99/4A

XAMARINERS

Prism for Xamarin.Forms

Ben Ishiyama-Levy
Xamarin Evangelist
ben@xamariners.com
http://www.meetup.com/SingaporeMobileDev/

Xamarin

Prism : History

XAMARINERS

- Started with Microsoft Patterns and Practices Team
- Originally build for WPF and Silverlight
- Release in open Source
- Brian Lagunas (Infragistics) took over

Prism for Xamarin.Forms

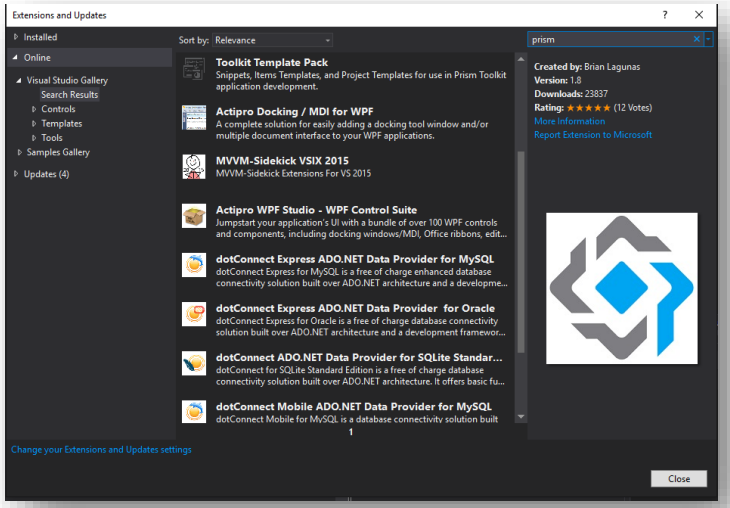
- Released Prism for Xamarin.Forms in 2016
- MVVM quick recap
- MVVM in Xamarin.Forms : Not really

Project Setup

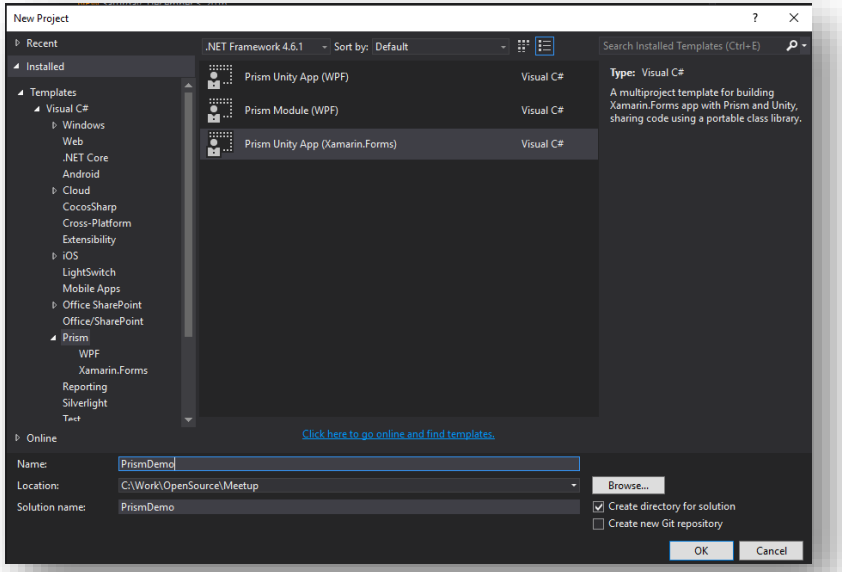
Visual Studio

XAMARINERS

- Install Prism Template: https://marketplace.visualstudio.com/items?itemName=BrianLagunas.PrismTemplatePack



