

Hand­s-on lab

Lab 2: Handling Page Navigation and Back

August 2015

Contents[Overview 3](#_Toc429138641)

[Exercise 1: Page Navigation 5](#_Toc429138642)

[Task 1 – Create a blank Universal Windows app 5](#_Toc429138643)

[Task 2 - Create a view 8](#_Toc429138644)

[Task 3 – Introduce navigation 11](#_Toc429138645)

[Task 4 – Pass a parameter to Page 2 13](#_Toc429138646)

[Task 5 – Display the message passed to Page2 14](#_Toc429138647)

[Exercise 2: Handling Back 16](#_Toc429138648)

[Task 1 – Enable the shell-drawn back button 16](#_Toc429138649)

[Task 2 - Define the standard back-requested pattern 19](#_Toc429138650)

[Task 3 – Virtual back button 20](#_Toc429138651)

[Task 4 – Handling forward navigation 24](#_Toc429138652)

[Summary 27](#_Toc429138653)

Overview

With UWP, Microsoft has introduced tools to help you provide a seamless navigation experience across devices. While a UWP app will run on any Windows 10 device, we still must plan for scenarios where the hardware differs.

The SystemNavigationManager class lets you opt-in to display a software back button when a hardware back button isn't available. This new ability allows you to handle back navigation without extension SDKs or adaptive code. Previously in Windows 8 apps, a back button drawn in the app content was the only back navigation option available for tablet and desktop devices. Now you may choose whichever method works best for your app's implementation, design, and user experience.

In this lab, we will cover the basics of navigation and how to implement it in a simple UWP app. We will explore how we can pass data between pages of the application as we navigate and handle back navigation with the shell-drawn back button.

# Objectives

* 1. This lab will show you how to:
  + Create a new XAML view
  + Navigate to a secondary page
  + Pass a parameter on navigation
  + Enable the shell-drawn back button when the backstack is populated
  + Implement the standard back-requested pattern

# System requirements

* 1. You must have the following to complete this lab:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015

# Optional add-ons

* 1. If you wish to complete the optional tasks in this lab, you will need:
  + Windows 10 Mobile Emulator

# Setup

* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 10.
  3. Install Microsoft Visual Studio 2015. Choose a custom install and ensure that the Universal Windows App Development Tools are selected from the optional features list.
  4. Optional: Install the Windows 10 Mobile Emulator.

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Page Navigation
  3. Handling Back
  4. Estimated time to complete this lab:  **45 to 60 minutes**.

Exercise 1: Page Navigation

* 1. Page navigation is an important part of any non-trivial app. When building a complex app, you may wish to use a framework that provides a navigation service to handle the details for you. You may also implement navigation without relying on a framework. In this exercise, you will create a secondary view, navigate directly to the view using Frame.Navigate, and learn how to pass data on navigation.

Task 1 – Create a blank Universal Windows app

We will begin by creating a project from the Blank App template.

1. In a new instance of Visual Studio 2015, choose File > New> Project to open the New Project dialog. Navigate to Installed > Templates > Visual C# > Windows > Universal and select the Blank App (Universal Windows) template.

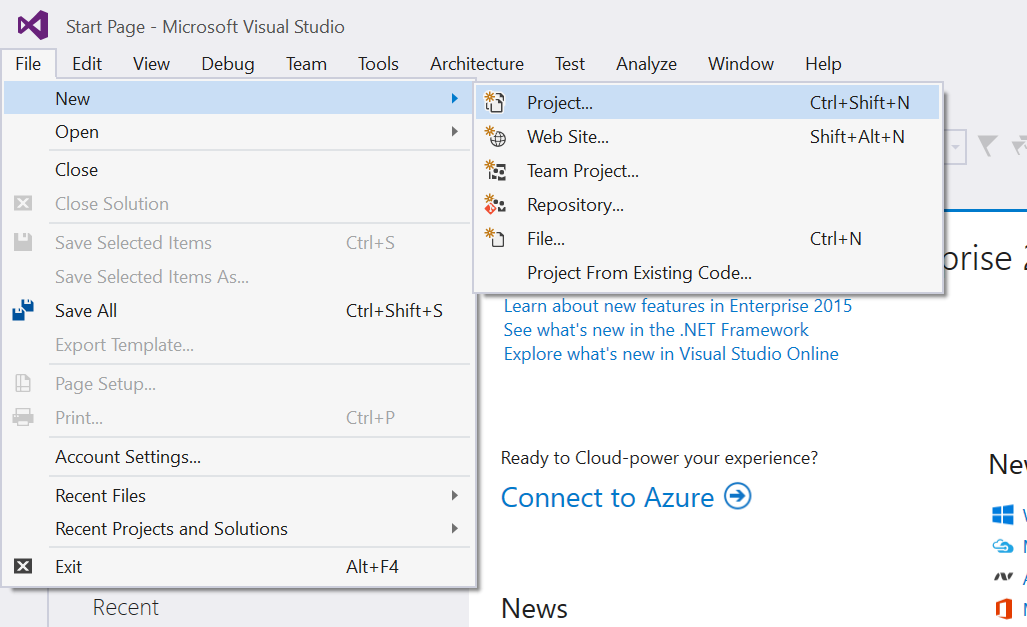


Figure 1

Open the New Project dialog in Visual Studio 2015.

1. Name your project **SimpleNavigation** and select the filesystem location where you will save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.
2. Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect **Add to source control** if you don't wish to version your work. Click **OK** to create the project.

