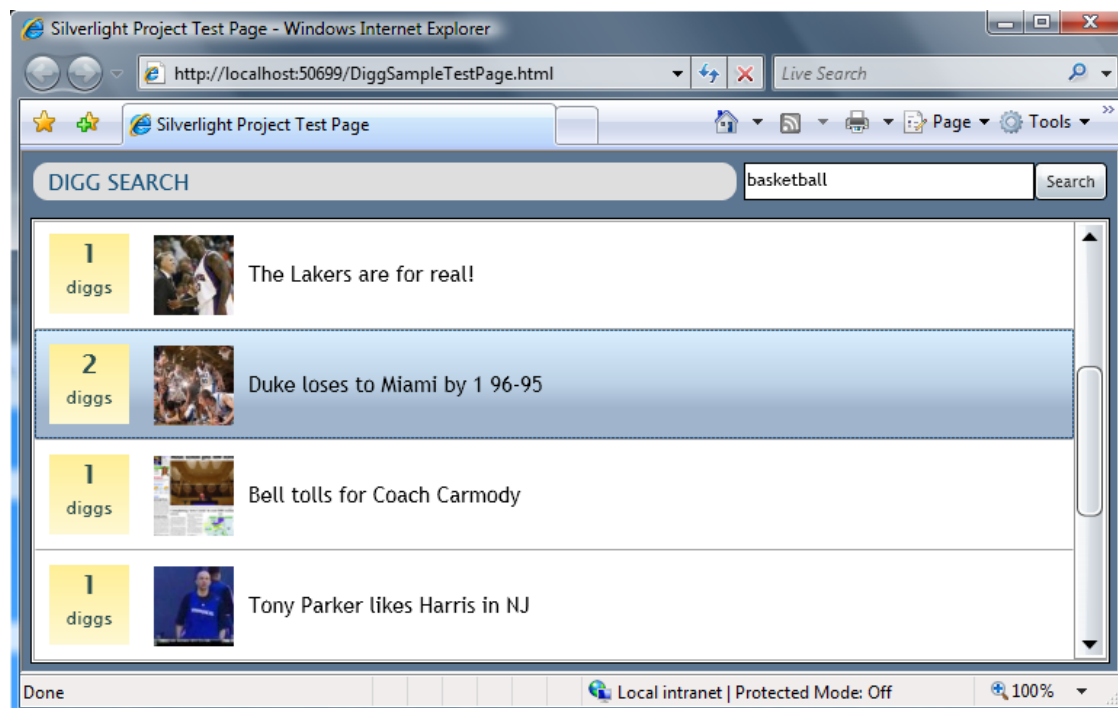
<Style x:Key="NumDigsBlock" TargetType="TextBlock">
  <Setter Property="HorizontalAlignment" Value="Center"/>
  <Setter Property="FontSize" Value="18"/>
  <Setter Property="FontWeight" Value="Bold"/>
  <Setter Property="Foreground" Value="DarkSlateGray"/>
</Style>

<Style x:Key="NumDigsSubBlock" TargetType="TextBlock">
  <Setter Property="HorizontalAlignment" Value="Center"/>
  <Setter Property="FontSize" Value="12"/>
  <Setter Property="Foreground" Value="DarkSlateGray"/>
</Style>

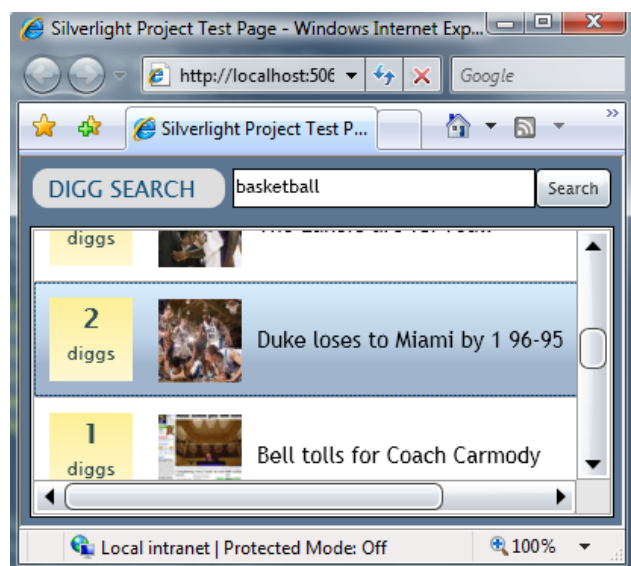
<Style x:Key="ThumbNailPreview" TargetType="Image">
  <Setter Property="Margin" Value="7,7,5,5"/>
  <Setter Property="Height" Value="55"/>
</Style>

