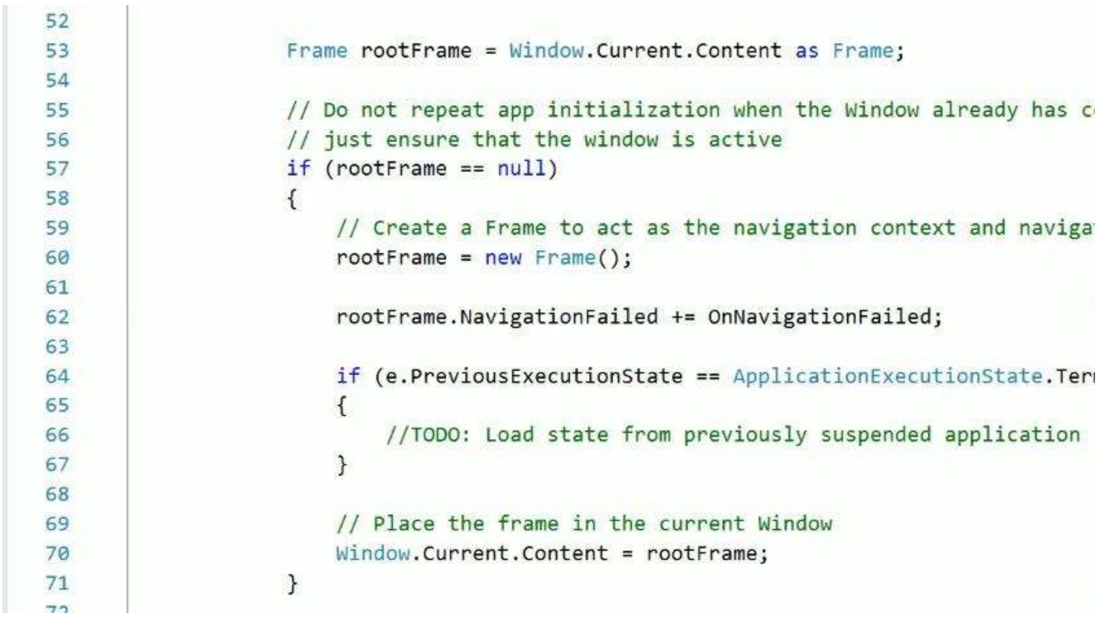
UWP-019 - Working with Navigation

Up until now we've only created apps with a single Page, the MainPage.XAML, and while that's fine for simple apps. However, it's likely that you will need to add additional pages and navigate to them. Typically, you would want to load a new page into your app whenever you need new functionality, new layout, or new content. It's possible to create a complex app and change out the layout and the functionality all in a single page, but by adding additional pages it allows you to decompose functionality into smaller, more manageable units, and this will reduce the complexity of the application's development and its maintenance.

So Pages in the Universal Windows Platform are roughly equivalent to web pages in web development. As a web developer you might create several webpages and create hyperlinks between the pages so that users can navigate in between pages in the website. The end user would load up their frame (i.e., their web browser) and would navigate to your home page. From there they might then navigate to each of the pages using the hyperlinks, or they might navigate then, at some point, using the back and the forward buttons on the toolbar of the frame, (again, their web browser).

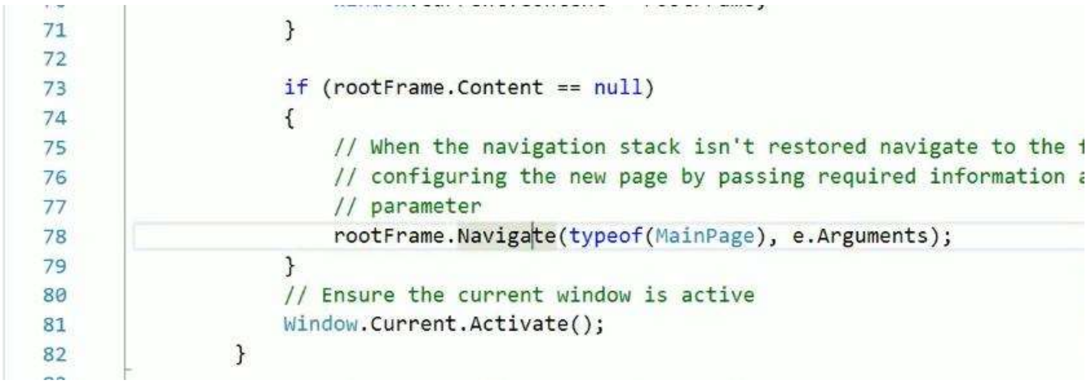
So in Universal Windows Platform your pages are hosted in Frame objects (that's why I keep using that word instead of the web browser). You may not realize this, but the Frame is critical to the application itself. As your application starts up, there is an Application object and then there is a Window inside of that, so that will give you all of the chrome in a Window's Desktop version -- the close, minimize, and the maximize buttons in the upper right-hand corner, the title, etc. along the top bar.

