Microsoft Press PreView

Unter dem Label *Microsoft Press PreView* veröffentlicht Microsoft Press kostenlos Vorabkapitel ausgewählter Entwicklertitel. Sie können diese Kapitel online auf msdn lesen oder sie herunterladen. Pro Quartal wird das Angebot um ein neues Buch ergänzt, bereits vorgestellte Bücher werden durch neue Kapitel aktualisiert. Sie haben auch die Möglichkeit, die Vorabkapitel zu bewerten. Hier geht es zu allen Inhalten von *Microsoft Press PreView*: *http://www.microsoft.com/germany/msdn/knowhow/press/*

Mehr Informationen zu ***Entwickeln für Windows Phone 2. Auflage. Architektur, Frameworks, APIs*** finden Sie unter *www.microsoft-press.de/?cnt=product&id=ms-5465&apid=60555*

P Entwickeln für Windows Phone 2. Auflage: Architektur, Frameworks, APIs (PreView-Kapitel)

Hinweis Beginn

Bei diesem Kapitel handelt es sich um eine Preview in Form eines Kapitels. Die dargestellten Inhalte basieren auf der Windows Phone SDK 7.1 RC -Version vom August 2011 und stellen daher auch nicht den finalen Stand der Inhalte dar.

Dieses Kapitel ist darüber hinaus eine Zusammenfassung mehrerer verschiedener Kapitelteile, um eine generelle Übersicht zu bieten und stellt den aktuellen Ist-Stand der Kapitel dar.

Einige Verweise werden hier auf das Kapitel »nn« genutzt. Dabei ist zu diesem Zeitpunkt die finale Kapitelnummer der verwiesenen Inhalte noch nicht bekannt, so dass »nn« hier als Platzhalter genutzt wird.

Beschriebene Quellcodes sind beim PreView-Kapitel nicht enthalten.

Mit diesen Informationen wünschen die Autoren Ihnen viel Spaß beim Lesen.

Hinweis Ende

SketchFlow mit dem Windows Phone

Streng genommen müsste dieses Thema ganz am Anfang des Buches stehen. Wir haben uns aber entschieden, diesen Teil extra in das letzte Kapitel des Buches zu verschieben, da wir der Meinung sind, dass man erst die Grundlagen der Entwicklung mit dem Windows Phone kennen sollte, bevor man sich dieser Technik widmet.

Aus der Sicht eines Entwicklungszyklus gehört dieses Thema, wenn denn eingesetzt, an den Anfang einer Projektphase. Doch was ist *SketchFlow*?

Wahrscheinlich ist einem der Begriff *Sketching* (zu Deutsch: »Zeichnen«) nicht geläufig, doch man macht es häufig implizit oder explizit: Man stellt sich die Anwendung, bevor sie erstellt wird, vor dem geistigen Auge vor, oder fertigt ein paar Zeichnungen/Skizzen an, die einzelne Oberflächen beschreiben oder vielleicht auch die Navigation zwischen den einzelnen Oberflächen darstellen.

Oftmals verwendet man diese Technik auch, um Absprachen direkt mit dem Kunden zu führen, bevor die Entwicklung in eine falsche Richtung geht und man ungewollt hohe und unnötige Kosten verursacht.

SketchFlow selbst ist ein Modul, welches Microsoft Expression Blend seit der Version 3 zum Sketching von WPF- und Silverlight-Anwendungen bietet. Durch das Sketching kann man sich schnell über Details und ein generelles Vorgehen einigen, anstatt erst in die Entwicklung einzusteigen.

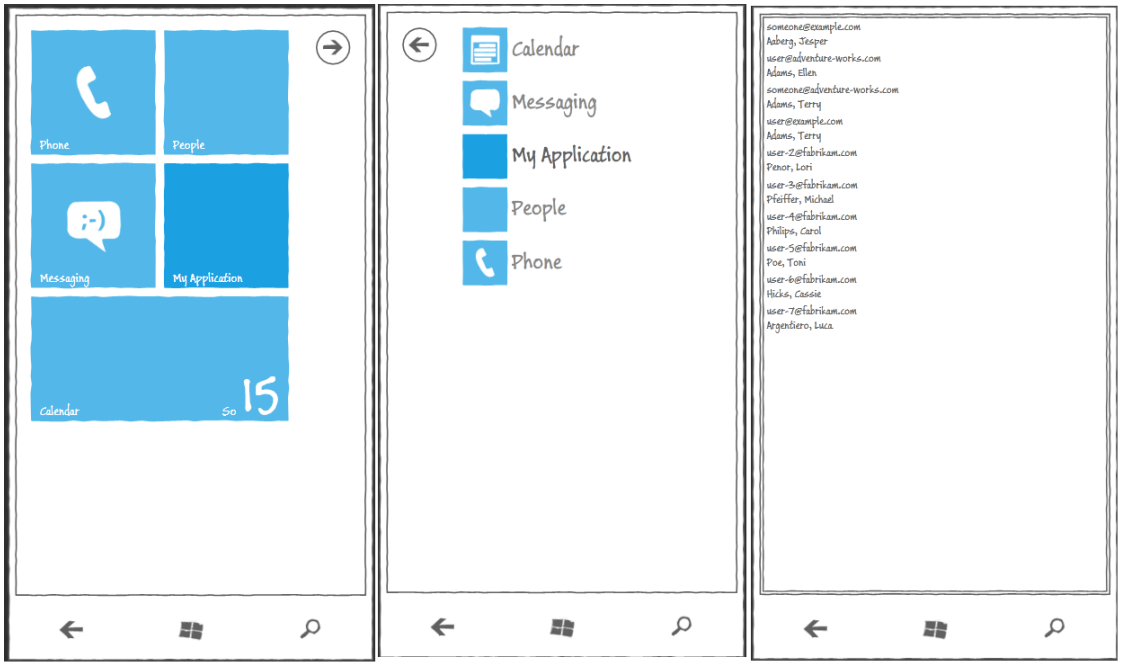

Abbildung P.1: *3 Verschiedene SketchFlow-Oberflächen*

Die Sketchflow-Oberflächen an sich wirken gezeichnet.Dies hat mehrere Gründe: So soll die strichhafte Darstellung darauf hinweisen, dass es sich wirklich um einen Entwurfsmodus handelt. Zusätzlich verhindert dies, dass man zu sehr auf einzelne Aspekte eingeht und sich daher besser auf das Wesentliche konzentrieren kann.

Beim Sketching kommt es auch nicht darauf an, wie der Quellcode der Anwendung aussieht, oder wie etwas funktioniert. Einzig und allein die Oberflächendarstellung steht im Vordergrund.

Animationen, wie auch Datenquellen lassen sich verwenden, da letztlich der volle Expression Umfang zur Verfügung steht. Man arbeitet hier ergebnisorientiert, da bis auf Ideen und Konzepte hinsichtlich Funktionalität und Aussehen in der darauf folgenden Realisierung nichts übernommen wird. Es ist und bleibt ein Wegwerfprojekt.

Auf *CodePlex* ist mittlerweile ein Projekt (*http://wp7sketchflow.codeplex.com/*) verfügbar, welches die Sketching-Funktionalität in Expression Blend auch für Windows Phone Anwendungen ermöglicht.

Leider gibt es ein kleines Problem mit der SketchFlow für Windows Phone Vorlage: Es wird ein Expression Blend mit SketchFlow benötigt, d.h. eine Version, die nicht mit den Windows Phone SDK installiert wird, da ansonsten die Projektvorlage nicht ausgewählt werden kann.

Die Projektvorlage ist schnell als MSI-Installationsdatei von der oben genannten Website heruntergeladen und installiert.

Ziel der folgenden Beschreibung ist es nicht, alle Funktionen von Expression Blend zu zeigen, denn dies würde viel zu weit gehen. Eher sollen die Sketch- und Feedback-Elemente dargestellt werden, die für diesen Prozess sinnvoll sind.

Eine erste Einführung in Expression Blend erhalten Sie in *Kapitel nn*.

Erstellen des ersten Sketches

Der erste Sketch ist schnell erstellt:

Starten Sie hierzu Expression Blend, nachdem Sie das Installationspaket vom oben genannten Projekt erfolgreich installiert haben. Erstellen Sie als nächstes ein neues Projekt vom Typ *Windows Phone SketchFlow Application* aus der Kategorie *Windows Phone*.

