Microsoft

Xamarin University

XAM290

Master-detail and Drawer Navigation

Download materials from university.xamarin.com

Objectives

1. Choose between *split* and *popover* for your master view
2. Switch between pages using popover drawer navigation
3. Display a collection using master-detail split view

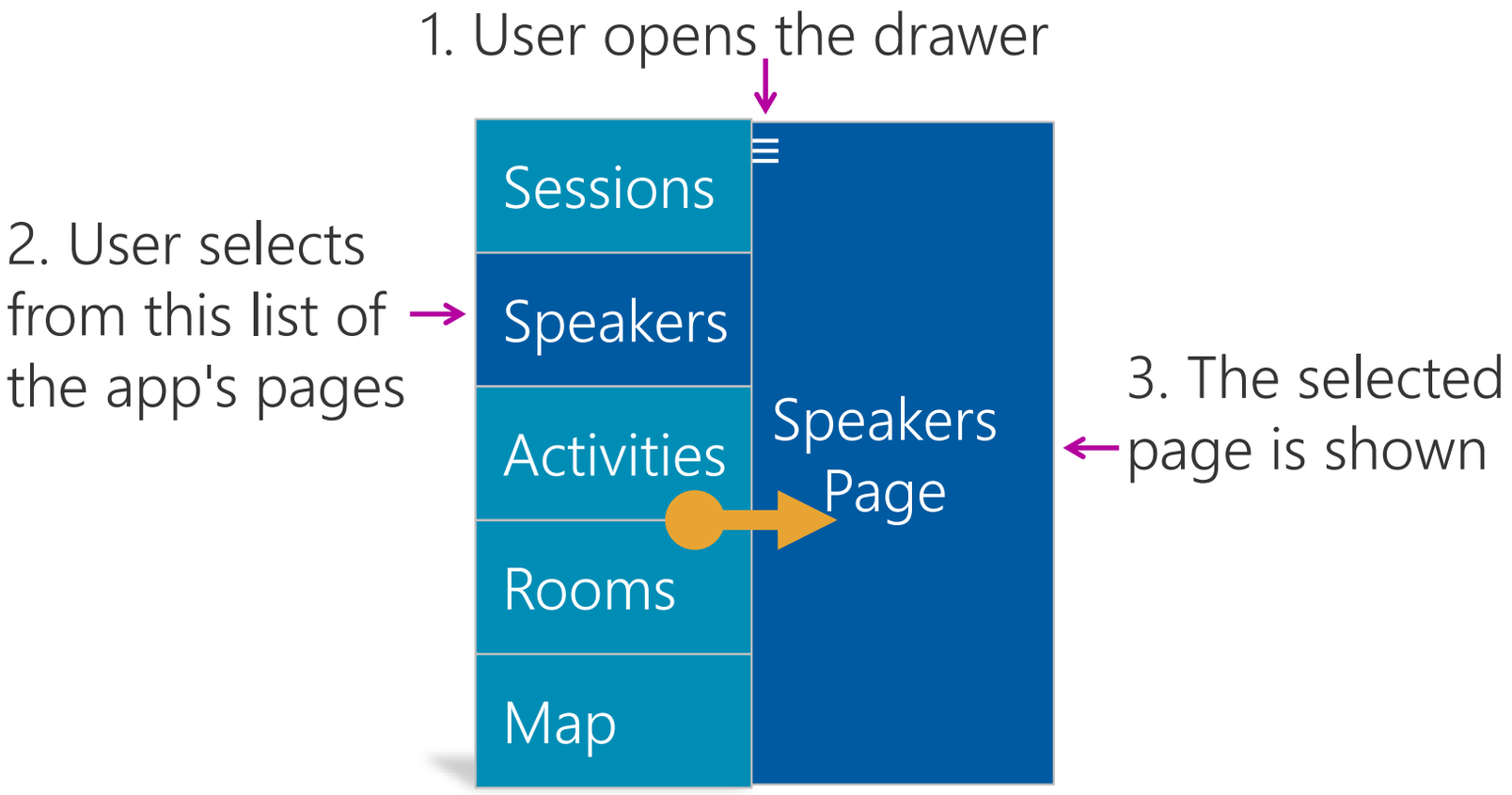
Choose between split and popover for your master view

Tasks

1. Decide whether split or popover behavior is appropriate for your data
2. Control the `Master` behavior of a `MasterDetailPage`

What is drawer navigation?

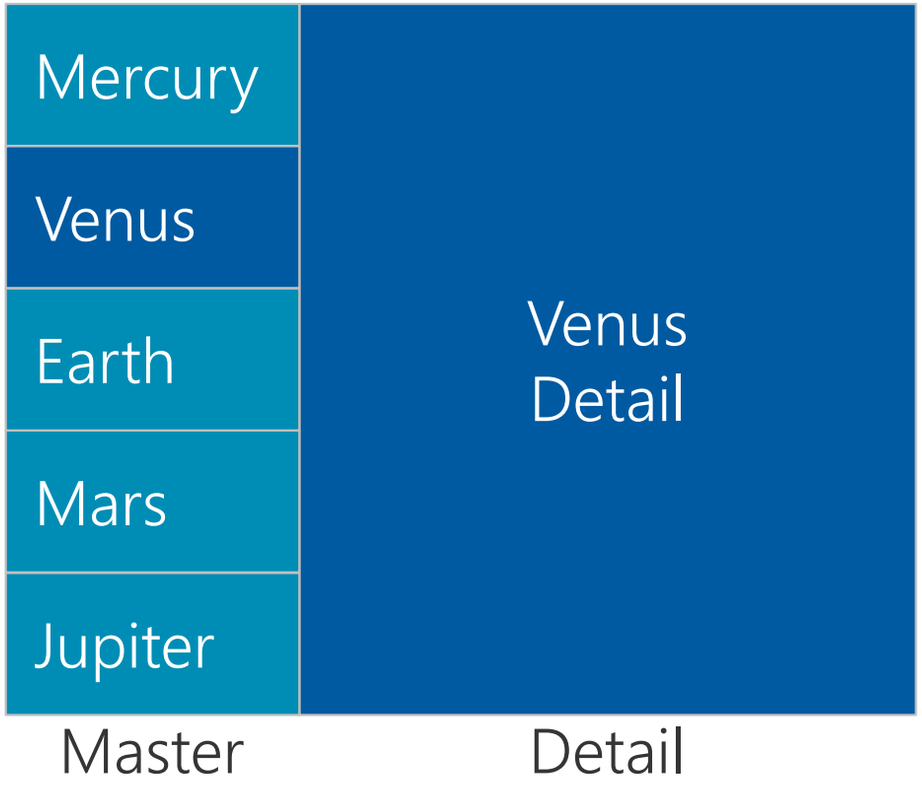
Drawer navigation is a navigation paradigm that uses a slide-out drawer to host a menu of pages and a content area to display the user's selected page



What is master-detail?

