Den Pictures Hub in eigenen Applikationen nutzen

Bilder bearbeiten im Telefon

Ein Windows Phone ist auch eine Kamera. Und zwar eine ziemlich gute, weil Microsoft eine Reihe von Hardwarevoraussetzungen vorschreibt. Wer will, kann seine Bilder mit diesem Gerät auch gleich bearbeiten. dotnetpro zeigt, wie Sie eine eigene Applikation für die Bildbearbeitung erstellen und diese als Erweiterung im Pictures Hub registrieren.

Auf einen Blick

René Schulte ist .NET-, Silverlight- und Windows-Phone-Entwickler, Dipl.-Inf. (FH) und Microsoft Silverlight MVP. Er ist Blogger, Autor und Entwickler verschiedener Open-Source-Projekte und hat die Windows-Phone-Bildeffekte-App Pictures Lab geschrieben. Auf Twitter finden Sie ihn unter dem Kürzel @rschu.

Inhalt

- Bilder laden, ändern, speichern.
- Die Multi-Touch-Oberfläche nutzen.
- Eigene Applikationen in den Pictures Hub integrieren.

Grundlagen

- Gestaltungsrichtlinien für Windows Phone 7, www.dotnetpro.de/A1011Metro

dnpCode

A1102PictureHub

Joe Belfiore, Corporate Vice President, Windows Phone Program Management, sagte: „Windows Phones are going to be absolutely amazing devices for taking pictures and dealing with pictures.“ Und er hat recht, denn Microsoft schreibt den Hardwareherstellern Mindestvoraussetzungen vor. So muss jedes Windows Phone mindestens eine 5-Megapixel-Kamera mit Blitzlicht haben. Zudem muss ein kleiner Hardwareknopf für den Kameraauslöser vorhanden sein. Dieser Knopf ermöglicht auch die Pocket-to-Picture-Funktion. Damit ist es möglich, die Kameraapplikation blitzschnell zu öffnen, auch wenn das Telefon gesperrt ist. So verpasst man nie wieder die besten Schnappschüsse, da man erst das Telefon entsperren und danach die Kameraapplikation öffnen muss.

Bei Windows Phones wurde also auch auf eine State-of-the-Art-Kamerafunktionalität geachtet. Zentraler Bestandteil ist der sogenannte Pictures Hub, die integrierte Bildapplikation und Fotobibliothek. Der Nutzer kann hier Bilder betrachten und öffnen. Für den Entwickler ist dieser Hub besonders interessant. Von hier aus können Bilder geladen werden, und es ist möglich, einen Einstiegspunkt für die eigene App zu registrieren.

Die Features und APIs lassen sich am besten an einer Beispielapplikation erklären. Das folgende Beispiel zeigt, wie Sie ein Bild aus dem Pictures Hub laden, Bearbeitungen mittels Multi-Touch vornehmen und das Bild im Anschluss zurück in den Pictures Hub schreiben. Dabei werden auch die vielen kleinen Dinge und Tricks erklärt, die eine Applikation zu einer richtigen App machen. Der vollständige Quellcode der App befindet sich auf der Heft-CD.

[Abb. 1] Ein Bild bearbeiten.

Die Oberfläche

Das Grundgerüst des Beispiels ist eine Windows-Phone-Silverlight-Applikation (Abbildung 1). Das User Interface richtet sich nach den offiziellen Windows-Phone-Designrichtlinien [1] [2].

Mit Windows Phone hat es Microsoft vor allem auf den Verbrauchermarkt abgesehen. Nicht nur deswegen ist es wichtig, besonderes Augenmerk auf das Design und die User Experience (UX) zu legen. Im XAML der *MainPage* wird die Oberfläche der Hauptseite definiert, siehe Listing 1.

Der *TextBlock* mit dem Namen *PageTitle* enthält den Titel der Page. Als Style wird hier die statische Ressource *PhoneTextTitle1Style* verwendet, die das Windows Phone SDK bereitstellt. Windows Phone unterstützt verschiedene Themes, welche vom Nutzer angepasst werden können. Für die beste User Experience empfiehlt es sich, keine fest codierten Werte zu verwenden. Sie finden die vordefinierten Styles in der Datei: *%ProgramFiles%\SDKs\Windows Phone\v7.0\Design\ThemeResources.xaml*.