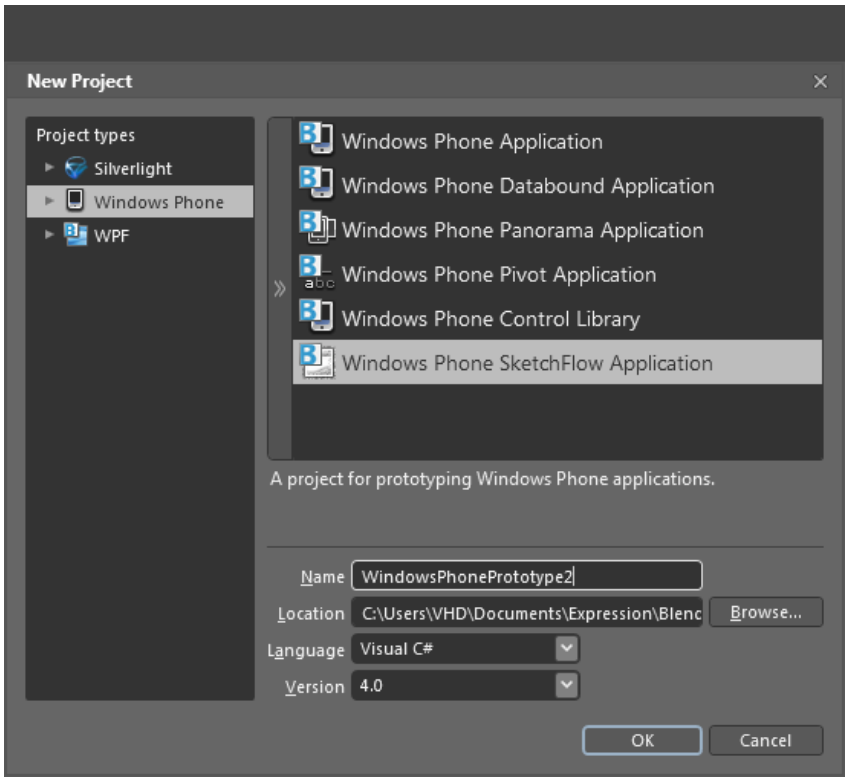

Abbildung P.2: *Mit dieser Vorlage lassen sich Windows Phone SketchFlow Projekte gestalten*

Nachdem Bestätigung von *OK* erhält man in Blend die folgende Darstellung.

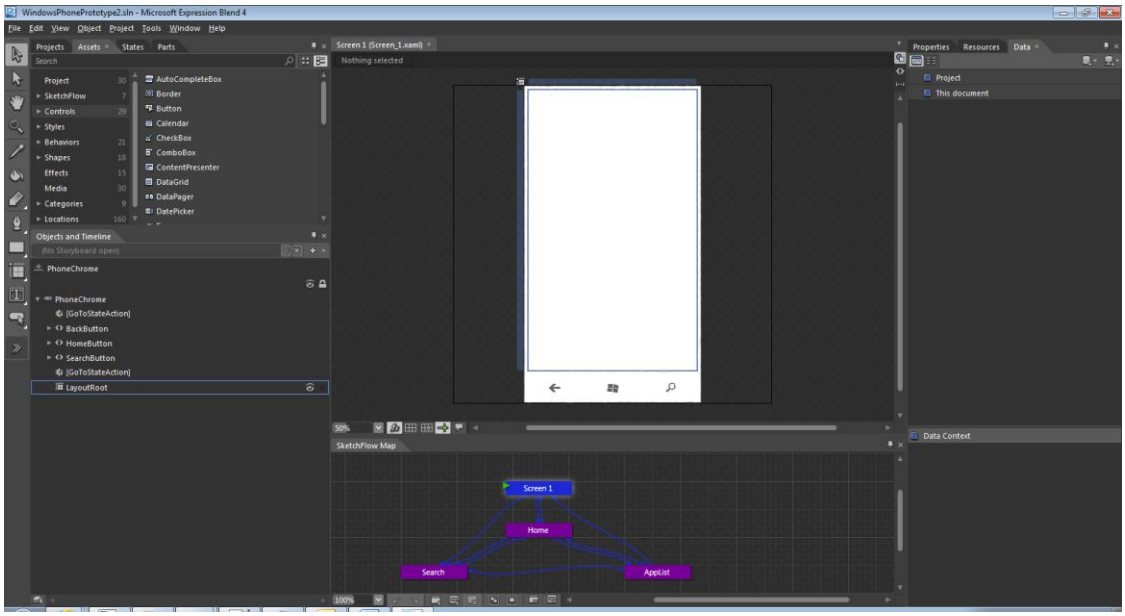

Abbildung P.3: *Die Arbeitsoberfläche eines SketchFlow Projektes*

Im mittleren Bereich von Blend ist im oberen Teil die Designoberfläche (*Screen1*) zu sehen. Dabei wirkt diese, wie eine gezeichnete Darstellung eines Windows Phone, was passend zum Sketching ist.

Der untere Bereich stellt die *SketchFlow Map* dar. In diesem Bereich sind alle *Screens*, also Bildschirmoberflächen, des Projekts enthalten.

Wie man sieht, sind bereits einige Oberflächen neben *Screen1* enthalten: *Home*, *AppList* und *Search*. Diese »emulieren« sozusagen das unterliegende Betriebssystem. So lässt sich die in Blend gezeichnete Anwendung, wie auf einem richtigen Gerät aus dem *Start* (hier also *Home* genannt) oder der *AppList* starten.

Zum Start gelangt man, wenn man beispielsweise die *Start*-Taste auf der Oberfläche aktiviert. Dazu später mehr.

Zwischen den Oberflächen sind Pfeile enthalten. Diese zeigen die Navigationswege zwischen den Oberflächen an.

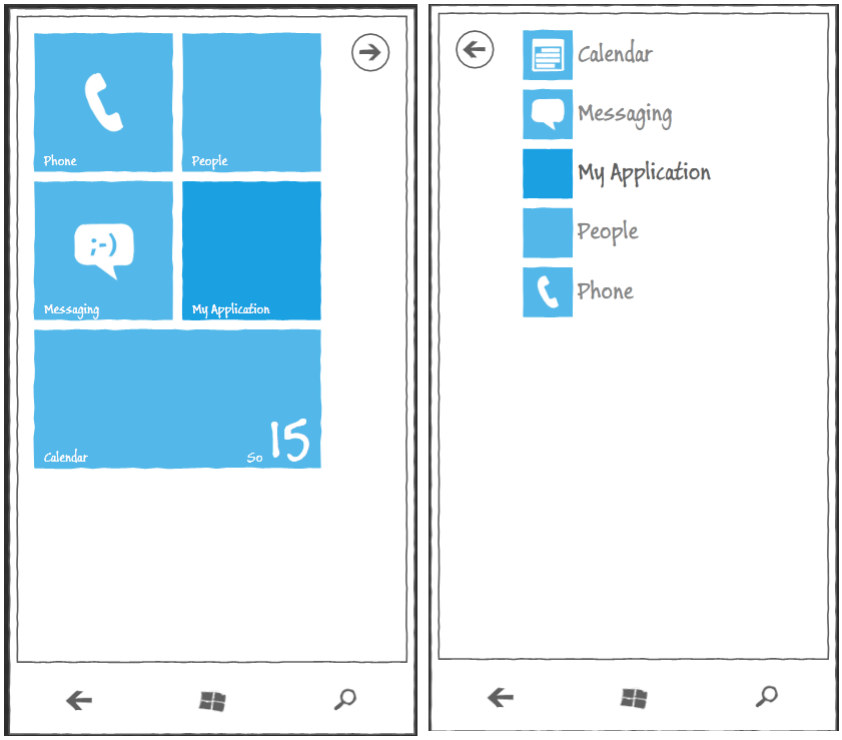

Abbildung P.4: *Der Start und die AppList sind bereit vordefiniert*

Der letzte Bereich sind die *Assets* im linken Bildschirmbereich von Blend, welche auch hier eine wichtige Rolle spielen. Genau, wie bei einer üblich designten Anwendung befinden sich hier Steuerelemente, die in SketchFlow-Projekten verwendet werden können.

Dabei ist die Verwendung recht simpel, da die Steuerelemente, die hier verwendet werden sollen mit dem Suffix *–Sketch* enden.

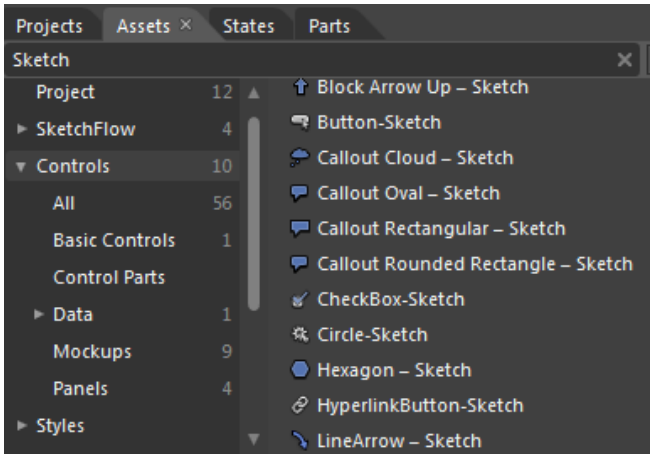

Abbildung P.5: *SketchFlow-Steuerelemente*

Die Verwendung der üblichen Steuerelementen ist genauso möglich, jedoch fehlt dann der gezeichnete Effekt. Somit spricht nichts gegen den Einsatz von eigens erstellten Steuerelementen. Damit lassen sich beispielsweise auch Teilergebnisse von selbst entwickelten Funktionen oder Steuerelementen bereits im

Sketch darstellen. Doch das ist nicht alles:

So gibt es zusätzlich eine Unterkategorie *Mockups* in den *Controls*.