Inside of the Window sits that Frame which will host pages. As you can see in this new example that I've created named "NavigationExample", in the App.xaml.cs file in line 53, you can see where this all gets set up.



```
52
53                  Frame rootFrame = Window.Current.Content as Frame;
54
55                  // Do not repeat app initialization when the Window already has c
56                  // just ensure that the window is active
57                  if (rootFrame == null)
58                  {
59                      // Create a Frame to act as the navigation context and naviga
60                      rootFrame = new Frame();
61
62                      rootFrame.NavigationFailed += OnNavigationFailed;
63
64                      if (e.PreviousExecutionState == ApplicationExecutionState.Ter
65                      {
66                          //TODO: Load state from previously suspended application
67                      }
68
69                      // Place the frame in the current Window
70                      Window.Current.Content = rootFrame;
71                  }
```

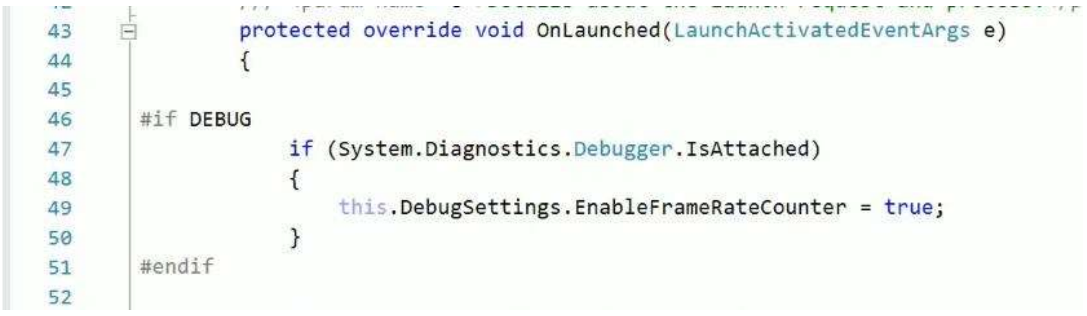
A lot of the hard work related to the interaction of the Application, Window and Frame is done for you automatically whenever you use this blank template. You can see where we get a reference to the Frame inside of the Window, and they call it rootFrame, and then in the rootFrame, you can navigate to a specific page in your application.



```
            }

            if (rootFrame.Content == null)
            {
                // When the navigation stack isn't restored navigate to the f
                // configuring the new page by passing required information a
                // parameter
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            // Ensure the current window is active
            Window.Current.Activate();
        }
```

In this case, we're navigating to MainPage, so we give it a type of a data type that we want it to navigate to, and then it creates the instance of that type, and it will pass in parameters as arguments.

While we're here, I'm going to delete this diagnostic …



```
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {

#if DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                this.DebugSettings.EnableFrameRateCounter = true;
            }
#endif

```

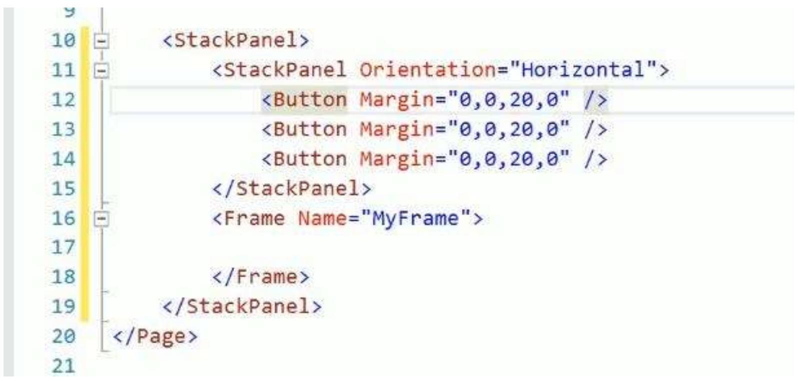
At some point later on we will learn about loading new Pages into this topmost Frame, the rootFrame, but for getting started I think it would be easier to just talk about adding a Frame to our MainPage.xaml so that we can load individual Pages in and out of that Frame that's hosted by the MainPage.