Auf der Page befindet sich auch ein *Image*-Control zur Anzeige des gewählten Bildes. Das *Canvas* wird später die Bearbeitungen enthalten, welche unter anderem in den Eventhandlern *Viewport_MouseMove* und *Viewport_MouseLeftButtonDown* vorgenommen werden. Im unteren Bereich der Page befindet sich die *ApplicationBar*. Sie wird ebenfalls im XAML der *MainPage* definiert, siehe Listing 2.

Die Application Bar des Beispiels enthält vier Icon-Buttons für das Laden und Sichern eines Bil-

Die Application Bar

Die Windows-Phone-Application-Bar ist ein nützliches Control, welches normalerweise als Toolbar verwendet wird. Die Application Bar kann bis zu vier Icon-Buttons und eine Reihe von Menu-Items enthalten. Die ApplicationBar ist ein natives System-Control. Ihr Silverlight-Wrapper unterstützt in der derzeitigen Fassung leider kein DataBinding. Auch kann keine Auflösung des *x:Name*-Attributs der Items vorgenommen werden, und eine Membervariable ist demzufolge immer *null*. Wird dynamischer Inhalt benötigt, empfiehlt es sich, das Control im Code-behind anzulegen, um eine gültige Referenz auf die Items zu erhalten. Diese Problematik ist Microsoft bekannt, und es ist damit zu rechnen, dass eine der nächsten Windows-Phone-SDK-Versionen eine bessere Variante bereitstellt.

des, für eine Rückgängigfunktion und für das Hinzufügen einer Bitmapgrafik. Als Icons werden drei selbst erstellte Icons und ein Standardicon verwendet. Die 32 vom Windows Phone SDK mitgelieferten Standardicons finden Sie im Ordner *%ProgramFiles%\SDKs\Windows Phone\v7.0\Icons*. Die Icons liegen dort als PNG-Dateien jeweils im Light-or-Dark-Theme und im Adobe-Illustrator-Vektorformat vor.

Um die PNG-Icons verwenden zu können, müssen Sie sie dem Visual-Studio-Projekt hinzufügen. Dabei ist es wichtig, in den Eigenschaften der Datei die *BuildAction* auf *Content* zu setzen. Standardmäßig wird hier ansonsten *Resource* verwendet.

Code-behind

Die UI ist vorbereitet. Nun ist es an der Zeit, die App mit Leben zu füllen. Um nicht von den wesentlichen Punkten abzulenken, wurde die Applikation so einfach und verständlich wie möglich gehalten. Aus diesem Grund wurde die Programmfunktionalität direkt im Code-behind implementiert. Für eine (größere) Real-Life-App empfiehlt es sich, bei der Entwicklung nach dem Model-View-ViewModel-Muster vorzugehen [3] [4] [5].

Als Erstes wird ein Bild in der Applikation benötigt. Um ein Bild aus dem Pictures Hub zu erhalten, wird eine Instanz der Klasse *PhotoChooserTask* verwendet, siehe Listing 3. Im Konstruktor der *MainPage* wird die *PhotoChooserTask*-Membervariable initialisiert und der *PhotoChooserTaskCompleted*-Eventhandler an das *Completed*-Ereignis angehängt. Betätigt der Nutzer den ersten Icon-Button (Open) der Application Bar, wird der *PhotoChooserTask* mittels der *Show*-Methode geöffnet, siehe Listing 4.

Daraufhin öffnet sich der Pictures Hub, und der Anwender kann ein Foto auswählen. Durch die Angabe von *ShowCamera* =

Listing 1

Oberfläche der Hauptseite.

