Asynchrones Downloaden eines Bildes mit Anzeige eines Progressbars

Nach dem ich schon gezeigt habe, wie man eine asynchrone ImageView unter MonoTouch realisiert, hier jetzt der XAML-Ansatz. Getestet mit Windows Phone 7, aber sollte auch in Silverlight laufen:

Als erstes brauchen wir mal den XAML-Teil

Quellcode

```
<StackPanel>
<ProgressBar x:Name="imagePerformanceProgressBar" Foreground="#FF41A50F" IsIndeterminate="False" Value="{Binding Image.DownloadProgress}" Visibility="{Binding Image.IsLoading, Converter={StaticResource VisibilityConverter}}" HorizontalAlignment="Left" Width="150" Height="20" VerticalAlignment="Center" />
<TextBlock Text="Laden..." Foreground="Black" FontSize="12" VerticalAlignment="Center" HorizontalAlignment="Left" Margin="10,20,0,0" Visibility="{Binding Image.IsLoading, Converter={StaticResource VisibilityConverter}}"/>
<Image Source="{Binding Image.Image}" VerticalAlignment="Bottom" ImageOpened="OnImageOpened" Stretch="None"/>
</StackPanel>
```

Diesen Binden wir an unser ImageViewModel

Quellcode

```
using System;
using System.ComponentModel;
using System.Windows.Media.Imaging;

namespace your.space
{
/// <summary>
/// Class representing a ViewModel of an image
/// </summary>
public class ImageViewModel : INotifyPropertyChanged
{
#region Members

bool isLoading;
int progress;
#endregion

#region Constructors
/// <summary>
/// Initializes a new instance of the <see cref="ImageViewModel"/> class.
/// </summary>
/// <param name="imageUrl">The image URL.</param>
public ImageViewModel(string imageUrl)
: this(imageUrl, null) { }

/// <summary>
/// Initializes a new instance of the <see cref="ImageViewModel"/> class.
/// </summary>
/// <param name="imageUrl">The image URL.</param>
/// <param name="fallbackImage">The fallback image.</param>
public ImageViewModel(string imageUrl, BitmapImage fallbackImage)
{
if (string.IsNullOrEmpty(imageUrl))
{
```

```
37. Image = fallbackImage;
38. IsLoading = false;
39. return;
40. }
41. IsLoading = true;
42. Image = new BitmapImage();
43. Image.ImageOpened += (s, e) => { IsLoading = false; };
44. Image.DownloadProgress += (s, e) => { DownloadProgress = e.Progress; };
45. Image.ImageFailed += (s, e) =>
46. {
47. IsLoading = false;
48. Image = FallbackImage;
49. OnPropertyChanged("Image");
50. };
51. Image.UriSource = new Uri(imageUrl, UriKind.RelativeOrAbsolute);
52. FallbackImage = fallbackImage;
53. }
54. #endregion
55. #region Properties
56. /// <summary>
57. /// Gets or sets a value indicating whether this instance is loading.
58. /// </summary>
59. /// <value>
60. /// <c>true</c> if this instance is loading; otherwise, <c>false</c>.
61. /// </value>
62. public bool IsLoading
63. {
64. get { return isLoading; }
65. private set
66. {
67. if (value == isLoading)
68. return;
69. isLoading = value;
70. OnPropertyChanged("IsLoading");
71. }
72. }
73. /// <summary>
74. /// Gets or sets the download progress.
75. /// </summary>
76. /// <value>The download progress.</value>
77. public int DownloadProgress
78. {
79. get { return progress; }
80. private set
81. {
82. if (value == progress)
83. return;
84. progress = value;
85. OnPropertyChanged("DownloadProgress");
86. }
87. }
88. /// <summary>
89. /// Gets the image.
90. /// </summary>
91. /// <value>The image.</value>
92. public BitmapImage Image { get; private set; }
93. /// <summary>
94. /// Gets or sets the fallback image.
```

```
107.  /// </summary>
108.  /// <value>The fallback image.</value>
109.  public BitmapImage FallbackImage
110.  {
111.  get;
112.  private set;
113.  }
114.  #endregion
115.  #region Events
116.  /// <summary>
117.  /// Called when [property changed].
118.  /// </summary>
119.  /// <param name="propertyname">The propertyname.</param>
120.  protected virtual void OnPropertyChanged(string propertyname)
121.  {
122.  if (PropertyChanged != null)
123.  PropertyChanged(this, new PropertyChangedEventArgs(propertyname));
124.  }
125.  /// <summary>
126.  /// Occurs when a property value changes.
127.  /// </summary>
128.  public event PropertyChangedEventHandler PropertyChanged;
129.  #endregion
130.  }
131.  }
```

Alles anzeigen

Wer den Code aufmerksam liest, erkennt, das es in dieser Version zusätzlich möglich ist, ein Fallback Image anzuzeigen, falls der Download schief läuft.

Und damit der Progressbar und der Platzhaltertext auch ein- bzw ausgeblendet wird brauchen wir noch den Visibility Converter. Dieser muss dann natürlich noch als Resource im XAML eingebunden werden:

Quellcode

```
1.  using System;
2.  using System.Globalization;
3.  using System.Windows;
4.  using System.Windows.Data;
5.  namespace your.space
6.  {
7.  /// <summary>
8.  /// Converter Class
9.  /// </summary>
10.  public class VisibilityConverter : IValueConverter
11.  {
12.  #region IValueConverter Members
13.  /// <summary>
14.  /// Modifies the source data before passing it to the target for display in the UI.
15.  /// </summary>
16.  /// <param name="value">The source data being passed to the target.</param>
17.  /// <param name="targetType">The <see cref="T:System.Type"/> of data expected by the target dependency property.</param>
18.  /// <param name="parameter">An optional parameter to be used in the converter logic.</param>
19.  /// <param name="culture">The culture of the conversion.</param>
20.  /// <returns>
```

```
23. /// The value to be passed to the target dependency property.
24. /// </returns>
25. public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
26. {
27. return ((bool) value) ? Visibility.Visible : Visibility.Collapsed;
28. }
29. /// <summary>
30. /// Modifies the target data before passing it to the source object. This method is called only in <see
    cref="F:System.Windows.Data.BindingMode.TwoWay"/> bindings.
31. /// </summary>
32. /// <param name="value">The target data being passed to the source.</param>
33. /// <param name="targetType">The <see cref="T:System.Type"/> of data expected by the source object.</param>
34. /// <param name="parameter">An optional parameter to be used in the converter logic.</param>
35. /// <param name="culture">The culture of the conversion.</param>
36. /// <returns>
37. /// The value to be passed to the source object.
38. /// </returns>
39. public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
40. {
41. return null;
42. }
43. #endregion
44. }
45. }
```

Alles anzeigen