- New Project with template:
  Prism Unity App

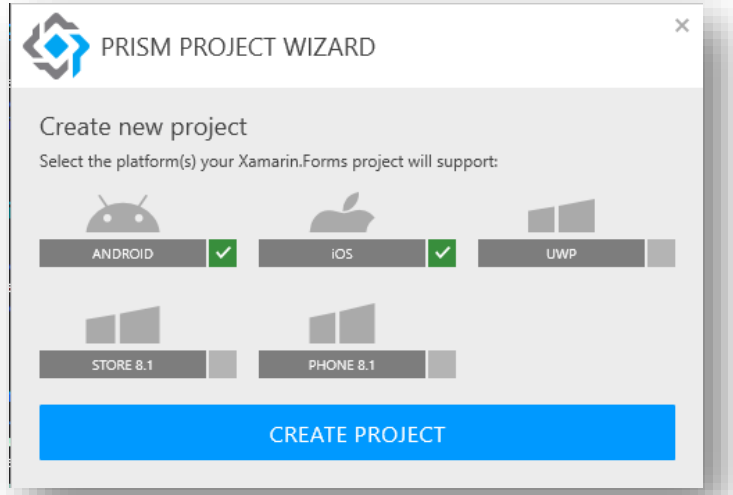


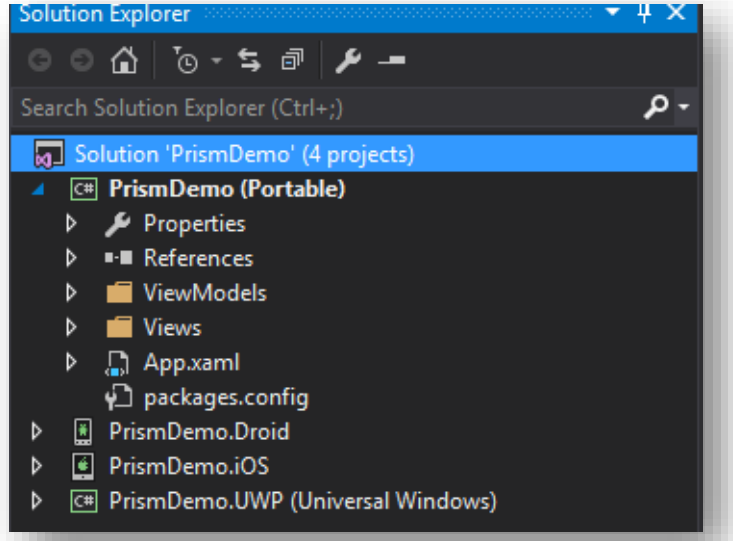
Project Setup

Visual Studio

XAMARINERS

- Prism template let you select your platforms (unlike Xamarin.Forms template)
- Creates a Xamarin.Forms solution with 1 project per plaftform and 1 PCL project
- Created Views and ViewModels folders on PCL project



