Session 3: Page Navigation and Data binding – Demos

# Demo 1: Navigation (5 minutes)

In this demo, you add a secondary command which navigates to a new page – the ‘about’ page.

1. Open solution **1 – Page navigation – BEGIN**.
2. Open **MainPage.xaml** if not already open.
3. Show the **Document Outline** window
   1. Expand **BottomAppBar** to locate **SecondaryCommands**
   2. Right-click on SecondaryCommands, then click **Add AppBarButton**
   3. Explain that although Windows Phone can only show text menu items for secondary commands, the XAML used can be the same on both Windows and Windows Phone – just on the latter, this AppBarButton does not render as a button, but just uses the ‘Label’ value as the text menu item.
   4. Change the **Label** of the AppbarButton to **about**.
   5. At the top of the Properties window, click the lightning flash button to switch to event handlers. Double-click on the box next to the **Click event** to create an event handler for the menu item.
4. Before coding the event handler, add the AboutPage.xaml
   1. Right-click on the project file in Solution Explorer, then click **Add existing item**…
   2. In the Add existing item window, navigate to the **Items** folder in the root of the demo files folder for this module. Double-click on **AboutPage.xaml**. This adds the page and its code file to the project.
5. Write the following code in the click event handler method:

private void AppBarButton\_Click(object sender, RoutedEventArgs e)

{

Frame.Navigate(typeof(AboutPage));

}

1. Run and show the appbar secondary command working.
   1. Click the Back key while the AboutPage is displaying. The app suspends!
   2. This is because this project was created originally using the Blank app new project template, which has no Hardware Back key handling in it.
2. Stop the app debugging so we can fix this.
   1. Go into App.xaml.cs and add the following into the App constructorhod:

protected override void OnNavigatedTo(NavigationEventArgs e)

{

Windows.Phone.UI.Input.HardwareButtons.BackPressed += HardwareButtons\_BackPressed;

}

* 1. Create the HardwareButtons\_BackPressed event handler as follows:

private void HardwareButtons\_BackPressed(object sender, Windows.Phone.UI.Input.BackPressedEventArgs e)

{

Frame rootFrame = Window.Current.Content as Frame;

if (rootFrame != null && rootFrame.CanGoBack)

{

rootFrame.GoBack();

e.Handled = true;

}

}

1. Re-run the demo, but show that now, when you press HW Back from the About page, the backwards page navigation happens, and Back from the MainPage still suspends the app.

# Demo 2: Cache Mode (5 minutes)

This simple demo shows how cache mode affects page usage.

1. Open solution **2 – CacheModeDemo – BEGIN**.
2. Run the app
   1. Demonstrate that the app has three pages – tell them that it has Hardware back key handling wired in, same as shown in the previous demo.
   2. Show that when you click on the button on MainPage, you navigate to Page 2, and the button on Page 2 navigates to page 3. Hardware Back navigates backwards – all very simple.
   3. Navigate to Page 2 and enter some random values in the TextBox fields, and click one of the radio buttons. Then click the **To Page 3** button.
   4. Navigate back again to Page 2 – all the va,yes you entered on age 2 are lost – not what the user would probably expect.
   5. Stop debugging.
3. Edit the Page 2 constructor.
   1. Change the **navigationCacheMode** to **NavigationCacheMode.Required**.
   2. Rerun the previous demo – this time the state of the page is retained and the values entered are saved between visits to the page. Optional: the hash code of the Page object is shown in the Debug window, so you can show that to drive home the point that this is the same object.
   3. Navigate back to MainPage, and then forward again to Page 2 once more – still the input values are retained because it’s the same object – but once again, perhaps on a Forward navigation, a user might expect to see a clean, empty page (though of course that depends on the application scenario).
4. Explain that one way of adding the correct behaviors to your app is to use NavigationHelper which is a set of helper classes that are included in most of the more complex new prohect templates.
   1. To add navigationHelper to a basic app such as this that doesn’t have it, simply Add a New Item of type BasicPage.
   2. Do this now – you will be prompted to add the helper files, which end up in the Common folder.
   3. Open navigationhelper.cs and show the comments block that tells you how to wire up a page to use it.
   4. We’ll look at programming it in just a moment after a few more slides…

# Demo 3: NavigationHelper (5 minutes)

This simple demo shows how to program NavigationHelper and how it helps you manage page state.

1. Open solution **3 – CacheModeNavigationHelperDemo – End**.
2. Explain that this is a modified version of the previous demo, where all the pages have been modified to use navigationhelper, and the Hardware backKey handling has been removed from App,xaml.cs – because HardwareBack key is now handled by navigationHelper.
   1. Show the Navigationhelper logic coded into MainPage.xaml.cs
   2. Show that the Hardware back key handling logic is commented out of App.xaml.cs
3. Run the app – show that values entered into page2 are retained across page navigations, and that Hardware back key handling is correct
4. Say that: There’s much more to NavigationHelper – and it’s partner SuspensionManager – which help you save the state of your app across Suspension and Termination app lifecycle states – we will discuss that in more details in Module 7 – App Lifecycle.

# Demo 4: Data binding (5 minutes)

In this demo, you replace the hard-wired data values set directly in the XAML with a much more realistic approach of having data classes that expose the data for the application, and using data binding to get the XAML to load data from a data object – or ‘viewmodel’ as we call such classes.