<Style x:Key="TitleBlock" TargetType="TextBlock">
  <Setter Property="FontFamily" Value="Trebuchet MS"/>
  <Setter Property="TextAlignment" Value="Left"/>
  <Setter Property="VerticalAlignment" Value="Center"/>
</Style>
```

Một điều quan trọng cần phải nhắc về ListBox là – dù bạn có tùy biến các mục dữ liệu như thế nào đi nữa, nó vẫn cung cấp các chức năng hover và chọn mục dữ liệu một cách tự động. Điều này đúng cho cả việc sử dụng chuột lẫn bàn phím (các phím up/down, home/end...):



ListBox cũng hỗ trợ thay đổi kích thước các dòng – và sẽ tự động cuộn nội dung của chúng ta nếu cần (nhớ rằng thanh cuộn ngang sẽ xuất hiện nếu chiều rộng của sổ nhỏ hơn chiều rộng nội dung)



Bước tiếp theo

Chúng ta đã chuyển sang dùng List, và đã dọn dẹp nội dung hiển thị bên trong. Giờ hãy cố gắng hoàn thiện các chức năng của chương trình thêm một chút, và xây dựng khả năng hiển thị theo kiểu

master/detail cho phép người dùng xem ngay một câu chuyện ngay khi họ chọn nó trong danh sách. Để làm điều đó, xin mời xem phần tiếp theo: Dùng User Control để cho phép xem theo dạng Master/Detail.

Bài 6: Dùng User Control để cho phép xem theo dạng Master/Detail

Đây là phần 6 trong loạt 8 bài hướng dẫn các bước để xây dựng một chương trình Digg đơn giản dùng bản Beta 1 của Silverlight 2. Các bài hướng dẫn này nên được đọc theo thứ tự, và sẽ giúp giải thích một số khái niệm cơ bản trong Silverlight.

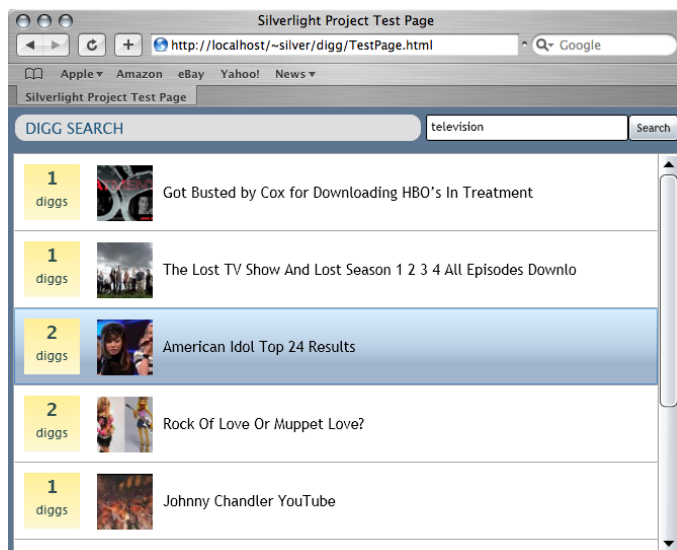
Bạn có thể download mã nguồn phiên bản hoàn chỉnh của chương trình mẫu Digg tại đây: <http://www.scottgu.com/blogposts/slbeta1apps/diggssample.zip>.

User Control là gì ?

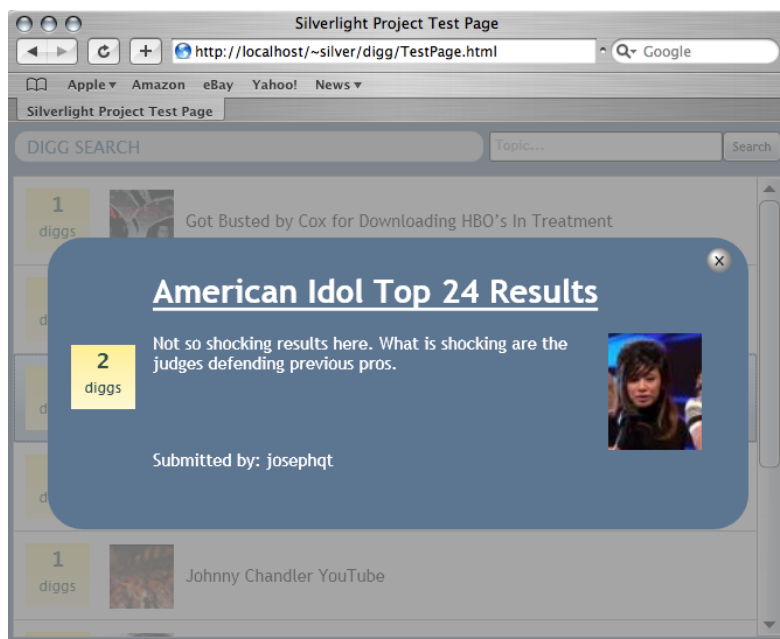
Một trong những mục tiêu thiết kế Silverlight và WPF là cho phép các nhà phát triển có thể dễ dàng đóng gói các thành phần giao diện thành những control có thể dùng lại được. Các nhà phát triển có thể tạo ra các control mới bằng cách tạo một lớp thừa kế từ một lớp Control đã có (có thể là chính bản thân lớp Control, cũng có thể là một lớp nào đó thừa kế từ Control, chẳng hạn như TextBox, Button...).

Hoặc bạn cũng có thể tạo một lớp User Control một cách dễ dàng từ các file XAML để tạo ra giao diện cho một control, và các lớp này cũng có thể dễ dàng dùng lại.

Đối với Digg, chúng ta muốn tạo ra một ứng dụng hoạt động theo kiểu master/details, nó sẽ cho phép người dùng tìm dữ liệu về một chủ đề nào đó rồi đưa vào danh sách, và cho phép người dùng chọn một kết quả để xem thông tin chi tiết. Chẳng hạn, nếu chọn một kết quả trong danh sách:



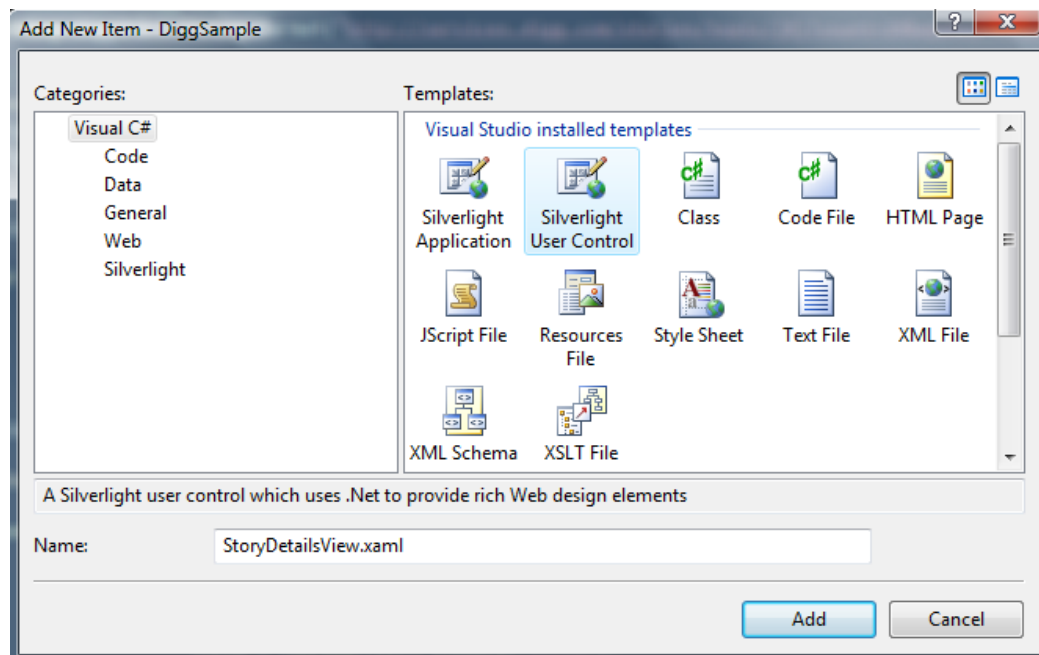
sẽ xuất hiện các thông tin chi tiết như sau:



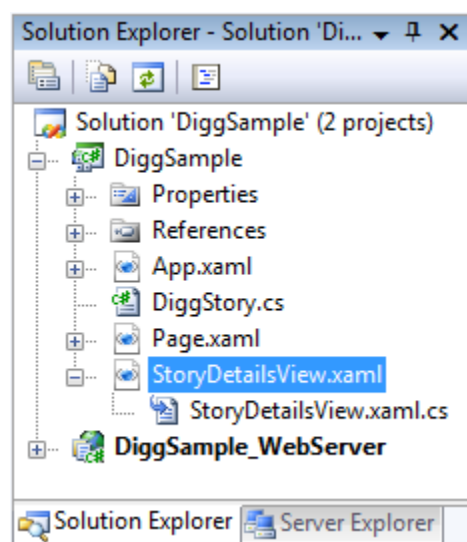
Chúng ta dự định sẽ xây dựng bảng thông tin chi tiết này bằng cách tạo ra một đối tượng UserControl có tên “StoryDetailsView”.

Tạo User Control “StoryDetailsView”

Chúng ta sẽ bắt đầu bằng cách nhấn chuột phải lên trên tên dự án DiggSample trong Visual Studio và nhấn chọn “Add new item”. Trong cửa sổ mới xuất hiện, bạn chọn UserControl và đặt tên cho control muốn tạo là “StoryDetailsView”:



Bạn sẽ thấy một UserControl với tên đã chọn được thêm vào danh sách:



Xây dựng một cửa sổ modal đơn giản bằng cách dùng một User Control:

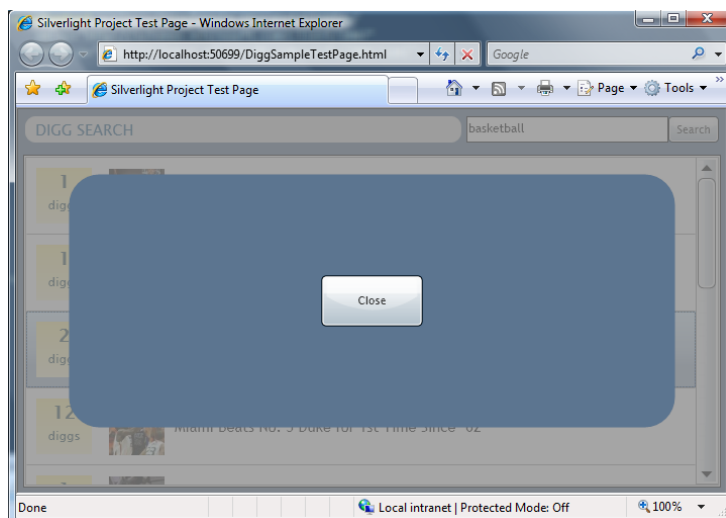
Chúng ta dự định sẽ dùng control StoryDetailsView để hiển thị một hộp thoại chứa nội dung chi tiết của kết quả tìm kiếm. Khi nó được hiển thị, chúng ta muốn rằng nó sẽ hiện lên trên tất cả các nội dung khác trên trang, và phải đảm bảo rằng người dùng sẽ không thể can thiệp vào bất kỳ phần nào khác nếu chưa đóng hộp thoại này lại.

Có một số cách để làm ra hộp thoại dạng modal này. Trong trường hợp này chúng ta sẽ làm bằng cách mở file StoryDetailsView.xaml và thêm vào đoạn XAML sau:

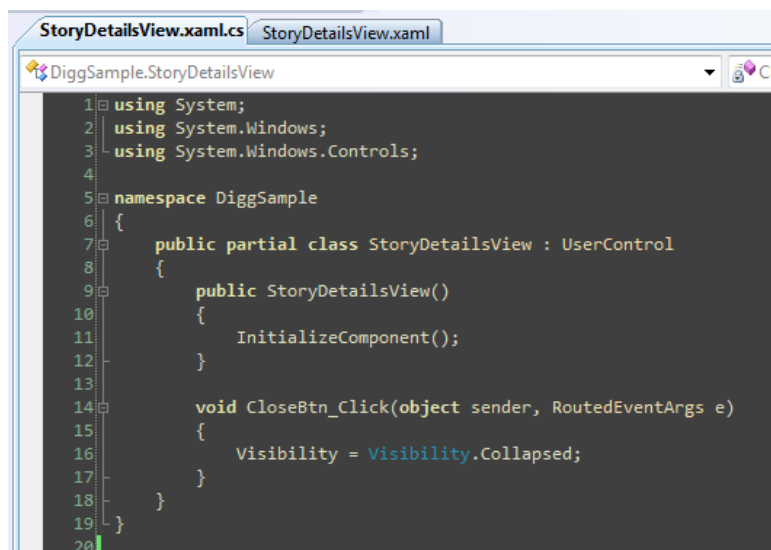
```
StoryDetailsView.xaml
1 <UserControl x:Class="DiggSample.StoryDetailsView"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
4
5   <Grid>
6
7       <Rectangle HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Opacity="0.765" Fill="#FF8A8A8A" />
8
9       <Border CornerRadius="30" Background="#FF5C7590" Width="600" Height="250">
10
11         <Button Content="Close" Width="100" Height="50" Click="CloseBtn_Click" />
12
13       </Border>
14
15   </Grid>
16
17 </UserControl>
18
```

Thẻ <Rectangle> ở trên dùng để chiếm lấy toàn bộ diện tích trên màn hình. Màu nền của nó sẽ màu xám tối mờ mờ (vì thuộc tính Opacity của nó là 0.765 nên bạn có thể nhìn thấy một chút phía sau nó). Thẻ <Border> thứ hai sẽ nằm lên trên hình chữ nhật trước, và chiếm một phần trên trang. Nó có màu nền xanh, và chứa một nút Close.

Khi chạy, control StoryDetailsView sẽ hiển thị như sau:



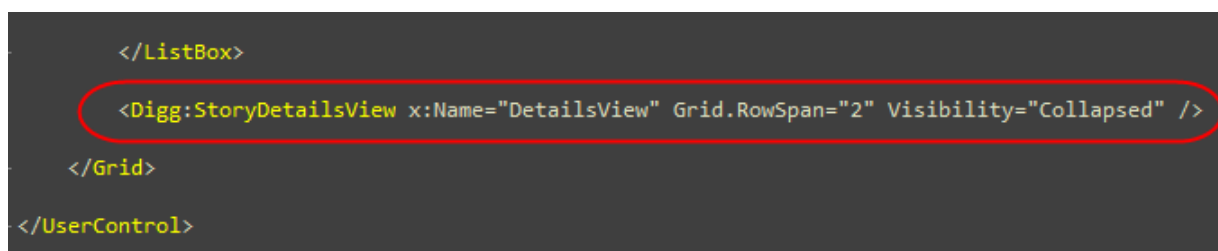
Chúng ta có thể tạo hàm xử lý sự kiện “CloseBtn_Click” trong file code-behind của user control. Khi nhấn lên, hàm này sẽ đặt thuộc tính Visibility của UserControl thành “Collapsed”, đối tượng sẽ biến mất và người dùng sẽ trở lại màn hình bên dưới nó.