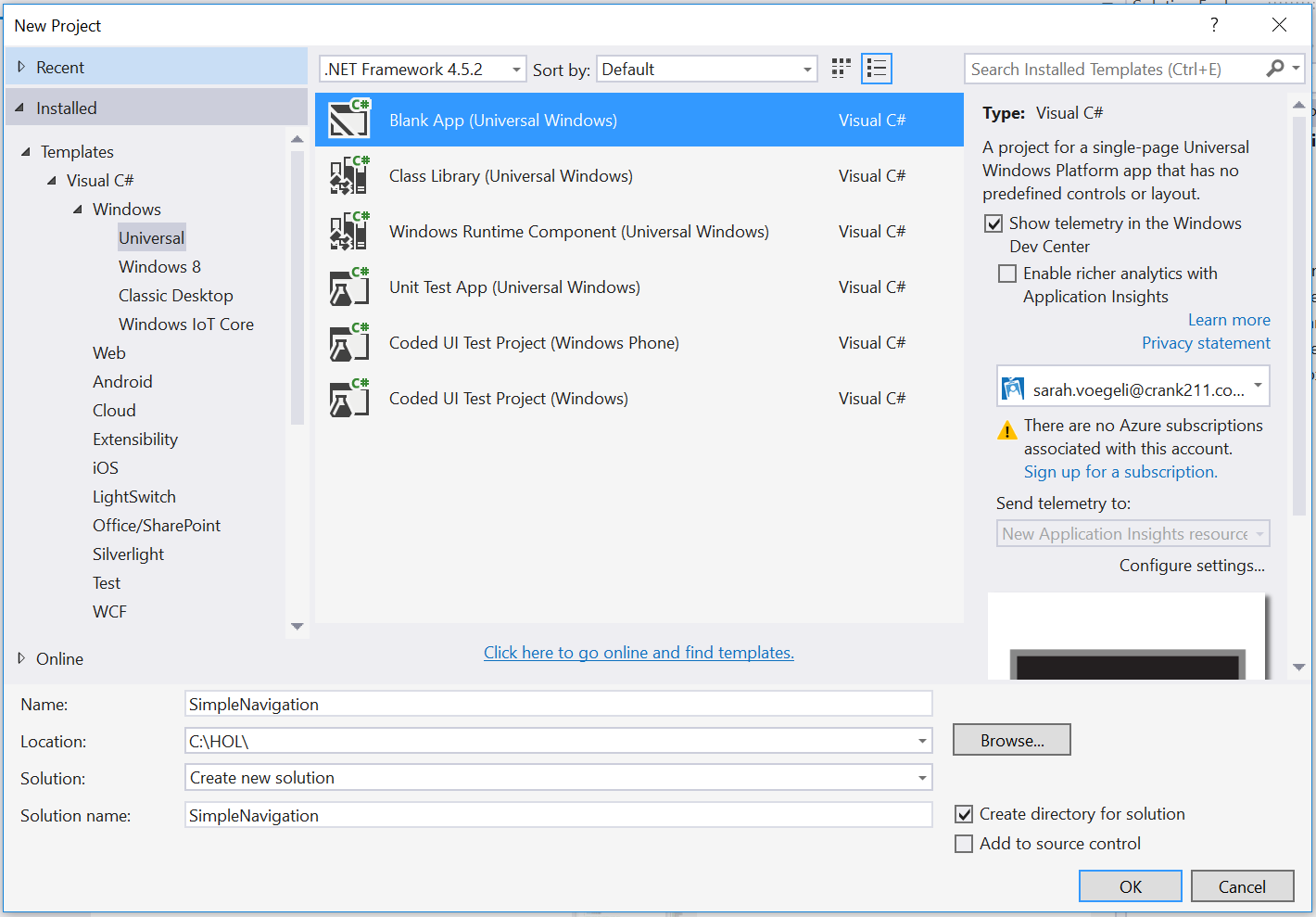


Figure 2

Create a new Blank App project in Visual Studio 2015.

1. Set your Solution Configuration to Debug and your Solution Platform to x86. Select Local Machine from the Debug Target dropdown next to the Start Debugging Button.



Figure 3

* + 1. Configure your app to run on the Local Machine.
  1. **Note:**  is the Start Debugging button.

1. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.

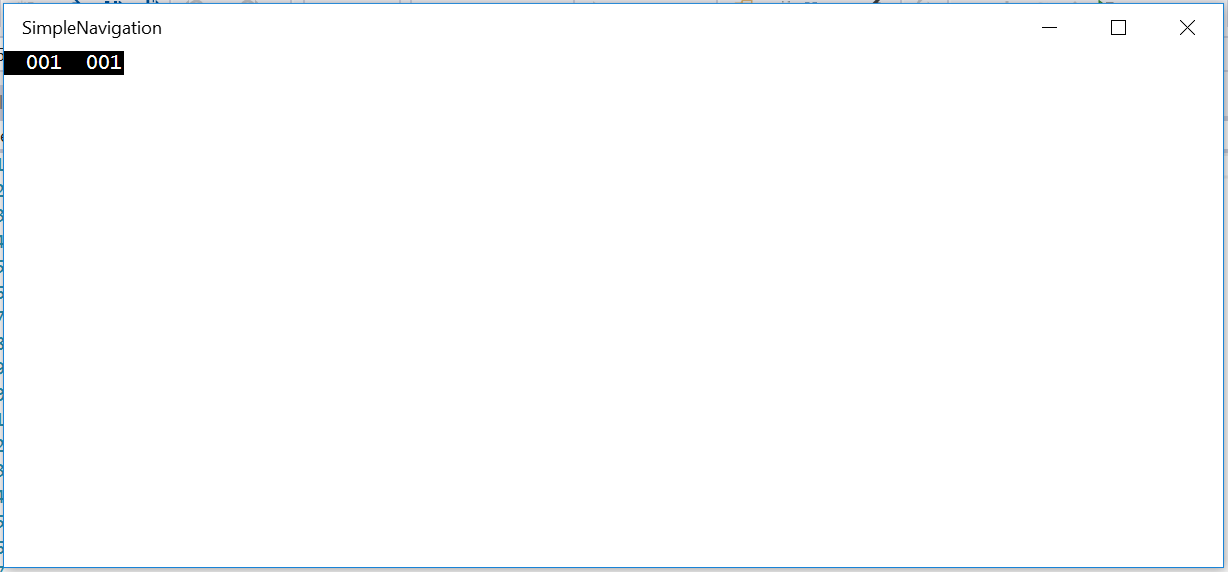


Figure 4

The blank universal app running in Desktop mode.

* 1. **Note:** The frame rate counter is a debug tool that helps to monitor the performance of your app. It is useful for apps that require intensive graphics processing but unnecessary for the simple apps you will be creating in the Hands-on Labs.
  2. In the Blank App template, the preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of the Hands-on Labs, we will turn it off.

1. Return to Visual Studio and stop debugging.
2. Open App.xaml.cs. Turn off the frame rate counter in the **#if DEBUG** preprocessor directive by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.
   * 1. C#
   1. #if DEBUG
   2. if (System.Diagnostics.Debugger.IsAttached)
   3. {
   4. this.DebugSettings.EnableFrameRateCounter = false;
   5. }
   6. #endif
3. Build and run your app again. This time, you will see a blank app window without the frame rate counter.

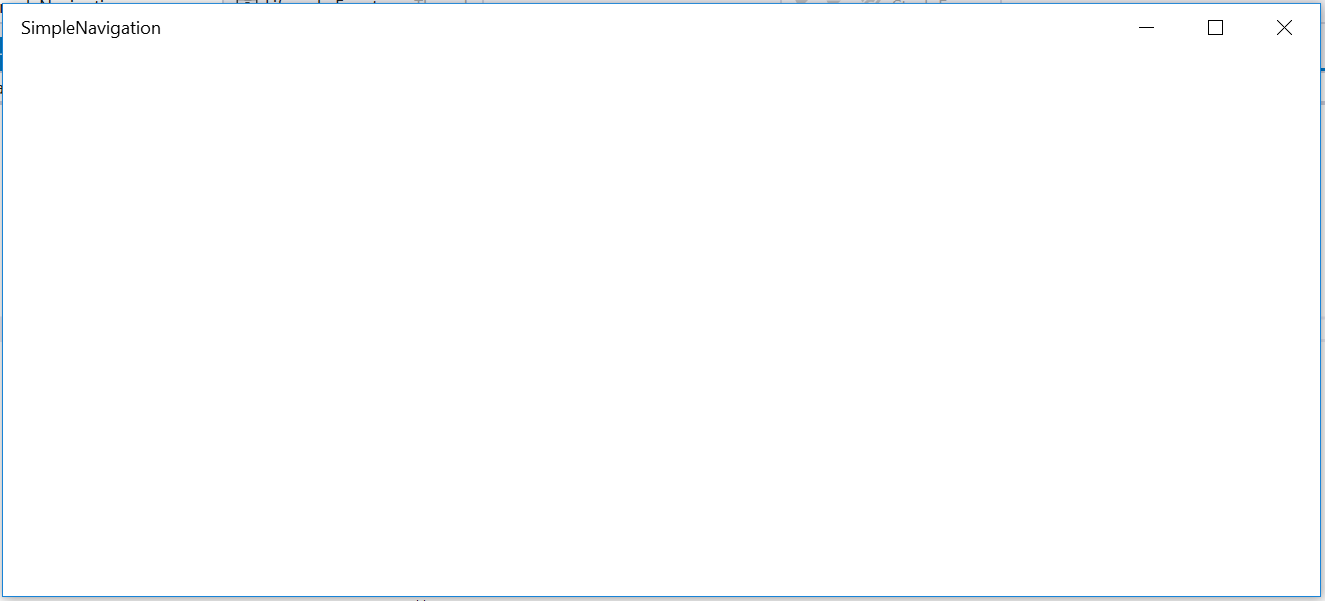


Figure 5

The Blank universal app after the frame rate counter has been disabled.

1. Stop debugging and return to Visual Studio.

Task 2 - Create a view

Before you can introduce navigation, you will need to create a secondary view to which you can navigate from the MainPage of the app.