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <StackPanel Margin="12,17,0,28" d:IsHidden="True">
        <TextBlock Name="PageTitle" Text="Fail Uploader"
            Style="{StaticResource PhoneTextTitle1Style}"/>
        <Grid Name="Viewport" Grid.Row="1" MouseMove="Viewport_MouseMove"
            MouseLeftButtonDown="Viewport_MouseLeftButtonDown">
            <Image Name="Image" VerticalAlignment="Top" HorizontalAlignment="Left" />
            <Canvas Name="DrawCanvas" />
        </Grid>
    </StackPanel>
</Grid>
```

Listing 2

Die Befehlsleiste definieren.

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True">
        <shell:ApplicationBarIconButton IconUri="/data/appbar/FolderCamera.png"
                Text="Open" Click="ApplicationBarIconOpenButton_Click" />
        <shell:ApplicationBarIconButton
                IconUri="/data/appbar/appbar.save.rest.png" Text="Save"
                Click="ApplicationBarIconSaveButton_Click" />
        <shell:ApplicationBarIconButton IconUri="/data/appbar/undo.png"
                Text="Undo" Click="ApplicationBarIconUndoButton_Click"  />
        <shell:ApplicationBarIconButton IconUri="/data/appbar/monkey.png"
                Text="Monkey!" Click="ApplicationBarIconClipArtButton_Click"  />
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Listing 3

Ein Bild aus dem Pictures Hub laden.

```
private readonly PhotoChooserTask photoChooserTask;
public MainPage() {
   InitializeComponent();
   // Init chooser
   photoChooserTask = new PhotoChooserTask {ShowCamera = true};
   photoChooserTask.Completed += PhotoChooserTaskCompleted;
}
private void PhotoChooserTaskCompleted(object sender, PhotoResult e) {
   Load(e);
}
```

Listing 4

Ein Foto anzeigen.

```
private void ApplicationBarIconOpenButton_Click(object sender, EventArgs e) {
   photoChooserTask.Show();
}
```

Listing 5

Ein Bild laden.

```
private void Load(PhotoResult photoResult){
  if (photoResult.TaskResult == TaskResult.OK){
     LoadImage(photoResult.ChosenPhoto);
  }
}
private void LoadImage(Stream stream) {
//Load original image and invalidate bitmap
//so it gets newly rendered when used
  var bitmapImage = new BitmapImage();
  bitmapImage.SetSource(stream);
  Image.Source = bitmapImage;
}
```

true wird ein kleines Kamerasymbol im Pictures Hub angezeigt. Hiermit kann der Nutzer auch direkt ein neues Foto aufnehmen, ohne dass zusätzlich ein *CameraCaptureTask* in der App implementiert werden muss. Ist die Auswahl abgeschlossen, kehrt das System zu der App zurück und löst das Ereignis *Completed* aus. Die *Load*-Methode erhält als Parameter das *PhotoResult* aus dem *Completed*-Eventhandler, siehe Listing 5. Falls die Auswahl nicht abgebrochen wurde, wird der JPEG-Stream des ausgewählten Bildes an ein *BitmapImage* übergeben. Diese Klasse verarbeitet die Bilddaten und stellt diese für das Silverlight-*Image*-Control als *Source* zur Anzeige bereit.

Das Bild bearbeiten

Mit diesen wenigen Zeilen Code ist es möglich, ein Bild zu laden. Das Bearbeiten eines Bildes ist nicht wesentlich schwieriger. Die Beispielapplikation ermöglicht es dem Nutzer, durch rote Striche bestimmte Bereiche hervorzuheben, siehe Abbildung 1. Das Zeichnen wird in den Maus-Eventhandlern des *Viewport*-Grids ermöglicht. Berührt der Nutzer mit seinem Finger das Display, wird die *AddPolyline*-Methode gerufen, siehe Listing 6. In dieser Methode wird eine Silverlight-*Polyline* erstellt und dem *DrawCanvas* hinzugefügt. Eine *Polyline* ist ein Linienzug, also eine Verbindung mehrerer einzelner Geraden. Der Linienzug soll mit einem roten Pinsel und der Pinselstärke von zehn Pixeln gezeichnet werden und abgerundete Enden besitzen. Bewegt der Nutzer seinen Finger über das Display, wird das *MouseMove*-Ereignis ausgelöst. In der Methode *AddPointToPolyline* wird der Polyline dann ein neuer Punkt hinzugefügt. Hebt der Nutzer seinen Finger und setzt ihn erneut auf, wird ein neues *MouseLeftButtonDown*-Ereignis ausgelöst und somit eine neue Polyline erzeugt.

Multi-Touch Monkey

Rote Linien sind nett, um bestimmte Dinge hervorzuheben; man kann damit auch Fotos verunstalten und Spaß haben. Noch mehr Freude kommt aber auf, wenn man dem Bild zum Beispiel einen Affenkopf hinzufügt, siehe Abbildung 2. In der Beispielapplikation kann der Nutzer einen freigestellten Affenkopf mit dem entsprechenden *Application Bar Button* hinzufügen. Dieses Affenbild kann dann mittels Multi-Touch verschoben, rotiert und skaliert werden, siehe Listing 7.

Das Beispielprojekt enthält ein PNG-Bild eines freigestellten Orang-Utan-Kopfes. Dieses Bild wird dynamisch zur Laufzeit geladen und einem neu erstellten *Image*-

Listing 6

Linien zeichnen.