Master-detail is a way to display a collection of homogenous data that shows an overview of the entire collection and an expanded view of a single item

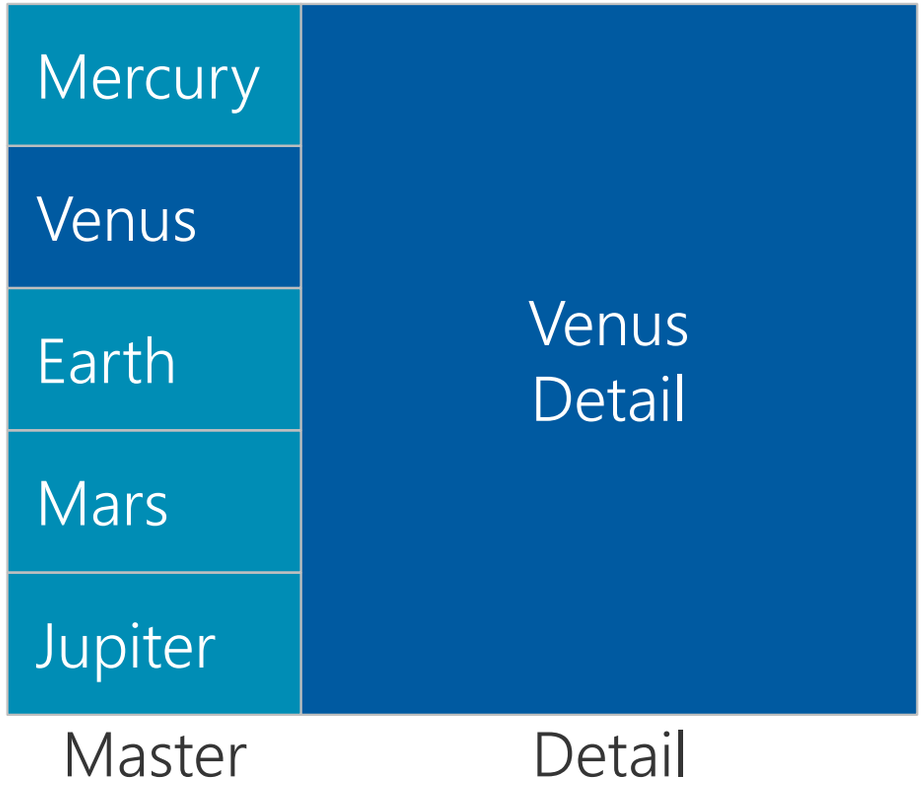
1. User selects one entry from a list of same-type items →

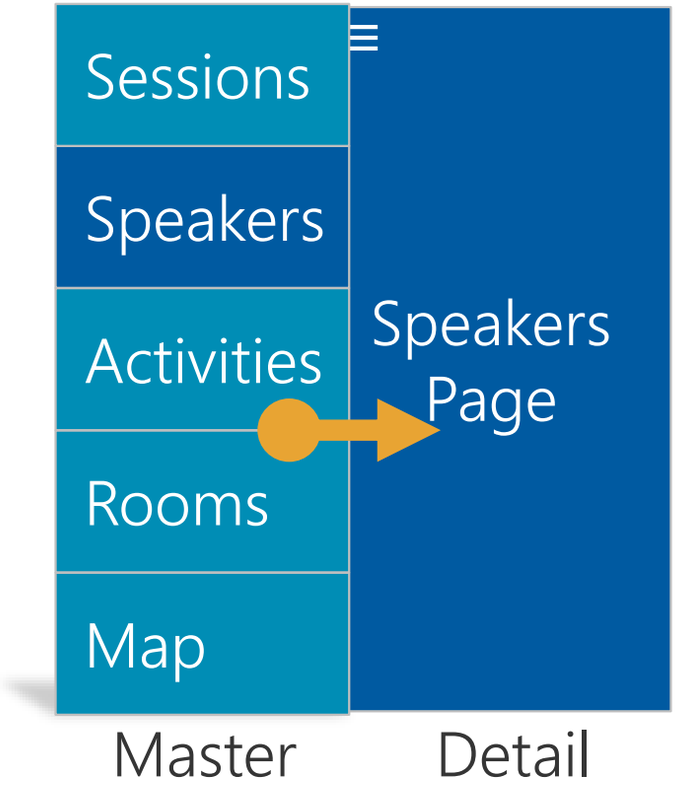


← 2. Long-form data for the selected Item is shown

Terminology

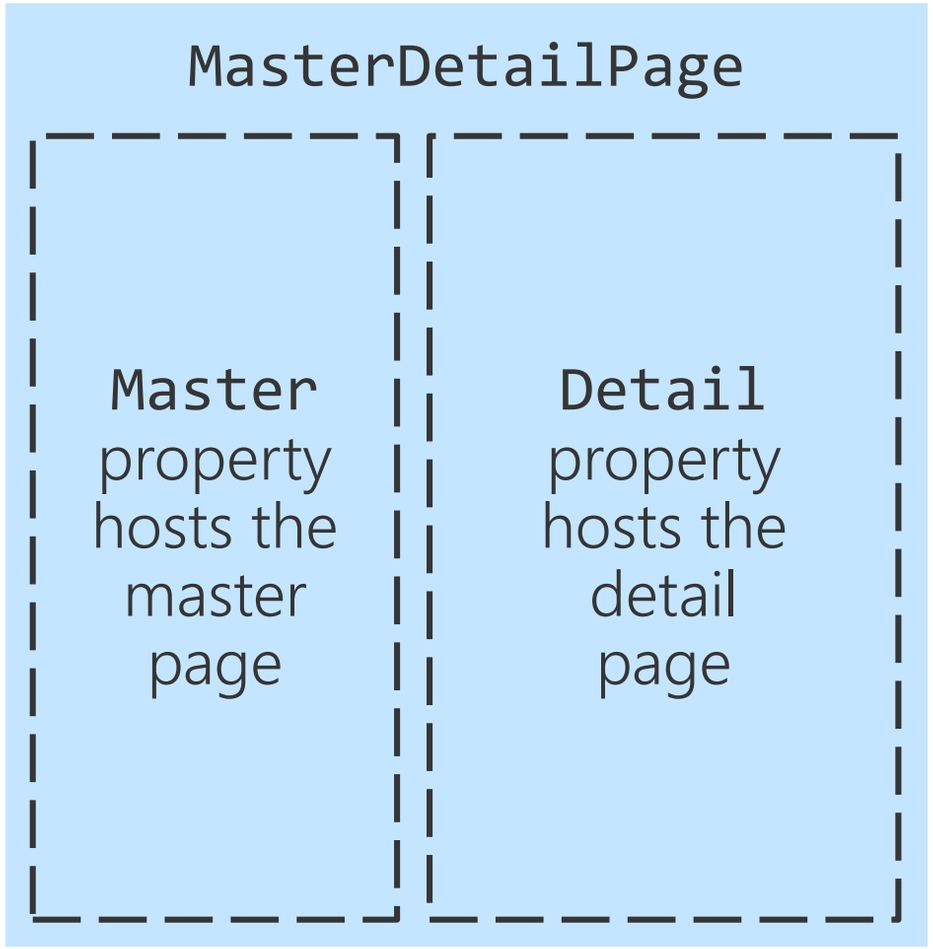
Both master-detail and drawer navigation have two content areas - these are referred to as *Master* and *Detail*