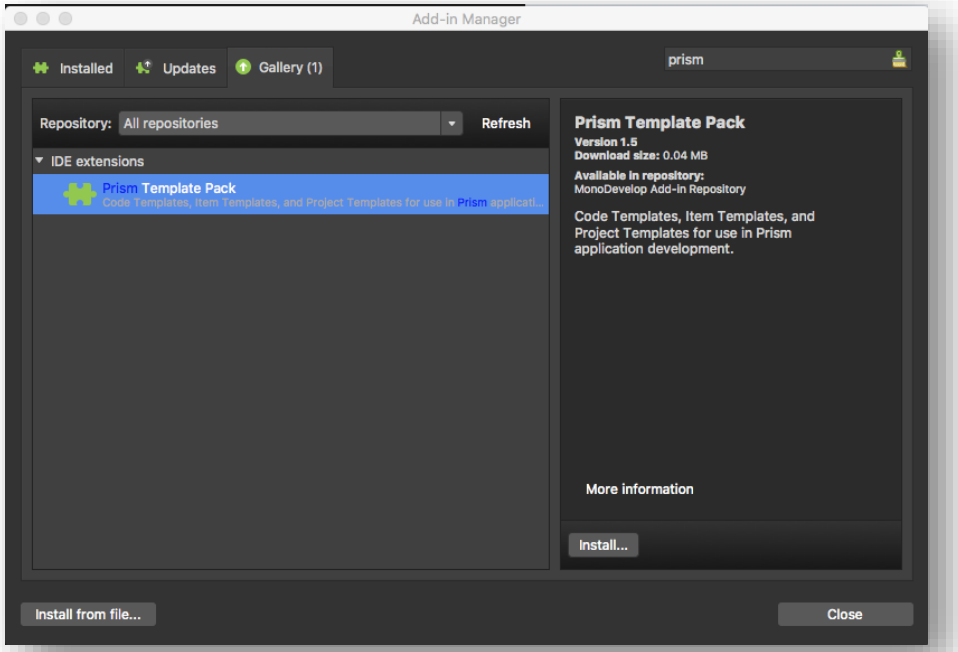


Project Setup

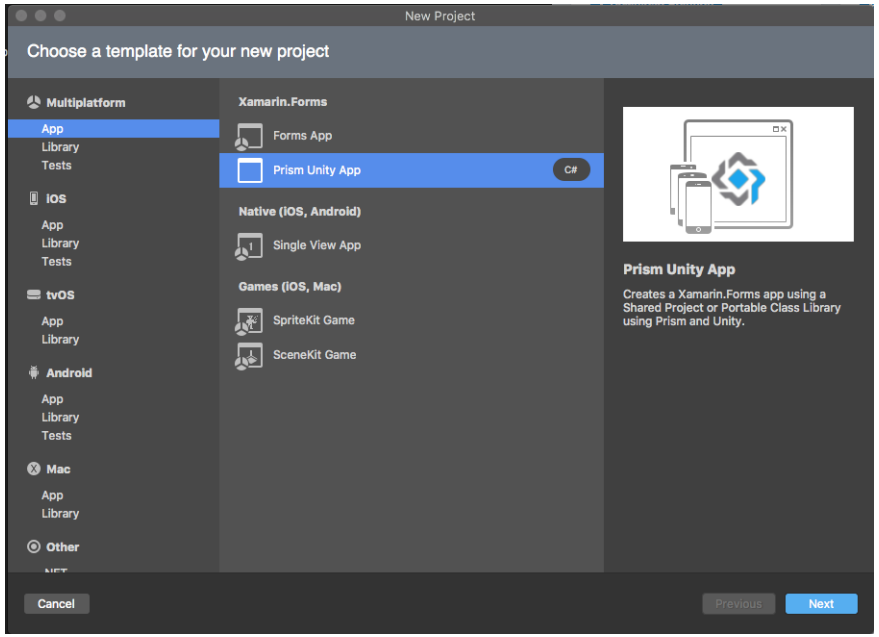
Xamarin Studio

XAMARINERS

- Install Prism Template from add-in manager:



- New Project with template:
  Prism Unity App

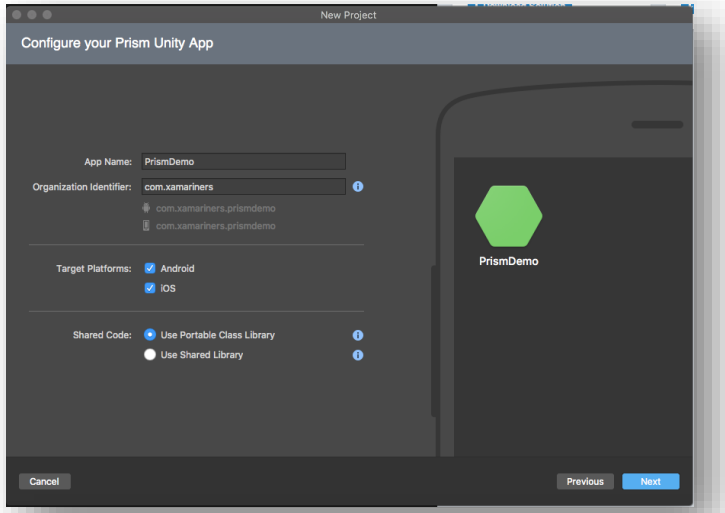


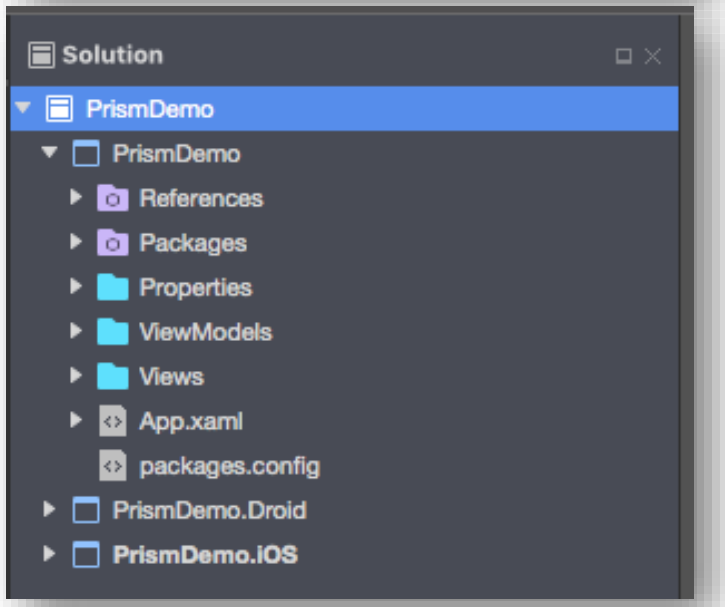
Project Setup

Xamarin Studio

XAMARINERS

- Prism template let you select your platforms (unlike Xamarin.Forms template)
- Creates a Xamarin.Forms solution with 1 project per plaftform and 1 PCL project
- Created **Views** and **ViewModels** folders on PCL project



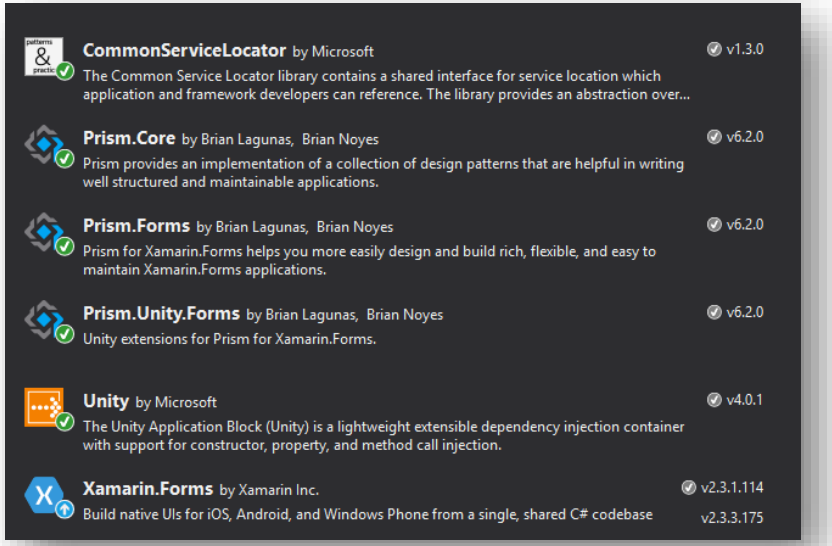


Project Setup

XAMARINERS

- Added packages: Prism (Core, Forms, Unity.Forms), Unity (Unity, CommonServiceLocator), Xamarin.Forms