To illustrate this, I'll add three new Pages by selecting the Project name in the Solution Explorer, then selecting the Project menu > Add New Item … In the dialog, make sure that you select the Blank Page Template. Create three Pages named Page1, Page2 and Page3.

These will be the Pages that are loaded into and out of the MainPage's Frame.

In the MainPage.xaml I add a StackPanel to replace the top-most Grid.

I’ll add another StackPanel and a Frame inside the top-most StackPanel, and I am going to give the Frame a name so I can access it programmatically. I’ll name it “MyFrame”. The StackPanel will be used to create a horizontal button bar, so I set the Orientation="Horizontal", and create three Buttons and set the Margin of each to "0,0,20,0".



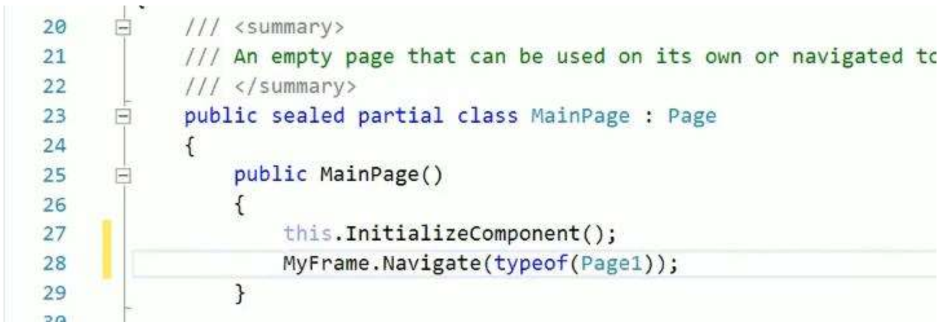
```
    <StackPanel>
        <StackPanel Orientation="Horizontal">
            <Button Margin="0,0,20,0" />
            <Button Margin="0,0,20,0" />
            <Button Margin="0,0,20,0" />
        </StackPanel>
        <Frame Name="MyFrame">

        </Frame>
    </StackPanel>
</Page>
```

The first Button will be named “HomeButton” and I’ll set the Content="Home", and add a click event named “HomeButton_Click”.

The second Button will be named “BackButton” and I’ll set the Content="Back", and add a click event named “BackButton_Click”.

The third Button will be named “ForwardButton” and I’ll set the Content="Forward" and add a click event named “ForwardButton_Click”.

As our Frame loads along with the MainPage, I want to load Page1 into view automatically. In the code behind file MainPage.xaml.cs I’ll located the constructor for the Page, I will access MyFrame, and call its Navigate method. Just like we saw in the App.xaml.cs, we will need to navigate to a typeof(Page1).

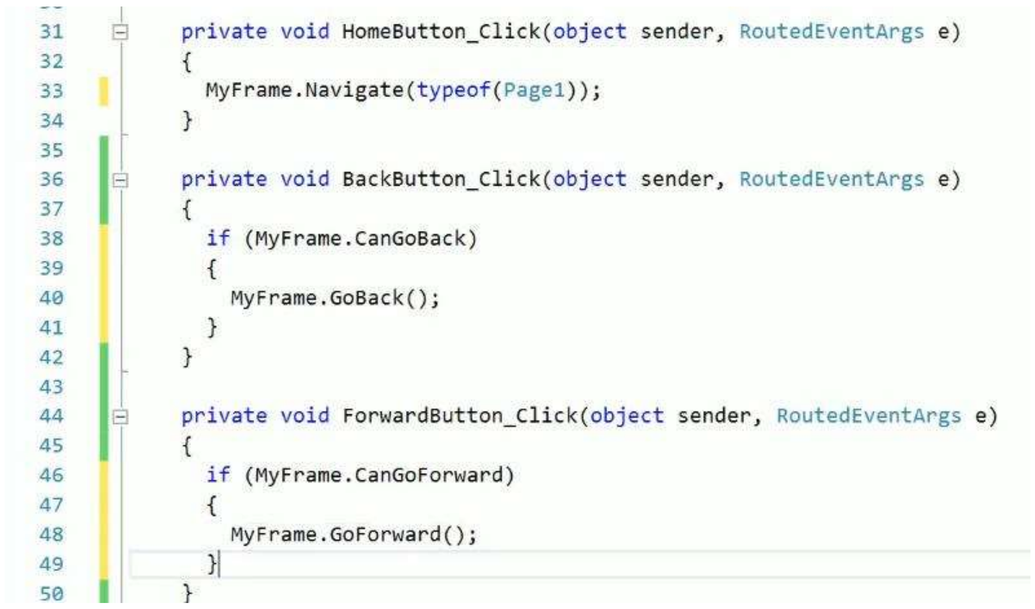


```
    /// <summary>
    /// An empty page that can be used on its own or navigated tc
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            MyFrame.Navigate(typeof(Page1));
        }
```