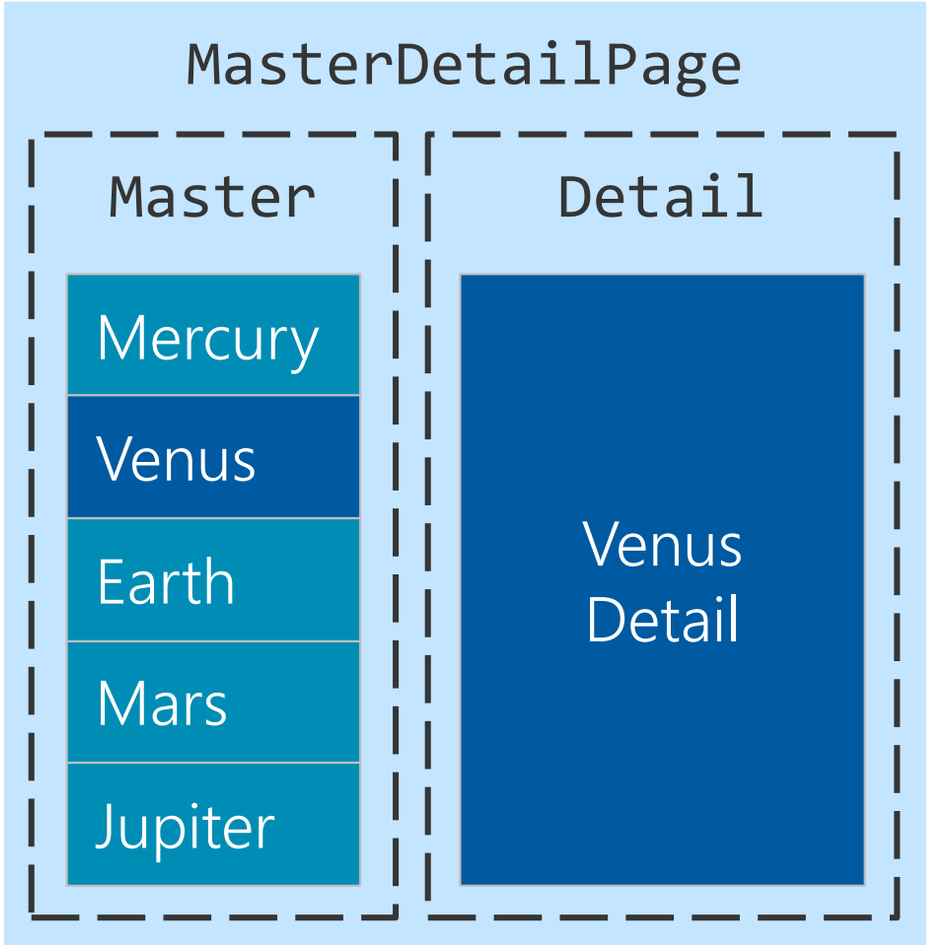
What is MasterDetailPage?

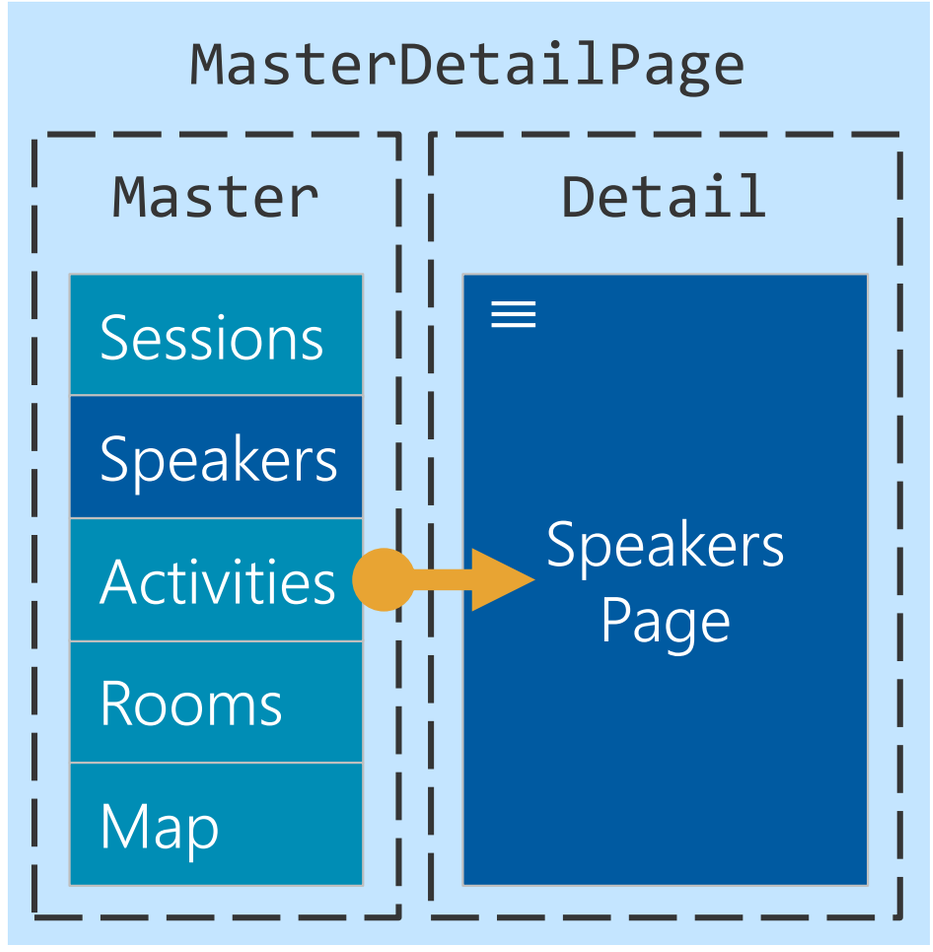
`MasterDetailPage` is a Xamarin.Forms page that displays a **`Master`** page and a **`Detail`** page and coordinates the synchronization between them



