Abstract

This article discusses porting advanced user interface features from an iOS app to a Windows Store app. We use an electronic medical record (EMR) application for this case study.

1 Introduction

In recent years, tablets, as new forms of mobile computing platforms, have quickly moved from the consumer electronics spaces into the business and enterprise computing domains. After the release of Windows 8 operating systems earlier this year, we felt there was a need to provide some quick start tutorials for developers on how to port their existing apps from other platforms such as iOS to Windows 8, and start developing new apps on Intel Ultrabook™ devices, tablets, and other Intel architecture-based devices running Windows 8. This article serves this objective and focuses on the advanced user interface topics.

On iOS, natively Objective-C is the main development language. For Windows Store apps, you have multiple choices available, including Visual C#, HTML / JavaScript*, and others. In this case study, we use Visual C#* as the development language of choice.

1.1 From Xcode* 4 to Visual Studio* 2012

Like the Xcode tools package on OS X* for iOS application developers, Visual Studio 2012 provides an integrated development environment (IDE) for Windows Store app developers. Also like the Interface Building design tool on Xcode 4, which supports storyboarding (Figure 1), Visual Studio 2012 includes a XAML Designer tool (Figure 2).
Figure 1: The Interface Builder in Xcode 4
1.2 The Case Study Project

This article is one of a series based on a case study project. In the project, we ported a medical record management application from iOS to Windows 8. The basic requirements of the application include:

- Show a list of patients
- Show the personal and medical information of a specific patient, which includes identity, billing, vitals, lab tests, medical images, etc.
- Display detailed graphs and images when selected

This article will cover the advanced UI features of the project.

2 High Level UI Design and Navigation Patterns

On iOS, we can use the split-view controller to present a master view and a detailed view on the screen. We can use table views or tab bar views to group different categories of information on the view. Figure 3 shows the split view along with the master table view and the detailed table view. The left pane of the split view shows the scrollable patient list. The right pane shows the medical records associated with the selected patient in the list. We use a table view to put the medical record categories on the same view. We can also use the tab bar view, with each tab view displaying a specific medical record category. Figure 4 shows how this view is created in Xcode 4 storyboard.
<table>
<thead>
<tr>
<th>Medical Records</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FOX LOGAN</strong></td>
<td></td>
</tr>
<tr>
<td>403-551-6894</td>
<td></td>
</tr>
<tr>
<td><strong>MARLENE BRYAN</strong></td>
<td></td>
</tr>
<tr>
<td>651-65-6895</td>
<td></td>
</tr>
<tr>
<td><strong>ANDREWS PERKINS</strong></td>
<td></td>
</tr>
<tr>
<td>773-38-9607</td>
<td></td>
</tr>
<tr>
<td><strong>MARION SWANSON</strong></td>
<td></td>
</tr>
<tr>
<td>981-85-6449</td>
<td></td>
</tr>
<tr>
<td><strong>GORDON GARDNER</strong></td>
<td></td>
</tr>
<tr>
<td>350-98-9184</td>
<td></td>
</tr>
<tr>
<td><strong>ANTONIA BLAKE</strong></td>
<td></td>
</tr>
<tr>
<td>400-14-3658</td>
<td></td>
</tr>
<tr>
<td><strong>HOWELL BECK</strong></td>
<td></td>
</tr>
<tr>
<td>195-85-3659</td>
<td></td>
</tr>
<tr>
<td><strong>PAT HOGAN</strong></td>
<td></td>
</tr>
<tr>
<td>374-23-6181</td>
<td></td>
</tr>
<tr>
<td><strong>FOWLER GILL</strong></td>
<td></td>
</tr>
<tr>
<td>869-15-6000</td>
<td></td>
</tr>
<tr>
<td><strong>DORA DICKERSON</strong></td>
<td></td>
</tr>
<tr>
<td>159-77-2329</td>
<td></td>
</tr>
<tr>
<td><strong>ELLIOTT JORDAN</strong></td>
<td></td>
</tr>
<tr>
<td>503-84-3892</td>
<td></td>
</tr>
<tr>
<td><strong>PAM HENSLEY</strong></td>
<td></td>
</tr>
<tr>
<td>263-86-7621</td>
<td></td>
</tr>
<tr>
<td><strong>BANKS FIGUEROA</strong></td>
<td></td>
</tr>
<tr>
<td>993-95-4166</td>
<td></td>
</tr>
<tr>
<td><strong>VICKY REYNOLDS</strong></td>
<td></td>
</tr>
<tr>
<td>137-56-9881</td>
<td></td>
</tr>
<tr>
<td><strong>SMITH HANSEN</strong></td>
<td></td>
</tr>
<tr>
<td>353-24-2076</td>
<td></td>
</tr>
<tr>
<td><strong>ERICA BUCHANAN</strong></td>
<td></td>
</tr>
<tr>
<td>633-39-6581</td>
<td></td>
</tr>
</tbody>
</table>