```
1 using System;
2 using System.Windows;
3 using System.Windows.Controls;
4
5 namespace DiggSample
6 {
7     public partial class StoryDetailsView : UserControl
8     {
9         public StoryDetailsView()
10        {
11            InitializeComponent();
12        }
13
14        void CloseBtn_Click(object sender, RoutedEventArgs e)
15        {
16            Visibility = Visibility.Collapsed;
17        }
18    }
19 }
20
```

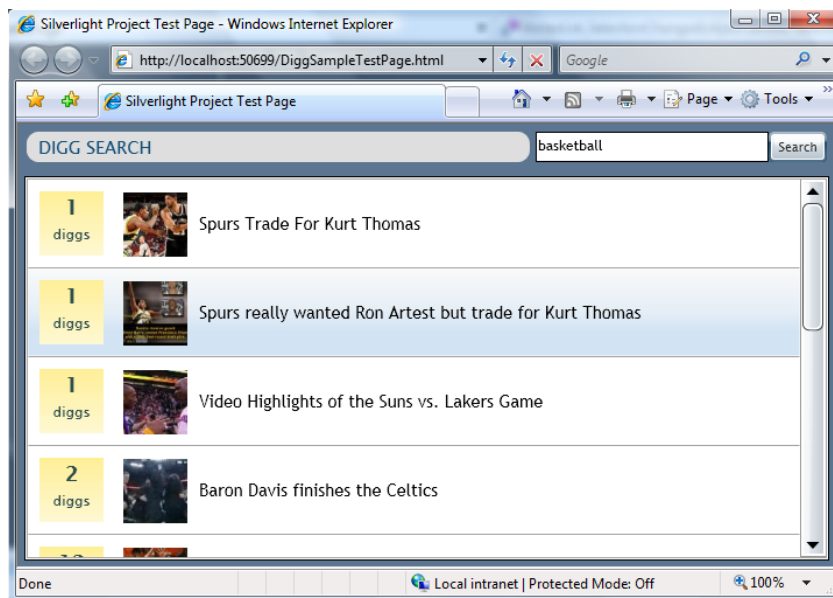
Hiện thị control StoryDetailsView

Một cách dễ dàng để hiển thị control StoryDetailsView trên màn hình là thêm nó vào cuối file Page.xaml, và đặt giá trị mặc nhiên cho thuộc tính Visibility của nó là Collapsed (do vậy mặc nhiên nó sẽ không hiển thị khi ứng dụng được nạp):



```
</ListBox>
<Digg:StoryDetailsView x:Name="DetailsView" Grid.RowSpan="2" Visibility="Collapsed" />
</Grid>
</UserControl>
```

Chúng ta có thể bắt sự kiện “SelectionChanged” từ ListBox bên trong file code-behind của Page.xaml:



Khi một người dùng chọn một mục nào đó trong danh sách, chúng ta sẽ dùng hàm xử lý sự kiện `SelectionChanged` của `ListBox` để đặt giá trị cho thuộc tính `Visibility` của control `ShowDetailView` thành “Visible”:

```
void StoriesList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    DetailsView.Visibility = Visibility.Visible;
}
```

Nó sẽ làm xuất hiện hộp thoại mà chúng ta đã tạo. Khi người dùng nhấn nút “Close”, nó sẽ biến mất, và người dùng lại có thể tiếp tục chọn một kết quả khác.

Chuyển dữ liệu cho đối tượng StoryDetailView

Khi `StoryDetailView` hiện ra, chúng ta cần đưa thông tin chi tiết kết quả tìm kiếm mà người dùng đã chọn trong `ListBox`.

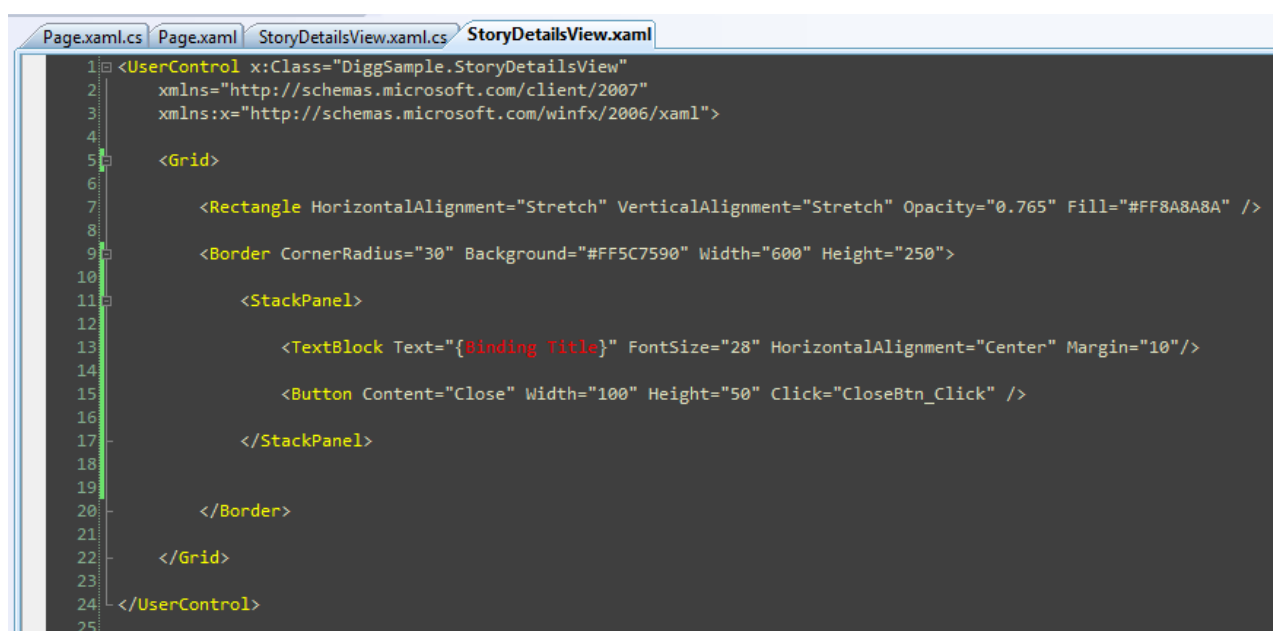
Bên trong hàm xử lý sự kiện “`SelectionChanged`” của `ListBox` (trong file code-behind), chúng ta sẽ truy cập vào đối tượng `DiggStory` tương ứng với dòng mà người dùng đã chọn trong `ListBox` bằng cách truy cập vào thuộc tính “`SelectedItem`”.

Một cách khác cũng có thể dùng là truyền đối tượng `DiggStory` này vào cho `StoryDetailView` bằng cách gán nó cho thuộc tính “`DataContext`” của `StoryDetailView` ngay trước khi cho control này hiện ra.