When to use MasterDetailPage

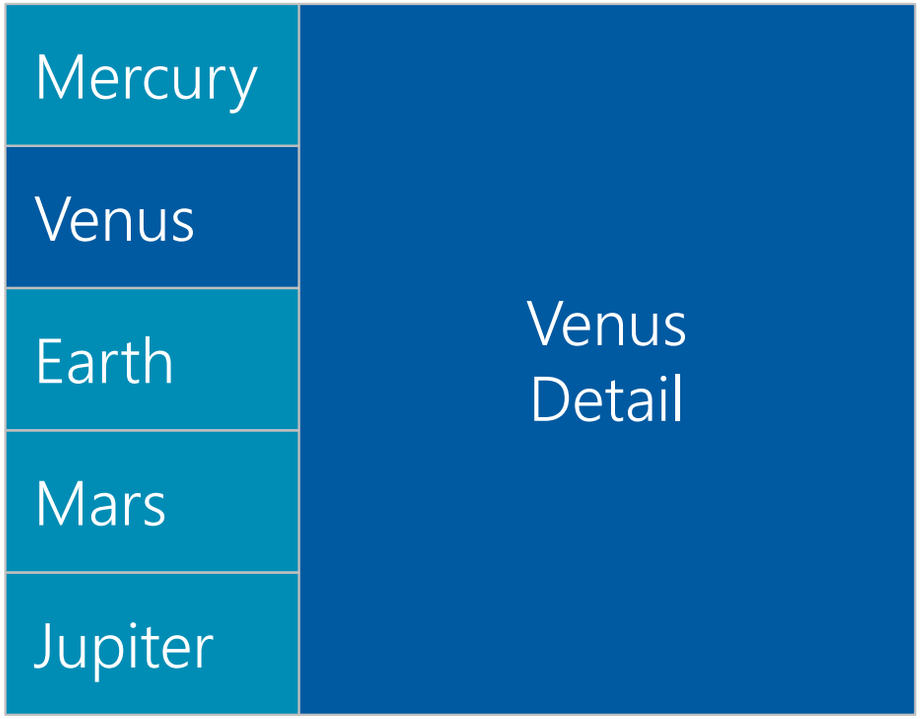
`MasterDetailPage` is used for both master-detail and drawer navigation



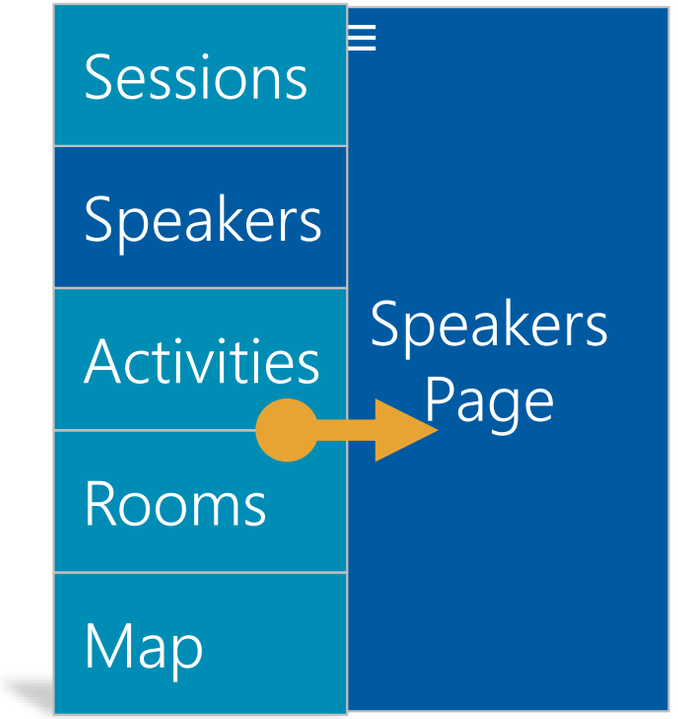


Split vs. popover behavior

The terms *split* and *popover* describe the visibility options for the master view in a master-detail or drawer-navigation display



Split has a menu that is always visible



Popover has a drawer that slides over the content

What is MasterBehavior?

The **`MasterBehavior`** property lets you influence the presentation of the master page in a **`MasterDetailPage`**

You choose your preferred behavior →

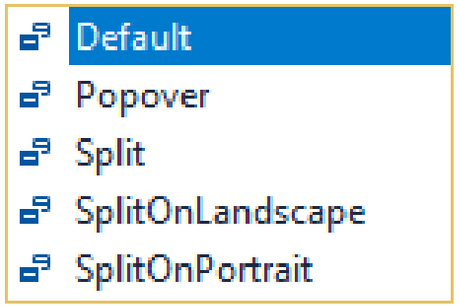
```
var md = new MasterDetailPage();

md.MasterBehavior = MasterBehavior.
```

- Default
- Popover
- Split
- SplitOnLandscape
- SplitOnPortrait

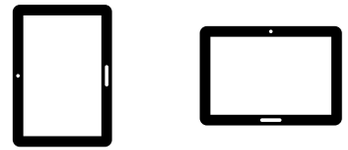
Factors that determine master behavior

`MasterDetailPage` computes the master behavior based on your setting for **`MasterBehavior`** and attributes of the runtime device



`MasterBehavior` setting

Device idiom

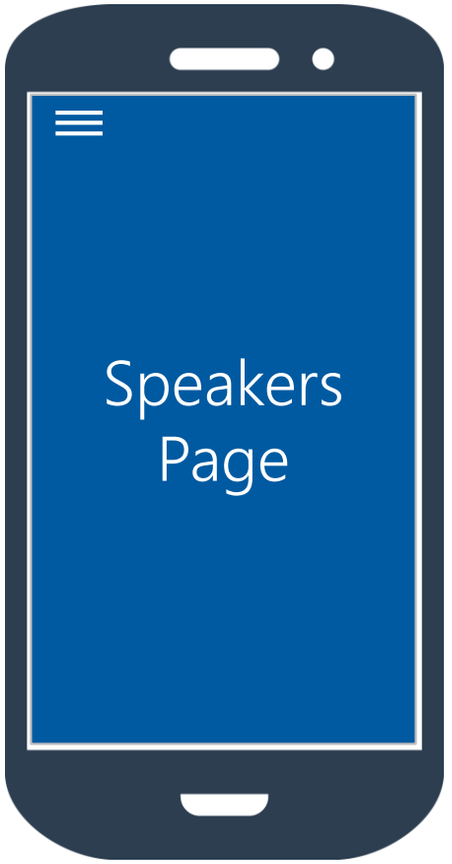


Screen orientation

Phone behavior

`MasterDetailPage` always uses popover behavior for the master when running on a phone

Your setting for `MasterBehavior` is ignored on a phone



Non-phone behavior

`MasterDetailPage` uses your **`MasterBehavior`** setting and device orientation to determine master behavior on non-phone devices

	Portrait	Landscape
`Default`	popover	split
`Popover`	popover	popover
`Split`	split	split
`SplitOnLandscape`	popover	split
`SplitOnPortrait`	split	popover

Which behavior should you choose?