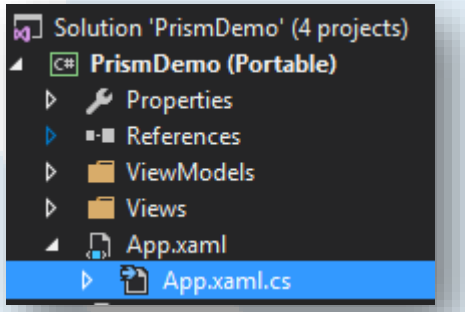
- Unity container can be replaced by Autofac, Dryloc, Ninject



App Class

XAMARINERS

- To add Prism support to Xamarin.Forms, The App class needs to inherit from **PrismApplication** instead of **Application**




```xml
<?xml version="1.0" encoding="utf-8" ?>
<prism:PrismApplication xmlns="http://xamarin.com/schemas/2014/forms"
                        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                        xmlns:prism="clr-namespace:Prism.Unity;assembly=Prism.Unity.Forms"
                        x:Class="PrismDemo.App">

</prism:PrismApplication>
```

App.xaml

```
namespace PrismDemo
{
    5 references
    public partial class App : PrismApplication
    {
        2 references
        public App(IPlatformInitializer initializer = null) : base(initializer) { }

        0 references
        protected override void OnInitialized()
        {
            InitializeComponent();

            NavigationService.NavigateAsync("MainPage?title=Hello%20from%20Xamarin.Forms");
        }

        0 references
        protected override void RegisterTypes()
        {
            Container.RegisterTypeForNavigation<MainPage>();
        }
    }
}
```

App.xaml.cs

App Class

Prism adds 3 features to app class:

- **Ctor** takes **IPlatformInitializer** parameter: provides a hook for platform specific Registations with DI Container
- **OnInitialized()** override: provides Xamarin.Forms initialization and navigation to app main page
- **RegisterTypes()** override: provides a hook for common Registations with DI Container

Connect Views & ViewModels

XAMARINERS

Assign ViewModel to View by Convention over Configuration:

- Xaml page is in the **Views** folder
- ViewModel is in the **ViewModels** folder
- ViewModel name = $"{PageName}ViewModel"
  - Ex: MainPage.xaml ⇔ MainPageViewModel.cs

Connect Views & ViewModels

XAMARINERS

Register Pages with **RegisterTypes()** on App class with **Container.RegisterTypeForNavigation<T>()** where T is the page type. Prism wires the **ViewModel** to the **View** thanks to convention

```
protected override void RegisterTypes()
{
    Container.RegisterTypeForNavigation<MainPage>();
}
```

You can also override conventions by using **Container.RegisterTypeForNavigation<TPageType, TViewModel>()**

```
protected override void RegisterTypes()
{
    Container.RegisterTypeForNavigation<MainPage, MainPageViewModel>();
}
```

Connect Views & ViewModels

XAMARINERS

Under the hood, prism registers the view with its type name string:

```
public static IUnityContainer RegisterTypeForNavigation<TView>(
{
  Type viewType = typeof (TView);
  if (string.IsNullOrWhiteSpace(name))
    name = viewType.Name;
  return container.RegisterTypeForNavigation(viewType, name);
}
```

Ultimately ending in Prism DI Container:

```
public static IUnityContainer RegisterTypeForNavigation(this IUnityContainer container, Type viewType, string name)
{
  PageNavigationRegistry.Register(name, viewType);
  return container.RegisterType(typeof (object), viewType, name, new InjectionMember[0]);
}
```

For consumption in navigation with PageNavigationService:

```
protected override Page CreatePage(string name)
{
  return this._container.Resolve<object>(name, new ResolverOverride[0]) as Page;
}
```

Connect Views & ViewModels

XAMARINERS

Navigation example:
Having a page type of **MainPage**, we use the string parameter **"MainPage"** as key on **NavigateAsync()** method to navigate to that page

```
NavigationService.NavigateAsync("MainPage");
```

The Page type name key can be overridden at registration with a custom string, hence using that custom key for navigation

```
Container.RegisterTypeForNavigation<MainPage>("CustomPage");
```

```
NavigationService.NavigateAsync("CustomPage");
```

ViewModel

XAMARINERS

No changes needed to XAML page to support Prism (for example, some MVVM frameworks require to change the ContentPage type with a custom one)