```
private void Viewport_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) {
   AddPolyline();
}
private void Viewport_MouseMove(object sender, MouseEventArgs e) {
   AddPointToPolyline(e.GetPosition(Viewport));
}
private Polyline polyline;
private void AddPolyline() {
   polyline = new Polyline {
      Stroke              = new SolidColorBrush(Colors.Red),
      StrokeThickness     = 10,
      StrokeStartLineCap  = PenLineCap.Round,
      StrokeEndLineCap    = PenLineCap.Round,
   };
   DrawCanvas.Children.Add(polyline);
}
private void AddPointToPolyline(Point point) {
   if (polyline != null) {
      polyline.Points.Add(point);
   }
}
```

WriteableBitmap

Die *WriteableBitmap* ist eine einfache Silverlight-Klasse, die eine Ansammlung von Pixeln als Array von int32-Werten enthält. Jedes Pixel ist aus vier Bytes zusammengesetzt. Das erste Byte des int32-Wertes entspricht dem Alpha-Kanal (Transparenz), das zweite Byte dem Rot-Kanal, das dritte der Grün-Komponente und das vierte dem Blau-Anteil. Ein Pixel beinhaltet somit die Farbkomponenten als ARGB-Werte.

Die *WriteableBitmap*-Klasse besitzt neben dem int32-Array *Pixels* noch die Properties *PixelWidth* und *PixelHeight*, welche die Dimension des Pixelbildes angeben.

An dieser Stelle sei auf die Open-Source-Bibliothek *WriteableBitmapEx* des Autors hingewiesen. Diese enthält eine Reihe nützlicher Erweiterungsmethoden für die ansonsten rudimentäre *WriteableBitmap*-Klasse. [7]

[Abb. 2] Einen Affenkopf hinzufügen.

Emulator versus Gerät

Ein moderner PC mit Emulator ist deutlich performanter als ein Windows Phone. Um böse Überraschungen zu vermeiden, sollte man jedoch generell jede Anwendung vor der Veröffentlichung auf einem Gerät getestet haben.

Um eine Applikation auf ein Gerät zu kopieren, muss man dieses für die Entwicklung freischalten. Dafür liefert das SDK das Tool „Windows Phone Developer Registration" mit. Jeder Entwickler kann standardmäßig drei Geräte im Entwicklerportal App Hub registrieren (http://create.msdn.com).

Das Ausführen von Code auf einem Gerät unterscheidet sich nicht wesentlich vom Debuggen mit dem Emulator. Um den Code auf das Gerät zu deployen, muss allerdings die Zune Software ausgeführt werden. Ist die Zune Software aktiv, kann eine App aber nicht auf die Medienfunktionalität, wie etwa den Pictures Hub, zugreifen. Somit ist kein Debuggen dieser Codebestandteile möglich. Um diese Funktionalität dennoch debuggen zu können, liefert das SDK das Tool „Windows Phone Connect" mit. Dieses Tool stellt eine Verbindung zum Gerät ohne die Zune Software bereit und ermöglicht dadurch das Debuggen aller Funktionen.

Listing 7

Einen Affenkopf hinzufügen.