Typically, drawer navigation uses popover while master-detail uses one of the split options

		Portrait	Landscape
Drawer →	**Default**	popover	split
	Popover	popover	popover
Master-detail	**Split**	split	split
	SplitOnLandscape	popover	split
	SplitOnPortrait	split	popover

Exercise

Control the Master behavior of a MasterDetailPage

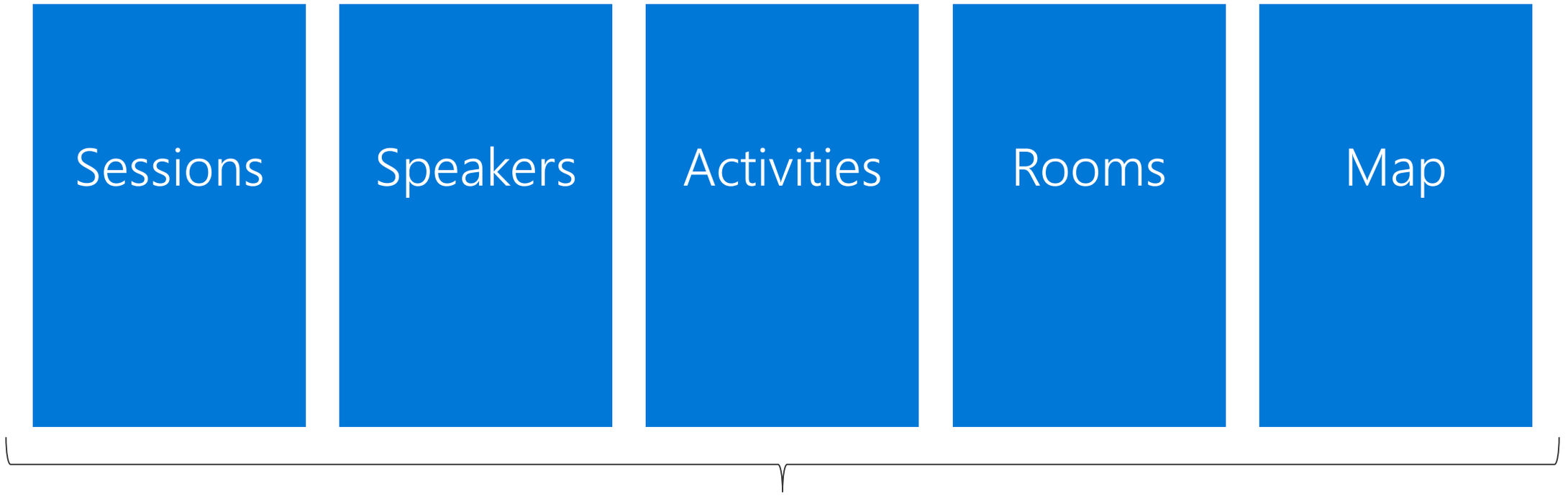
Switch between pages using popover drawer navigation

Tasks

1. Create a drawer menu
2. Create the content pages
3. Instantiate a `MasterDetailPage`
4. Navigate to the appropriate page when the user selects an item in drawer

Motivation

Apps with many top-level pages need a navigation model that makes all pages discoverable but does not occupy too much screen space

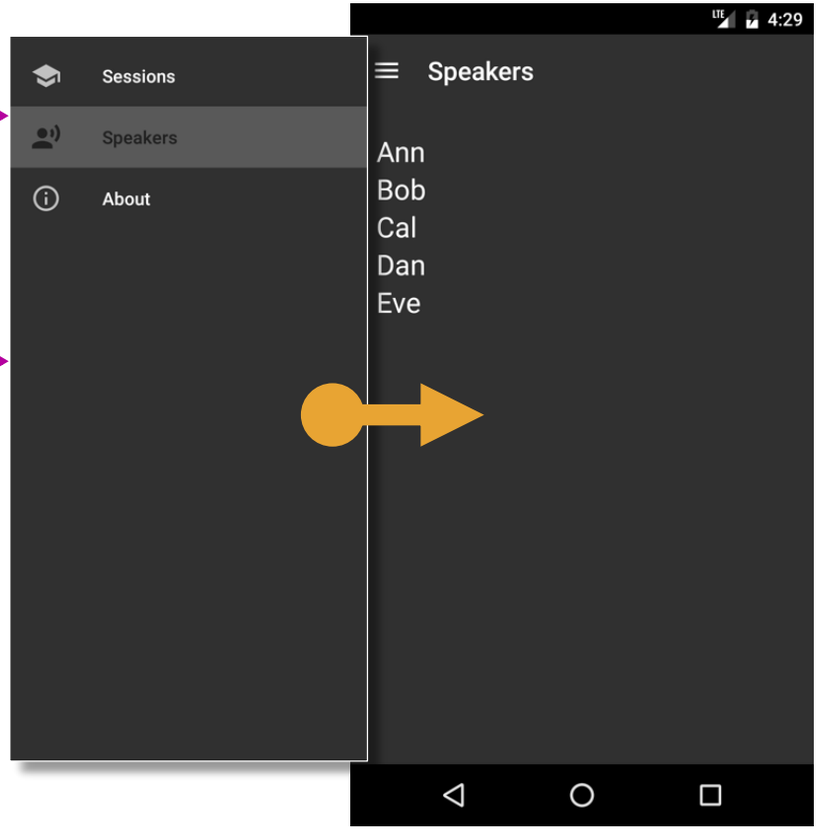


What is drawer navigation?

Drawer navigation is a navigation paradigm that uses a menu in a sliding panel for navigation