That should load Page1 during MainPage construction.

Next, I’ll create method stubs for each of the three buttons by putting my mouse cursor in the Click property and pressing F12.

Next, I’ll load pages into MyFrame with each button click.



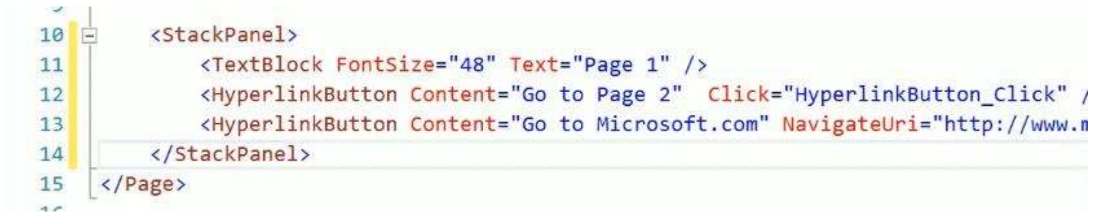
```
private void HomeButton_Click(object sender, RoutedEventArgs e)
{
  MyFrame.Navigate(typeof(Page1));
}

private void BackButton_Click(object sender, RoutedEventArgs e)
{
  if (MyFrame.CanGoBack)
  {
    MyFrame.GoBack();
  }
}

private void ForwardButton_Click(object sender, RoutedEventArgs e)
{
  if (MyFrame.CanGoForward)
  {
    MyFrame.GoForward();
  }
}
```

The HomeButton is easy. Whenever it is clicked the Frame will load Page1.

However, in the Back and Forward buttons I'll invoke the Frame's history, or rather the BackStack. The Frame will maintain the history of pages that the user navigates through, and it will replay those navigation requests whenever we go back or forward. To use this, we should ask the navigation's BackStack, "Can we go forward? "Can we go backward? And if we can, then go ahead and go back, or go ahead and go forward, respectively."

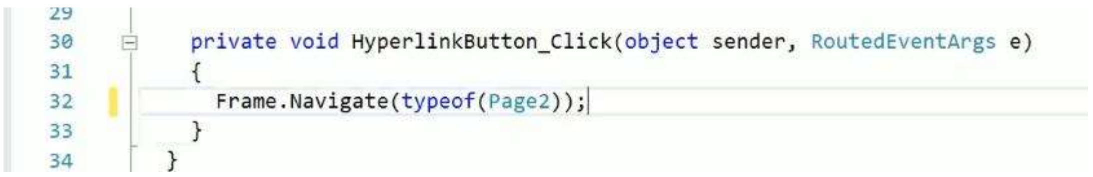
If MyFrame.CanGoBack is true then we'll call GoBack(). Similarly, if MyFrame.CanGoForward is true then we'll call GoForward().

Next I'll open Page1.xaml and replace the Grid with a StackPanel. Inside the StackPanel I'll add TextBlock, I'll make the font size large, and I'll set the Text equal to "Page1". Also, I'll add a HyperlinkButton. I could add any type of control where I can handle an event, and handle it to tell the Frame to navigate to a given Page. I use the HyperlinkButton because it connotes navigation, just like a web page's hyperlink connotes navigation. The HyperlinkButton has a special property that makes it different from a Button that's simply styled to look like a hyperlink. The HyperlinkButton has a NavigateURI property that will allow us to navigate out to a website. I'll add a second copy of the HyperlinkButton just to demonstrate that and I'll set the NavigateURI property to http://www.microsoft.com:



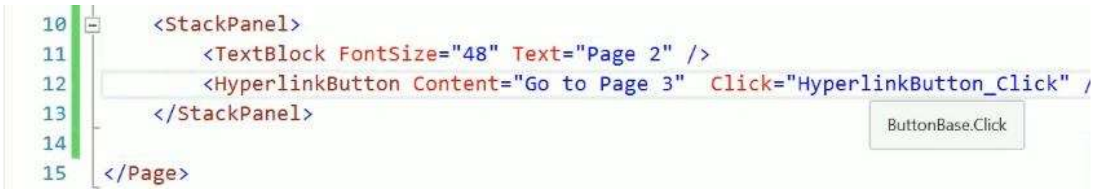
```
    <StackPanel>
        <TextBlock FontSize="48" Text="Page 1" />
        <HyperlinkButton Content="Go to Page 2"  Click="HyperlinkButton_Click" /
        <HyperlinkButton Content="Go to Microsoft.com" NavigateUri="http://www.m
    </StackPanel>
</Page>
```

And I’ll handle the HyperlinkButton_Click event:



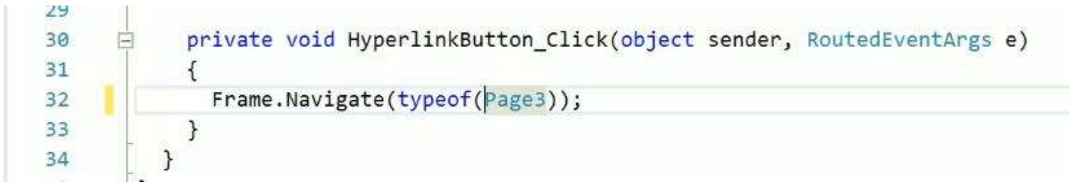
```
    private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
    {
      Frame.Navigate(typeof(Page2));
    }
  }
```

I’ll do the same on Page2.xaml:



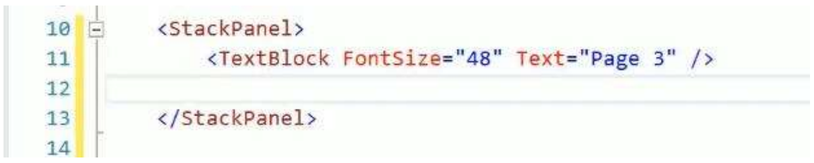
```
    <StackPanel>
        <TextBlock FontSize="48" Text="Page 2" />
        <HyperlinkButton Content="Go to Page 3"  Click="HyperlinkButton_Click" /
    </StackPanel>

</Page>
```

And I’ll add the code to navigate to Page3:



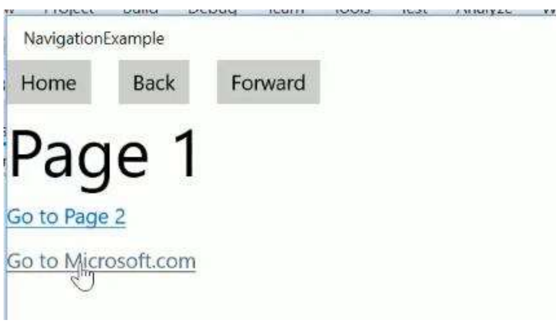
```
    private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
    {
      Frame.Navigate(typeof(Page3));
    }
  }
```

Finally, on Page3 I’ll simply add a TextBlock:



```
    <StackPanel>
        <TextBlock FontSize="48" Text="Page 3" />

    </StackPanel>

```

If you run the application you will be able to click the “Go to Microsoft.com” to load that website into a web browser, or click the “Go to Page 2” and “Go to Page 3” links and navigate to those Pages in the Frame.

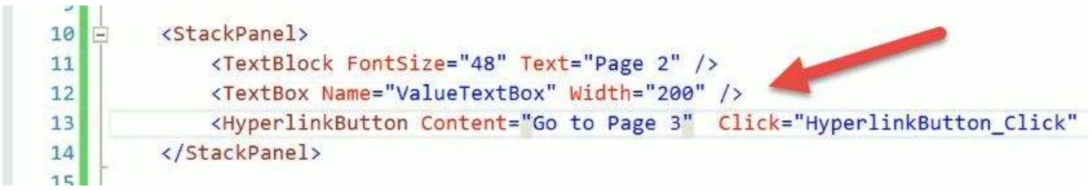


I should also be able to click the Home, Back and Forward buttons to traverse the BackStack.