```
private void ApplicationBarIconClipArtButton_Click(object sender, EventArgs e) {
   AddMonkeyHead();
}
private void AddMonkeyHead() {
   using (var bmpStream = Application.GetResourceStream
    (new Uri("DnpPicHubFun;component/data/monkey.png", UriKind.Relative)).Stream){
      // Monkey-PNG als Image-Control hinzufügen
      var bmpi = new BitmapImage();
      bmpi.SetSource(bmpStream);
      var image = new Image { Source = bmpi };
      DrawCanvas.Children.Add(image);
      // Dem Image das MultiTouchBehavior hinzufügen
      var behavior = new MultiTouchBehavior { IsInertiaEnabled = false, MaximumScale = 500 };
      Interaction.GetBehaviors(image).Add(behavior);
      behavior.Move(new Point(240, 400), 0, 150);
   }
}
```

Control im *DrawCanvas* hinzugefügt. Um das Bild auf diese einfache Art und Weise laden zu können, ist es wichtig, dass in den Eigenschaften der PNG-Datei die *Build-Action* auf *Resource* eingestellt ist.

Im zweiten Schritt wird die Multi-Touch-Funktionalität über ein spezielles Behavior ergänzt, und das Bild wird durch die *Move*-Methode auf dem Display zentriert.

Über ein Behavior können Sie einem Element nachträglich bestimmte Funktionalität hinzufügen. Dies ist vor allem für Designer und im Zusammenhang mit Expression Blend interessant, da Behaviors vorrangig in XAML verwendet werden und keine Programmierung erfordern.

Das im Beispiel verwendete *MultiTouch-Behavior* wurde von den Silverlight-MVPs Davide Zordan und Laurent Bugnion entwickelt. Das Behavior ist Bestandteil einer Open-Source-Bibliothek für Multi-Touch, die Sie unter http://multitouch.codeplex.com finden. Das Behavior ist sehr einfach zu verwenden und kann mittels zahlreicher Properties angepasst werden.

Undo

Mithilfe der Rückgängigfunktion kann der Nutzer den zuletzt gezeichneten Linienzug oder Affenkopf aus dem *DrawCanvas* entfernen, siehe Listing 8.

Der Code prüft, ob an dem zuletzt hinzugefügten *DrawCanvas*-Unterelement ein Behavior vorhanden ist, und entfernt dieses gegebenenfalls.

Das Bild sichern

Nachdem der Anwender das Bild verschönert hat, will er es zurück in den Pictures Hub sichern. Leider gibt es im Windows Phone Silverlight SDK keinen *PhotoSaveTask* oder etwas Ähnliches. Glücklicherwei-

PERFECTION IN SOFTWARE PROTECTION
CodeMeter: Sicherer & flexibler Softwareschutz!

- Hardwarebasiert mit CodeMeter in vielen Bauformen: USB, PC-, Express-, CF-, SD- und µSD-Card
- Softwarebasiert durch Aktivierung mit CodeMeterAct
- Server-, Desktop- und Embedded-Betriebssysteme
- Flexibles und umfangreiches Lizenzmanagement: Netzwerk, Einzelplatz, Pay-Per-Use, Overflow, u.v.m.
- Integration in Vertriebsprozesse und Backend

MEDIA ACCESS
PERFECTION IN SOFTWARE PROTECTION
DOCUMENT

Secure Code Seminar
Roadshow 2011
Besuchen Sie unser kostenloses Secure Code Seminar!
Alle Termine & Informationen finden Sie unter:
www.wibu.de
Bestellen Sie noch heute Ihr kostenloses SDK!
WIBU-SYSTEMS AG
Rüppurrer Straße 52-54
D-76137 Karlsruhe
Tel.: 0721-93172-0
www.wibu.de
WIBU SYSTEMS

Listing 8

Kommando zurück.

```
private void Undo() {
   if (DrawCanvas != null && DrawCanvas.Children.Count > 0) {
      // Behavior entfernen, falls vorhanden
      var uiElement = DrawCanvas.Children[DrawCanvas.Children.Count - 1];
      var behaviors = Interaction.GetBehaviors(uiElement);
      if (behaviors.Count > 0) {
         behaviors.Clear();
      }
      // Element entfernen
      DrawCanvas.Children.Remove(uiElement);
   }
}
```

Listing 9

Die geänderte Datei speichern.