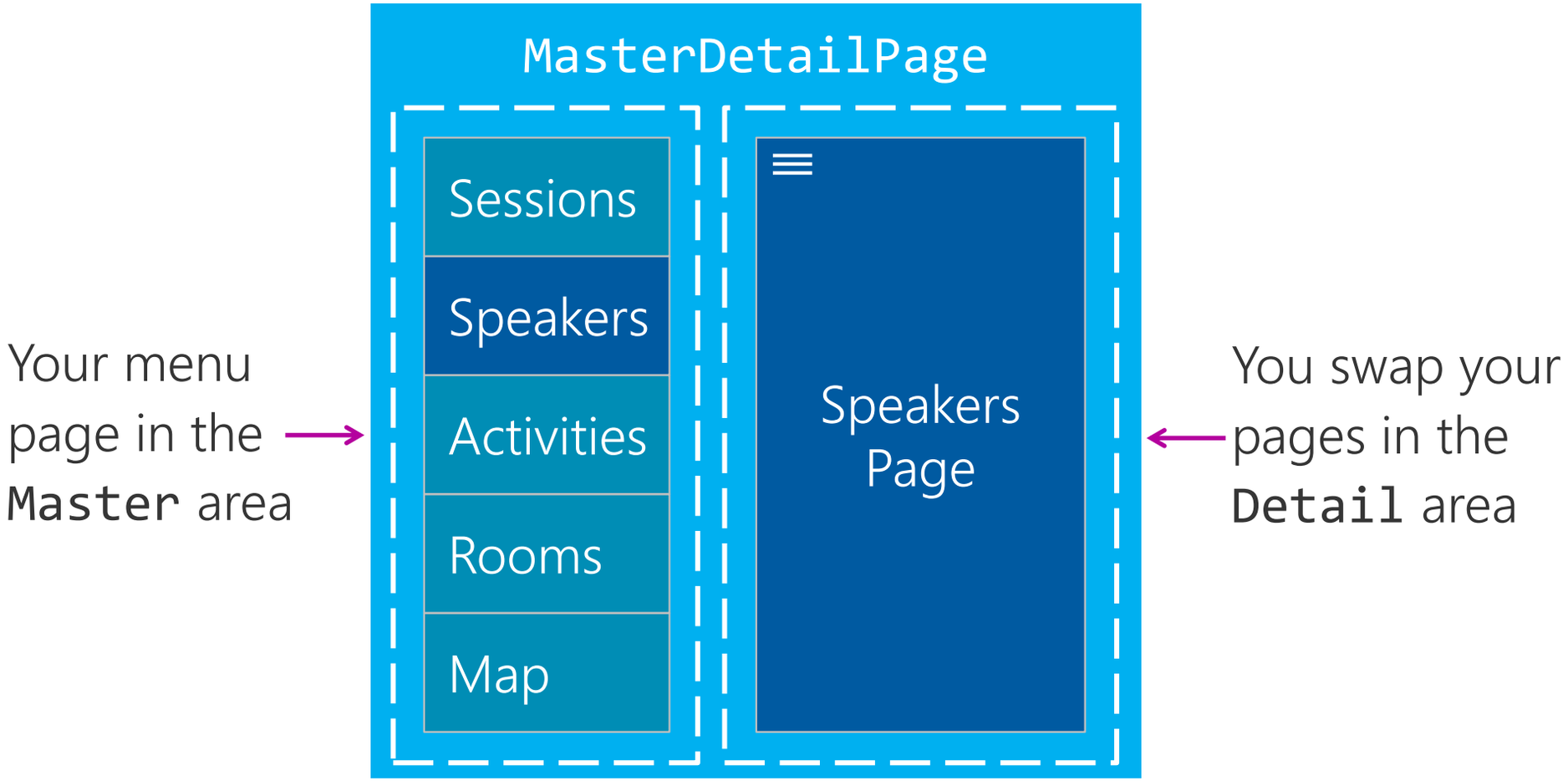
Menu has room for many entries

Menu overlays the content when open, but disappears when closed



Drawer-navigation architecture

`MasterDetailPage` hosts your content and implements the sliding-drawer animation



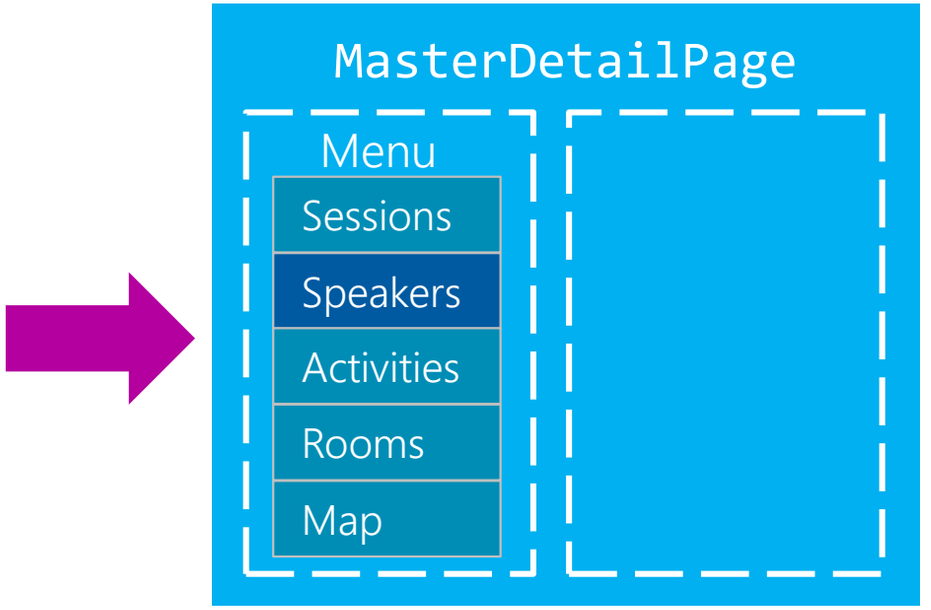
How to code the drawer menu

Your drawer menu presents the list of your app's pages to the user

ConferenceMenuPage.xaml

`Title` must be set

```
<ContentPage Title="Menu">
    <StackLayout>
        <Button Text="Sessions" />
        <Button Text="Speakers" />
        <Button Text="Activities" />
        <Button Text="Rooms" />
        <Button Text="Map" />
    </StackLayout>
</ContentPage>
```

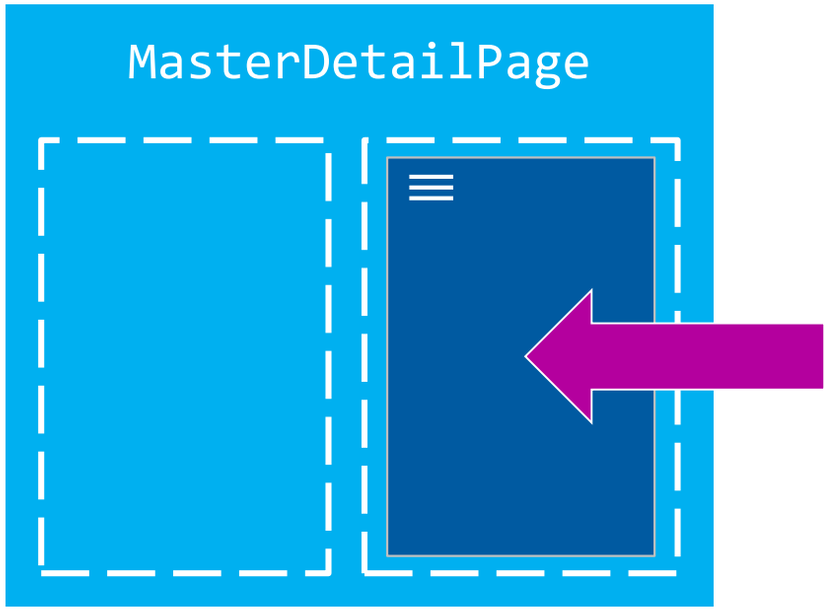


Must be a **`Page`**

You can use **`Buttons`**, a **`ListView`**, etc.