Only addition (although set to true as default so not visible unless for disabling): **ViewModelLocator.AutowireViewModel** on contentPage

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"
             prism:ViewModelLocator.AutowireViewModel="True"
             x:Class="PrismDemo.Views.MainPage"
             Title="MainPage">
```

Sets ViewModel to this Page **BindingContext**

BindableBase

XAMARINERS

ViewModels need to inherit Prism **BindableBase**

- Implements **INotifyPropertyChanged**
- Provides **SetProperty()** that stores property value and send a notification to all controls bound to property

```
public class MainPageViewModel : BindableBase, INavigationAware
{
    private string _title;
    1 reference
    public string Title
    {
        get { return _title; }
        set { SetProperty(ref _title, value); }
    }
```

Dependency Injection

XAMARINERS

To register types such as services, use the selected DI Container:

- For platform agnostic types, use **RegisterTypes()** on App class on PCL project

- For platform specific types, use **RegisterTypes()** on **AndroidInitializer** class (Android), **iOSInitializer** class (iOS), **UwpInitializer** class (UWP)

```
protected override void RegisterTypes()
{
    Container.RegisterTypeForNavigation<MainPage>();
    Container.RegisterType<ITodoService, FakeTodoService>();
}
```

Dependency Injection

XAMARINERS

As Pages (and therefore ViewModels through auto wiring) and services are registered in the DI container, resolved registered types instances are injected in all registered dependencies Constructors (along with their resolved child dependencies)

```
private string _title;
private readonly ITodoService _todoService;

1 reference
public string Title
{
    get { return _title; }
    set { SetProperty(ref _title, value); }
}

0 references
public MainPageViewModel(ITodoService todoService)
{
    _todoService = todoService;
}
```

Navigation LifeCycle

XAMARINERS

- We want to retrieve data asynchronously
- We need a hook other than the ViewModel constructor (not async)
- Xamarin.Forms offers **OnAppearing()** and **OnDisappearing()** on Page code behind, but we need those hooks on the ViewModel

Navigation LifeCycle

XAMARINERS

- ViewModels need to implement Prism **INavigationAware**:
  - **OnNavigatedTo() – Initialisation (ex: fetch data)**
  - **OnNavigatedFrom() - Cleanup**

Example:
OnNavigatedTo fetches data async

ObservableCollection property is initialized with the data

```
1 reference
public class MainPageViewModel : BindableBase, INavigationAware
{
    private readonly ITodoService _todoService;

    private ObservableCollection<TodoItem> _todoItems;
    1 reference
    public ObservableCollection<TodoItem> TodoItems
    {
        get { return _todoItems; }
        set { SetProperty(ref _todoItems, value); }
    }
    0 references
    public MainPageViewModel(ITodoService todoService)
    {
        _todoService = todoService;
    }

    0 references
    public void OnNavigatedFrom(NavigationParameters parameters)
    {

    }

    0 references
    public async void OnNavigatedTo(NavigationParameters parameters)
    {
        var todoItems = await _todoService.GetTodoItems();
        TodoItems = new ObservableCollection<TodoItem>(todoItems);
    }
}
```

Navigation LifeCycle

XAMARINERS

The Observable collection is bound to the Page, here in a **ListView:**

```
<ListView ItemsSource="{Binding TodoItems}" SelectedItem="{Binding SelectedTodoItem}">
  <ListView.ItemTemplate>
    <DataTemplate>
      <ViewCell>
        <controls:TodoListItem />
      </ViewCell>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

The item properties are encapsulated in a control for reusability and maintenance sake