1. In your SimpleNavigation solution, right-click on the project name in the Solution Explorer and choose **Add > New Item**.

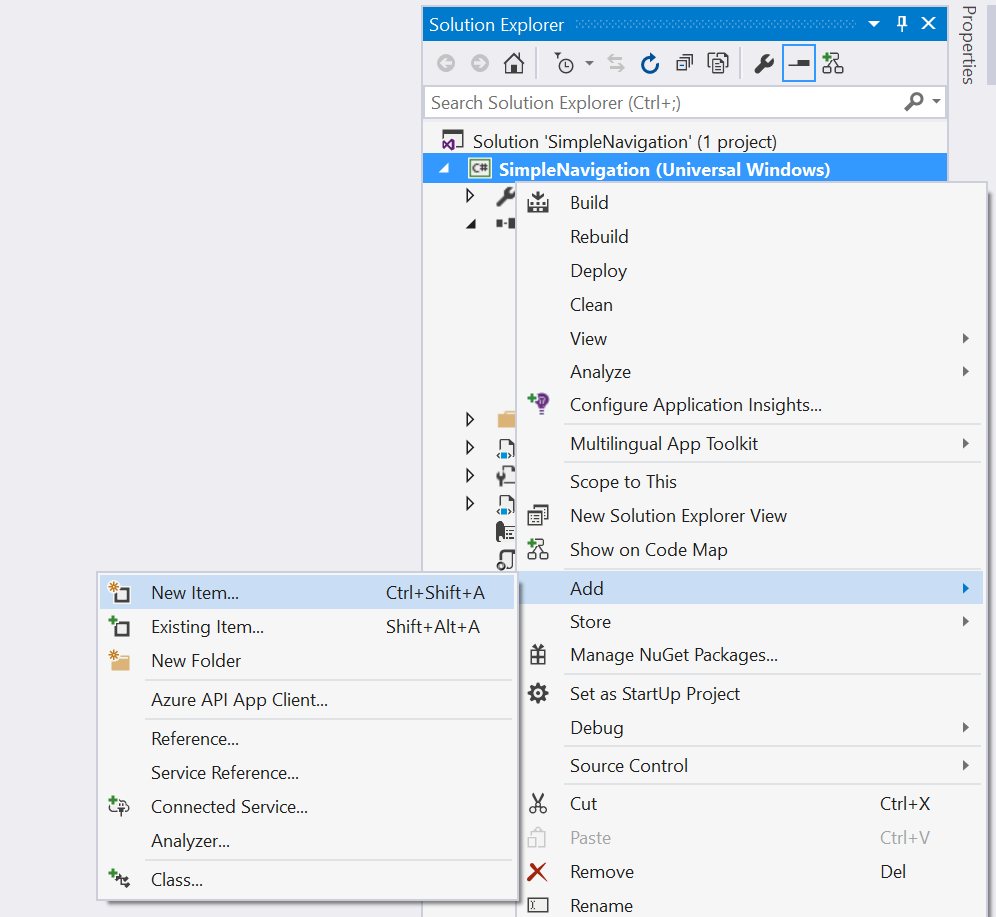


Figure 6

Add a new item in the Solution Explorer.

1. Select the **Blank Page** item type in the Visual C# item list. Name the item **Page2.xaml** and click **Add** to create it.

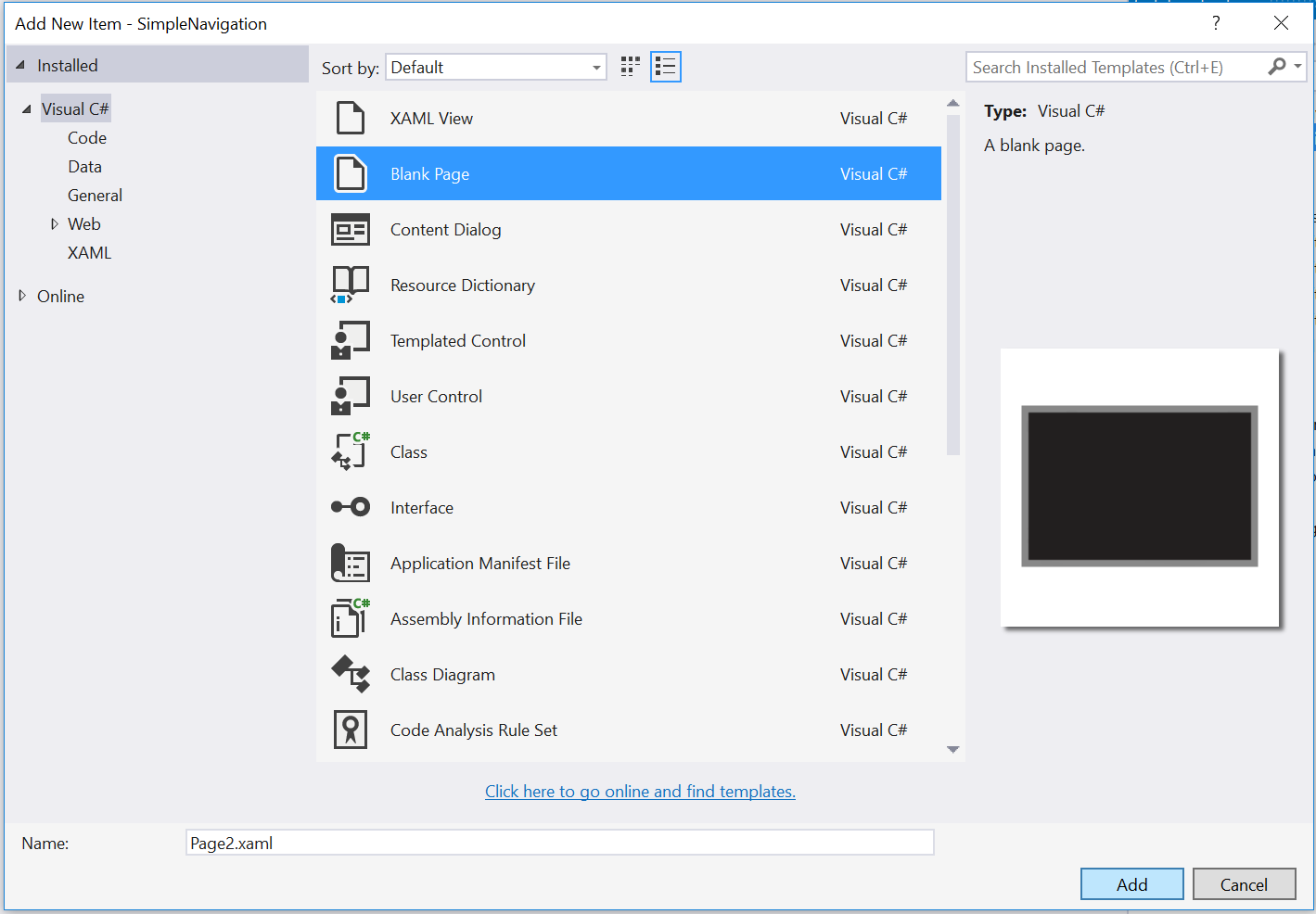


Figure 7

Add an item of type Blank Page to the SimpleNavigation project.

1. Open **Page2.xaml** and add a TextBlock to display the page title.
   * 1. XAML
   1. <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
   2. <TextBlock Text="Page 2" FontWeight="Light" FontSize="24" Margin="12" />
   3. </Grid>
   4. **Note:** ThemeResource is a XAML markup extensions that allows you to reference XAML styles defined elsewhere in a XAML Resource Dictionary. ThemeResource can dynamically use different resource dictionaries to reflect the user’s active system theme at runtime. A StaticResource reference differs in that it does not update at runtime.