How to code your content pages

Typically, each of your top-level pages will be its own **ContentPage**




```
<ContentPage> <!-- SessionsPage   --> </ContentPage>
<ContentPage> <!-- SpeakersPage   --> </ContentPage>
<ContentPage> <!-- ActivitiesPage --> </ContentPage>
<ContentPage> <!-- RoomsPage      --> </ContentPage>
<ContentPage> <!-- MapPage        --> </ContentPage>
```

Must be **Pages**

You build the pages as appropriate

How to build the drawer navigation UI [overview]

Several steps are required to assemble the drawer navigation user interface

1. Derive from `MasterDetailPage`
2. Set the `Master` property
3. Set the `Detail` property
4. Wrap the `Detail` page in a `NavigationPage`
5. Use your derived type as your App's `MainPage`

How to build the drawer navigation UI [step 1]

Derive from **`MasterDetailPage`** so your class inherits master-detail functionality like the drawer and the **`Master`**/**`Detail`** properties

```
public class ConferenceMasterDetailPage : MasterDetailPage
{
    public ConferenceMasterDetailPage()
    {
        base.Master = ...
        base.Detail = ...
    }
}
```

How to build the drawer navigation UI [step 2]

Set the **`Master`** property to your menu page

Load your menu into the drawer

```
public class ConferenceMasterDetailPage : MasterDetailPage
{
    public ConferenceMasterDetailPage()
    {
        this.Master = new ConferenceMenuPage();
        this.Detail = ...
    }
}
```

How to build the drawer navigation UI [step 3]

Set the **`Detail`** property to a page of your choice to be displayed at startup

Set the initial page the user will see

```
public class ConferenceMasterDetailPage : MasterDetailPage
{
    public ConferenceMasterDetailPage()
    {
        this.Master = new ConferenceMenuPage();
        this.Detail = new SessionsPage();
    }
}
```

How to build the drawer navigation UI [step 4]

Wrap the **`Detail`** page in a **`NavigationPage`** so it includes a navigation bar which ensures the menu button is visible

```
public class ConferenceMasterDetailPage : MasterDetailPage
{
    public ConferenceMasterDetailPage()
    {
        this.Master = new ConferenceMenuPage();
        this.Detail = new NavigationPage(new SessionsPage());
    }
}
```

Required for Android but often used on all platforms to keep the code simple

How to build the drawer navigation UI [step 5]

Use your derived type as your App's **`MainPage`**

```
public partial class App : Application
{
    public App()
    {  ...
       MainPage = new ConferenceMasterDetailPage();
    }
}
```

Should be the root page of your UI

Always set your **`MasterDetailPage`** as the root page, nesting in other page types is not supported

Menu icon

Android and Windows include a menu icon for drawer navigation, for iOS you must provide an image

```
<ContentPage Title="Menu">
    <ContentPage.Icon>
        <OnPlatform x:TypeArguments="FileImageSource">
            <On Platform="iOS" Value="nav-menu-icon.png" />
        </OnPlatform>
     </ContentPage.Icon>
     ...
</ContentPage>
```

```
public ConferenceMasterDetailPage()
{
   if (Device.RuntimePlatform == Device.iOS)
      master.Icon = (FileImageSource)ImageSource.FromFile("nav-menu-icon.png");
   ...
}
```

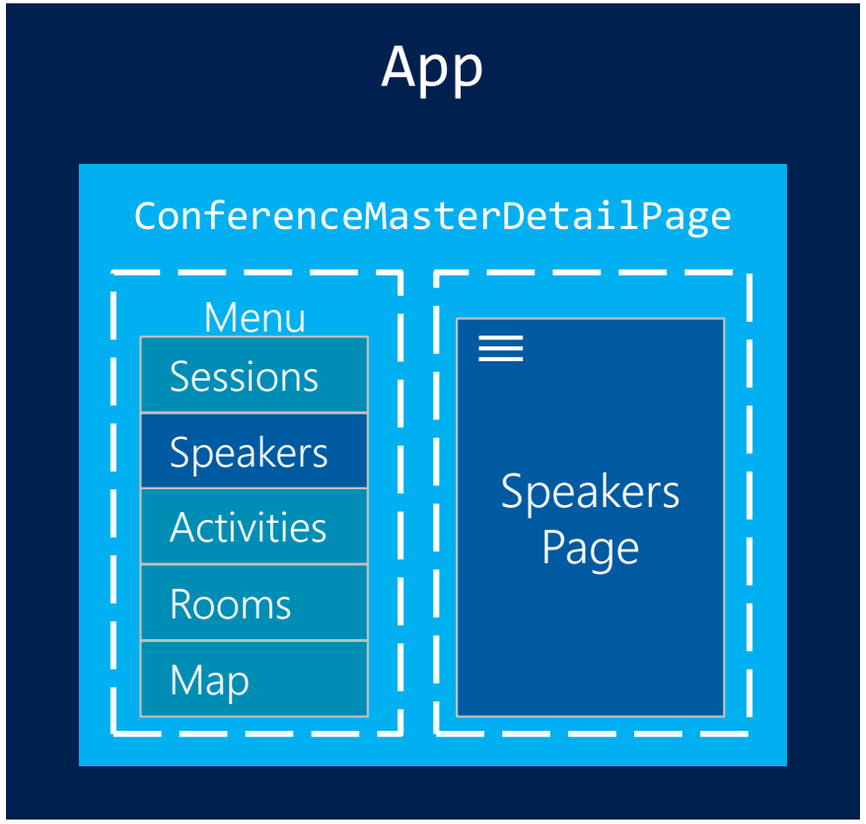
Exercise

Define the UI for drawer navigation

Detail-page update responsibility

You must write code to update the page displayed in the **`Detail`** area

1. User selects a menu item in **`Master`**

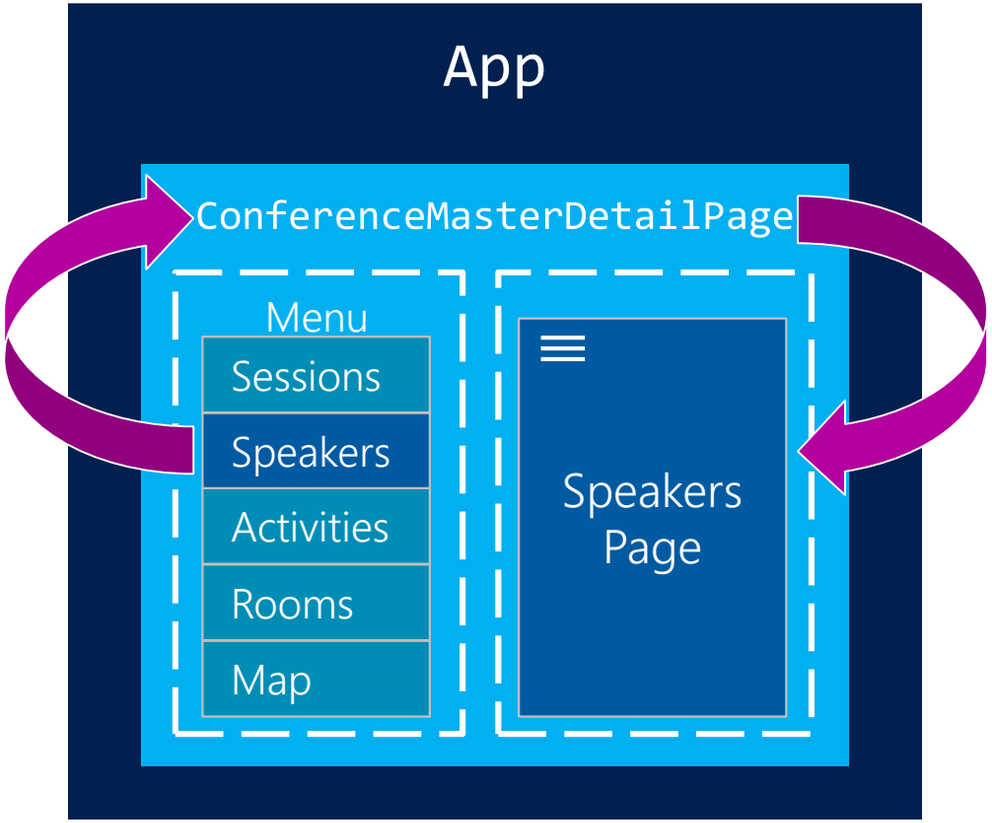


2. You swap your pages in the **`Detail`** area

Coordinating page

Typically, your master-detail page coordinates between **`Master`** and **`Detail`**