Same rules applies to value converters

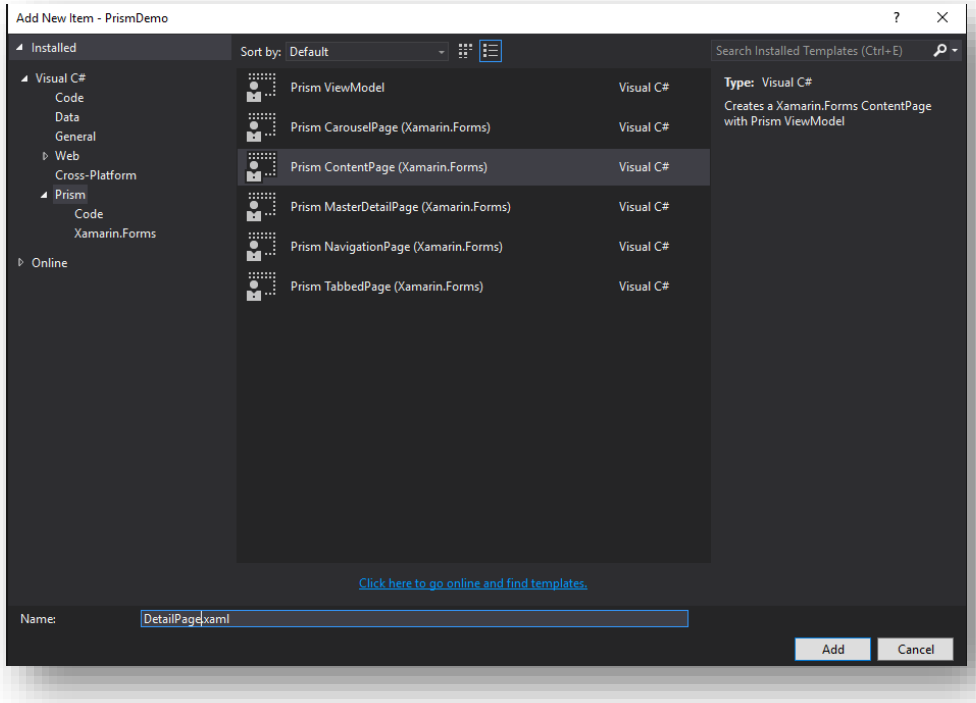
Any trick of the trade that makes your xaml more readable is a good trick

```
<?xml version="1.0" encoding="utf-8" ?>
<Grid xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            xmlns:valueConverters="clr-namespace:PrismDemo.ValueConverters;assembly=PrismDemo"
            x:Class="PrismDemo.Controls.TodoListItem"
            Padding="10, 0, 10, 0">
  <Label Text="{Binding MainText}" VerticalOptions="Center" HorizontalOptions="Center" />

  <Grid.Resources>
    <ResourceDictionary>
      <valueConverters:TodoStatusToIconConverter x:Key="TodoStatusToIconConv" />H
    </ResourceDictionary>
  </Grid.Resources>

  <StackLayout Grid.Row="0" Grid.Column="1" Padding="0,3,0,0" >
    <Label Text="{Binding Name}" FontSize="16" LineBreakMode="TailTruncation" />
  </StackLayout>
  <Label Grid.Row="1" Grid.Column="1" Text="{Binding Details}" FontSize="9" />

  <Image Grid.Row="0" Grid.Column="0" Grid.RowSpan="2"
         Source="{Binding TodoStatus, Converter={StaticResource TodoStatusToIconConv}}"/>

  <Grid.RowDefinitions>
    <RowDefinition Height="23" ></RowDefinition>
    <RowDefinition Height="*" ></RowDefinition>
  </Grid.RowDefinitions>

  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="40" ></ColumnDefinition>
    <ColumnDefinition Width="*" ></ColumnDefinition>
  </Grid.ColumnDefinitions>
</Grid>
```

Navigation With Parameters

XAMARINERS

- **NavigationParameters** is a **Dictionary<string, object>** that can be optionally provided to the **NavigateAsync()** method
- **NavigationParameters** in injected in the **OnNavigatedTo()** and **OnNavigatedFrom()** methods
- Add a **ContentPage** to the **Views** folder using the Prism templates. This will create the **Page**, **ViewModel**, and register them with the DI **Container** automatically



Navigation With Parameters

XAMARINERS

- Add a command hook to the **ItemTapped** event of the **ListView**
- this can achieved by binding the **SelectedItem** property of the **ListView** with a **ViewModel** property and in turn execute a command on property changed
- Another way is to use Xamarin.Forms behaviors to bind a command to an event (use the nuget **Corcav.Behaviors**):

```
<ListView.ItemTemplate>
  <behaviors:Interaction.Behaviors>
    <behaviors:BehaviorCollection>
      <behaviors:EventToCommand EventName="ItemTapped" Command="{Binding NavigateToTodoItem}" />
    </behaviors:BehaviorCollection>
  </behaviors:Interaction.Behaviors>
  <DataTemplate>
    <ViewCell>
      <controls:TodoListItem />
    </ViewCell>
  </DataTemplate>
</ListView.ItemTemplate>
</ListView>
```

Navigation With Parameters

XAMARINERS

Add a delegate parameterized command (saves you creating a ICommand all the time), that provides the selected item object as command parameter

Instantiate a **NavigationParameters** object and add the selected item to this Dictionary

```
0 references
public DelegateCommand<ItemTappedEventArgs> NavigateToTodoItem
{
    get
    {
        if (_navigateToTodoItem == null)
        {
            _navigateToTodoItem = new DelegateCommand<ItemTappedEventArgs>(async selectedTodoItem =>
            {
                NavigationParameters param = new NavigationParameters();
                param.Add("todoItem", selectedTodoItem.Item);
                await _navigationService.NavigateAsync("TodoItemPage", param);
            });
        }
        return _navigateToTodoItem;
    }
}
```

Add the **INavigation** service to the Ctor to get its resolved instance and use it to navigate to the detail page.

```
0 references
public TodoListPageViewModel(ITodoService todoService, INavigationService navigationService)
{
    _navigationService = navigationService;
    _todoService = todoService;
}
```

Navigation With Parameters

Retrieve the **NavigationParameters** on the **OnNavigatedTo()** method of the detail page **ViewModel**