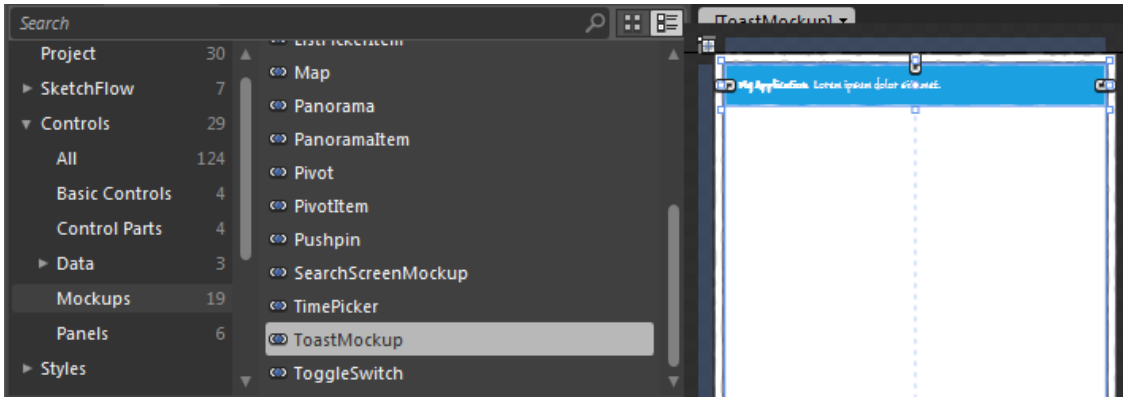

Abbildung P.6: *Mockups enthalten einige Beispielelemente aus dem Windows Phone*

In dieser Kategorie befinden sich Oberflächen oder Elemente, die auf dem Windows Phone vorhanden sind, so dass, wie dargestellt, sich eine Toastbenachrichtigung genauso im Sketch unterbringen lässt, wie auch ein *DatePicker*-Steuerelement. Auch *Panorama*- oder *Pivot*-Oberflächendarstellung sind hiermit eine Leichtigkeit.

In diesem Beispiel soll einfach eine zweite Oberfläche erstellt werden, zu welcher über eine Sketch-Schaltfläche navigiert wird. Durch Aktivierung der *Back*-Taste soll dann zur vorherigen Ansicht zurückgekehrt werden. Dieses Beispiel ist recht einfach gehalten, da wirklich nur die Eigenheiten des *SketchFlow* dargestellt werden sollen.

Um nun eine neue Oberfläche zu erstellen, aktiviert man aus der *SketchFlow Map* den Eintrag *Screen1*, indem man diesen mit der Maus markiert.

Abbildung P.7: *Optionen eines Screen-Eintrags*

Durch die Aktivierung werden neue Schaltflächen unterhalb des Eintrags sichtbar.

Die erste Schaltfläche erzeugt eine verbundene Oberfläche, wohingegen die zweite Schaltfläche eine Verbindung zu einer bereits bestehenden Oberfläche erlaubt.

Die dritte Schaltfläche ermöglicht die Erstellung einer zusammengesetzten Oberfläche.

Die letzte Schaltfläche erlaubt eine andere Farbwahl des Eintrags. Somit lassen sich Oberflächen farblich logisch gruppieren, was man somit auf den ersten Blick erkennt.

Hinweis Beginn

Der kleine Pfeil auf der linken Seite des Rechtecks beschreibt, dass diese Oberfläche beim Starten als Erste dargestellt wird.

Hinweis Ende

Aktivieren Sie die erste Schaltfläche und ziehen dabei den neu erstellten Eintrag *New Screen* zur Seite.

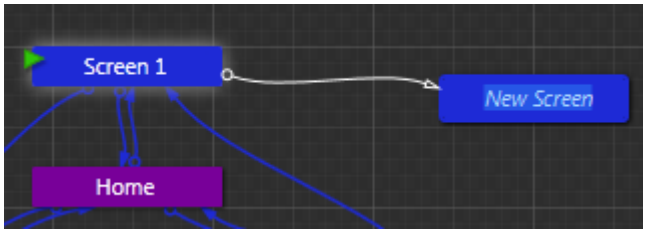

Abbildung P.8: *Eine neu erstellte Oberfläche*

Dieser Eintrag ist bereits mit der vorherigen Oberfläche verbunden, was durch den Pfeil dargestellt wird.

Aktivieren Sie ein anderes Element, so wird aus *New Screen* dann *Screen 2*.

Fügen Sie *Screen 1* nun ein Objekt vom Typ *Button-Sketch* hinzu. Aus dessen Kontextmenü wählen Sie den Eintrag *Navigate to* gefolgt von *Screen 2*.

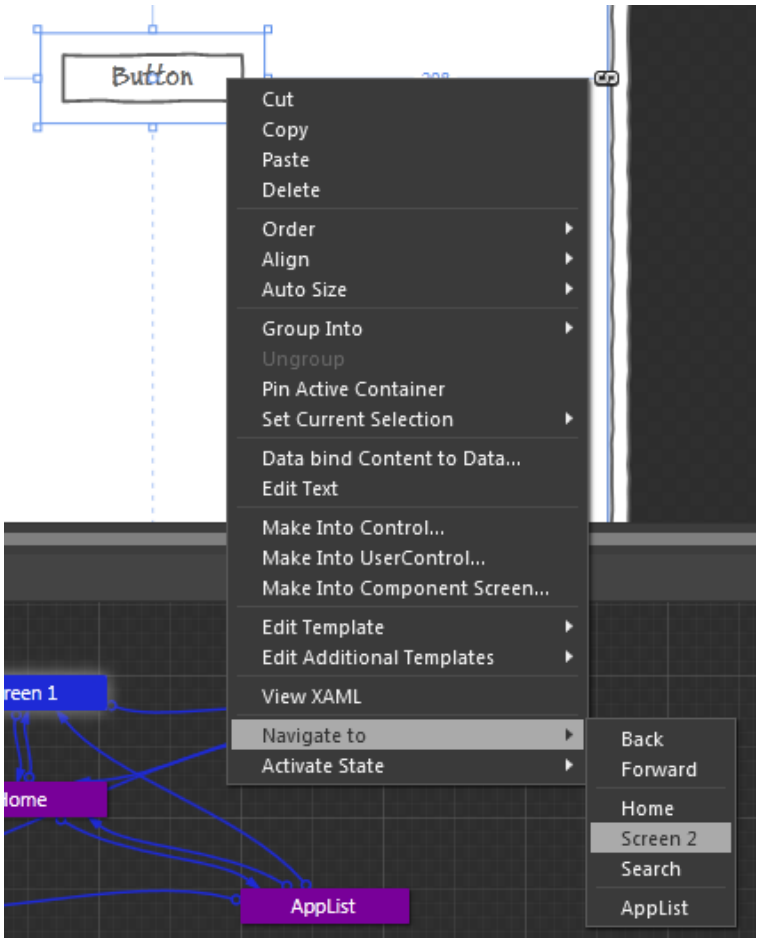

Abbildung P.9: *Das Erstellen einer Navigation kann einfacher nicht sein*

Durch diese Aktion ist nun eine Navigation einfach zu *Screen 2* möglich, wenn die Schaltfläche bei der Ausführung ausgelöst wird.

Wechseln Sie in die Ansicht *Screen 2*.

Dort soll durch die *Back*-Taste zur vorherigen Oberfläche zurückgekehrt werden.

Wählen Sie dazu aus dem Kontextmenü der *Back*-Taste den Eintrag *Navigate to* gefolgt von *Screen 1* aus.

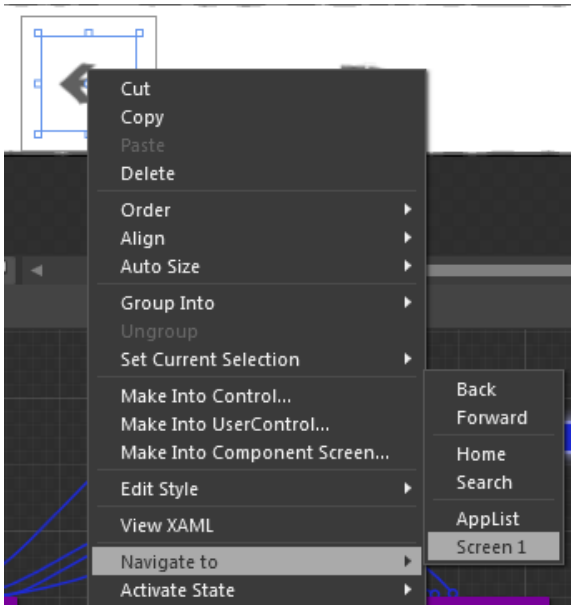

Abbildung P.10: *Erstellen der Rücknavigation*

Durch die Erstellung der Rücknavigation ist in der *SketchFlow Map* nun ein neuer Pfeil hinzugekommen. Beide Pfeile symbolisieren somit, dass in beide Richtungen navigiert werden kann. Natürlich sind auch Pfeile für die *Start*- oder die *Suchen*-Taste vorhanden.

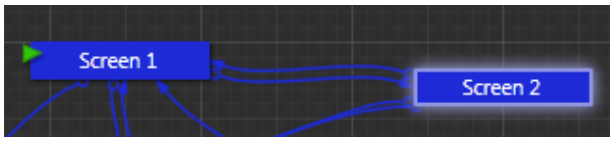

Abbildung P.11: *In beide Richtungen kann nun navigiert werden.*

Damit wäre dieses Beispiel fertig.

Starten Sie nun das Projekt im Debug-Modus.

Hinweis Beginn

Um Feedback vom Kunden oder anderen Mitarbeitern einzuholen, kann Expression Blend auch ein Paket in Form eines Ordners im Dateisystems zusammenschnüren.