```
private void ApplicationBarIconSaveButton_Click(object sender, EventArgs e) {
   Save();
}
private void Save() {
   using (var stream = new MemoryStream()) {
      // Rendern
      var bitmap = new WriteableBitmap(Viewport, null);
      // Bitmap als JPEG in den Stream encodieren
      bitmap.SaveJpeg(stream, bitmap.PixelWidth, bitmap.PixelHeight, 0, 100);
      stream.Seek(0, SeekOrigin.Begin);
      // Bild in die Medienbibliothek schreiben
      var name = String.Format("MyApp_{0:yyyy-MM-dd_hh-mm-ss-tt}.jpg", DateTime.Now);
      new MediaLibrary().SavePicture(name, stream);
   }
}
```

Listing 10

Den PageTitle ausblenden.

```
private void PhoneApplicationPage_OrientationChanged(object sender,
OrientationChangedEventArgs e) {
   if ((e.Orientation & PageOrientation.Landscape) == PageOrientation.Landscape) {
      PageTitle.Visibility = Visibility.Collapsed;
   } else {
      PageTitle.Visibility = Visibility.Visible;
   }
}
```

se kann man Windows Phones aber auch mit dem XNA-Framework programmieren. In der Assembly *Microsoft.Xna.Framework* finden Sie auch die hier benötigte *MediaLibrary*-Klasse. Diese Klasse besitzt unter anderem eine *SavePicture*-Methode, die einen JPEG-Datenstrom in den Pictures Hub schreibt. Alle gesicherten Bilder werden im Pictures-Album *Gespeicherte Bilder* abgelegt. Es gibt zurzeit keine Möglichkeit, ein anderes Zielalbum anzugeben. Um überhaupt Bilder in die Medienbibliothek schreiben zu können, muss die Applikation die Capability *ID_CAP_MEDIALIB* in der Datei *WMAppManifest.xml* angeben. Wird diese Capability nicht angegeben, löst ein Aufruf der Methode *SavePicture* eine Exception aus. Wie in Listing 9 gezeigt, löst der Anwender den Sichern-Vorgang aus, indem er den Diskettenbutton betätigt.

Als Erstes wird der Inhalt des *Viewport*-Grids in eine *WriteableBitmap* gerendert. Das Grid beinhaltet das *Image*-Control und den *DrawCanvas*. Beide Elemente werden somit zusammen in eine *WriteableBitmap* geschrieben. Wichtig ist, dass der *DrawCanvas* eine Ebene über dem *Image*-Control liegt. Ansonsten wären die Linienzüge oder der Affenkopf nicht sichtbar.

Das eigentliche Rendern erfolgt mit einem der drei *WriteableBitmap*-Konstruktoren. Alternativ könnte man auch die *WriteableBitmap.Render*-Methode mit den gleichen Parametern verwenden.

Als zweiter Schritt wird die *WriteableBitmap* mithilfe einer Erweiterungsmethode in einen JPEG-Stream übertragen. Das gerenderte Pixelbild wird in der vollen Auflösung in bester Qualität encodiert. Die Qualitätsangabe muss im Bereich von 0 bis 100 liegen, wobei 100 der besten Qualität entspricht. Eine hohe Qualität bedeutet aber auch eine größere Datei. JPEG ist ein verlustbehaftetes Codierungsverfahren, und es ist nicht empfehlenswert, eine Qualität unter 70 zu verwenden. Die *SaveJpeg*-Erweiterungsmethode ist Bestandteil der Windows Phone Silverlight Runtime und Teil des Namespaces *System.Windows.Media.Imaging*. Im letzten Schritt wird die *SavePicture*-Methode der *MediaLibrary*-Klasse verwendet, um den JPEG-Stream in die Medienbibliothek zu schreiben.

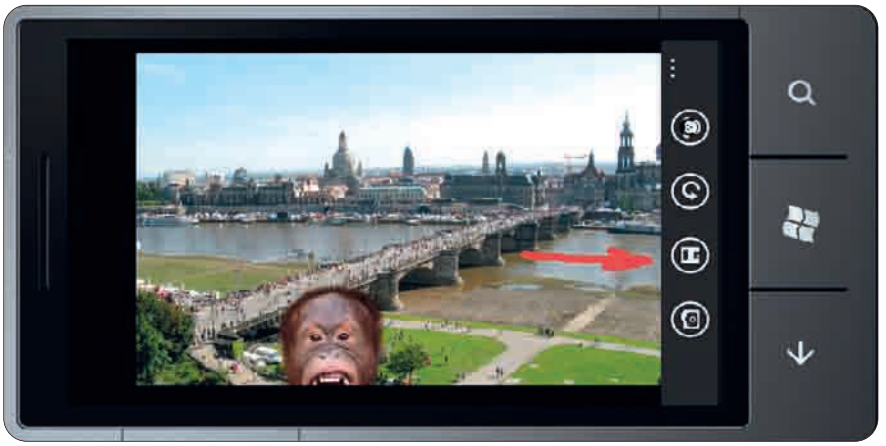
Die Orientierung behalten

Alle aktuellen Windows Phones müssen eine Auflösung von 480 x 800 Pixeln besitzen. Normalerweise wird das Gerät im Hochformatmodus (Portrait) verwendet. Zum Schreiben von Texten ist aber das Querformat (Landscape) besser geeignet. Das Gleiche trifft auch auf das Betrachten von Bildern zu, welche im Querformat aufgenommen wurden. Die meisten Anwender sind daran gewöhnt, dass die Oberfläche der Applikation auch um 90° rotiert, wenn sie das Gerät drehen, siehe Abbildung 3.

Die meisten Fotos werden im Querformat aufgenommen, deswegen ist die Möglichkeit, das Foto zu drehen, essenziell für eine Foto-App. Glücklicherweise ist dies relativ einfach zu implementieren. Im XAML der Page muss in diesem Fall die Property *SupportedOrientations* vom Standardwert *Portrait* einfach nur auf *PortraitOrLandscape* gesetzt werden.