1. Open solution **4 – Data Binding – BEGIN**.
2. In **Windows Explorer**, select the **Data** and **DataModel** folders from the **Items** folder in the root of the demo files for this module, and drag them onto the Project file in Solution Explorer.
3. Show the contents of **Recipes.txt** in the **Data** folder which contains the raw data – find item with the id 2000 – which is our quiche Lorraine.
4. Open **RecipeDataSource.cs** in the DataModel folder, and find the methods **LoadLocalDataAsync** and **GetItem** (in class **RecipeDataSource**) – these are the methods we will use (the other methods will be useful later on when we build out the full ContosoCookbook). Also say that we’ll come to SampleDataSource.cs in a moment.
5. Add the code to load in the data:
   1. In **App.xaml.cs**, add the following to **OnLaunched** (add the **async** keyword to the method signature):

protected async override void OnLaunched(LaunchActivatedEventArgs e)

{

**// Load recipe data**

**await ContosoCookbookSimple.DataModel.RecipeDataSource.LoadLocalDataAsync();**

**…**

* 1. In MainPage.xaml.cs, OnNavigatedTo, comment out the existing code, and add the following instead:

protected override void OnNavigatedTo(NavigationEventArgs e)

{

//List<string> ingredients = new List<string> {

// "Shortcrust Pastry: ","220g plain flour","120g cold butter",

// "1 tablespoon very cold water","1 large egg",

// "separated into yolk and white","Filling: ","1 teaspoon oil",

// "1 small onion","220g bacon","finely chopped 5 large eggs",

// "125ml cream","1/4 teaspoon ground nutmeg Salt and pepper",

// "110g grated tasty cheddar cheese"

//};

//IngredientsListBox.ItemsSource = ingredients;

var item = ContosoCookbookSimple.DataModel.RecipeDataSource.GetItem("2000");

this.DataContext = item;

}

* 1. Modify the XAML in MainPage.xaml.cs:

<Page.BottomAppBar>

<CommandBar>

<CommandBar.SecondaryCommands>

<AppBarButton Label="about" Icon="Help" Click="AppBarButton\_Click"/>

</CommandBar.SecondaryCommands>

<AppBarButton Icon="List" Label="ingredients">

<AppBarButton.Flyout>

<Flyout>

<Grid Height="400" Margin="0,25,0,0">

<ListBox x:Name="IngredientsListBox" ItemsSource="{Binding Ingredients}"/>

</Grid>

</Flyout>

</AppBarButton.Flyout>

</AppBarButton>

</CommandBar>

</Page.BottomAppBar>

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<StackPanel Margin="10,25,0,30">

<TextBlock Style="{StaticResource TitleTextBlockStyle}" Text="CONTOSO COOKBOOK" Margin="0,20,0,0"/>

<TextBlock Style="{StaticResource HeaderTextBlockStyle}" Text="{Binding ShortTitle}" Margin="0,20,0,0"/>

</StackPanel>

<Grid Grid.Row="1" Margin="10,0,0,70">

<Grid.RowDefinitions>

<RowDefinition Height="109\*"/>

<RowDefinition Height="155\*"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Image Source="{Binding ImagePath}" Stretch="UniformToFill"/>

<ScrollViewer Grid.Row="1" Margin="0,10,0,0">

<TextBlock x:Name="InstructionsTextBlock" TextWrapping="Wrap"

Style="{StaticResource BodyTextBlockStyle}"

Text="{Binding Directions}"/>

</ScrollViewer>

<StackPanel Grid.Row="2" Orientation="Horizontal">

<TextBlock Text="Prep time:" Style="{StaticResource BodyTextBlockStyle}"

Foreground="{StaticResource PhoneAccentBrush}"/>

<TextBlock x:Name="PrepTimeTextBlock" Text="{Binding PrepTime}" Style="{StaticResource BodyTextBlockStyle}" Margin="10,0,0,10"/>

</StackPanel>

</Grid>

1. Run – show that it works.
2. Change the ID in OnNavigatedTo to 2001, show that it works with different data.

# Demo 5: Design-time Data (5 minutes)

1. Continue with the previous solution – or solution **5 – Data Binding – BEGIN**.
2. Problem here is that we have degraded the design time experience – it’s very hard for designers or developers to visualize what the output will look like when the tools display nothing.  
   We can solve this by providing some data that is used only at design time, not at runtime.
   1. Open **DataModel\SampleDataSource** and show what’s in it. It mimics the properties of the real RecipeDataSource but plugs in some dummy values.  
      Point out that it uses some dummy images called LIghtGray.png, MediumGray.png and DarkGray.png.
   2. Right-click on the **Assets** folder, then click **Add existing items**
      1. In the Add existing item window, navigate to **Items\Assets** and add **LIghtGray.png**, **MediumGray.png** and **DarkGray.png**.
   3. In MainPage.xaml, modify the <Page> element as follows:

<Page

x:Class="ContosoCookbookSimple.MainPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="using:ContosoCookbookSimple"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:data="using:ContosoCookbookSimple.DataModel"

mc:Ignorable="d"

d:DataContext="{Binding Source={d:DesignInstance Type=data:SampleDataItem, IsDesignTimeCreatable=True}}">

* 1. Explain that the **xmlns:data** attribute declares an xml namespace we can use to refer to objects in the ContosoCookbookSimple.DataModel namespace – such as SampleDataSource
  2. The **d:DataContext** line creates an instance of SampleDataItem which it uses as the DataContext at runtime only.

1. Build the project. The designer should now be showing some meaningful data and a dummy gray image.