You might wonder why am I referenced Frame and not MyFrame in the code above? After all, the Frame's name is MyFrame, right? True, whenever we call the Navigate method, and we pass in the type that create a new instance of, that Navigate method will create that instance and will act as a factory in so much that it will set up the new instance of that object by setting its various properties. One of the properties that it sets for a given page is the Frame property, and sets it to whatever the parent Frame is, so that's how we get a reference to the current Frame that we're sitting inside of in order to call its Navigate method.

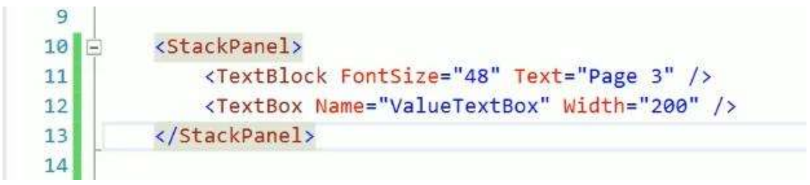
So that's the most basic scenario that we can cover. Let's add a wrinkle and pass values from Page2 to Page3.

I add a TextBox to Page2 and name that ValueTextBox.



```
<StackPanel>
    <TextBlock FontSize="48" Text="Page 2" />
    <TextBox Name="ValueTextBox" Width="200" />
    <HyperlinkButton Content="Go to Page 3"  Click="HyperlinkButton_Click"
</StackPanel>
```

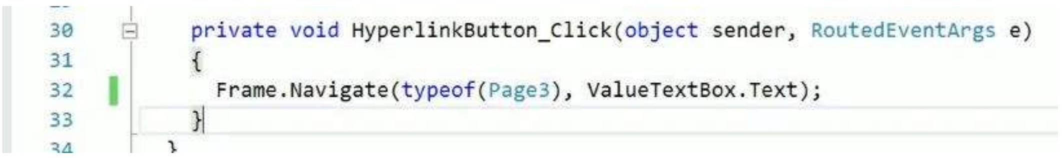
I add a TextBox to Page3 and also name that ValueTextBox.



```
<StackPanel>
    <TextBlock FontSize="48" Text="Page 3" />
    <TextBox Name="ValueTextBox" Width="200" />
</StackPanel>
```

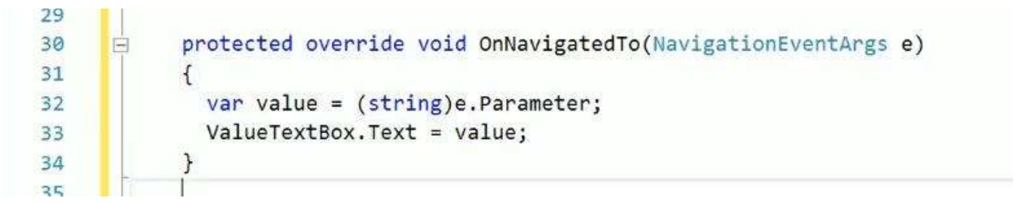
Now what I want to do when I navigate from Page 2 to Page 3, I want to pass along the value that the user typed in the ValueTextBox on Page2 and then display that value in the ValueTextBox in Page 3. How do we do that?

If we take a look at Page2.xaml.cs, there is an overloaded version of the Navigate method that allows us to pass in anything we want as a parameter that is then retrieved by the other page that we're navigating to, so in this case just take ValueTextbox.Text, and pass it over to the other side.



```
private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
  Frame.Navigate(typeof(Page3), ValueTextBox.Text);
}
```

Now retrieving it on the other side is a little bit trickier. On the Page3.xaml.cs we will have to override the OnNavigatedTo method which fires every time that we navigate to this Page.



```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
  var value = (string)e.Parameter;
  ValueTextBox.Text = value;
}
```

Notice that we access the value using the NavigationEventArgs' Parameter property. Since the value we pass is passed as an Object data type, it can hold any type, therefore we must cast it to string.

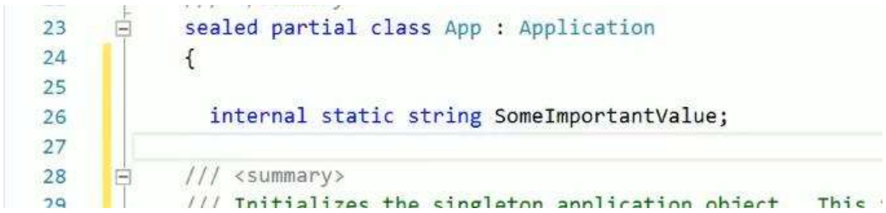
Running the application again, we can send test from Page2, navigate to Page3 and display the value on Page3.