Über das Menü *File* gefolgt von *Package SketchFlow Project...* erhält man einen neuen Dialog, in welchen man einen Ordner und einen Namen für das Paket vergibt.

Dieses Paket kann man dann als beispielsweise zusätzlich verpackte ZIP-Datei verschicken. Der Empfänger muss lediglich diese entpacken und die Datei *Default.html* ausführen. Das Ergebnis ist dasselbe, wie das folgend beschriebene.

Hinweis Ende

Erstellen von Feedback

Es mag verwirrend sein, dass nun eine Silverlight-Anwendung im eingestellten Webbrowser gestartet wird, obwohl eine Windows Phone SketchFlow Anwendung erstellt wurde.

Grund hierfür ist, dass die Entwickler der Windows Phone SketchFlow-Projektvorlage sich das Leben erleichtern wollten, was in diesem Fall völlig legitim ist. So wird die generelle Projektvorlage für Silverlight SketchFlow-Projekte als Basis verwendet. Durch reines Skinning wird jedoch daraus die Vorlage für Windows Phone Anwendungen, so dass beide denselben Mechanismus verwenden. Dies ist auch ein Vorteil, so dass eine erneute Eingewöhnung nicht notwendig ist.

Die Anwendungsoberfläche sieht nun wie in ***Abbildung P.12*** aus.

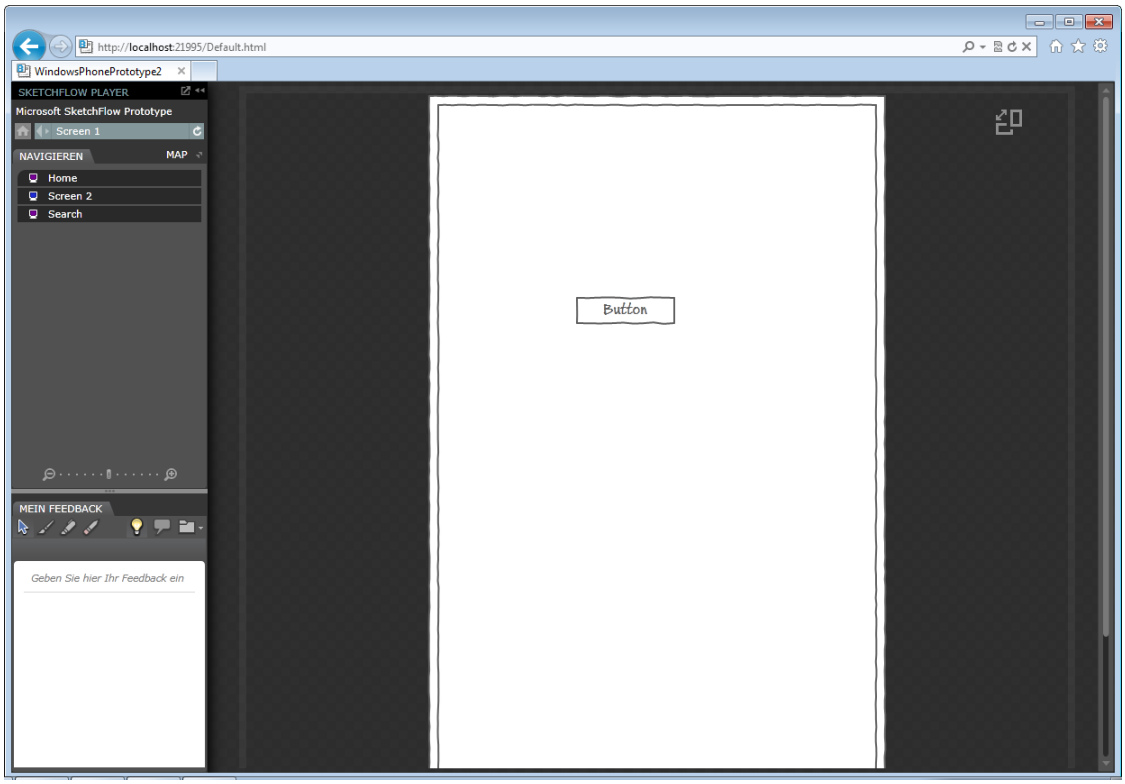

Abbildung P.12: *Die Anwendungsoberfläche für das zu sammelnde Feedback*

Mittig auf dem Bildschirm ist die Anwendungsoberfläche zu sehen. Dabei wird die Oberfläche als erstes gestartet, die in der *SketchFlow Map* mit dem grünen Pfeil gekennzeichnet war.

Bevor nun aber die Schaltfläche aktiviert wird, soll erst auf die einzelnen Elemente der SketchFlow-Oberfläche eingegangen werden.

In der rechten oberen Ecke ist ein kleines Symbol, mit welchem sich die Darstellung zwischen Hoch- und Quermodus wechseln lässt.

Der linke untere Bereich des Bildschirms beinhaltet alle benötigten Funktionen für das Feedback. So lassen sich dort textuelle Feedbacks verfassen, sowie auch ein Stift und ein Marker auswählen, mit dem man auf der Anwendungsoberfläche Anmerkungen machen kann, oder aber bestimmte Teile in verschiedenen Farben hervorhebt.

Diese Anmerkungen lassen sich auch ein-/ausblenden, wie auch wieder entfernen.

Das Ordner-Symbol rechts außen in der Symbolleiste des *Mein Feedback*-Karteireiters ermöglicht das Exportieren von Feedback.

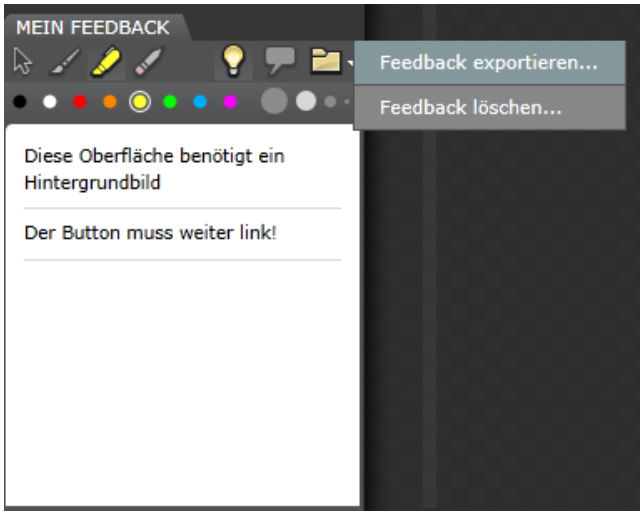

Abbildung P.13: *Feedback lässt sich direkt exportieren.*

Nach Angabe eines Kürzels und eines Namens wird das Feedback in einer SketchFlow-Feedbackdatei abgelegt, welches nichts anderes als eine XML-Datei ist.

Der obere linke Bereich ermöglicht die direkte Navigation zu Oberflächen. Dies erspart einem eine umständliche Navigation über mehrere Bildschirme hinweg.

Möchte man stattdessen auch über die Navigationswege sprechen, dann ist dies auch möglich. So lässt sich über den Doppelpfeil neben *Map* die bekannte *SketchFlow Map* darstellen, in welcher die Oberflächen ebenfalls direkt angewählt werden können.

Abbildung P.14: *Aktivierung der SketchFlow Map ermöglicht die Darstellung der verknüpften Oberflächen*

Sollte die Darstellung sehr viele Oberflächen beinhalten und dadurch unübersichtlich werden, so ist ein Hinein- und Heraus-Zoomen in die Map ebenfalls möglich.

Auswerten von Feedback

Das gesammelte Feedback lässt sich in Expression Blend wieder darstellen.

Hierzu aktiviert man im Menü *Window* den Eintrag *SketchFlow Feedback*.

Durch + lassen sich die einzelnen Feedback-Dateien laden, so dass dann beispielsweise die Darstellung wie folgt aussieht.