**Notes**
- Dr Thomas 2012-03-12
- Dr Lee 2012-03-09

**Vitals**
- RECORD 2012-03-12 11205
- RECORD 2012-03-09 10235
- RECORD 2011-10-18 10250

**Lab Tests**
- LAB RESULT 2012-03-11 11205
- LAB RESULT 2012-03-08 10235

**Images**
- X-Ray 2012-03-05 11205
- ULTRASOUND 2012-03-01 10235

**Name:** FOX LOGAN  
**Gender:** male  
**Status:** Living  
**Date of Birth:** 10-25-1957  
**Last Visit:** 2012-04-23T  
**Insurance:** Aetna Group

Figure 3 On iOS, a split view controller and its master table view and detailed table view
In a Windows Store app, we can accommodate this design by following the Windows Store app hierarchical system of navigation pattern (Figure 5). The first level page shows a grid view that includes a tile for each patient (Figure 6). The second level page is a grouped item page that includes the medical records for the patient selected from the first level page (Figure 7). The third level page is a group detail page that shows the specific category of medical records selected from the second level page (Figure 8). We can also have a fourth level page that shows the item details, for example, the actual X-ray image selected from the third level page.
Figure 5 Windows Store app hierarchical system of navigation.

Figure 6 In the Windows Store app, the root level grid view includes tiles for the patient list.
Figure 7 In the Windows Store app, the second level page shows the medical records associated with the selected patient.

Figure 8 In the Windows Store app, the third level page shows the group selected from the second level page.
In Visual Studio 2012 projects, the UI page is defined with a "XAML" file, and a C# implementation file (.cs) associated with it. Because the transitions from one page to another page usually originate from user actions, for example, when a grid view item is pressed, naturally the event listeners are the places used to handle the navigations. Figure 9 shows our first level page PatientsListPage where we define an ItemsGridView_ItemClick event listener.

```xml
<common:LayoutAwarePage
    x:Name="pageRoot"
    x:Class="PRApp.PatientsListPage"
    DataContext="{(Binding DefaultViewModel, RelativeSource={RelativeSource Self})}"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:PRApp"
    xmlns:common="using:PRApp.Common"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d">
    ...
    <GridView
        x:Name="itemGridView"
        AutomationProperties.AutomationId="ItemsGridView"
        AutomationProperties.Name="Items"
        TabIndex="1"
        Grid.RowSpan="2"
        Padding="116,136,116,46"
        ItemsSource="{(Binding Source={StaticResource itemsViewSource})}"
        ItemTemplate="{(StaticResource PatientsItemStyle1)}"
        SelectionMode="None"
        IsSwipeEnabled="false" IsItemClickEnabled="True" ItemClick="ItemsGridView_ItemClick"/>
    ...
</common:LayoutAwarePage>
```

Figure 9 The GridView in PatientsListPage XAML file includes an ItemGridView_ItemClick event listener (**)

Figure 10 shows in PatientsListPage.xaml.cs, we implement the ItemsGridView_ItemClick method, which calls the Windows Runtime Frame.Navigate() method to create a second level page PatientGroupItemsPage object and show as the current page on screen. When we construct the PatientGroupItemsPage object, we pass in the clicked item, which is a PatientsViewModel object.

```csharp
namespace PRApp
{
    /// <summary>
```
/// A page that displays a collection of item previews. In the Split Application this page
/// is used to display and select one of the available groups.
/// </summary>

{
...
    void ItemsGridView_ItemClick(object sender, ItemClickEventArgs e)
    {
        this.Frame.Navigate(typeof(PatientGroupItemsPage), ((PatientsViewModel)e.ClickedItem));
    }
...}

---

In the LoadState method in PatientGroupItemsPage.xaml.cs (Figure 11), we retrieve the patient object from this parameter and construct the Groups collection for the grid view data model.

/// <summary>
/// Populates the page with content passed during navigation. Any saved state is also
/// provided when recreating a page from a prior session.
/// </summary>
/// <param Title="navigationParameter">The parameter value passed to
/// <see cref="Frame.Navigate(Type, Object)"/> when this page was initially requested.
/// </param>
/// <param Title="pageState">A dictionary of state preserved by this page during an earlier
/// session. This will be null the first time a page is visited.</param>
protected override void LoadState(Object navigationParameter, Dictionary<String, Object> pageState)
{
    // TODO: Assign a collection of bindable groups to this.DefaultViewModel["Groups"]

    PatientsViewModel patient = (PatientsViewModel)navigationParameter;
    this.DefaultViewModel["Groups"] = pgipvm.GetGroups(patient.id);
}

---

Figure 10 PatientsListPage.cs implements the ItemsGridView_ItemClick event listener (**)

Figure 11 The LoadState method of PatientGroupItemsPage.xaml.cs
3  Windows Project Templates and Data Binding

In Figure 7 and Figure 8, items are grouped and shown nicely in the grid views. Visual Studio 2012 Windows Store project templates provide a powerful basis to construct these user interface pages. The predefined project templates include grouped items page, group detail page, item detail page, etc. We use the X-rays group detail page as an example here.