For more information on using the ThemeResource extension, visit <https://msdn.microsoft.com/en-us/library/windows/apps/dn263118.aspx>

1. Open **MainPage.xaml** and add a page title there as well. Our pages are simple, so it will be helpful to know where we are once we introduce navigation.
   * 1. XAML
   1. <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
   2. <TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" Margin="12" />
   3. </Grid>

Task 3 – Introduce navigation

Now that you have two views in your project, you can navigate between them. In-app navigation takes place in a frame, which acts as a container for your pages. When your app starts up, the root frame is built in App.xaml.cs and attaches to the window. The frame is important in the sense that it manages navigation between the pages. In this task, you will create a button on the main page to trigger navigation to Page2.

1. Open **MainPage.xaml** and add a button that will trigger navigation to Page2. You may wish to enclose the page title and the button in a StackPanel to improve the layout of the page.
   * 1. XAML
   1. <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
   2. <StackPanel HorizontalAlignment="Left" Margin="12">
   3. <TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" Margin="12" />
   4. <Button Content="Go to Page 2" Margin="0,12,0,0" />
   5. </StackPanel>
   6. </Grid>
2. Add a click event to the button. You will create the **Button\_Click** method to handle navigation in the next step.
   * 1. XAML
   1. <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
   2. <StackPanel HorizontalAlignment="Left" Margin="12">
   3. <TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" Margin="12" />
   4. <Button Content="Go to Page 2" Click="Button\_Click" Margin="0,12,0,0" />
   5. </StackPanel>
   6. </Grid>
3. Now let's add the **Button\_Click** event handler to **MainPage.xaml.cs**. **Frame.Navigate** triggers the frame to load the content of the specified page. It accepts as a parameter the page to which you wish to navigate, specified as a type reference, and an optional second parameter to pass to that page. We will pass a parameter later on in this exercise, but for now we will navigate without it.
   * 1. C#
   1. public sealed partial class MainPage : Page
   2. {
   3. public MainPage()
   4. {
   5. this.InitializeComponent();
   6. }
   7. private void Button\_Click(object sender, RoutedEventArgs e)
   8. {
   9. Frame.Navigate(typeof(Page2));
   10. }
   11. }
4. Build and run your app on the Local Machine. When you click the **Go to Page 2** button on the main page, your frame will navigate to Page 2. We haven't enabled back behavior, so you won't be able to to return to MainPage yet. You will learn how to handle back behavior in the next exercise.

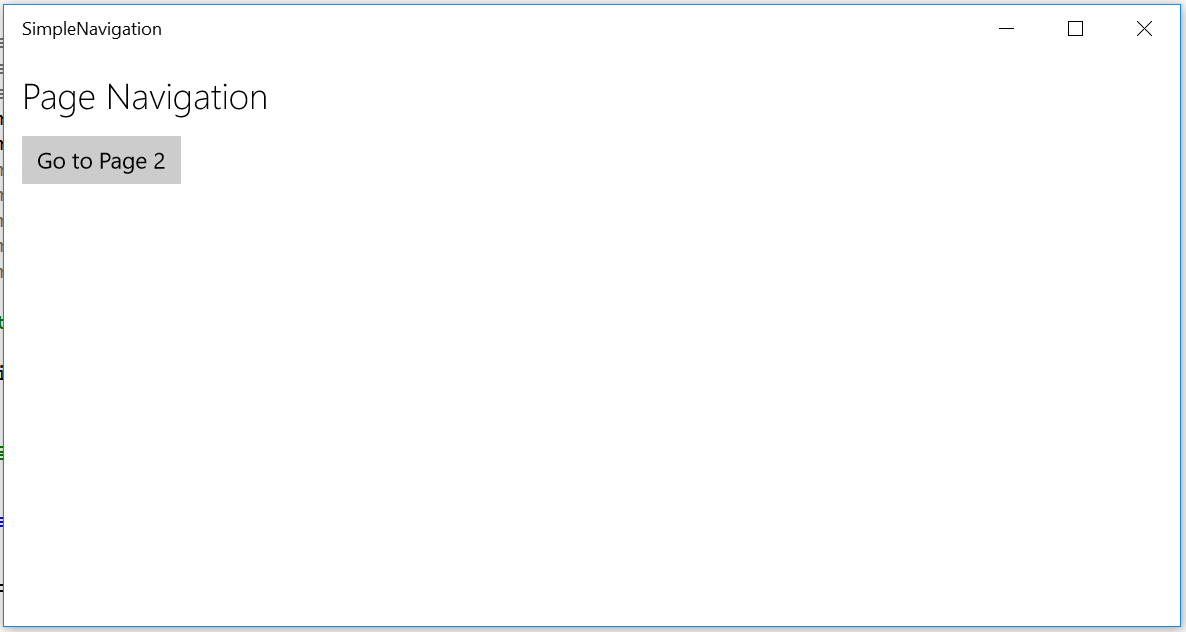




Figure 8

The **Go to Page 2** button triggers navigation to another page in the app.

1. Stop debugging and return to Visual Studio.

Task 4 – Pass a parameter to Page 2

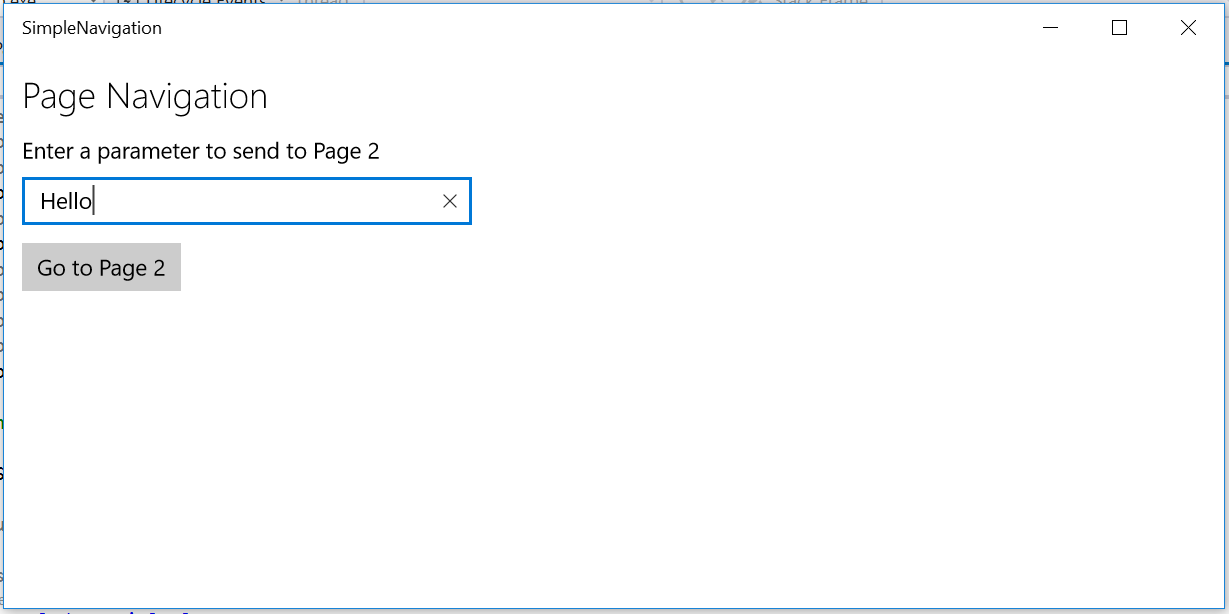
* 1. You have successfully navigated between pages in your app. It is often useful to pass information to the new page on navigation. In this task, you will pass a parameter from the main page to Page2.