Set the data to a **ViewModel** bindable property

Consume that object on the detail page Xaml

```
1 reference
public class TodoItemPageViewModel : BindableBase
{

    private TodoItem _todoItem;
    1 reference
    public TodoItem TodoItem
    {
        get { return _todoItem; }
        set { SetProperty(ref _todoItem, value); }
    }
    0 references
    public TodoItemPageViewModel()
    {
    }

    0 references
    public void OnNavigatedTo(NavigationParameters parameters)
    {
        TodoItem = parameters["todoItem"] as TodoItem;
    }
}
```

Deep Linking

Prism Support complex queries as parameters of the **NavigateAsync()** method

Supports queries like "**MainPage/DetailPage?id=1**" to navigate directly to the detail page of the app and, at the same time, passing a parameter called id with value 1

Example: link a specific application page from another application, a website or a section of your app

Deep Linking: Navigation Page

XAMARINERS

- Register the base **NavigationPage** type included in Xamarin.Forms as a type for navigation in the **Container**
- use the query "**NavigationPage/MainPage**" to tell to the **NavigationService** that we need to navigate first to the page identified by the **NavigationPage** key and then to the one identified by the **MainPage** key
- As **NavigationPage** is just a container, the **MainPage** (and every consequent page in the navigation flow) will be embedded into a **NavigationPage**

```
public partial class App : PrismApplication
{
    3 references | Ben Levy, 25 minutes ago | 1 author, 1 change
    public App(IPlatformInitializer initializer = null) : base(initializer) { }

    0 references | Ben Levy, 25 minutes ago | 1 author, 1 change
    protected override void OnInitialized()
    {
        InitializeComponent();

        NavigationService.NavigateAsync("NavigationPage/TodoListPage");

    }

    0 references | Ben Levy, 25 minutes ago | 1 author, 1 change
    protected override void RegisterTypes()
    {
        Container.RegisterTypeForNavigation<NavigationPage>();

        Container.RegisterTypeForNavigation<TodoListPage>();
        Container.RegisterTypeForNavigation<TodoItemPage>();
        Container.RegisterType<ITodoService, FakeTodoService>();
    }
```

Deep Linking: NavigationParameters

XAMARINERS

- Deep linking allows QueryString NavigationParameters in queries:

```
NavigationService.NavigateAsync("FirstPage?id=1&title=First page");
```

- the destination page will receive, in the **NavigationParams** object of the **OnNavigatedTo()** method, two items:
- one with key id and value 1
- one with key title and value First page.

```
1 reference | Ben Levy, 38 minutes ago | 1 author, 1 change
public void OnNavigatedTo(NavigationParameters parameters)
{
    string id = parameters["id"].ToString();
    string title = parameters["title"].ToString();

    Title = $"Page with id {id} and title {title}";
}
```

Page Dialog Services

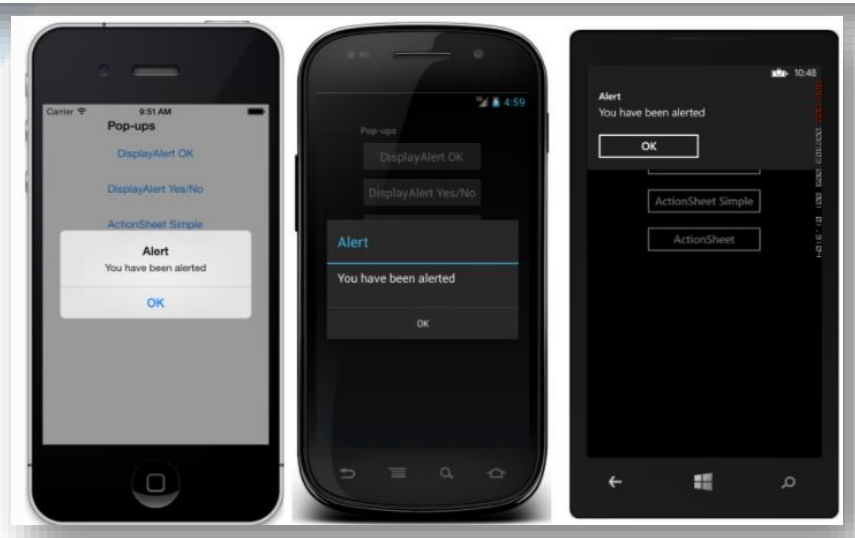
XAMARINERS

- Provides control of **DisplayAlert** and **DisplayActionSheet** from **ViewModel**
- Get an instance of the IPageDialogService from the ViewModel constructor using DI

```
public MainPageViewModel(IPageDialogService dialogService)
{
    _dialogService = dialogService;
}
```

- DisplayAlertAsync method shows a modal pop-up to alert the user or ask simple questions of them.
- To display these alerts with Prism's IPageDialogService, use the DisplayAlertAsync method:

```
_dialogService.DisplayAlertAsync("Alert", "You have been alerted", "OK");
```