But what happens when click the Back button. The value is no longer in Page2's TextBox. However, if we then click the Forward button the value is still there. What happened?

Remember I said whenever you're navigating through the BackStack -- through the history of the Frame – it will replay the Navigation event. In our case, the step of navigating from Page 2 to Page 3 included some text, so it gets picked up as we replay that event and displayed into Page3's TextBox. However, when we go back to Page 2, when we navigated from Page 1 to Page 2, we didn't send anything over. It replays that event, and there is nothing in Page2's TextBox as a result of it.

But what if we wanted to maintain state between these pages? There are several different ways we can go about this. Some experienced developers might caution you against this approach, and they're probably right – if you were building an enterprise scale application, however I want to keep this as simple as possible, and as long as you don't do anything abusive to your application this technique should be fine. Just keep in mind that there are probably more elegant solutions than what I'm about to do.

So since we said that the Application sits at the very top of the stack when you launch an app, there is an Application object that hosts a Window, that hosts a Frame, that hosts MainPage. What we can do is treat this App class, this instance of Application, as a global variable. In other words, it is accessible throughout the application. So inside of the class's definition I can add a public (internal) field:

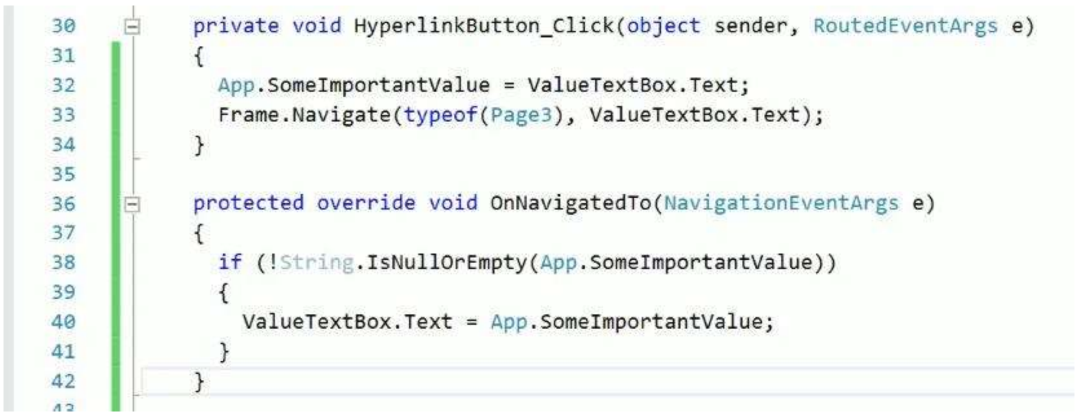


```
sealed partial class App : Application
{

  internal static string SomeImportantValue;

/// <summary>
/// Initializes the singleton application object.  This
```

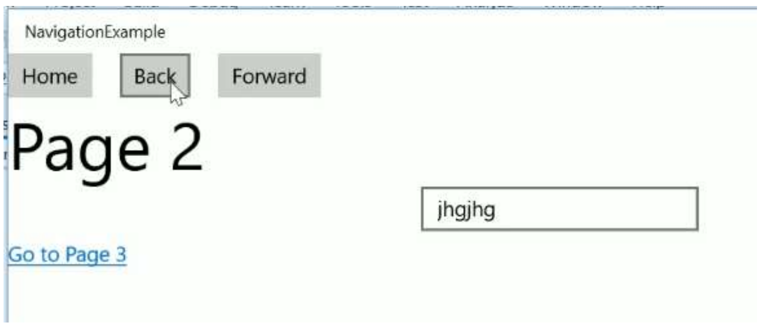
SomeImportantValue is a static string. Internal means that anything inside of this app can access that variable, but nothing outside of the application.

Back in Page2.xaml.cs I modify the Hyerplinkbutton_Click event to save the value typed into the ValueTextBox into the SomeImportantValue field.