1. Add a **TextBox** to **MainPage.xaml**. This TextBox will accept user input that you will pass to Page2 for display later on.
   * 1. XAML
   1. <StackPanel HorizontalAlignment="Left" Margin="12">
   2. <TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" Margin="12" />
   3. <TextBox x:Name="Message" Header="Enter a parameter to send to Page 2" Width="300" Margin="0,12,0,0" />
   4. <Button Content="Go to Page 2" Click="Button\_Click" Margin="0,12,0,0" />
   5. </StackPanel>
2. In your **Button\_Click** handler in the code-behind, add the optional second parameter to **Frame.Navigate** to pass the TextBox text to Page2.
   * 1. C#
   1. private void Button\_Click(object sender, RoutedEventArgs e)
   2. {
   3. Frame.Navigate(typeof(Page2), Message.Text);
   4. }
   5. **Note:** The optional parameter you pass into **Frame.Navigate** does not have to be a string (it is an object in the API), but it must be serializable. An object that is serializable can be converted to a stream of bytes for storage in order to save its state and recreate that state later on. The frame keeps track of the app history and navigation parameters, which it needs to resume from suspend, for instance, and it does so by serializing the parameters. If you wish to pass a parameter that is not serializable, you may write code to serialize it yourself for advanced use cases.

Task 5 – Display the message passed to Page2

Your message will now be passed as a parameter when you navigate to Page2. However, we haven't done anything to handle the message in Page2, so it won't yet display. Let's add a pop-up dialog to display the message.

1. Open **Page2.xaml.cs**. Create an override for the **OnNavigatedTo()** method.
   * 1. C#
   1. public Page2()
   2. {
   3. this.InitializeComponent();
   4. }
   5. protected override void OnNavigatedTo(NavigationEventArgs e)
   6. {
   7. base.OnNavigatedTo(e);
   8. }
   9. **Note:** The page constructor may not be called every time you navigate to a page if the page has already been loaded. The **OnNavigatedTo()** method is called every time you navigate to a page, however, so we can use it to consistently trigger the display of the incoming message.
2. Await a message dialog in the **OnNavigatedTo()** override and pass in the incoming parameter. You will need to make the override an async method to handle the await.
   * 1. C#
   1. protected async override void OnNavigatedTo(NavigationEventArgs e)
   2. {
   3. await new MessageDialog("You sent: " + e.Parameter).ShowAsync();
   4. base.OnNavigatedTo(e);
   5. }
   6. **Note:** Asynchronous processes allow the application to continue execution without having to wait for operations that may potentially block the UI thread. The UI will continue to respond to the user while the async process runs in the background.
   7. An asynchronous method includes the async modifier, and its name typically ends in Async by convention. It usually contains at least one await, which suspends the method and returns control to the method’s caller along with a task value. A task typically runs asynchronously on a thread pool thread instead of the main application thread, and its state can be ascertained through the Status property. When an async method is complete, its associated task is marked as complete, and any return values can be accessed through the task.
   8. For more information and examples of asynchronous programming patterns, visit <https://msdn.microsoft.com/en-us/library/hh191443.aspx>
3. Add the **Windows.UI.Popups** namespace to the **Page2** code-behind.
   * 1. C#
   1. using Windows.UI.Popups;
4. Build and run your app. Type a message into the TextBox on the MainPage and navigate to Page2. You will see a pop-up appear on Page2 with your message.



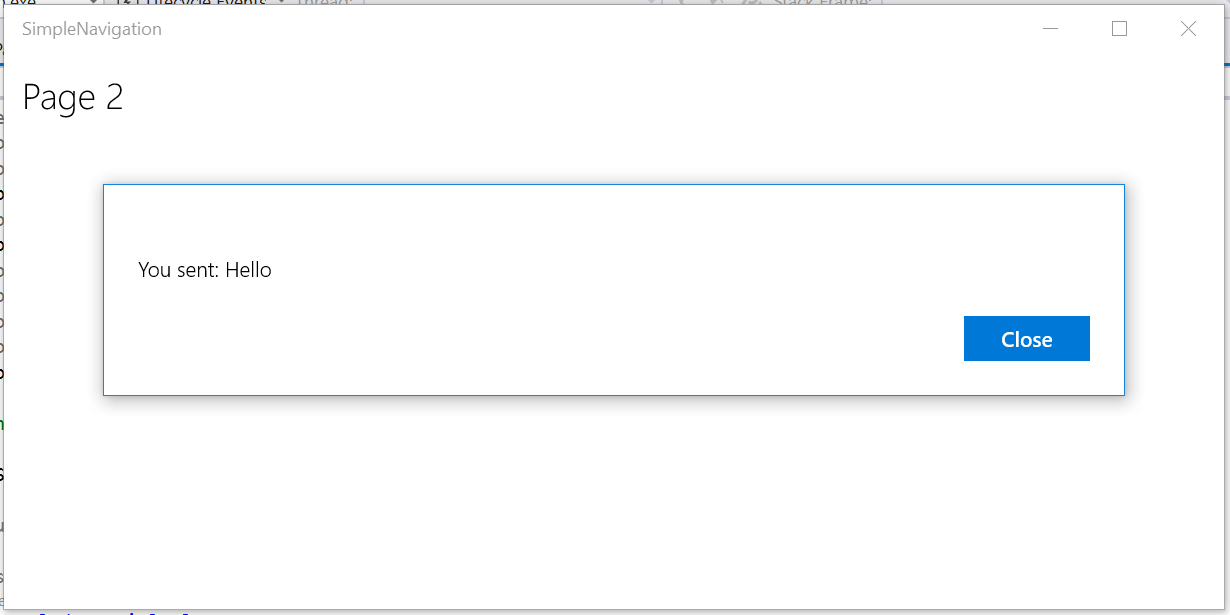


Figure 9

The user-generated message is passed to Page2 as a parameter.

1. Stop debugging and return to Visual Studio.

Exercise 2: Handling Back

* 1. UWP apps run on a range of devices which differ in how they handle back navigation. Whereas Windows 10 Mobile devices typically provide a hardware back button, tablet and desktop devices typically do not. There are several ways to handle back navigation within your app. In this exercise, you will use the shell-drawn back button to return to the main page from a secondary page.

Task 1 – Enable the shell-drawn back button

* 1. The **SystemNavigationManager** provides options to enable the **AppViewBackbutton** for Desktop and Tablet modes. This shell-drawn back button displays in the task bar when in tablet mode and in the titlebar when in windowed Desktop mode. The visibility of the shell-drawn back button is ignored on Mobile, because a hardware back button is expected. In this task, you will enable the system-provided back button when the ability to go back is available within the app.

1. Open **App.xaml.cs**. Add a **rootFrame.Navigated** event handler at the end of the **OnLaunched** override. This event handler will fire every time there is navigation on the root frame and will display the app view back button if the app backstack isn't empty.
   * 1. C#
   1. // Ensure the current window is active
   2. Window.Current.Activate();
   3. rootFrame.Navigated += (s, a) =>
   4. {
   5. SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility = rootFrame.CanGoBack ?
   6. AppViewBackButtonVisibility.Visible :
   7. AppViewBackButtonVisibility.Collapsed;
   8. };
   9. **Note:** We wish to only show the shell back button when there is something in the back stack. If the user clicks the back button and there's no code to handle back, the shell back button will navigate to the previous app by default.
2. Build and run your app and navigate to Page2. The back button will appear in the title bar in Desktop mode.