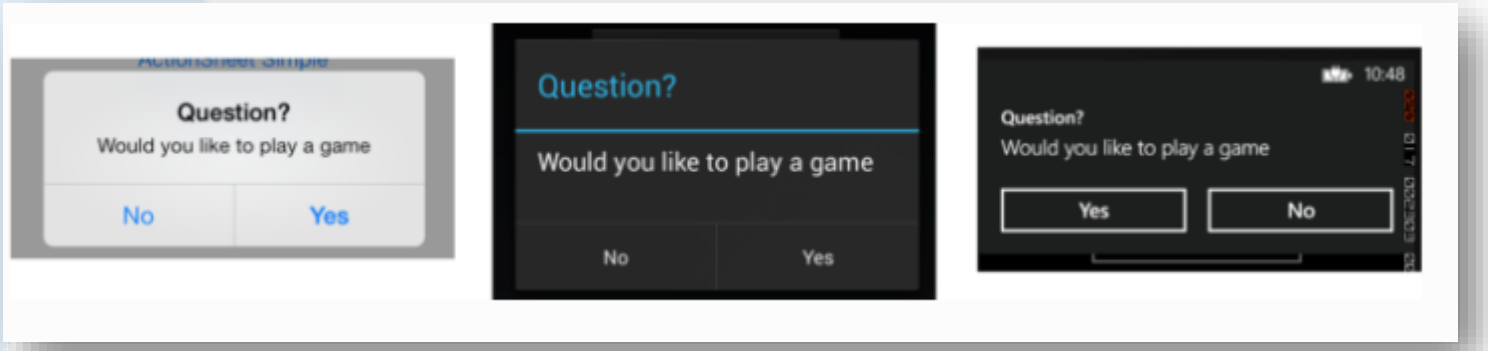


DisplayAlertAsync

XAMARINERS

- To get a response from an alert, supply text for both buttons and await the method.
- After the user selects one of the options the answer will be returned to your code

```
var alertButton2 = new Button { Text = "DisplayAlert Yes/No" }; // triggers alert
alertButton2.Clicked += async (sender, e) =>
{
    var answer = await DisplayAlertAsync ("Question?", "Would you like to play a game", "Yes", "No");
    Debug.WriteLine("Answer: " + answer); // writes true or false to the console
};
```



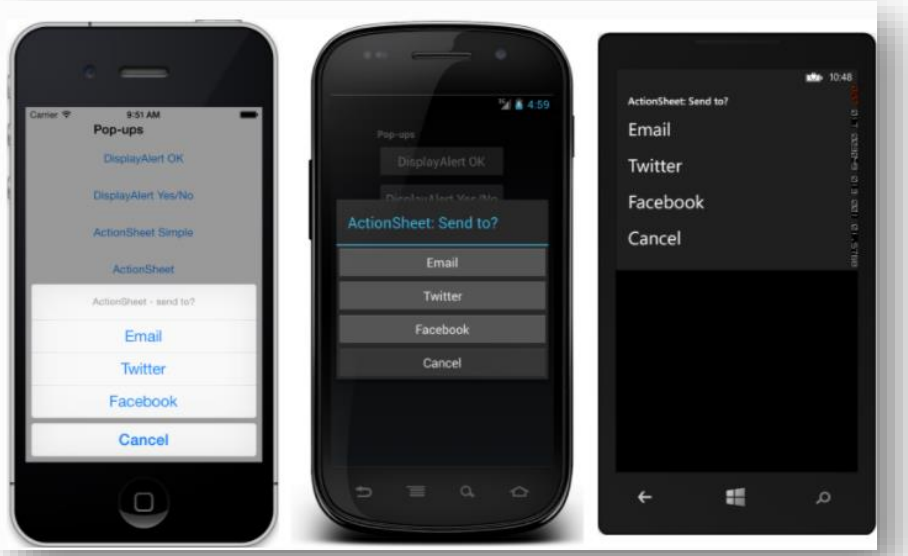
`

DisplayActionSheetAsync

XAMARINERS

- To display an action sheet, await DisplayActionSheetAsync in any ViewModel, passing the message and button labels as strings.
- The method returns the string label of the button that was clicked by the user

```
var actionButton1 = new Button { Text = "ActionSheet Simple" };
actionButton1.Clicked += async (sender, e) =>
{
    var action = await DisplayActionSheetAsync ("ActionSheet: Send to?", "Cancel", null, "Email", "Twitter",
    Debug.WriteLine("Action: " + action); // writes the selected button label to the console
};
```



`

More Info

XAMARINERS

- Prism GitHub: https://github.com/PrismLibrary/Prism
- Prism Doc: http://prismlibrary.readthedocs.io/en/latest/
- Prism template pack: https://marketplace.visualstudio.com/items?itemName=BrianLagunas.PrismTemplatePack
- This demo repo : https://github.com/PrismLibrary/Prism

XAMARINERS

Thank You! Questions?

Ben Ishiyama-Levy
Xamarin Evangelist

ben@xamariners.com