```
SupportedOrientations="PortraitOrLandscape"
```

Um noch mehr Platz für das Foto im Landscape-Modus zu gewinnen, empfiehlt



[Abb. 3] Darstellung im Landscape-Modus.

es sich, den *PageTitle* auszublenden. Dazu können Sie das *OrientationChanged*-Event verwenden. Dieses Event wird ausgelöst, wenn das Telefon rotiert wurde, siehe Listing 10.

Integration in den Pictures Hub

Eins der besonderen Highlights von Windows Phone ist die Möglichkeit, den Hub durch Drittapplikationen zu erweitern. Wählt der Anwender beispielsweise ein Foto im Pictures Hub aus, kann dieser direkt von dort aus eine Applikation starten. Der Anwendung wird dann beim Aufruf das ausgewählte Foto übergeben. Der Anwender kann somit direkt aus dem Kontext heraus die App öffnen und muss nicht erst die Anwendung manuell starten und dort ein Bild auswählen.

Für den Pictures Hub gibt es zum einen die Share Picker Extension [8] und zum anderen die Extras Extension [9]. Eine Share Picker Extension wird im *Share...*-Menüeintrag des Bildes ergänzt und soll dem Hochladen und Verteilen von Fotos dienen. Hier könnte sich beispielweise eine Flickr-Upload-App registrieren. Die Extras Extension ist für Anwendungen zur Bildmanipulation gedacht. Registrierte Applikationen werden innerhalb des *Extras*-Eintrags im Kontextmenü des Fotos angezeigt.

Um eine eigene Anwendung dafür zu registrieren, müssen Sie dem Projekt eine *Extras.xml* hinzufügen. Die *BuildAction* der Datei muss auf *Content* (Inhalt) gesetzt sein und den Inhalt aus Listing 11 enthalten.

Zum Konsumieren des übergebenen Fotos müssen Sie innerhalb der *MainPage* die *OnNavigatedTo*-Methode überschreiben. Den Code dazu finden Sie auf der Heft-CD.

Der Applikation wird eine Referenz zum ausgewählten Bild mittels eines Tokens übergeben. Dieser Token wird verwendet, um das Bild mittels der *MediaLibrary.GetPictureFromToken*-Methode abzurufen. Der daraus resultierende JPEG-Stream wird wieder dem Silverlight-*Image*-Control als *Source* zur Anzeige bereitgestellt.

Zusammenfassung

Aufgrund der Hardwarevoraussetzungen bietet die Windows-Phone-Plattform sehr gute Fotofunktionen. Der Zugriff auf die Bilder aus dem Pictures Hub kann über einen Chooser erfolgen; die Applikation kann sich aber auch im Pictures Hub registrieren und direkt von dort gestartet werden. Die Multi-Touch-Oberfläche erlaubt eine intuitive Manipulation von Bildern. Dies alles lässt sich mit relativ wenig Aufwand umsetzen. **[ml]**

[1] UI Design and Interaction Guide for Windows Phone 7, www.dotnetpro.de/SL1102Architektur1

[2] Bernhard Pichler, Klarheit auf engstem Raum, Gestaltungsrichtlinien für Windows Phone 7, dotnetpro 11/2010, Seite 24 ff., www.dotnetpro.de/A1011Metro

[3] Jeremy Likness, Model-View-ViewModel (MVVM) Explained, www.dotnetpro.de/SL1102Architektur2

[4] Golo Roden, Endlich sauber getrennt, Vom Model-View-Controller zum Model-View-ViewModel, dotnetpro 8/2008, Seite 129 ff., www.dotnetpro.de/A0808MVVM

[5] Holger Fleck, ViewModel für Silverlight, Das MVVM-Pattern mit Silverlight umsetzen, dotnetpro 9/2009, Seite 42 ff., www.dotnetpro.de/A0909MVVM

[6] Execution Model Overview for Windows Phone, www.dotnetpro.de/SL1102Architektur3

[7] WriteableBitmapEx, www.dotnetpro.de/SL1102Architektur4

[8] Share Picker Extensibility for Windows Phone, www.dotnetpro.de/SL1102Architektur5

[9] Photo Extras Application Extensibility for Windows Phone, www.dotnetpro.de/SL1102Architektur6

[10] René Schulte, PicFx – Windows Phone Picture Effects Application, www.dotnetpro.de/SL1102Architektur7

Listing 11

Extras hinzufügen.

