# Chapter 5: Navigation in Xamarin.Forms

## Section 5.1: NavigationPage flow with XAML

App.xaml.cs file (App.xaml file is default, so skipped)

```
using Xamrin.Forms

namespace NavigationApp
{
    public partial class App : Application
    {
        public static INavigation GlobalNavigation { get; private set; }

        public App()
        {
            InitializeComponent();
            var rootPage = new NavigationPage(new FirstPage());

            GlobalNavigation = rootPage.Navigation;

            MainPage = rootPage;
        }
    }
}
```

FirstPage.xaml file

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage
   xmlns="http://xamarin.com/schemas/2014/forms"
   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
   x:Class="NavigationApp.FirstPage"
   Title="First page">
    <ContentPage.Content>
        <StackLayout>
            <Label
               Text="This is the first page" />
            <Button
               Text="Click to navigate to a new page"
               Clicked="GoToSecondPageButtonClicked"/>
            <Button
               Text="Click to open the new page as modal"
               Clicked="OpenGlobalModalPageButtonClicked"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

In some cases you need to open the new page not in the current navigation but in the global one. For example, if your current page contains bottom menu, it will be visible when you push the new page in the current navigation. If you need the page to be opened over the whole visible content hiding the bottom menu and other current page's content, you need to push the new page as a modal into the global navigation. See `App.GlobalNavigation` property and the example below.

FirstPage.xaml.cs file

```
using System;
using Xamarin.Forms;
```

```
namespace NavigationApp
{
    public partial class FirstPage : ContentPage
    {
        public FirstPage()
        {
            InitializeComponent();
        }

        async void GoToSecondPageButtonClicked(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new SecondPage(), true);
        }

        async void OpenGlobalModalPageButtonClicked(object sender, EventArgs e)
        {
            await App.GlobalNavigation.PushModalAsync(new SecondPage(), true);
        }
    }
}
```

SecondPage.xaml file (xaml.cs file is default, so skipped)

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage
   xmlns="http://xamarin.com/schemas/2014/forms"
   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
   x:Class="NavigationApp.SecondPage"
   Title="Second page">
    <ContentPage.Content>
        <Label
           Text="This is the second page" />
    </ContentPage.Content>
</ContentPage>
```

## Section 5.2: NavigationPage flow

```
using System;
using Xamarin.Forms;

namespace NavigationApp
{
    public class App : Application
    {
        public App()
        {
            MainPage = new NavigationPage(new FirstPage());
        }
    }

    public class FirstPage : ContentPage
    {
        Label FirstPageLabel { get; set; } = new Label();

        Button FirstPageButton { get; set; } = new Button();

        public FirstPage()
        {
            Title = "First page";
```

```
            FirstPageLabel.Text = "This is the first page";
            FirstPageButton.Text = "Navigate to the second page";
            FirstPageButton.Clicked += OnFirstPageButtonClicked;

            var content = new StackLayout();
            content.Children.Add(FirstPageLabel);
            content.Children.Add(FirstPageButton);

            Content = content;
        }

        async void OnFirstPageButtonClicked(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new SecondPage(), true);
        }
    }

    public class SecondPage : ContentPage
    {
        Label SecondPageLabel { get; set; } = new Label();

        public SecondPage()
        {
            Title = "Second page";

            SecondPageLabel.Text = "This is the second page";

            Content = SecondPageLabel;
        }
    }
}
```

## Section 5.3: Master Detail Navigation

The code below shows how to perform asynchronous navigation when the app is in a MasterDetailPage context.