Next, I override the OnNavigatedTo() method and whenever navigation returns to this page I can see if the ValueTextBox.Text was ever set to any value by checking to see if it is not null or empty. If it was, then I can reset the ValueTextBox to the field SomeImportantValue:


```
private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
  App.SomeImportantValue = ValueTextBox.Text;
  Frame.Navigate(typeof(Page3), ValueTextBox.Text);
}

protected override void OnNavigatedTo(NavigationEventArgs e)
{
  if (!String.IsNullOrEmpty(App.SomeImportantValue))
  {
    ValueTextBox.Text = App.SomeImportantValue;
  }
}
```

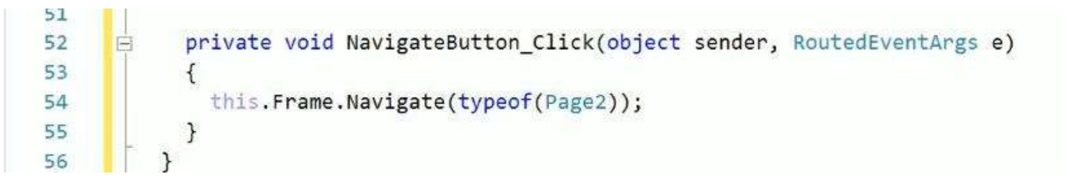
Replaying that scenario from earlier, we can retrieve the original value typed into Page2 when navigating back to that Page.



Next, I'll demonstrate how to navigate to a whole different page at the RootFrame level.

Let's go to MainPage.xaml and add one more Button named NavigateButton, and I'll set the Content="Navigate Root Frame", and set the Click equal to a new event handler.

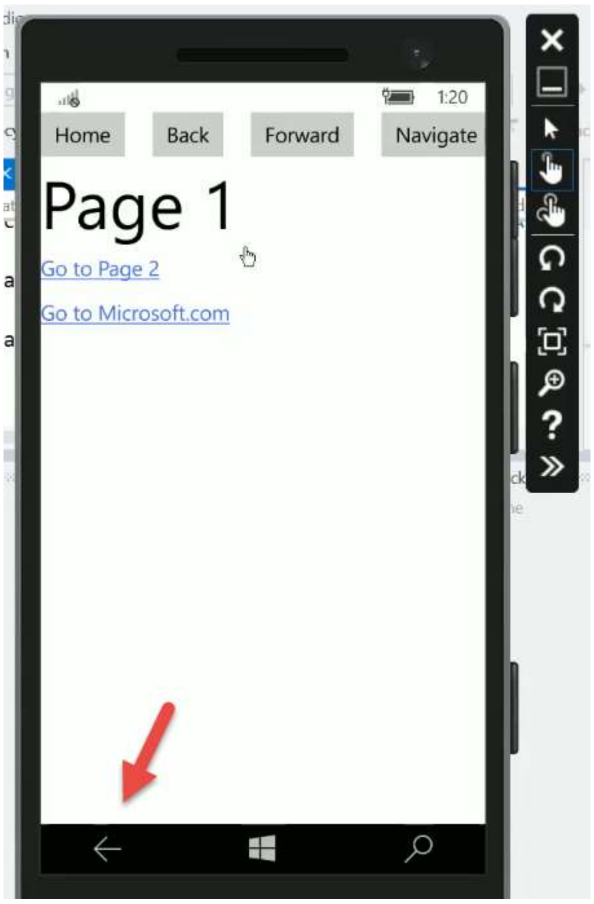
I press F12 to work in the button's click event method stub. Remember we said that Page has a Frame property, so I'm just going to use the "this" keyword to access this instance of MainPage, then reference the Frame's Navigate() method and will pass in type of Page2:


```
  private void NavigateButton_Click(object sender, RoutedEventArgs e)
  {
    this.Frame.Navigate(typeof(Page2));
  }
}
```

When I run the application I now automatically navigate to Page 2. However, as a result of that, I've lost the chrome, the outermost StackPanel, and I've lost my navigation buttons because I'm loading a new Page (Page2) to replace MainPage.

Finally, let's talk about what happens whenever we run this on a mobile device because the phone is unique regarding navigation when compared to other device families.

Below you can see that we have this representation of the back button here on the chrome of our emulator.



If we were to go to Page 2, type in data and go to Page 3, and then we can go back, and the can go forward using the buttons we created at the top of the Page.

What if I were to tap this back arrow at the bottom? What we might want to happen is that go back to Page 2, then to Page One, but when I hit the back button, it completely escapes out of our application and goes back to the Home Page, and, unfortunately, how to fix this will require a little bit more explanation and we won't cover that in this series. There are lengthy articles that explain how to handle this scenario.