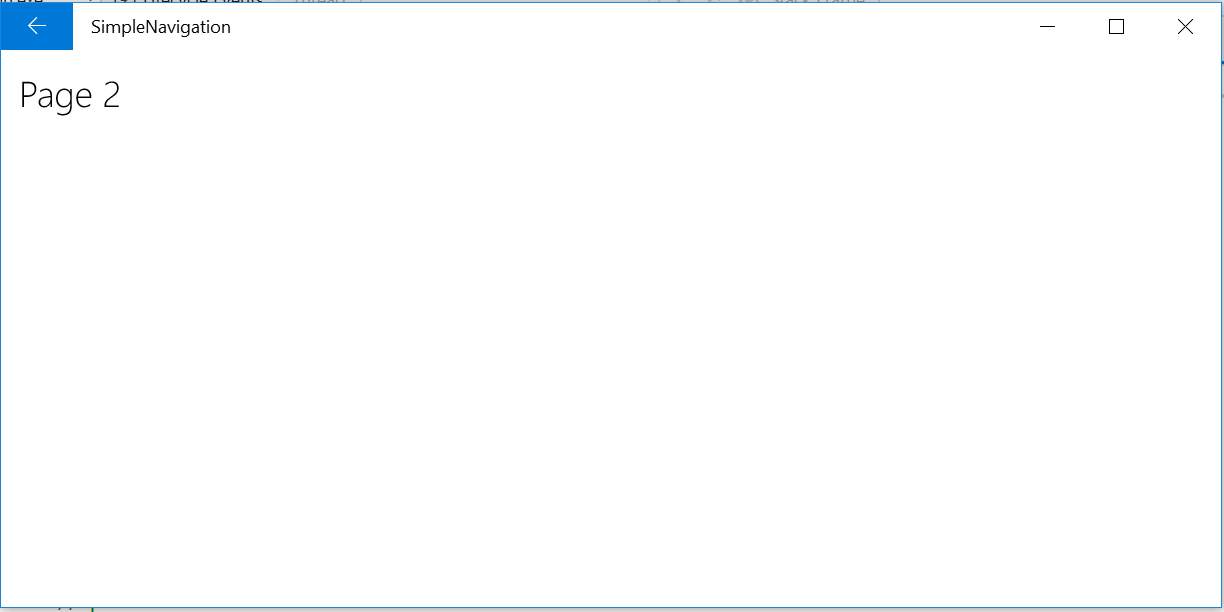


Figure 10

The AppViewBackButton is visible in the title bar in Desktop mode.

* 1. **Note:** The color of the shell-drawn back button is defined by the sytem theme. If you wish to control the placement or style of the back button, you can still implement your own back button control within the app.

1. With your app still running, use the notifications panel to switch into Tablet mode. The back button will appear in the task bar.



Figure 11

The AppViewBackButton is visible in the shell in Tablet mode.

* 1. **Note:** In Split Screen, there is a back stack available for each side of the screen.

1. Stop debugging and return to Visual Studio. Run the app again, this time in the Mobile emulator. When you navigate to Page2, the shell back button will not appear, because it is ignored on Mobile.

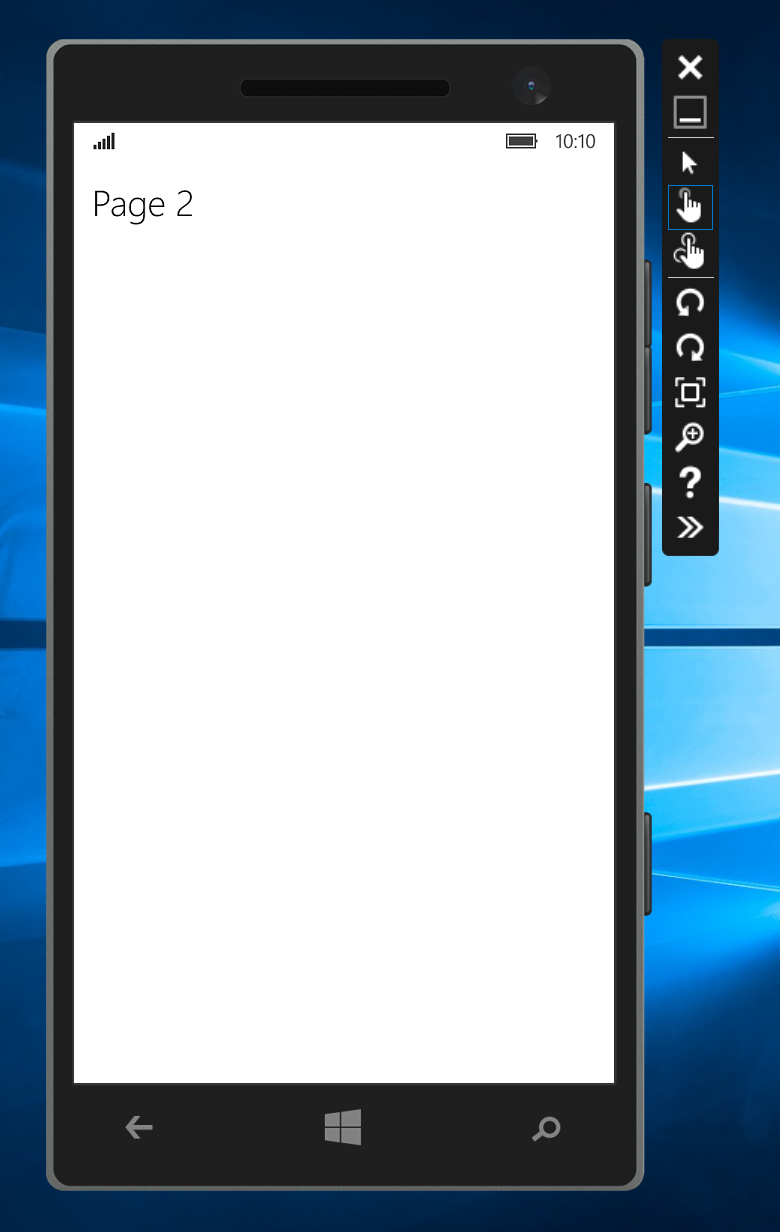


Figure 12

The AppViewBackButton is hidden on Mobile.

1. Stop debugging and return to Visual Studio. You may have noticed that the back button doesn't navigate back in the app. Although you enabled its visibility, it is not yet hooked up to handle back navigation. In the next task, you will enable back behavior.

Task 2 - Define the standard back-requested pattern

Your app view back button displays only when the root frame of the app can go back. Let's define the standard back requested pattern to handle back requests.

1. In **App.xaml.cs**, create and subscribe to an **App\_BackRequested** event.
   1. **Note:** The subscription to App\_BackRequested must take place after the view has been created, because you will need access to GetForCurrentView().BackRequested to proceed.
      1. C#
   2. SystemNavigationManager.GetForCurrentView().BackRequested += App\_BackRequested;
   4. rootFrame.Navigated += (s, a) =>
   5. {
   6. SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility = rootFrame.CanGoBack ?
   7. AppViewBackButtonVisibility.Visible :
   8. AppViewBackButtonVisibility.Collapsed;
   9. };
   10. }
   12. private void App\_BackRequested(object sender, BackRequestedEventArgs e)
   13. {
   14. }
2. In your **App\_BackRequested** event handler, check to see if the BackRequested event has already been handled. If not, set the default behavior to navigate back within the frame. Make sure to set **e.Handled** to **true** when done.
   * 1. C#
   1. private void App\_BackRequested(object sender, BackRequestedEventArgs e)
   2. {
   3. // Check that no one has already handled this
   4. if (!e.Handled)
   5. {
   6. // Default is to navigate back within the Frame
   7. Frame frame = Window.Current.Content as Frame;
   8. if (frame.CanGoBack)
   9. {
   10. frame.GoBack();
   11. // Signal handled so the system doesn't navigate back through the app stack
   12. e.Handled = true;
   13. }
   14. }
   15. }
3. Build and run your app. Navigate to **Page2**, then use the back button to return to the main page.
   1. **Note:** In Windows 10, the back button (if shown or available as a hardware back button) will no longer exit the app when the backstack is empty.
4. Stop debugging and return to Visual Studio.

Task 3 – Virtual back button

If you would like to have more control over the styling and behavior of the back button, you can create your own button to handle back navigation.