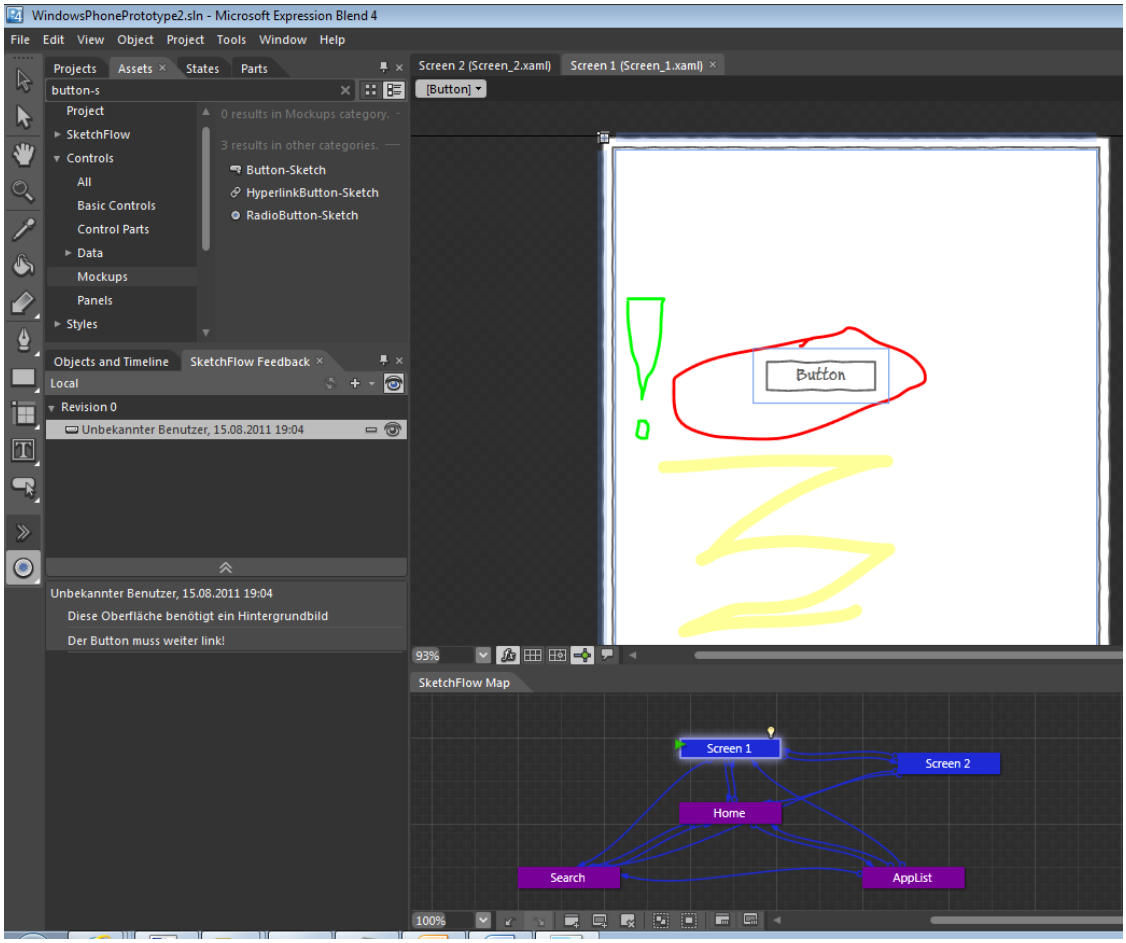

Abbildung P.15: *Das gesammelte Feedback wird ähnlich wie beim Sammeln des Feedbacks dargestellt*

Es sind somit nicht nur die Kommentare, sondern auch die grafischen Anmerkungen sichtbar. Empfindet man diese kurzfristig als störend, oder will nur das Feedback von bestimmten Anwendern sehen, so ist durch das Auge-Symbol das Feedback optisch (de-)aktivierbar. Möchte man stattdessen einen gesammelten Report als Word-Dokument erstellen, so ist dies ebenfalls möglich. Dieses kann beispielsweise dem Chef als Bericht zur Verfügung gestellt werden. Um diese Form des Exports anzustoßen, ist aus dem Menü *File* der Eintrag *Export to Microsoft Word…* zu aktivieren, welches einen neuen Dialog darstellt.

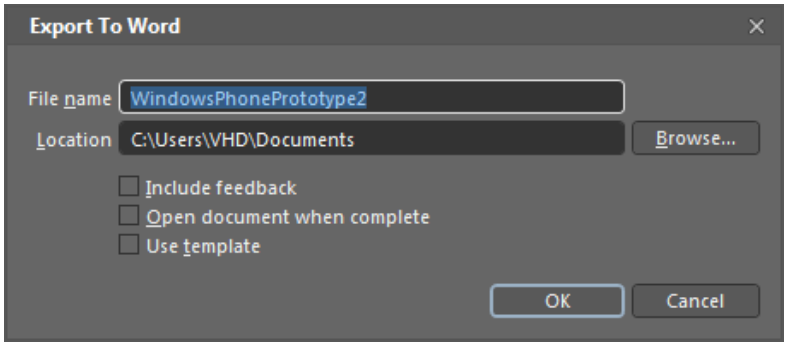

Abbildung P.16: *Das Exportieren als Word-Dokument ist ebenfalls mit Expression Blend möglich*

Dort ist neben dem Dateinamen und dem Ablageort die Option *Include feedback* zu aktivieren, wenn dieser im exportierten Dokument enthalten sein soll.

Das Ergebnis sieht in etwa wie in folgt aus.

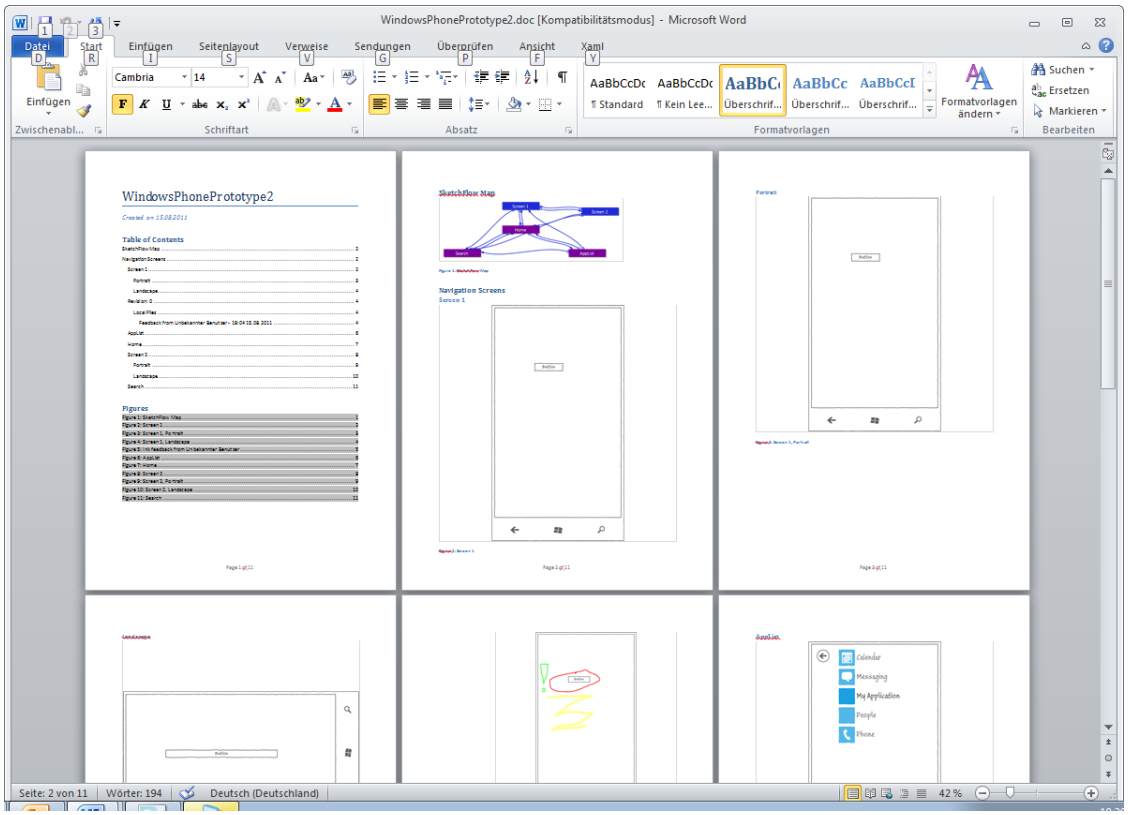

Abbildung P.17: *Das exportierte Word Dokument*

Gerade mit dieser Option lässt sich das müßige Erstellen von Standarddokumenten extrem beschleunigen.

Sketching für Windows Phone ist ein probates Mittel schnell Ergebnisse zu genaueren Absprachen zu erhalten. Zwar beginnt dadurch die Entwicklung nicht schneller, jedoch sollte hierdurch die Entwicklungszeit durch ansonsten später stattfindende Reviews und Nacharbeiten verkürzt werden.

Zugriff auf die Kamera

Musste man für einen Zugriff auf die Kamera bisher den Umweg über den `CameraCaptureTask` gehen, ermöglicht Windows Phone 7.5 nun auch den direkten Zugriff auf die Kamera um Fotos und Videos aufzunehmen, oder aber auch den Livestream der Kamera in der eigenen Anwendung anzuzeigen. Stichwort: Augmented Reality.

Der `CameraCaptureTask` ist natürlich weiterhin enthalten und bietet eine einfache Möglichkeit ein Foto aufzunehmen. Wenn Sie jedoch mehr Kontrolle über die Aufnahme haben wollen oder einfach nur an den Livestream der Kamera gelangen wollen, ist der `CameraCaptureTask` nicht geeignet.

Der direkte Kamerazugriff kann über zwei verschiedene Schnittstellen erfolgen:

2. `PhotoCamera`-Klasse
3. Silverlight 4 Webcam API

Die *Silverlight 4 Webcam API* ist eine Entwicklerschnittstelle der *Silverlight Runtime* und ermöglicht es, bestehenden Silverlight Code einer Desktop Anwendung weiterhin auf dem Windows Phone zu nutzen.

Die API bietet die Möglichkeit Videos aufzunehmen, was mit der `PhotoCamera` Klasse leider nicht funktioniert. Allerdings ist die Qualität der Videos nicht gleichzusetzen mit den Aufnahmemöglichkeiten der nativen Kameraapplikation.

Die `PhotoCamera` Klasse hingegen bietet eine umfangreichere Unterstützung der integrierten Hardware und mehr Optionen an. So kann das Blitzlicht beeinflusst, die Auflösung für die Aufnahmen geändert, sowie die automatische Fokussierung gestartet werden.

Je nach Hardware kann auch eine Fokussierung auf einen bestimmten Punkt vorgenommen werden, aber dies bietet nicht jede Kamera.