In Visual Studio 2012’s Solution Explorer window, right click the project name and select “Add -> New Item...” from the pop-up menu. Select “Visual C#” on the left pane. On the center pane, we see the list of the predefined page templates. Among those templates, we select “Group Item Page.” A preview of the template is shown on the right pane. We also enter a name for the page in the text box at the bottom of the dialog (Figure 12), and press the “Add” button. Visual Studio 2012 now generates a file named “XRayImagesGroupDetailPage.xaml” file and a file named “XRayImagesGroupDetailPage.xaml.cs” in the project.

![Add New Item dialog shows the Window Store project templates](image)

Figure 12 Add New Item dialog shows the Window Store project templates (***)
If we inspect the generated XRayImagesGroupDetailPage.xaml file, we can see this page is bound to “DefaultViewModel” data context and the grid view items in this page are bound to the “Items” as the collection view source (Figure 13).

```xml
<common:LayoutAwarePage
  x:Name="pageRoot"
  x:Class="PRApp.XRayImagesGroupDetailPage"
  DataContext="{(Binding DefaultViewModel, RelativeSource={RelativeSource Self})}"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:PRApp"
  xmlns:common="using:PRApp.Common"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  mc:Ignorable="d">
  <Page.Resources>
    <!-- Collection of items displayed by this page -->
    <CollectionViewSource
      x:Name="itemsViewSource"
      Source="{Binding Items}"/>
  </Page.Resources>
</common:LayoutAwarePage>

Figure 13 XRayImagesGroupDetailPage.xaml specifies the binding data source (**).
In Visual Studio 2012, if we expand the "Common" folder generated under the project (Figure 15), we can see Visual Studio has generated a group of files under it. Among these files, LayoutAwarePage.cs contains the class that we derive the XRayImagesGroupDetailPage from.

Figure 15 The "Common" folder in the project.

The “Common” folder also includes the “BindableBase.cs” file. We derive a data model for the view from this class. Figure 16 provides the outlines of the XRayImagesGroupDetailPageViewModel class.

```csharp
namespace PRApp.ViewModels
{
    public abstract class XRayImageDataCommon : BindableBase
    {

    }

    public class XRayImageDataItem : XRayImageDataCommon
    {

    }
}
```
public class XRayImageDataGroup : XRayImageDataCommon
{
    
    private ObservableCollection<XRayImageDataItem> _items = new ObservableCollection<XRayImageDataItem>();
    public ObservableCollection<XRayImageDataItem> Items
    {
        get { return this._items; }
    }
    
    
}

public sealed class XRayImagesGroupDetailPageViewModel
{
    
    public static XRayImageDataGroup GetGroup(string uniqueId)
    {
    
    }
    
}

Figure 16 XRayImagesGroupDetailPageViewModel.cs, which the group and item classes derived from BindableBase class (**)

To connect the view and the data sources, in XRayImagesGroupDetailPage.xaml class's LoadState method, we pass the PRApp.CommonBindableBase derived group and items object to DefaultViewModel (Figure 17).

/// <summary>
/// Populates the page with content passed during navigation. Any saved state is also
/// provided when recreating a page from a prior session.
/// </summary>
/// <param name="navigationParameter">The parameter value passed to
/// <see cref="Frame.Navigate(Type, Object)"/> when this page was initially requested.
/// </param>
/// <param name="pageState">A dictionary of state preserved by this page during an earlier
/// session. This will be null the first time a page is visited.\</param>
protected override void LoadState(Object navigationParameter, Dictionary<String, Object> pageState)
Figure 17 In XRayImagesGroupDetailPage.cs, the view items are connected with the data source in the LoadState method. (**)

4 Summary

In this article, we used an EMR app to investigate how to port some advanced UI features from iOS to the new Windows Store app platforms. From this case study, we can see Windows Runtime provides a rich set of features and development tools for developing impressive UI features.

About the Author

Miao Wei is a software engineer with Intel Corporation's Software and Services Group. He currently works on the Intel Atom™ Scale and Enabling projects. He has over 15 years of experience in developing mobile platforms, web browsers, location-based applications and services, digital map databases, and other software products.

Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel, the Intel logo, and Atom are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others

**This sample source code is released under the Intel OBL Sample Source Code License (MS-LPL Compatible), Microsoft Limited Public License and Visual Studio 2012 License. Copyright© 2012 Intel Corporation. All rights reserved.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain
optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804