1. Comment out the visibility of the shell back button **in App.xaml.cs**.
   * 1. C#
   1. //rootFrame.Navigated += (s, a) =>
   2. //{
   3. // SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility = rootFrame.CanGoBack ?
   4. // AppViewBackButtonVisibility.Visible :
   5. // AppViewBackButtonVisibility.Collapsed;
   6. //};
2. Add a button to **Page2.xaml** and assign its style to the StaticResource **NavigationBackButtonNormalStyle**. Enclose the button and the page title in a horizontal StackPanel for alignment.
   * 1. XAML
   1. <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
   2. <StackPanel Orientation="Horizontal">
   3. <Button Style="{StaticResource NavigationBackButtonNormalStyle}"
   4. VerticalAlignment="Top" />
   5. <TextBlock Text="Page 2" FontWeight="Light" FontSize="24" VerticalAlignment="Top" />
   6. </StackPanel>

</Grid>

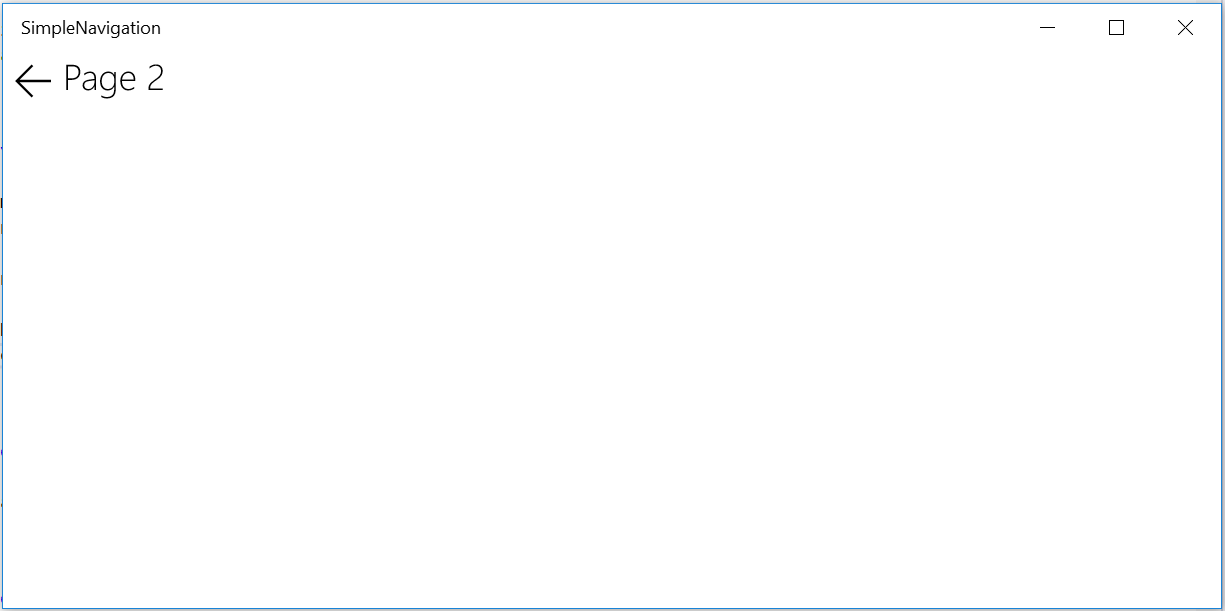
1. Add a click event to the button and bind it to a handler called GoBack(). You will create the GoBack() handler in the next step.
   * 1. XAML
   1. <Button Style="{StaticResource NavigationBackButtonNormalStyle}"
   2. Click="{x:Bind GoBack}" VerticalAlignment="Top" />
2. In the Page2 code behind, add the GoBack() function to handle the back event.
   * 1. C#
   1. private void GoBack()
   2. {
   3. if (Frame.CanGoBack)
   4. Frame.GoBack();
   5. }
3. Build and run your app. When you navigate to Page2, you will see your custom back button instead of the shell back button. The custom button will return you to the main page.
   * 1. 

Figure 13

The custom back button on Page2.

1. Stop debugging and return to Visual Studio.
2. Let’s make the custom back button visible only when there is something in the back stack. Add a private **Visibility** field to the Page2 code behind.
   * 1. C#
   1. public sealed partial class Page2: Page
   2. {
   3. private Visibility \_canGoBack;
3. In the Page2 **OnNavigatedTo** override, check if the frame can go back and set the Visibility field accordingly.
   * 1. C#
   1. protected async override void OnNavigatedTo(NavigationEventArgs e)
   2. {
   3. if (Frame.CanGoBack)
   4. \_canGoBack = Visiblity.Visible;
   5. else
   6. \_canGoBack = Visibility.Collapsed;

await new MessageDialog("You sent: " + e.Parameter).ShowAsync();

1. }
2. Bind the visibility of your back button to the **\_canGoBack** field in **Page2.xaml**.
   * 1. XAML
   1. <Button Style="{StaticResource NavigationBackButtonNormalStyle}"
   2. Click="{x:Bind GoBack}" Visibility="{x:Bind\_canGoBack}"

VerticalAlignment="Top" />

1. In **App.xaml.cs**, set the start up page for your app to **Page2** instead of **MainPage**.
   * 1. C#

if (rootFrame.Content == null)

{

// When the navigation stack isn’t restored navigate to the first page,

// configuring the new page by passing required information as a

// navigation parameter

rootFrame.Navigate(typeof(Page2)), e.Arguments);

}

1. Build and run your app. The custom back button will not appear on Page 2, because there is nothing in the back stack.

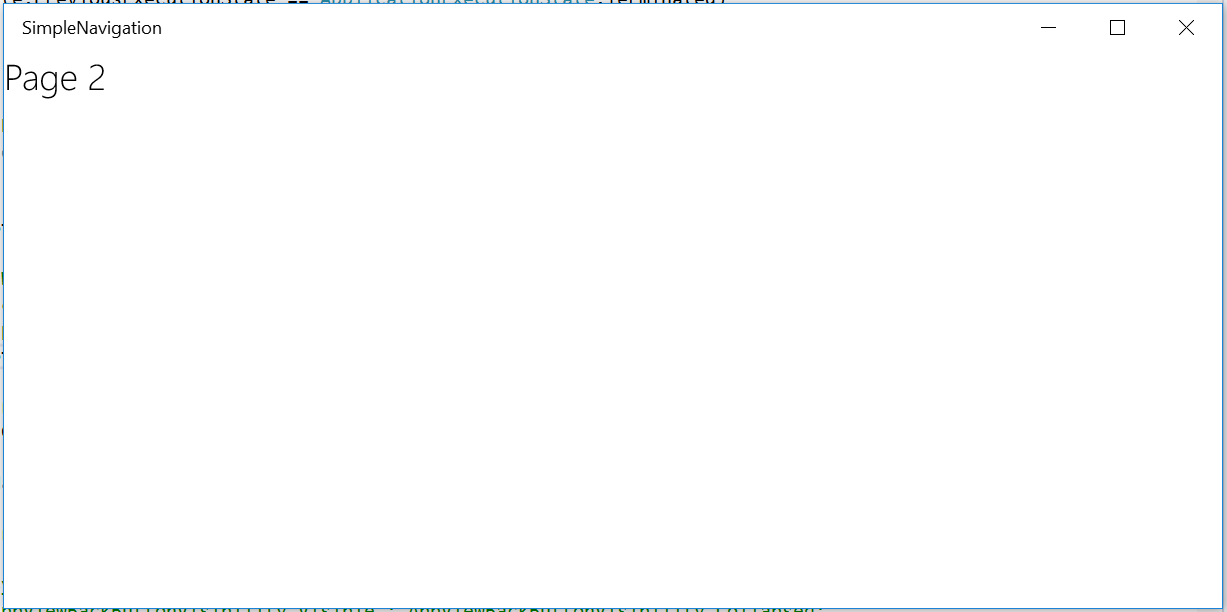


Figure 14

The custom back button does not appear when the back stack is empty.