```
public async Task NavigateMasterDetail(Page page)
{
        if (page == null)
            {
                return;
            }

        var masterDetail = App.Current.MainPage as MasterDetailPage;

        if (masterDetail == null || masterDetail.Detail == null)
            return;

        var navigationPage = masterDetail.Detail as NavigationPage;
        if (navigationPage == null)
        {
            masterDetail.Detail = new NavigationPage(page);
            masterDetail.IsPresented = false;
            return;
        }

        await navigationPage.Navigation.PushAsync(page);


navigationPage.Navigation.RemovePage(navigationPage.Navigation.NavigationStack[navigationPage.Navig
ation.NavigationStack.Count - 2]);
```

```
        masterDetail.IsPresented = false;
    }
```

## Section 5.4: Using INavigation from view model

First step is create navigation interface which we will use on view model:

```
public interface IViewNavigationService
{
    void Initialize(INavigation navigation, SuperMapper navigationMapper);
    Task NavigateToAsync(object navigationSource, object parameter = null);
    Task GoBackAsync();
}
```

In `Initialize` method I use my custom mapper where I keep collection of pages types with associated keys.

```
public class SuperMapper
{
    private readonly ConcurrentDictionary<Type, object> _typeToAssociateDictionary = new
ConcurrentDictionary<Type, object>();

    private readonly ConcurrentDictionary<object, Type> _associateToType = new
ConcurrentDictionary<object, Type>();

    public void AddMapping(Type type, object associatedSource)
    {
        _typeToAssociateDictionary.TryAdd(type, associatedSource);
        _associateToType.TryAdd(associatedSource, type);
    }

    public Type GetTypeSource(object associatedSource)
    {
        Type typeSource;
        _associateToType.TryGetValue(associatedSource, out typeSource);

        return typeSource;
    }

    public object GetAssociatedSource(Type typeSource)
    {
        object associatedSource;
        _typeToAssociateDictionary.TryGetValue(typeSource, out associatedSource);

        return associatedSource;
    }
}
```

Enum with pages:

```
public enum NavigationPageSource
{
    Page1,
    Page2
}
```

`App.cs` file:

```
public class App : Application
{
```

```
    public App()
    {
        var startPage = new Page1();
        InitializeNavigation(startPage);
        MainPage = new NavigationPage(startPage);
    }

    #region Sample of navigation initialization
    private void InitializeNavigation(Page startPage)
    {
        var mapper = new SuperMapper();
        mapper.AddMapping(typeof(Page1), NavigationPageSource.Page1);
        mapper.AddMapping(typeof(Page2), NavigationPageSource.Page2);

        var navigationService = DependencyService.Get<IViewNavigationService>();
        navigationService.Initialize(startPage.Navigation, mapper);
    }
    #endregion
}
```

In mapper I associated type of some page with enum value.

`IViewNavigationService` implementation:

```
[assembly: Dependency(typeof(ViewNavigationService))]
namespace SuperForms.Core.ViewNavigation
{
    public class ViewNavigationService : IViewNavigationService
    {
        private INavigation _navigation;
        private SuperMapper _navigationMapper;

        public void Initialize(INavigation navigation, SuperMapper navigationMapper)
        {
            _navigation = navigation;
            _navigationMapper = navigationMapper;
        }

        public async Task NavigateToAsync(object navigationSource, object parameter = null)
        {
            CheckIsInitialized();

            var type = _navigationMapper.GetTypeSource(navigationSource);

            if (type == null)
            {
                throw new InvalidOperationException(
                    "Can't find associated type for " + navigationSource.ToString());
            }

            ConstructorInfo constructor;
            object[] parameters;

            if (parameter == null)
            {
                constructor = type.GetTypeInfo()
                                  .DeclaredConstructors
                                  .FirstOrDefault(c => !c.GetParameters().Any());

                parameters = new object[] { };
            }
```

```
            else
            {
                constructor = type.GetTypeInfo()
                                  .DeclaredConstructors
                                  .FirstOrDefault(c =>
                                    {
                                        var p = c.GetParameters();
                                        return p.Count() == 1 &&
                                            p[0].ParameterType == parameter.GetType();
                                    });

                parameters = new[] { parameter };
            }

            if (constructor == null)
            {
                throw new InvalidOperationException(
                    "No suitable constructor found for page " + navigationSource.ToString());
            }

            var page = constructor.Invoke(parameters) as Page;

            await _navigation.PushAsync(page);
        }

        public async Task GoBackAsync()
        {
            CheckIsInitialized();

            await _navigation.PopAsync();
        }

        private void CheckIsInitialized()
        {
            if (_navigation == null || _navigationMapper == null)
                throw new NullReferenceException("Call Initialize method first.");
        }
    }
}
```

I get type of page on which user want navigate and create it's instance using reflection.

And then I could use navigation service on view model:

```
var navigationService = DependencyService.Get<IViewNavigationService>();
await navigationService.NavigateToAsync(NavigationPageSource.Page2, "hello from Page1");
```

## Section 5.5: Master Detail Root Page

```
public class App : Application
{
    internal static NavigationPage NavPage;

    public App ()
    {
        // The root page of your application
        MainPage = new RootPage();
    }
}
public class RootPage : MasterDetailPage
```

```
{
    public RootPage()
    {
        var menuPage = new MenuPage();
        menuPage.Menu.ItemSelected += (sender, e) => NavigateTo(e.SelectedItem as MenuItem);
        Master = menuPage;
        App.NavPage = new NavigationPage(new HomePage());
        Detail = App.NavPage;
    }
    protected override async void OnAppearing()
    {
        base.OnAppearing();
    }
    void NavigateTo(MenuItem menuItem)
    {
        Page displayPage = (Page)Activator.CreateInstance(menuItem.TargetType);
        Detail = new NavigationPage(displayPage);
        IsPresented = false;
    }
}
```

# Section 5.6: Hierarchical navigation with XAML

By default, the navigation pattern works like a stack of pages, calling the newest pages over the previous pages. You will need to use the NavigationPage object for this.

**Pushing new pages**

```
...
public class App : Application
{
    public App()
    {
        MainPage = new NavigationPage(new Page1());
    }
}
...
```

**Page1.xaml**

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 1" />
        <Button Text="Go to page 2" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...
```

**Page1.xaml.cs**

```
...
public partial class Page1 : ContentPage
{
    public Page1()
    {
        InitializeComponent();
    }

    protected async void GoToNextPage(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new Page2());
    }
}
```

```
...
```

**Page2.xaml**

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 2" />
        <Button Text="Go to Page 3" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...
```

**Page2.xaml.cs**

```
...
public partial class Page2 : ContentPage
{
    public Page2()
    {
        InitializeComponent();
    }

    protected async void GoToNextPage(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new Page3());
    }
}
...
```

### Popping pages

Normally the user uses the back button to return pages, but sometimes you need to control this programmatically, so you need to call the method **NavigationPage.PopAsync()** to return to the previous page or **NavigationPage.PopToRootAsync()** to return at the beggining, such like...

**Page3.xaml**

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 3" />
        <Button Text="Go to previous page" Clicked="GoToPreviousPage" />
        <Button Text="Go to beginning" Clicked="GoToStartPage" />
    </StackLayout>
</ContentPage.Content>
...
```

**Page3.xaml.cs**

```
...
public partial class Page3 : ContentPage
{
    public Page3()
    {
        InitializeComponent();
    }

    protected async void GoToPreviousPage(object sender, EventArgs e)
    {
        await Navigation.PopAsync();
    }

    protected async void GoToStartPage(object sender, EventArgs e)
    {
```

```
        await Navigation.PopToRootAsync();
    }
}
...
```

## Section 5.7: Modal navigation with XAML

Modal pages can created in three ways:

- From **NavigationPage** object for full screen pages
- For Alerts and Notifications
- For ActionSheets that are pop-ups menus

**Full screen modals**

```
...
// to open
await Navigation.PushModalAsync(new ModalPage());
// to close
await Navigation.PopModalAsync();
...
```

**Alerts/Confirmations and Notifications**

```
...
// alert
await DisplayAlert("Alert title", "Alert text", "Ok button text");
// confirmation
var booleanAnswer = await DisplayAlert("Confirm?", "Confirmation text", "Yes", "No");
...
```

**ActionSheets**

```
...
var selectedOption = await DisplayActionSheet("Options", "Cancel", "Destroy", "Option 1", "Option
2", "Option 3");
...
```