Einziges Manko ist die fehlende Möglichkeit zu zoomen, und dass, wie der Name der Klasse jedoch schon verrät, keine Videoaufnahme möglich sind.

Dafür bietet die `PhotoCamera` Klasse jedoch auch hervorragende Möglichkeiten den Livestream der Kamera in der eigenen Applikation anzuzeigen, und diesen auch nach Belieben zu manipulieren.

So können beispielsweise Overlays für Lokationsdaten, Übersetzungen von Text oder 3D Objekte eingebunden und mit dem Kamerabild kombiniert werden.

Der Livestream kann aber auch für Anwendungsfälle, wie Barcodescanning, Gesichtserkennung und vielen weiteren Zwecken verwendet werden.

Hinweis Beginn

Beide Schnittstellen ermöglichen die Anzeige und Verwendung des Livestreams. Im Regelfall bietet die PhotoCamera-API jedoch eine etwas bessere Performanz, ist jedoch nicht kompatibel, beziehungsweise nicht im großen Silverlight Framework enthalten.

Hinweis Ende

Silverlight 4 Webcam API

Da die *Silverlight 4 Webcam API* generell eine bereits bestehende Silverlight Framework API ist, wird an dieser Stelle nicht in aller Tiefe auf die Möglichkeiten der API eingegangen.

Doch auch wenn die API kompatibel zwischen der Desktop Version und dem Windows Phone ist, gibt es speziell für das Windows Phone einige Änderungen, beziehungsweise Optimierungen, die nicht unerwähnt bleiben sollten:

1. Speicherung des Audiostreams und Videostreams per `FileSink` als MP4
2. Unterstützung des Y-Formates (Luminaz)
3. `rycycleBuffer` Parameter für `VideoSink`

Ansonsten ist die Verwendung der API gleich zu der Verwendung in einer Desktop Anwendung. Daher wird lediglich anhand einer Beispielapplikation gezeigt, wie mit der API ein Video aufgenommen werden kann.

Das vollständige Beispielprojekt finden Sie im Quellcode Ordner dieses Kapitels unter dem Namen *SLCameraVideo*.

Im ersten Schritt wird eine »Leinwand« benötigt, um den Videostream in der Oberfläche anzuzeigen.

Dazu wird im XAML-Code der *MainPage* ein `Rectangle` hinzugefügt. Diesem `Rectangle` wird dann als Füllung eine `VideoBrush` zugewiesen. Die Kamera zeichnet später Ihren Inhalt direkt in diese

`VideoBrush`, womit das Kamerabild in dem `Rectangle` angezeigt wird.

```
<Rectangle HorizontalAlignment="Left" Height="480" Stroke="Black" VerticalAlignment="Top" Width="640">
   <Rectangle.Fill>
      <VideoBrush x:Name="VorschauBrush"/>
   </Rectangle.Fill>
</Rectangle>
```

Listing P.1: *Zuweisen der Fill Eigenschaft*

Es muss nicht zwingend ein `Rectangle` verwendet werden. Jede Form, die eine `Fill`-Eigenschaft besitzt, kann zur Vorschau verwendet werden. Des Weiteren werden drei Schaltflächen hinzugefügt, die es ermöglichen die Aufnahme zu starten, zu stoppen und abzuspielen.

```
<Button Content="Rec" Height="72" HorizontalAlignment="Left" Margin="646,6,0,0" Name="Record"
VerticalAlignment="Top" Width="130" Click="Record_Click" />
<Button Content="Stop" Height="72" HorizontalAlignment="Left" Margin="646,84,0,0" Name="Stop"
VerticalAlignment="Top" Width="130" Click="Stop_Click" />
<Button Content="Play" Height="72" HorizontalAlignment="Left" Margin="646,162,0,0" Name="Play"
VerticalAlignment="Top" Width="130" Click="Play_Click" />
```

Listing P.2: *Erstellen der Schaltflächen*

Innerhalb der Codebehind-Datei werden nun die weiteren Schritte durchgeführt.

Zuerst muss ein `CaptureSource` Objekt angelegt werden, dem später die Aufnahmegeräte zugeordnet werden können und die Kontrolle über diese ermöglicht.

Da das Video gespeichert werden soll, bietet sich ein `FileSink` Objekt an. Dieses kann direkt an die Kamera gebunden werden und »kümmert« sich um die Aufnahme.

```
public partial class MainPage : PhoneApplicationPage
{
   //Kamera anlegen
   CaptureSource kamera;
   //FileSink zum Speichern des Videos
   FileSink fileS;
   // Constructor
   ...
```

Listing P.3: *Deklaration der benötigten Objekte*

Beide Klassen befinden sich im *System.Windows.Media* Namespace, der von der Projektvorlage automatisch per `using`-Direktive eingebunden wurde.

Innerhalb der *OnNavigatedTo* Überschreibung wird nun die Kamera initialisiert, die Aufnahmegeräte für Bild und Ton zugeordnet, die Vorschau konfiguriert und die Übertragung des Livebildes Streamen der Kamera gestartet.

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
   //Kamera erzeugen
   kamera = new System.Windows.Media.CaptureSource();
   //Default Kamera verwenden
   kamera.VideoCaptureDevice = CaptureDeviceConfiguration.GetDefaultVideoCaptureDevice();
   //Default Microfon verwenden
   kamera.AudioCaptureDevice = CaptureDeviceConfiguration.GetDefaultAudioCaptureDevice();
   //Vorschau festlegen
   VorschauBrush.SetSource(kamera);
```

```
    //Kamera starten
    kamera.Start();
}
```

Listing P.4: *Konfiguration der Kamera sowie der Vorschau*

Die Aufnahme eines Videos erfolgt nun über die Schaltfläche *Rec*. Da die Kamera schon läuft, um den Livestream anzuzeigen, muss im ersten Schritt die `Stop`-Funktion aufgerufen werden, da sonst nicht das `FileSink`-Objekt verwendet werden kann, beziehungsweise genauer gesagt dem `FileSink`-Objekt keine laufende Kamera zugewiesen werden kann.

Wenn ein Dateiname und eine Aufnahmequelle festgelegt wurde, kann die Kamera wieder gestartet werden.

```
private void Record_Click(object sender, RoutedEventArgs e)
{
    //Die Kamera muss erst beendet werden
    kamera.Stop();
    //FileSink erstellen
    fileS = new FileSink();
    //Datei festlegen
    fileS.IsolatedStorageFileName = "aufnahme.mp4";
    //Aufnahmequelle festlegen
    fileS.CaptureSource = kamera;
    //Die Kamera und somit auch dei Aufnahme starten
    kamera.Start();
}
```

Listing P.5: *Starten der Aufnahme*

Um die Aufnahme zu beenden, wird lediglich ein erneuter Aufruf der `Stop` Funktion benötigt.