1. Stop debugging and return to **App.xaml.cs**. Change the start up page back to **MainPage**.
   * 1. C#
2. if (rootFrame.Content == null)
3. {
4. // When the navigation stack isn’t restored navigate to the first page,
5. // configuring the new page by passing required information as a
6. // navigation parameter
7. rootFrame.Navigate(typeof(MainPage)), e.Arguments);
8. }
9. Build and run your app. Navigate to Page2. The custom back button will appear again now that the back stack is populated.
10. Stop debugging and return to Visual Studio.
11. Task 4 – Handling forward navigation

Although there is currently no shell-drawn forward button, you may handle your forward navigation with a custom button that displays in the app content area. The Frame has a **CanGoForward** property in addition to **CanGoBack**, so we will check to see if the forward button makes sense contextually before displaying it.

1. Add a button to **MainPage.xaml** and give it the same style as the back button you added to Page2. There isn’t a StaticResource for the forward button, so we will transform the back button icon instead. We have added column definitions to the Grid to facilitate the layout.
   * 1. XAML
   1. <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
   2. <Grid.ColumnDefinitions>
   3. <ColumnDefinition />
   4. <ColumnDefinition Width="Auto" />
   5. </Grid.ColumnDefinitions>
   6. <TextBlock Grid.Column="0" Text="Page Navigation" FontWeight="Light" FontSize="24" Margin="12,0,0,0" />
   8. <Button Grid.Column="1" Style="{StaticResource

NavigationBackButtonNormalStyle}" VerticalAlignment="Top" HorizontalAlignment="Right">

* 1. </Button>
  2. <StackPanel Grid.Column="0" Margin="12, 40" HorizontalAlignment="Left">
  3. <TextBox x:Name="Message" Header="Enter a parameter to send to Page 2" Width="300" Margin="0,12,0,0" />
  4. <Button Content="Go to Page 2" Click="Button\_Click" Margin="0,12,0,0"/>
  5. </StackPanel>
  6. </Grid>

1. Use a **RenderTransform** to rotate the button 180 degrees.
   * 1. XAML

<Button Grid.Column="1" Style="{StaticResource NavigationBackButtonNormalStyle}" VerticalAlignment="Top" HorizontalAlignment="Right" RenderTransformOrigin="0.5,0.5" >

* 1. <Button.RenderTransform>
  2. <CompositeTransform Rotation="180"/>
  3. </Button.RenderTransform>
  4. </Button>

1. Add a click event to your forward button. You will create the **GoForward()** event handler in the next step.
   * 1. C#
   1. <Button Grid.Column="1" Style="{StaticResource
   2. NavigationBackButtonNormalStyle}" Click="{x:Bind GoForward}" VerticalAlignment="Top" HorizontalAlignment="Right" RenderTransformOrigin="0.5,0.5" >
2. In **MainPage.xaml.cs**, add the **GoForward()** event handler. You may notice that this handler is identical to the **GoBack()** handler but uses **Frame.CanGoForward** and **Frame.GoForward** instead of **Frame.CanGoBack** and **Frame.GoBack**.
   * 1. C#
   1. private void GoForward()
   2. {
   3. if (Frame.CanGoForward)
   4. Frame.GoForward();

}

1. As with the back button, let’s make it so the forward button only displays if there is something in the forward stack. Add the private field **\_canGoForward** to the **MainPage** class.
   * 1. C#
   1. public sealed partial class MainPage : Page
   2. {
   3. private Visibility \_canGoForward;
2. Create an **OnNavigatedTo()** override and use it to determine the visibility of the forward button.
   * 1. C#
   1. protected override void OnNavigatedTo(NavigationEventArgs e)
   2. {
   3. if (Frame.CanGoForward)
   4. \_canGoForward = Visibility.Visible;
   5. else
   6. \_canGoForward = Visibility.Collapsed;
   8. base.OnNavigatedTo(e);
   9. }
3. Bind the visibility of the forward button to **\_canGoForward** in **MainPage.xaml**.
   * 1. XAML
   1. <Button Grid.Column="1" Style="{StaticResource NavigationBackButtonNormalStyle}" Click="{x:Bind GoForward}" Visibility="{x:Bind \_canGoForward}" VerticalAlignment="Top" HorizontalAlignment="Right" RenderTransformOrigin="0.5,0.5" >
   2. <Button.RenderTransform>
   3. <CompositeTransform Rotation="180"/>
   4. </Button.RenderTransform>
   5. </Button>
4. Build and run your app. Navigate to Page2 and then back to the main page. The forward button will show up when you return to the main page. Use the forward button to navigate forward to Page2.

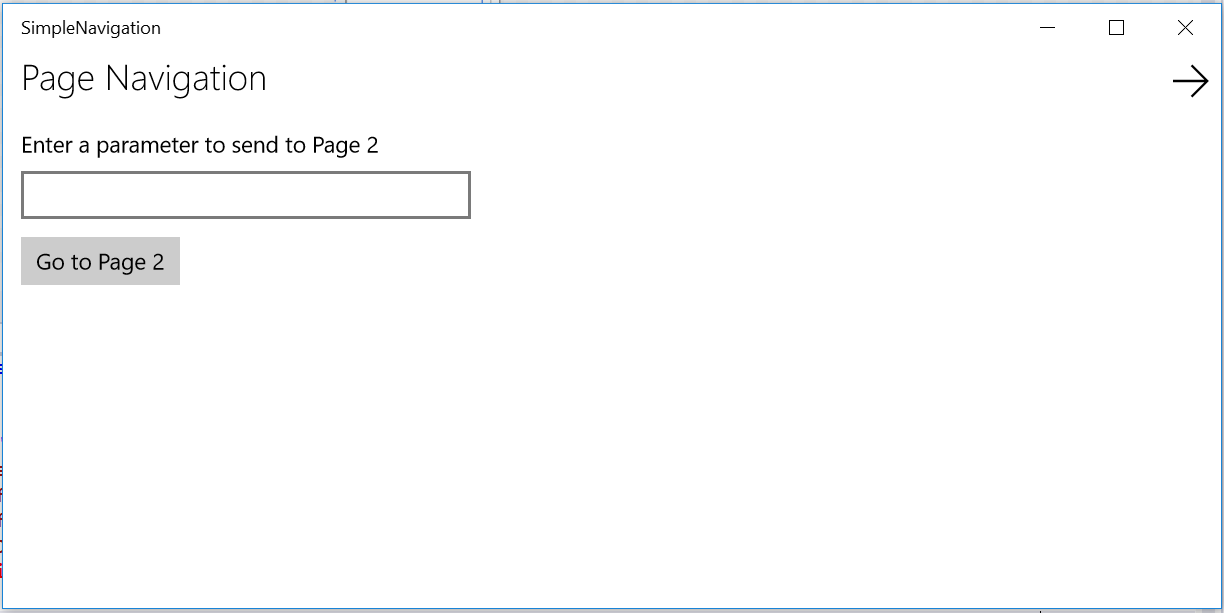


Figure 15

The forward button becomes available on the MainPage view.

* 1. **Note:** You may notice when navigating forward to Page2 that your original parameter message appears in the message dialog. The frame stores this information and restores it when you navigate forward.
  2. That same parameter does not show up when you navigate back to the MainPage, however. Try setting **NavigationCacheMode = NavigationCacheMode.Enabled** in your MainPage constructor after **this.InitializeComponent();**. Rerun the project and navigate to Page2 and back. This time, you will see the cached value appear in the MainPage TextBox.
  3. Although caching is handy, it has limited memory. If you wish to ensure cached information appears when needed, it may be best to handle it programmatically.

1. Stop debugging and return to Visual Studio.

Summary

* 1. In this lab, you learned about frames, page navigation, and back and forward behavior within a UWP app. In the next lab, you will expand on your knowledge of UWP to create an adaptive UI that is tailored across a broad range of screen resolutions.