```
<?xml version="1.0" encoding="utf-8" ?>
<Extras>
  <PhotosExtrasApplication>
    <Enabled>true</Enabled>
  </PhotosExtrasApplication>
</Extras>
```

Launchers und Choosers

Aus Sicherheitsgründen wird jede Windows-Phone-Applikation in einer isolierten Sandbox ausgeführt. Dadurch können Applikationen nicht direkt auf Ressourcen des Geräts zugreifen. Für den Zugriff auf die Ressourcen, wie etwa die Bildbibliothek, werden daher sogenannte Launchers und Choosers bereitgestellt. Launchers und Choosers sind eigenständige Applikationen.

Ein Launcher startet eine integrierte Applikation, welche keine Daten an die aufrufende Applikation zurückliefert. Als Beispiel sei der *EmailComposeTask* genannt. Dieser öffnet die eingebaute E-Mail-App. Ein Chooser hingegen liefert Daten an die aufrufende Applikation zurück. Beispielsweise öffnet der *CameraCaptureTask* die Kameraapplikation und liefert als Ergebnis ein neu aufgenommenes Foto an die aufrufende App. Sie finden die Launchers und Choosers im Namespace *Microsoft.Phone.Tasks*.

Die aktuell vorhandene Windows-Phone-Version verhindert, dass mehrere Apps parallel ausgeführt werden. Verwendet eine App einen Launcher oder Chooser, wird somit die rufende App beendet, in den sogenannten Tombstone-Modus versetzt und faktisch beendet. Dies ist ein essenzielles Konzept der Windows-Phone-Entwicklung.

Allerdings beendet nicht jeder Launcher oder Chooser zwangsläufig die eigene App. Aus Performancegründen beenden bestimmte „lite Tasks" die App nur dann, wenn etwa die Speicherauslastung zu groß ist. Folgende „lite Tasks" sind zurzeit bekannt: *PhotoChooserTask, CameraCaptureTask, MediaPlayerLauncher, EmailAddressChooserTask, PhoneNumberChooserTask,* Multiplayer Game Invite (XNA), Gamer You Card (XNA). Weitere Informationen finden Sie unter [6].