```
void StoriesList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    DiggStory story = (DiggStory) StoriesList.SelectedItem;

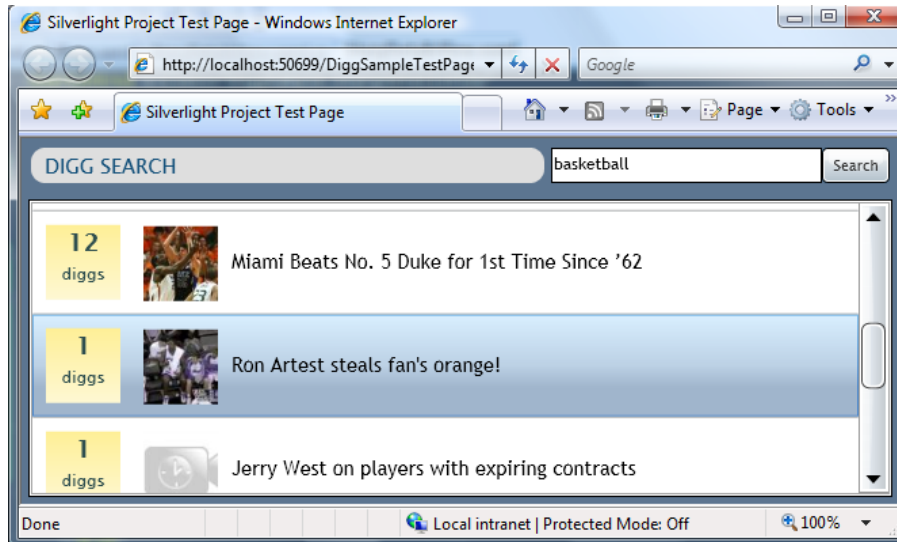
    DetailsView.DataContext = story;
    DetailsView.Visibility = Visibility.Visible;
}
```

Sau đó ta có thể viết các lệnh bên trong UserControl của chúng ta để hiển thị kết quả. Hoặc một cách khác là chúng ta có thể dùng các biểu thức gắn nối dữ liệu để gắn các control với giá trị của nó. Lấy ví dụ, chúng ta có thể cập nhật lại đoạn XAML của StoryDetailsView để hiển thị kết quả được chọn bằng cách dùng biểu thức gắn nối như sau:

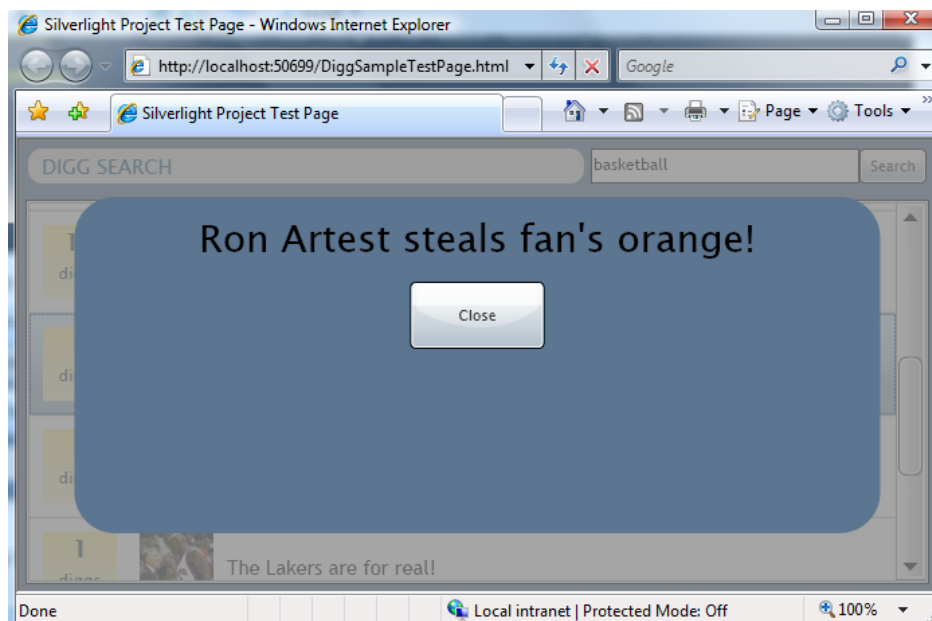


```
1 <UserControl x:Class="DiggSample.StoryDetailsView"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
4
5   <Grid>
6
7       <Rectangle HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Opacity="0.765" Fill="#FF8A8A8A" />
8
9       <Border CornerRadius="30" Background="#FF5C7590" Width="600" Height="250">
10
11         <StackPanel>
12
13             <TextBlock Text="{Binding Title}" FontSize="28" HorizontalAlignment="Center" Margin="10"/>
14
15             <Button Content="Close" Width="100" Height="50" Click="CloseBtn_Click" />
16
17         </StackPanel>
18
19     </Border>
20
21 </Grid>
22
23 </UserControl>
24
25
```

Và đây là kết quả khi người dùng nhấn vào danh sách:



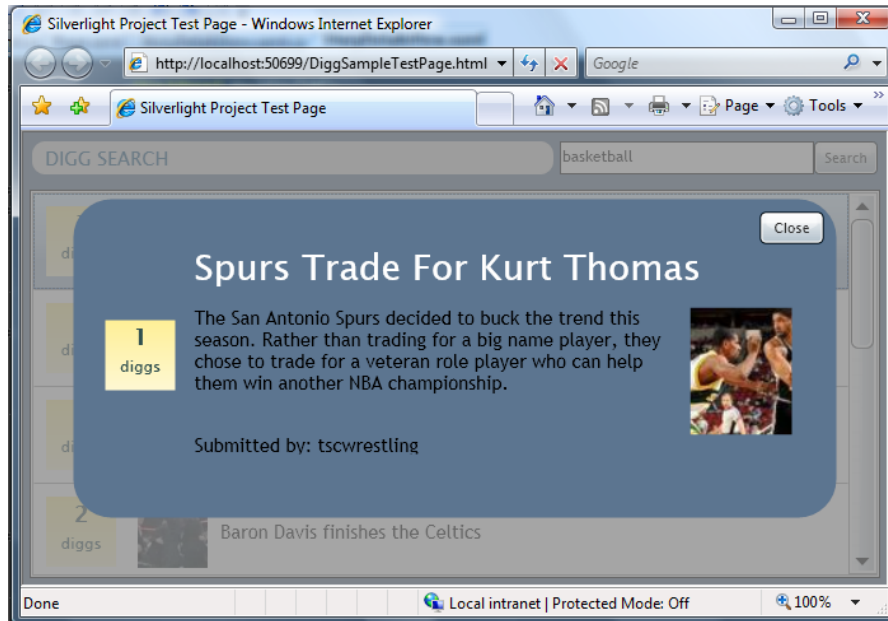
Hàm xử lý sự kiện của ListBox sẽ xử lý việc lựa chọn, đặt DataContext của UserControl thành đối tượng DiggStory trước khi làm nó hiện ra:



Chú ý là tiêu đề của DiggStory sẽ xuất hiện trong control, đó là vì chúng ta đã thêm biểu thức gắn nối dữ liệu và cho nó.

Hoàn chỉnh User Control Layout

Chương trình mẫu ở trên đã biểu diễn các bước cơ bản để thêm vào một hộp thoại dạng master/details. Giờ chúng ta sẽ hoàn chỉnh StoryDetailsView bằng cách thêm một số điều khiển và biểu thức gắn nối dữ liệu vào cho User Control.



Chúng ta có thể cập nhật StoryDetailsView để nó trông giống như trên bằng cách sửa lại <Border> cho nó có nội dung bên trong:

```
<StackPanel Margin="10">
    <!-- Top Right Close Button -->
    <Button Click="CloseBtn_Click" Content="Close" Style="{StaticResource CloseButton}"/>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="50"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="15"/>
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="85"/>
            <ColumnDefinition Width="380" />
            <ColumnDefinition Width="100" />
        </Grid.ColumnDefinitions>

        <!-- Top: Story Title (hyperlink) -->
        <HyperlinkButton Content="{Binding Title}" NavigateUri="{Binding HrefLink}" Style="{StaticResource TitleLink}" />

        <!-- Left: Yellow Digg Panel with NumDiggs-->
        <StackPanel Style="{StaticResource DiggPanelDetail}" >
            <TextBlock Text="{Binding NumDiggs}" Style="{StaticResource NumDigsBlock}" />
            <TextBlock Text="diggs" Style="{StaticResource NumDigsSubBlock}" />
        </StackPanel>

        <!-- Center: Story Description -->
        <TextBlock Text="{Binding Description}" Style="{StaticResource DescriptionBlock}"/>

        <!-- Right: Story Preview Picture -->
        <Image Source="{Binding Thumbnail}" Style="{StaticResource DetailsThumbnailPreview}"/>