```
private void Stop_Click(object sender, RoutedEventArgs e)
{
    //Kamera (und ggf. die Aufnahme) stoppen
    kamera.Stop();
}
```

Listing P.6: *Stoppen der Aufnahme*

Anschließend kann das Video über die *Play* Schaltfläche abgespielt werden, dazu wird der bereits in diesem Kapitel beschriebene `MediaPlayerLauncher` Task verwendet.

Hinweis Beginn

Es existiert leider keine Möglichkeit ein aufgenommenes Video in die Medienbibliothek zu kopieren, daher müssen Sie das Video über einen anderen Weg, wie beispielsweise einen Webdienst oder Sockets übertragen.

Hinweis Ende

Abbildung P.18: *Die SLCameraVideo Applikation im Emulator*

PhotoCamera Klasse

Entgegen der *Silverlight 4 Webcam API* ist die `PhotoCamera` Klasse speziell auf Windows Phone zugeschnitten. Dies muss nicht zwangsweise bedeuten, dass diese Schnittstelle besser geeignet ist, jedoch bietet sie je nach Szenario einige Optionen, die der *Silverlight 4 Webcam API* fehlen.

Steuerungsmöglichkeiten

Die `PhotoCamera`-Klasse bietet einige erweiterte Steuerungsmöglichkeiten für die Kamerahardware.

Die Steuerungsmöglichkeiten unterscheiden sich teilweise jedoch in Abhängigkeit der verwendeten Hardware.

1. Kameratyp

Wie auch die *Silverlight 4 Webcam API* unterstützt die `PhotoCamera` Klasse mehrere Kameratypen.

Diese sind in der Enumeration `CameraType` definiert und beinhaltet in der derzeitigen Version die beiden Varianten `Primary` und `FrontFacing`.

`Primary` ist die Kamera auf der Rückseite, so wie man Sie vom Windows Phone her gewöhnt ist.

Die `FrontFacing` Kamera ist eine optionale Kamera die beispielsweise für Videotelefonie verwendet werden könnte und sich auf der Frontseite des Gerätes befindet, sofern Sie denn vorhanden ist.

Beim Anlegen eines `PhotoCamera` Objektes kann über den Konstruktor der Kameratyp übergeben werden.

Über die statische Funktion `IsCameraTypSupported` der Basisklasse `Camera` kann überprüft werden, ob das Gerät über den gewünschten Kameratyp verfügt.

2. Autofokus

Über die Eigenschaft `IsFocusSupported` kann überprüft werden, ob die Kamera die automatische Fokussierung per `Focus` Funktion beherrscht. Für `CameraType.Primary` sollte dies eigentlich immer zutreffen, wird jedoch eine Frontkamera verwendet, ist es durchaus möglich, dass diese Funktion nicht

unterstützt wird.

3. Fokus auf bestimmten Punkt

Über die Eigenschaft `IsFocusAtPointSupported` kann überprüft werden, ob eine Fokussierung auf einen Punkt mithilfe der Funktion `FocusAtPoint` durchgeführt kann. Dieser Punkt kann vom Anwender in der Kameravorschau ausgewählt werden. Diese Art der Fokussierung wird nicht von jeder Kamera unterstützt.

4. Auflösung

Nicht jede Kamera unterstützt jede Auflösung. Über die Eigenschaft `AvailableResolutions` kann eine Liste der unterstützten Auflösungen abgefragt werden. Über die Eigenschaft `Resolution` kann die Auflösung, die derzeit für die Aufnahme verwendet wird, ausgelesen oder gesetzt werden.

Achtung Beginn

Verwechseln Sie nicht die Auflösung der Aufnahme mit der Auflösung der Vorschau. Die Vorschau besitzt eine eigene Eigenschaft `PreviewResolution`, welche jedoch nur lesbar ist.

Achtung Ende

5. Blitz

Über die Eigenschaft `FlashMode` kann das aktuell konfigurierte Blitzverhalten ausgelesen werden.Auch das Setzen des Blitzverhaltens erfolgt über diese Eigenschaft. Über die Funktion `IsFlashModeSupported` kann überprüft werden, ob die Kamera den gewählten Blitztypen unterstützt.

Hinweis Beginn

Das Setzen eines nicht unterstützen Blitztyps schaltet den Blitz automatisch aus.

Hinweis Ende

Die Blitztypen sind in der Enumeration `FlashMode` deklariert und besitzen folgende Werte:

Wert	Beschreibung
On	Blitz immer an
Off	Blitz immer aus
Auto	Blitz wird automatisch aktiviert wenn nötig
RedEyeReduction	In der Regel erfolgt ein Vorblitz um den bekannten »Roten Augen« auf Bildern vorzubeugen.

Tabelle P.1: *Blitztypen*

6. Kamerataste

Über die Klasse `CameraButtons` kann auch die Kamerataste für die eigene Applikation eingebunden werden.

Die Klasse bietet drei Ereignisse für den jeweiligen Zustand der Kamerataste an, sodass Funktionen, wie Autofokus oder Aufnahme, wie aus der nativen Kameraapplikation heraus gewohnt, implementiert werden können.

Achtung Beginn

Die `CameraButtons` Klasse kann nur in Kombination mit der `PhotoCamera` Klasse verwendet werden. Wird sie ohne ein aktives `PhotoCamera`-Objekt verwendet, werden die Ereignisse nicht ausgelöst.

Achtung Ende

Implementierung

Nachfolgend wird anhand einer Windows Phone Applikation die Implementierung der `PhotoCamera` Funktionen inklusive einiger Erweiterungen gezeigt. Das vollständige Beispiel finden Sie im Quellcode Ordner dieses Kapitels unter dem Namen *SLCamera*.

Die Beispielapplikation besitzt zwei Seiten, wobei eine der beiden jedoch nur einige Einstellungen bietet. In dieser Seite wird zwar auch die Kamera verwendet, jedoch werden lediglich lesende Operationen ausgeführt, die mehr oder weniger selbsterklärend sind und hier nicht weiter erläutert werden.

Viel interessanter ist die Hauptseite der Applikation, da hier die eigentliche Verwendung der Kamera gezeigt wird. Neben der Option Fotos aufzunehmen, speichert die Beispielapplikation die Fotos in der Medienbibliothek ab und zeigt außerdem eine mögliche Implementierung eines Fotoeffektes, indem per *Overlay* ein Schnurrbart eingeblendet und auch im Foto abgespeichert wird.

Die *MainPage.xaml* ist eine gängige `PhoneApplicationPage` im *Landscape* Modus. Das standardmäßig enthaltene *TitlePanel* wurde entfernt, dafür zwei `Rectangle` und ein `TextBlock`-Steuerelement in das *ContentPanel* eingefügt.

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
   <Rectangle x:Name="KameraBild" Margin="0,0,64,0" DoubleTap="KameraBild_DoubleTap">
      <Rectangle.Fill>
         <VideoBrush x:Name="kameraBildBrush"/>
      </Rectangle.Fill>
   </Rectangle>
   <Image x:Name="schnurrbart" Margin="0,0,64,0" Source="Schnurrbart.png" Stretch="None" Width="640"
Height="480" DoubleTap="schnurrbart_DoubleTap"/>
   <TextBlock x:Name="txtStatus" Margin="0,0,0,8" TextWrapping="Wrap" VerticalAlignment="Bottom"
Foreground="#DEFF0000" FontWeight="Bold"/>
</Grid>
```

Listing P.7: *XAML Code der MainPage*

Das hintere `Rectangle` dient als *Leinwand* für die Kameravorschau, das vordere dient als *Overlay* für den Fotoeffekt. Das `TextBlock`-Control dient zur Anzeige von Statusinformationen. Die restlichen Schritte sind alle in der *MainPage.xaml.cs* durchzuführen.

Für die einzelnen Funktionalitäten werden die folgenden Verweise und Referenzen benötigt.

```
//Kamerazugriff
using Microsoft.Devices;
//MediaLibrary
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework;
//IsolatedStorageSettings
using System.IO.IsolatedStorage;
//WriteableBitmapEx
using System.Windows.Media.Imaging;
```

Listing P.8: *Die benötigten using Direktiven*

Die Kamera, beziehungsweise ein Objekt der `PhotoCamera`-Klasse, werden zunächst als globales Klassenobjekt angelegt.

```
public partial class MainPage : PhoneApplicationPage
{
```

```
   //Kamera deklarieren
   PhotoCamera kamera = new PhotoCamera();
   // Constructor
   …
```

Listing P.9: *Anlegen des Kameraobjektes*

Die Initialisierung der Kamera, sowie das Anlegen der benötigten Ereignisbehandlungen, erfolgt in der Überschreibung der `OnNavigatedTo` Funktion.

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
   //Die Kamera im Dispatcher angelegen
   this.Dispatcher.BeginInvoke(delegate()
   {
      //Kamera anlegen
      kamera = new PhotoCamera();
      //Wird aufgerufen wenn die Kamera initialisiert ist
      kamera.Initialized += new EventHandler<CameraOperationCompletedEventArgs>(kamera_Initialized);
      //Wird aufgerufen wenn der Aufnahmevorgang abgeschlossen ist
      kamera.CaptureCompleted += new EventHandler<CameraOperationCompletedEventArgs>(kamera_CaptureCompleted);
      //Wird aufgerufen wenn der Aufnahmevorgang startet
      kamera.CaptureStarted += new EventHandler(kamera_CaptureStarted);
      //Wird aufgerufen wenn das aufgenommene Bild vorhanden ist
      kamera.CaptureImageAvailable += new EventHandler<ContentReadyEventArgs>(kamera_CaptureImageAvailable);
      //Kamera Hardwarebutton: Halb gedrückt
      CameraButtons.ShutterKeyHalfPressed += new EventHandler(CameraButtons_ShutterKeyHalfPressed);
      //Kamera Hardwarebutton: Vollständig gedrückt.
      CameraButtons.ShutterKeyPressed += new EventHandler(CameraButtons_ShutterKeyPressed);
      //VideoBrush festlegen
      kameraBildBrush.SetSource(kamera);
   });
}
```

Listing P.10: *Initialisierung der Kamera sowie festlegen der Ereignisbehandlung*

Sobald die Kamera initialisiert ist, wird das `Initialized`-Ereignis der Klasse PhotoCamera ausgelöst. Innerhalb der Ereignisbehandlung werden die Auflösung, das Blitzverhalten, sowie der Fotoeffekt eingestellt.

```
void kamera_Initialized(object sender, CameraOperationCompletedEventArgs e)
{
   //Die Blitz Einstellung, Auflösung und Schnurrbart Effekt auswerten und setzen
   if (e.Succeeded)
   {
      string strBlitz = "";
      string strAufloesung = "";
      bool boSchnurrbart = false;
      try
      {
```

```
      if (IsolatedStorageSettings.ApplicationSettings.Contains("Blitz"))
         strBlitz = (string)IsolatedStorageSettings.ApplicationSettings["Blitz"];
      if (IsolatedStorageSettings.ApplicationSettings.Contains("Aufloesung"))
         strAufloesung = (string)IsolatedStorageSettings.ApplicationSettings["Aufloesung"];
      if (IsolatedStorageSettings.ApplicationSettings.Contains("Schnurrbart"))
         boSchnurrbart = (bool)IsolatedStorageSettings.ApplicationSettings["Schnurrbart"];
   }
   catch (Exception ex)
   {
   }
   switch (strBlitz)
   {
      case "An":
         kamera.FlashMode = FlashMode.On;
         break;
      case "Aus":
         kamera.FlashMode = FlashMode.Off;
         break;
      case "Rote Augen Reduktion":
         kamera.FlashMode = FlashMode.RedEyeReduction;
         break;
      case "Automatisch":
         kamera.FlashMode = FlashMode.Auto;
         break;
   }
   if (strAufloesung.Length > 0)
   {
      double dbWidth = 800;
      double dbHeight = 600;
      string[] strSizes = strAufloesung.Split('x');
      double.TryParse(strSizes[0], out dbWidth);
      double.TryParse(strSizes[1], out dbHeight);
      IEnumerable<Size> lstAufloesungen = kamera.AvailableResolutions;
      int Count = lstAufloesungen.Count<Size>();
      Size curAufloesung;
      for (int i = 0; i < Count; i++)
      {
         curAufloesung = lstAufloesungen.ElementAt<Size>(i);
         if (curAufloesung.Height == dbHeight && curAufloesung.Width == dbWidth)
         {
            kamera.Resolution = curAufloesung;
            break;
          }
      }
   }
```