        <!-- Bottom Center: Submitter Details -->
        <StackPanel Style="{StaticResource SubmitDetails}">
            <TextBlock Text="Submitted by: " Style="{StaticResource PosterBlock}"/>
            <TextBlock Text="{Binding UserName}" Style="{StaticResource PosterBlock}" />
        </StackPanel>
    </Grid>
</StackPanel>
```

Như bạn thấy, không có bất kỳ đoạn mã lệnh nào được thêm vào, vì chúng ta đã dùng phép gắn nối dữ liệu (databinding) để “kéo” các giá trị từ DataContext, do vậy chúng ta không cần thêm bất kỳ đoạn lệnh nào.

Bước tiếp theo

Giờ chúng ta đã có tất cả các chức năng cơ bản và các tính năng tương tác với người dùng đã được xây dựng.

Bước cuối cùng, chúng ta sẽ chỉnh sửa lại giao diện thêm một chút, đặc biệt là thay đổi hình thức của ListBox và Button.

Để làm điều này, bạn hãy đọc bài hướng dẫn tiếp theo: Dùng các mẫu để tùy biến Look and Feel của control.

Bài 7: Dùng các mẫu để tùy biến Look and Feel của control

Đây là phần 7 trong loạt 8 bài hướng dẫn các bước để xây dựng một chương trình Digg đơn giản dùng bản Beta 1 của Silverlight 2. Các bài hướng dẫn này nên được đọc theo thứ tự, và sẽ giúp giải thích một số khái niệm cơ bản trong Silverlight.

Bạn có thể download mã nguồn phiên bản hoàn chỉnh của chương trình mẫu Digg tại đây: <http://www.scottgu.com/blogposts/slbeta1apps/diggsample.zip>.

Tùy biến Look and Feel

Một trong những tính năng mạnh mẽ nhất của mô hình lập trình trong WPF và Silverlight là khả năng cho phép tùy biến hoàn toàn LnF (Look and Feel: cách thức hiển thị và xử lý hành vi của các đối tượng giao diện) của các control. Nó cho phép người phát triển và người thiết kế có thể có thể làm cho các đối tượng giao diện trở nên cực kỳ tinh tế, và tính mềm dẻo rất cao của nó cũng cho phép tạo ra các trải nghiệm người dùng xuất sắc.

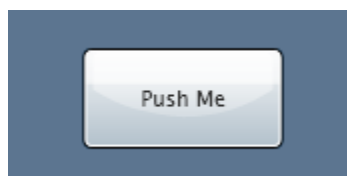
Trong phần này chúng ta sẽ xem qua một vài cách mà bạn có thể dùng để tùy biến các control, và sẽ dùng nó để “tỉa tốt” lại chương trình Digg.

Tùy biến nội dung của các control

In Part 1 of our tutorial we added a simple button control to the page and demonstrated how to set a Trong phần 1, chúng ta đã thêm một nút bấm vào một trang và giới thiệu cách đặt lại chuỗi nội dung “Push Me!” của nó. Chúng ta cũng đã tạo ra hàm để xử lý sự kiện “Click” mỗi khi người dùng nhấn chuột.

```
<Button x:Name="MyButton" Content="Push Me!" Width="100" Height="50" Click="MyButton_Click" />
```

Những thay đổi này sẽ làm cho nút bấm hiển thị giống như dưới đây:



Một trong những điều có thể làm bạn ngạc nhiên, đó là thuộc tính “Content” của Button không nhất thiết phải là một chuỗi đơn giản kiểu như “Push Me!”, mà thực sự, chúng ta thậm chí có thể đặt các Control và Shape như giá trị cho thuộc tính “Content”.

Lấy ví dụ, chúng ta có thể nhúng một StackPanel với một <Image> and <TextBlock> bên trong:

```
<Button x:Name="MyButton" Width="400" Height="200" Click="MyButton_Click">
  <Button.Content>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center">
      <Image Source="/scott.jpg" Height="175"/>
      <TextBlock Text="ScottGu" FontSize="36" VerticalAlignment="Center" Margin="10"/>
    </StackPanel>
  </Button.Content>
</Button>
```

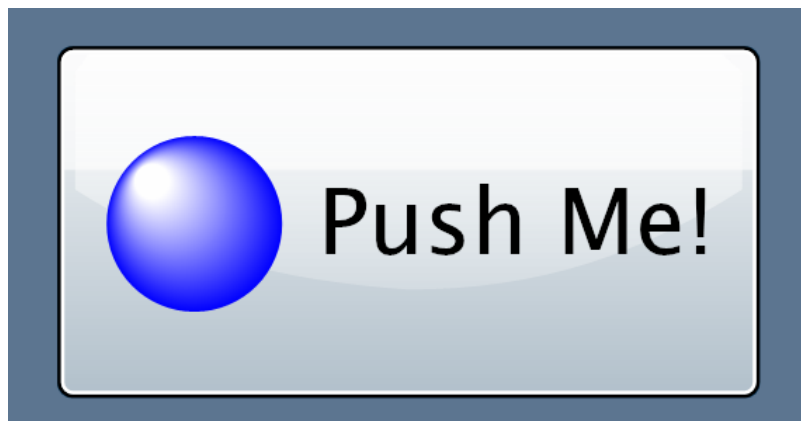
Điều này sẽ làm Button của chúng ta trông giống như dưới đây. Nhớ rằng nó vẫn giữ nguyên các hành vi và tính năng như cũ (khi nhấn vào sẽ lún xuống, và phát ra sự kiện Click...)



Chúng ta cũng có thể dùng một Shape để hiển thị một ảnh vector bên trong nút, giống như hình Ellipse dưới đây:

```
<Button x:Name="MyButton" Width="200" Height="100" Click="MyButton_Click">
  <Button.Content>
    <StackPanel Orientation="Horizontal">
      <Ellipse Margin="10" Width="50" Height="50">
        <Ellipse.Fill>
          <RadialGradientBrush GradientOrigin=".2,.2">
            <GradientStop Offset="0.2" Color="white"/>
            <GradientStop Offset="1" Color="blue"/>
          </RadialGradientBrush>
        </Ellipse.Fill>
      </Ellipse>
      <TextBlock VerticalAlignment="Center" FontSize="26">Push Me!</TextBlock>
    </StackPanel>
  </Button.Content>
</Button>
```

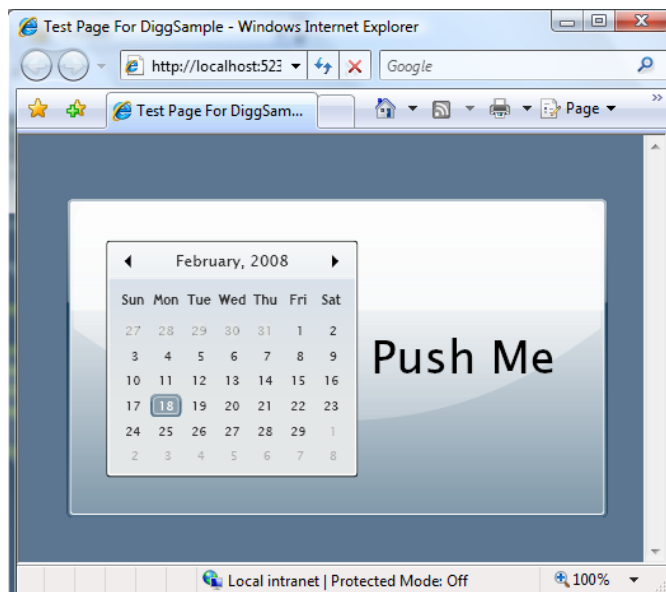
Ở trên tôi đã tô màu Ellipse với một RadialGradientBrush để nó trông đẹp hơn:



Nếu bạn là người lập dị ☺, thậm chí bạn còn có thể nhúng các control tương tác như Calendar trong ví dụ dưới đây:

```
<Button x:Name="MyButton" Width="425" Height="250" Click="MyButton_Click">
  <Button.Content>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
      <Calendar />
      <TextBlock VerticalAlignment="Center" FontSize="36" Text="Push Me" Padding="10"/>
    </StackPanel>
  </Button.Content>
</Button>
```

Ở ví dụ trên, control Calendar vẫn mang tính tương tác hoàn toàn – có nghĩa là người dùng có thể chuyển qua lại giữa các tháng và chọn một ngày trong tháng, và sau đó nhấn vào nút chứa nó để tạo ra sự kiện “Click”. (Tôi không chắc liệu điều này có là một trải nghiệm tốt cho người dùng hay không – nhưng nó cho ta thấy mức độ mềm dẻo mà bạn có thể làm).



Các cách tùy biến nội dung như trên sẽ làm việc không chỉ với Button, mà cho tất cả các control thừa kế từ lớp ContentControl.

Tùy biến các control dùng Control Template

Mô hình xây dựng nên các control trong Silverlight và WPF cho phép bạn làm được nhiều hơn là chỉ tùy biến lại nội dung bên trong control. Về cơ bản, nó cho phép bạn thay thế hoàn toàn các thành phần trực quan trên một control với bất kỳ thứ gì bạn muốn – trong khi vẫn giữ lại những hành vi như nó vốn có. Chẳng hạn, cho là bạn không muốn các nút bấm mang hình chữ nhật như mặc nhiên, mà muốn nó có hình tròn như dưới đây:



Chúng ta có thể làm điều này bằng việc tạo ra một kiểu “RoundButton” trong file App.xaml. Trong đó chúng ta sẽ thay thế thuộc tính Template của nút bấm, và cung cấp một ControlTemplate mà nó sẽ thay thế hình chữ nhật mặc nhiên với một hình Ellipse và một TextBlock bên trong:

```

<Style x:Key="RoundButton" TargetType="Button">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="Button">

        <Grid>

          <Ellipse Width="200" Height="200">
            <Ellipse.Fill>
              <RadialGradientBrush GradientOrigin=".2,.2">
                <GradientStop Offset="0.2" Color="White"/>
                <GradientStop Offset="1" Color="blue"/>
              </RadialGradientBrush>
            </Ellipse.Fill>
          </Ellipse>

          <TextBlock Text="Push Me!"
            FontSize="28"
            HorizontalAlignment="Center"
            VerticalAlignment="Center" />

        </Grid>

      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>

```

Chúng ta có thể tham chiếu đến Style này để dùng RoundButton như là mẫu cho nút bấm:

```

<Button Width="200" Height="200" Click="MyButton_Click" Style="{StaticResource RoundButton}" />

```

Kết hợp nội dung bên trong Control Template

Một điều mà có lẽ bạn đã biết trong “RoundButton” control template ở trên là kích thước của Button, và nội dung hiển thị bên trong nó đã được hard-coded (nó luôn hiển thị “Push Me!”). Một tin tốt lành là WPF và Silverlight cho phép ta có thể tùy biến cả các cài đặt đó. Chúng ta có thể làm điều này bằng cách dùng cú pháp mở rộng {TemplateBinding ControlProperty} bên trong control template để gắn nối vào các thuộc tính của control. Điều này sẽ cho phép các control template của chúng ta có thể thích ứng được với việc thay đổi các thuộc tính của các nhà phát triển khác:

```
<Style x:Key="RoundButton" TargetType="Button">
  <Setter Property="Margin" Value="15"/>
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="Button">
        <Grid>
          <Ellipse Width="{TemplateBinding Width}" Height="{TemplateBinding Height}">
            <Ellipse.Fill>
              <RadialGradientBrush GradientOrigin=".2,.2">
                <GradientStop Offset="0.2" Color="White"/>
                <GradientStop Offset="1" Color="blue"/>
              </RadialGradientBrush>
            </Ellipse.Fill>
          </Ellipse>
          <ContentPresenter Content="{TemplateBinding Content}"
            FontSize="{TemplateBinding FontSize}"
            FontFamily="{TemplateBinding FontFamily}"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"/>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Hãy để ý ở trên thay vì chúng ta thêm một `<TextBlock>` để hiển thị nội dung, chúng ta đã dùng control `<ContentPresenter>`. Điều này sẽ cho phép chúng ta không chỉ hiển thị các chuỗi văn bản mà cả các nội dung khác (giống như chúng ta đã làm trước đây trong loạt bài này).

Chúng ta sau đó có thể dùng Style ở trên trong 3 nút bấm bên dưới (mỗi cái đều có nội dung và các cài đặt khác nhau):

```

<StackPanel Orientation="Horizontal">

  <!-- First Button -->
  <Button Width="100" Height="100" Content="Button One" FontSize="14" Style="{StaticResource RoundButton}" />

  <!-- Second Button -->
  <Button Width="200" Height="200" Content="Button Two" FontSize="22" Style="{StaticResource RoundButton}" />

  <!-- Third Button -->
  <Button Width="200" Height="200" Style="{StaticResource RoundButton}">

    <Button.Content>

      <Calendar HorizontalAlignment="Center" VerticalAlignment="Center">
        <Calendar.RenderTransform>
          <ScaleTransform ScaleX=".5" ScaleY=".5" CenterX="100" CenterY="100"/>
        </Calendar.RenderTransform>
      </Calendar>

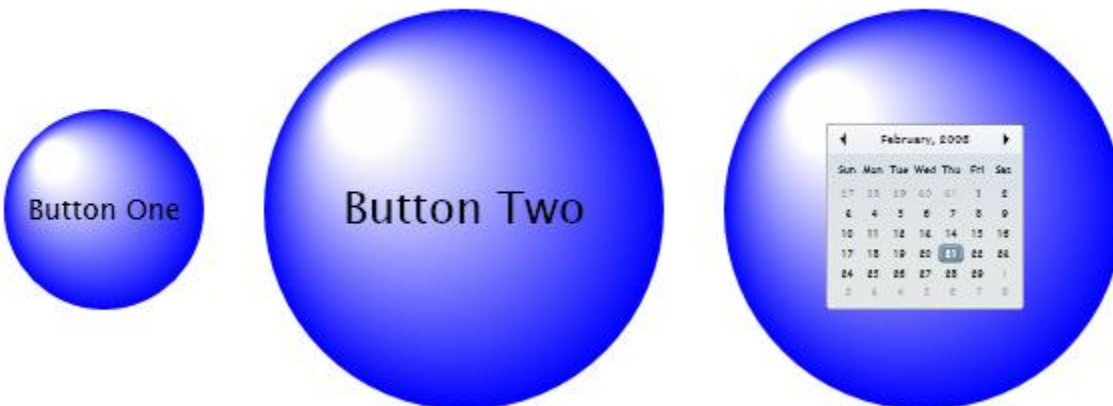
    </Button.Content>

  </Button>

</StackPanel>

```

Các Button trên sẽ hiển thị giống như dưới đây (và tất nhiên – control Calendar vẫn sẽ hỗ trợ chuyên trang và chọn ngày):

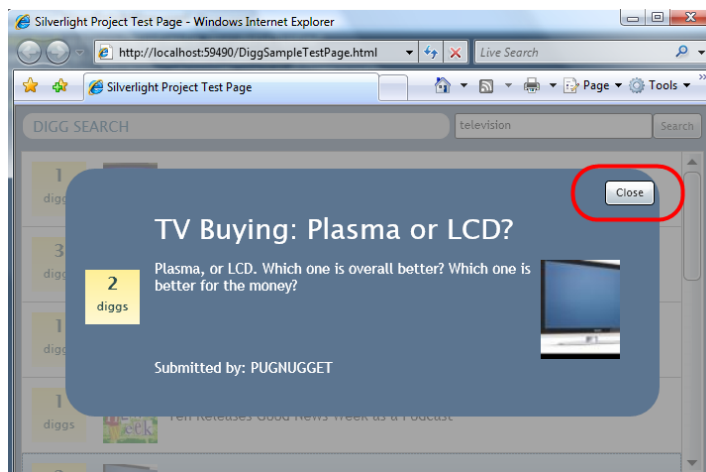


Nếu muốn đẹp mắt hơn, chúng ta còn có thể thêm các hoạt hình vào ControlTemplate (để bắt các trạng thái của nút bấm như: “hover”, “focus” và “pushed”). Khả năng này cho phép ta tạo ra các giao diện tương tác bóng bẩy, mà bạn thực sự không thể làm được với HTML.

Các nhà phát triển làm việc với các control trong một ứng dụng có thể quên đi tất cả các thuộc tính và kiểu dáng đã được tùy biến, họ chỉ cần quan tâm đến việc bắt các sự kiện và xử lý dữ liệu, việc tùy biến giao diện là của các nhà thiết kế.

Đánh bóng lại chương trình Digg

Giờ chúng ta đã khảo sát qua một số điều cơ bản về các mà các Control Template làm việc, chúng ta sẽ dùng chúng trong một vài chỗ để làm cho chương trình Digg bóng bẩy hơn. Có một nơi trong chương trình của chúng ta cần phải được chỉnh sửa lại – đó là nút Close trong User Control:



Một tin tốt lành là điều này rất dễ dàng cho chúng ta (hoặc cho nhà thiết kế của chúng ta) để có thể chỉnh sửa. Chúng ta có thể thêm một ControlTemplate vào style “CloseButton” trong file App.xaml và thêm một số hình họa để làm cho nút Close trông ấn tượng hơn. (ghi chú: một người thiết kế giỏi hơn tôi có thể thêm một số hoạt hình cho các hình họa để làm cho nó đẹp hơn):

```
<Style x:Key="CloseButton" TargetType="Button">
  <Setter Property="HorizontalAlignment" Value="Right"/>
  <Setter Property="Width" Value="50"/>
  <Setter Property="Height" Value="25"/>

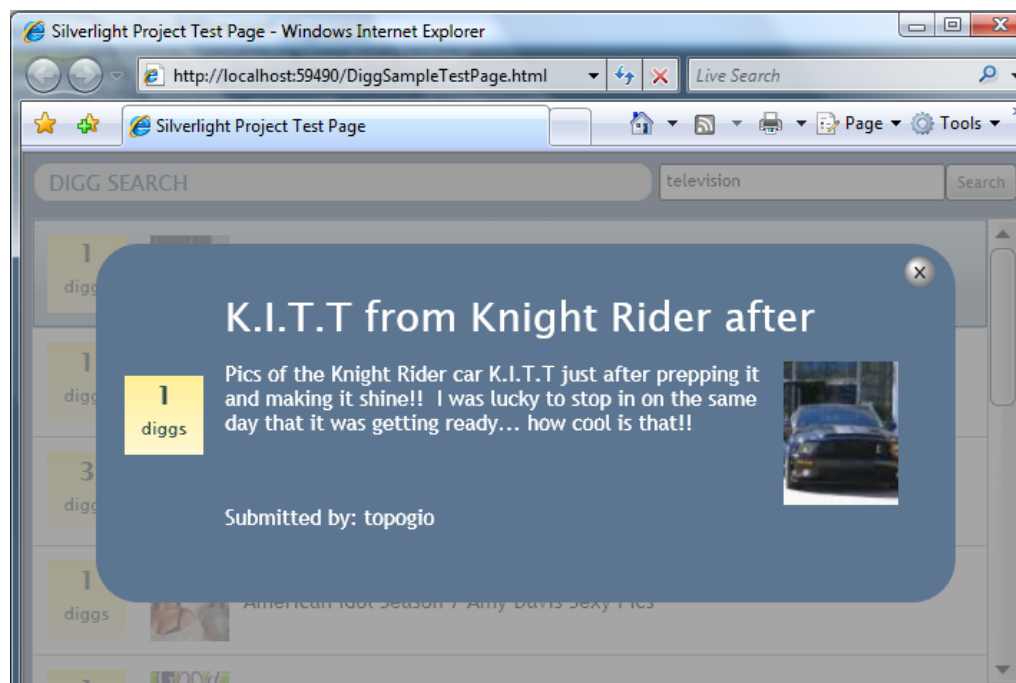
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate>
        <Border Width="22" Height="22" CornerRadius="15">

          <TextBlock Foreground="#222" TextAlignment="center" Text="r"
            FontSize="11" VerticalAlignment="center" FontFamily="Webdings"/>

          <Border.Background>
            <RadialGradientBrush GradientOrigin=".3, .3">
              <GradientStop Color="#FFF" Offset=".15"/>
              <GradientStop Color="#777" Offset="1"/>
            </RadialGradientBrush>
          </Border.Background>

        </Border>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```


Khi chạy lại chương trình, nút Close sẽ trông như sau:



Một chỗ nữa mà tôi nghĩ cũng nên được chỉnh sửa thêm là phần viền bên ngoài của ListBox. Nếu nhìn gần, bạn sẽ thấy ListBox trong bản Beta1 có một đường viền lồng bên ngoài như sau:



Chúng ta có thể bỏ viền này để nó có một viền phẳng xung quanh ListBox bằng việc tùy biến lại Control Template. Dưới đây là style của ListBox với một mẫu tùy biến:

```

<Style x:Key="StoriesList" TargetType="ListBox">
  <Setter Property="Margin" Value="5"/>
  <Setter Property="Grid.Row" Value="1"/>

  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate>
        <Grid>

          <ScrollViewer x:Name="ELEMENT_SV"
            Background="White"
            VerticalScrollBarVisibility="Auto"
            HorizontalScrollBarVisibility="Auto">

            <ItemsPresenter/>

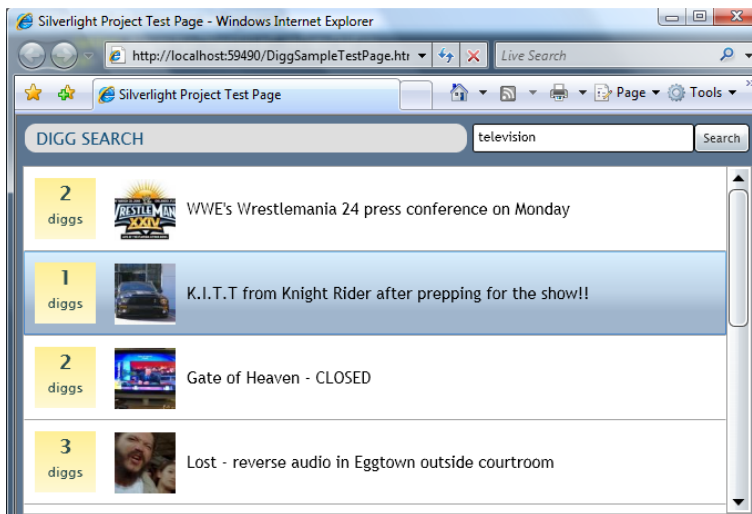
          </ScrollViewer>

        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>

```

Hãy để ý cách chúng ta xóa bỏ đường viền của ListBox. Chúng ta chỉ dùng <ScrollViewer> trong Silverlight (nó cho phép cuộn nội dung bên trong) và nhúng vào trong một <ItemsPresenter/> để nó tự vẽ các mục dữ liệu bên trong (nó dùng <DataTemplate> mà chúng ta đã tạo ở bài 4 để vẽ các mục đó).

Dưới đây là giao diện phẳng hơn mà chúng ta đã làm với ListBox:



Điều thực sự hấp dẫn là chúng ta đã không phải thay đổi bất kỳ điều gì trong chương trình, và cũng chẳng phải thay đổi mã XAML của các điều khiển để làm cho giao diện của chúng thay đổi. Khả năng phân tách code/design cho phép các nhà phát triển và thiết kế có thể làm việc một cách hữu hiệu trên các ứng dụng Silverlight và WPF.

Expression Blend và bộ Expression Studio làm cho khả năng thiết kế các control mạnh mẽ hơn, và mang đến một bộ công cụ thiết kế phong phú mà nó sẽ giúp việc tùy biến dễ dàng hơn.

Bước tiếp theo

Chúng ta đã hoàn thành xong việc xây dựng chương trình Digg bằng Silverlight. Bước cuối cùng là hiện thực hóa một phiên bản trên desktop cho nó. Một tin tốt là điều này cũng không khó khăn lắm – vì Silverlight vốn là một tập con của bộ WPF và .NET Framework đầy đủ, vậy nên các khái niệm, mã lệnh và nội dung đều có thể chuyển đổi tương đối dễ dàng. Để làm điều này, xin mời xem tiếp phần cuối: Xây dựng phiên bản chạy trên desktop với WPF.

Bài 8: Xây dựng phiên bản chạy trên desktop với WPF

Đây là phần 8 trong loạt 8 bài hướng dẫn các bước để xây dựng một chương trình Digg đơn giản dùng bản Beta 1 của Silverlight 2. Các bài hướng dẫn này nên được đọc theo thứ tự, và sẽ giúp giải thích một số khái niệm cơ bản trong Silverlight.

Bạn có thể download mã nguồn phiên bản hoàn chỉnh của chương trình mẫu Digg tại đây: <http://www.scottgu.com/blogposts/slbeta1apps/diggsample.zip>.

Mã nguồn phiên bản WPF có thể tải tại đây:

<http://www.scottgu.com/blogposts/slbeta1apps/diggdesktopsample.zip>.

Xây dựng phiên bản chạy trên desktop với WPF

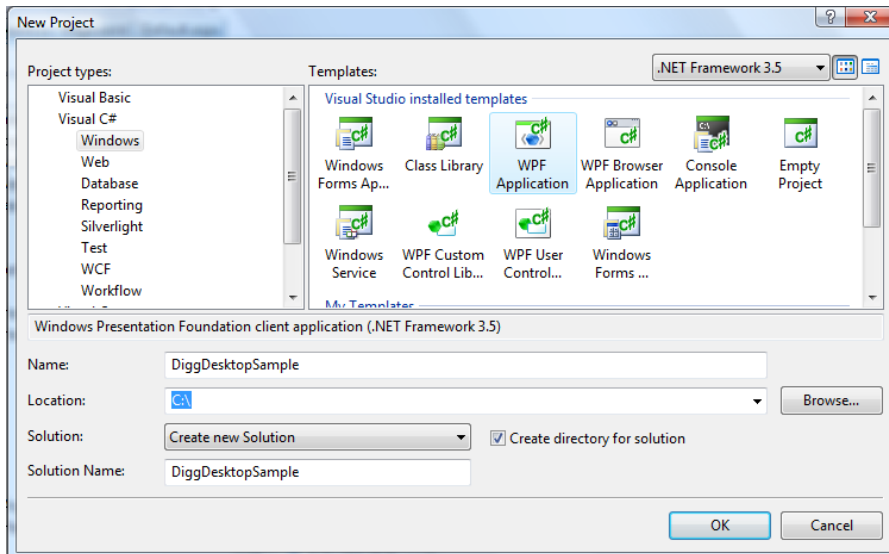
Mục đích của chúng ta trong bài cuối cùng này hơi khác bảy bài trước một chút. Chúng ta sẽ không định chạy các lệnh Silverlight trong bài này, thay vì đó chúng ta sẽ dùng WPF và .NET 3.5. Chúng ta sẽ làm cho chương trình Digg – vốn được xây dựng để chạy trên trình duyệt, có thể chạy trên desktop như một ứng dụng Windows bình thường.

Silverlight mang đến một tập hàm API là tập con của bộ đầy đủ trong .NET Framework. Mục đích là để các nhà phát triển có thể tận dụng lại các kiến thức và kỹ năng đã học, và dùng lại mã lệnh và nội dung xuyên suốt các loại ứng dụng khác nhau: các ứng dụng web RIA, các chương trình Windows và các giải pháp Office.

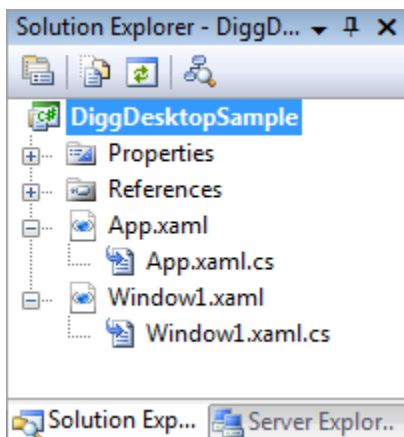
Dưới đây là các bước tôi dùng để dùng lại các lệnh trong chương trình Digg viết bằng Silverlight để xây dựng nên phiên bản chạy trên desktop (chạy bên ngoài trình duyệt như một ứng dụng Windows).

Bước 1: Tạo ứng dụng WPF

Chúng ta sẽ bắt đầu tạo một chương trình WPF (WPF Desktop Application) trong VS 2008. Ta sẽ đặt tên nó là “DiggDesktopSample”:



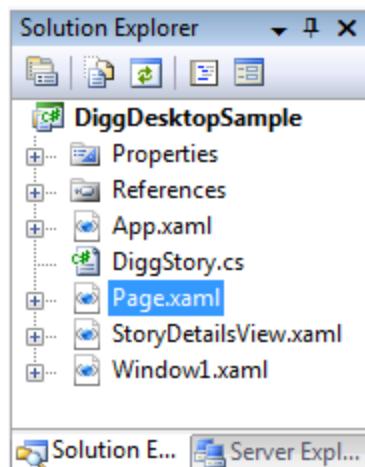
Dự án mới sẽ được tạo với 2 file: App.xaml và Window.xaml:



Bạn sẽ thấy rằng cấu trúc project này tương tự như project mà chúng ta đã tạo với Silverlight (bao gồm file App.xaml và Page.xaml):

Bước 2: Copy mã lệnh từ chương trình Digg vào ứng dụng WPF

Chúng ta sẽ sao chép/dán từ chương trình Silverlight có sẵn vào chương trình DiggDesktopSample mới:



Hiện tại với phiên bản Bate1, các bước sao chép/dán này đều phải làm bằng tay, nhưng trong tương lai, chúng tôi sẽ thêm vào chức năng chỉnh sửa tự động khi bạn sao chép giữa các loại chương trình khác nhau.

Bước 3: Khắc phục một số vấn đề

Tôi đã phải sửa lại hai chỗ để làm cho chương trình Digg có thể biên dịch được:

- 1) Schema của Silverlight Beta1 XAML (xmlns:) khác mới WPF. Tôi cần sửa lại các file XAML được sao chép vào để nó chỉ đến schema của WPF. Đây là thứ mà chúng tôi muốn cập nhật trước khi đưa ra phiên bản tiếp theo.

- 2) Tôi phải thay đổi `<WaterMarkTextBox>` thành `<TextBox>` và `<HyperlinkButton>` thành `<TextBlock>`. Đó là hai control mới có trong Silverlight Beta1 và hiện không có trong phiên bản WPF (chúng tôi sẽ thêm vào sau này). Tôi không phải thay đổi bất cứ đoạn lệnh nào để làm việc với chúng, bao gồm cả các lệnh về mạng, LINQ to XML, hoặc các lệnh gắn nối dữ liệu. Sau khi sửa lại, chương trình đã có thể biên dịch.

Bước 4: Chạy chương trình Digg trên một cửa sổ desktop

Tôi tiếp tục mở file Window1.xaml (đây là cửa sổ mặc nhiên được mở khi chạy chương trình) trong project và cập nhật lại tiêu đề thành "Digg Desktop Version" và tăng chiều rộng và ngang của cửa sổ. Sau đó tôi thêm vào đối tượng Page.xaml từ chương trình Digg cũ và đặt nó như đối tượng gốc trên Window, điều này sẽ làm nó được nạp và hiển thị mỗi khi Window được hiển thị. Tôi không cần phải thay đổi bất kỳ thứ gì bên trong mã lệnh của Page.xaml, vì nó thừa kế từ UserControl nên nó có thể được chứa bên trong bất kỳ Window hay Control nào của WPF.

```

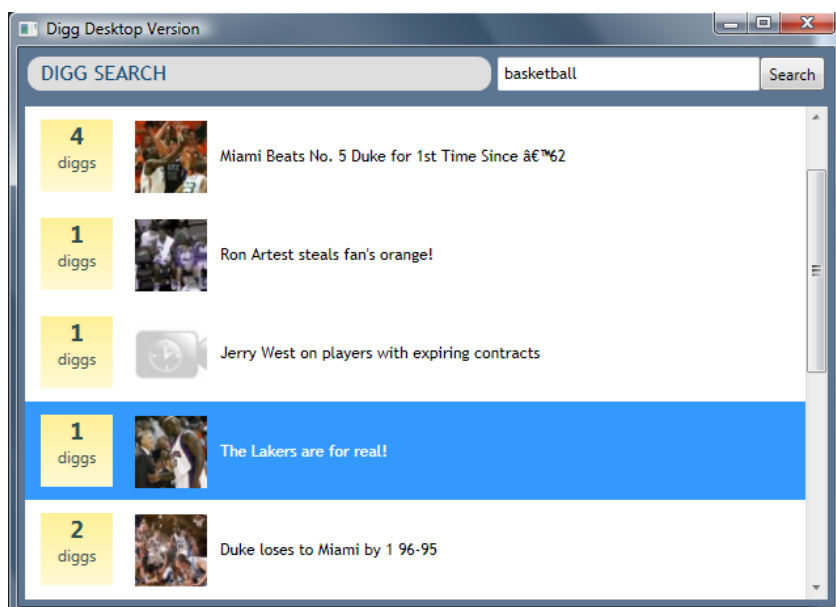
Window1.xaml
1 <Window x:Class="DiggDesktopSample.Window1"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:Digg="clr-namespace:DiggDesktopSample"
5   Title="Digg Desktop Version" Height="600" Width="700">
6
7   <Grid>
8     <Digg:Page/>
9   </Grid>
10
11 </Window>
12

```

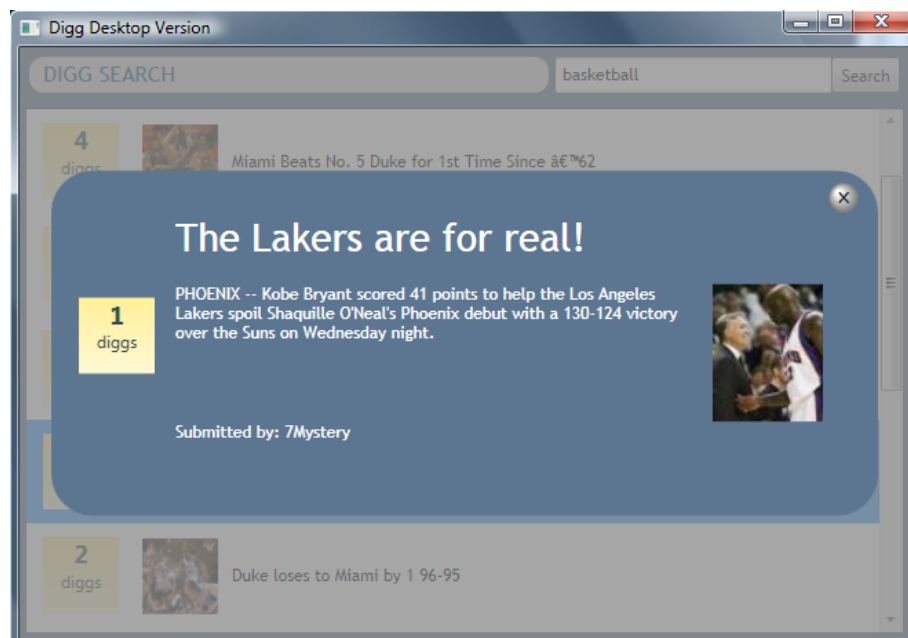
Việc cuối cùng cần làm là sửa lại một trục trục với Digg REST API, đó là vì Digg server có xác định xem có phải một server hay một chương trình (không chạy trên trình duyệt) đang truy cập nó hay không và đưa ra thông báo cấm truy cập (access denied) – có lẽ là để tránh bị các script tự động tấn công vào dịch vụ của họ. Tôi khắc phục bằng cách cho kết nối thông qua một proxy URL.

Bước 5: Chạy chương trình

Giờ chúng ta có thể chạy chương trình Digg Desktop Application. Tất cả các chức năng sẽ làm việc giống như phiên bản Silverlight:



Và khi một kết quả được chọn từ danh sách, thông tin chi tiết sẽ được hiển thị:



Có một vài điểm khác biệt về hình thức giữa phiên bản trình duyệt và phiên bản desktop. Nguyên nhân chính là vì WPF thừa kế lại giá trị mặc nhiên của các cài đặt hiển thị (font, màu sắc, thanh cuộn...) của hệ điều hành, trong khi Silverlight lại dùng một cài đặt hiển thị chung cho các hệ điều hành khác nhau. Nếu muốn hai phiên bản hiển thị hoàn toàn giống nhau, chúng ta cũng có thể làm được bằng cách chỉ ra giá trị cụ thể cho Style và Control Template.

Tổng kết

Chúng ta đã nắm được các thông tin cũng như kinh nghiệm để dùng lại mã lệnh giữa các ứng dụng Silverlight và WPF. Tôi nghĩ bạn sẽ thấy các kỹ năng và kiến thức bạn học khi xây dựng các ứng dụng Silverlight sẽ giúp ích rất nhiều khi chuyển sang các ứng dụng WPF.

[Giới thiệu thêm về Scott Gu](#): Scott Gu hiện là phó chủ tịch tập đoàn của Microsoft phụ trách bộ phận phát triển .NET. Anh là một trong những người xây dựng nên dự án .NET, và đóng một vai trò quan trọng trong việc thiết kế và phát triển .NET.

Hiện tại, anh là người quản lý trực tiếp các nhóm phát triển CLR, ASP.NET, Silverlight, WPF, IIS, Commerce Server và các công cụ Visual Studio cho Web, Client và Silverlight.

Tham khảo:

- First Look at Silverlight 2: <http://weblogs.asp.net/scottgu/archive/2008/02/22/first-look-at-silverlight-2.aspx>
- Microsoft® Silverlight™ 2 Tools for Visual Studio 2008 SP1: <http://www.microsoft.com/downloads/details.aspx?familyid=C22D6A7B-546F-4407-8EF6-D60C8EE221ED&displaylang=en>