```
      this.Dispatcher.BeginInvoke(delegate()
      {
         if (boSchnurrbart)
            schnurrbart.Visibility = System.Windows.Visibility.Visible;
         else
            schnurrbart.Visibility = System.Windows.Visibility.Collapsed;
         txtStatus.Text = "Kamera fertig...";
      });
   }
}
```

Listing P.11: *Setzen der Kameraeigenschaften nach der Initialisierung*

Der Livestream der Kamera wird nun in der Oberfläche der Anwendung angezeigt und die Kamera ist bereit für eine Aufnahme. Die Kamerataste soll analog zur nativen Kameraapplikation funktionieren.

Somit soll bei dem `ShutterHalfKeyPressed`-Ereignis automatisch fokussiert werden, was über einen einfachen Aufruf der `Focus`-Funktion bewerkstelligt werden kann.

```
void CameraButtons_ShutterKeyHalfPressed(object sender, EventArgs e)
{
   this.Dispatcher.BeginInvoke(delegate()
   {
      txtStatus.Text = "Fokussiere (Auto)";
   });
   try
   {
      //Auto Fokussierung starten
      kamera.Focus();
   }
   catch (Exception ex)
   {
      this.Dispatcher.BeginInvoke(delegate()
      {
         txtStatus.Text = ex.Message;
      });
   }
}
```

Listing P.12: *Automatische Fokussierung*

Beim Auslösen des `ShutterKeyPressed`-Ereignis soll eine Aufnahme durchgeführt werden. Auch dies ist ziemlich einfach umsetzbar, indem die Funktion `CaptureImage` aufgerufen wird.

```
void CameraButtons_ShutterKeyPressed(object sender, EventArgs e)
{
   try
   {
      //Eine Aufnahme starten
      kamera.CaptureImage();
   }
```

```
   catch (Exception ex)
   {
      this.Dispatcher.BeginInvoke(delegate()
      {
         txtStatus.Text = ex.Message;
      });
   }
}
```

Listing P.13: *Auslösen der Aufnahme*

Über das `CaptureImageAvailable`-Ereignis wird die Applikation benachrichtigt, sobald der Datenstrom der Aufnahme vorliegt und abgespeichert werden kann.

In dieser Beispielapplikation stehen nun zwei verschiedene Wege zur Behandlung dieses Ereignisses zur Verfügung. Zum einen das direkte Abspeichern des Fotos in die Medienbibliothek, zum anderen das Manipulieren des Bildes vor dem Abspeichern, für den Fall, dass der Fotoeffekt aktiviert wurde.

Der erste Weg ist ziemlich simple, wenn man sich eines `MediaLibrary`-Objektes aus dem *XNA Framework* bemächtigt. Ist ein solches Objekt angelegt, kann der Datenstrom des Fotos direkt über die `SavePictureToCameraRoll`-Funktion in der Medienbibliothek gespeichert werden.

```
void kamera_CaptureImageAvailable(object sender, ContentReadyEventArgs e)
{
   Dispatcher.BeginInvoke(delegate()
   {
      txtStatus.Text = "Speichere Bild...";
   });
   //Kurz warten im die UI Änderung durchzuführen
   System.Threading.Thread.Sleep(1);
   this.Dispatcher.BeginInvoke(delegate()
   {
      string strFile = "SLKamera.jpg";
      if (schnurrbart.Visibility == System.Windows.Visibility.Collapsed)
      {
         //Das Bild kann direkt gespeichert werden
         MediaLibrary medLib = new MediaLibrary();
         medLib.SavePictureToCameraRoll(strFile, e.ImageStream);
      }
      else
      {
         ...
```

Listing P.14: *Speichern des Fotos in der Medienbibliothek*

Der zweite Weg, der den Datenstrom vor dem Abspeichern noch manipulieren muss, ist ein wenig aufwändiger.

Generell gilt es die Grafik des Fotoeffektes auf dieselbe Größe des eigentlichen Fotos zu bringen. Dann müssen die beiden Grafiken noch miteinander vermischt werden, also das Effekt-Bild noch transparent über das Foto gelegt werden, bevor das so manipulierte Bild abgespeichert werden kann. Um diese Grafikoperationen durchzuführen, wird die `WriteableBitmap`-Klasse, sowie die Erweiterungsbibliothek `WritableBitmapEx`, verwendet.

Auf dem Blog des `WriteableBitmapEx` Entwicklers René Schulte gibt es zusätzlich noch eine weitere Extensionklasse, die es ermöglicht die `WriteableBitmap` direkt in die Medienbibliothek zu speichern. Sie finden den dazugehörigen Blogeintrag unter *http://kodierer.blogspot.com/2010/07/photos-photos-photos-how-to-save-load.html*. Die Exensionklasse wurde jedoch für dieses Beispiel noch um die Funktion `SaveToCameraRoll` ergänzt, um auch das editierte Foto in dem Album für eigene Aufnahmen abzuspeichern.

```
…
      else
      {
         //Die beiden Bilder (Aufnahme + Schnurrbart) übereinanderlegen
         //1.Auflösung auslesen
         int Width = Convert.ToInt32(kamera.Resolution.Width);
         int Height = Convert.ToInt32(kamera.Resolution.Height);
         //WritableBitmap erstellen und den Fotostream einlesen
         WriteableBitmap stream = new WriteableBitmap(Width, Height);
         stream.LoadJpeg(e.ImageStream);
         //WriteableBitmap erstellen und die Fotoeffekt Grafik laden
         WriteableBitmap bart = new WriteableBitmap(0, 0).FromResource("Schnurrbart.png");
         //Fotoeffekt Grafik an die Auflösung anpassen
         bart = bart.Resize(Width, Height, WriteableBitmapExtensions.Interpolation.Bilinear);
         //Die Grafiken übereinanderlegen
         Rect rc = new Rect(0, 0, Width, Height);
         stream.Blit(rc, bart, rc, WriteableBitmapExtensions.BlendMode.Alpha);
         //Über die Extension Methode der WritebleBitmapEx Klasse speichern
         stream.SaveToCameraRoll(strFile);
      }
      txtStatus.Text = "Bild gespeichert...";
   });
}
```

Listing P.15: *Anwenden des Grafikeffektes und abspeichern des Bildes*

Die letzte verbleibende Funktionalität der Beispielanwendung ist die Fokussierung auf einen bestimmten Punkt. Dieser Punkt kann vom Anwender über eine `DoubleTap`-Geste ausgewählt werden. Dazu wurde bei der XAML-Deklaration für die Kameravorschau und dem Fotoeffekt eine direkte Ereignisbehandlung implementiert. Innerhalb des Codebehinds leiten beide Ereignisse die jeweiligen Parameter an die Funktion `HandleDoubleTap` weiter. Die eigentliche Fokussierung erfolgt mittels der Funktion `FocusAtPoint` der `PhotoCamera`-Klasse. Dabei muss jedoch der zu fokussierende Punkt erst in ein spezielles Format konvertiert werden, der in einem Wertebereich von *0* bis *1* innerhalb des Vorschaubildes liegt. Der Wert $X = 0$, $Y = 0$ steht dabei für die linke, obere Ecke der Vorschau, der Wert $X = 1$, $Y = 1$ steht für die rechte, untere Ecke und $X = 0.5$, $Y = 0.5$ würde genau der Mitte entsprechen.

```
private void HandleDoubleTap(object sender, System.Windows.Input.GestureEventArgs e)
{
   //Fokusierung auf bestimmten Punkt wird nicht von jedem Gerät unterstützt
   if (kamera.IsFocusAtPointSupported)
   {
      //Die genaue Position innerhalb des Viewfinders bestimmen
```

```
      UIElement uiElem = sender as UIElement;
      double d1 = (uiElem.RenderSize.Width - e.GetPosition(sender as UIElement).X) / (uiElem.RenderSize.Width
/ 100);
      double x = 1 - (d1 / 100);
      double d2 = (uiElem.RenderSize.Height - e.GetPosition(sender as UIElement).Y) /
(uiElem.RenderSize.Height / 100);
      double y = 1 - (d2 / 100);
      this.Dispatcher.BeginInvoke(delegate()
      {
         txtStatus.Text = "Fokussiere" + x.ToString() + "," + y.ToString();
      });
      try
      {
         //Den Punkt fokusieren
         kamera.FocusAtPoint(x, y);
      }
      catch (Exception ex)
      {
         this.Dispatcher.BeginInvoke(delegate()
         {
            txtStatus.Text = ex.Message;
         });
      }
   }
}
```

Listing P.16: *Fokussierung eines bestimmten Punktes*

Abbildung P.19: *Die SLCamera Applikation im Emulator*