`Master` notifies your page of User selection

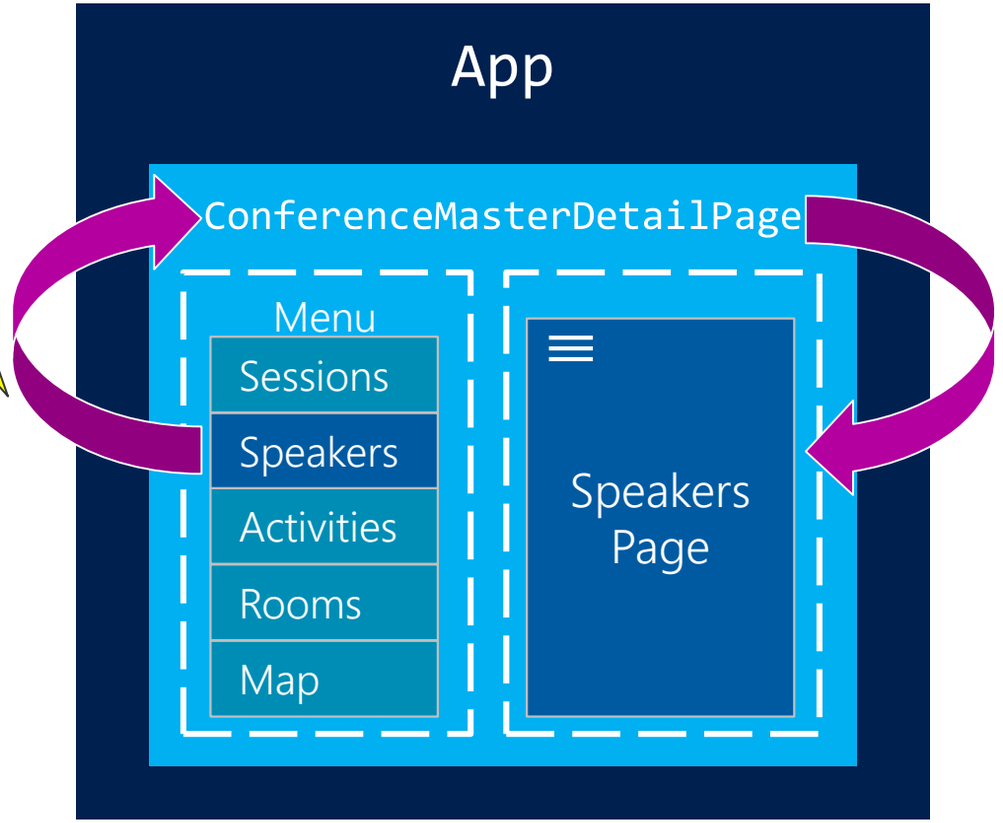


Your page updates **`Detail`** page

Minimize coupling

Typical to use an event to so your master-detail page does not know about the internal workings of your menu in the **`Master`** page

Publish an event, pass an enum to identify the page



Use the enum value to decide which page to display

```
public enum PageType
{
    Sessions,
    Speakers,
    Activities,
    Rooms,
    Map
}
```

How to respond to menu selection [overview]

Several steps required to update content when the user selects a menu item

1. Code an enum to describe your page types
2. Publish an event in your menu class
3. Subscribe to the menu event
4. Update `Detail` to the selected page
5. Close the drawer

How to respond to menu selection [step 1]

Code an enum to describe your page types

One value for each of your content-page types

```
public enum PageType
{
    Sessions,
    Speakers,
    Activities,
    Rooms,
    Map
}
```

How to respond to menu selection [step 2]

Publish an event in your menu class and pass the enum page-type as the event args

```
public partial class ConferenceMenuPage : ContentPage
{  ...
   public event EventHandler<PageType> PageSelected;

   public ConferenceMenuPage()
   {
      btnSessions.Clicked   += (s, e) => PageSelected?.Invoke(this, PageType.Sessions);
      btnSpeakers.Clicked   += (s, e) => PageSelected?.Invoke(this, PageType.Speakers);
      btnActivities.Clicked += (s, e) => PageSelected?.Invoke(this, PageType.Activities);
      btnRooms.Clicked      += (s, e) => PageSelected?.Invoke(this, PageType.Rooms);
      btnMap.Clicked        += (s, e) => PageSelected?.Invoke(this, PageType.Map);
   }
}
```

Raise the event when the user makes a menu selection

Identify the page the user selected

How to respond to menu selection [step 3]

Subscribe to the menu event in master-detail class

Subscribe to be notified when the user selects a menu item

```
public class ConferenceMasterDetailPage : MasterDetailPage
{  ...
   public ConferenceMasterDetailPage()
   {
      var master = new ConferenceMenuPage();

      master.PageSelected += OnPageSelected;

      this.Master = master
      ...
   }
}
```

How to respond to menu selection [step 4]

Update **`Detail`** to the selected page based on the enum value

Create the page based on the user's selection

Display the new page in the **`Detail`** area

```
public class ConferenceMasterDetailPage : MasterDetailPage
{  ...
   void OnPageSelected(object sender, PageType pageType)
   {
      Page page;

      switch (pageType)
      {
         case PageType.Sessions:   page = new SessionsPage(); break;
         case PageType.Speakers:   page = new SpeakersPage(); break;
         case PageType.Activities: page = new ActivitiesPage(); break;
         case PageType.Rooms:      page = new RoomsPage(); break;
         case PageType.Map:        page = new MapPage(); break;
      }

      Detail = new NavigationPage(page);

      ...
   }
}
```

Close the drawer [step 5]

If you are displaying the master page using popover then you need to hide the **`Master`** (or drawer) after the user makes a selection

Setting **`IsPresented`** will throw an exception for some values of **`MasterBehavior`** if the UI is split

```
void PresentDetailPage(PageType pageType)
{
    ...

    Detail = new NavigationPage(page);

    try
    {
        IsPresented = false;
    }
    catch { }
}
```

Exercise

Respond to drawer-menu selection to update your UI

Display a collection using master-detail split view

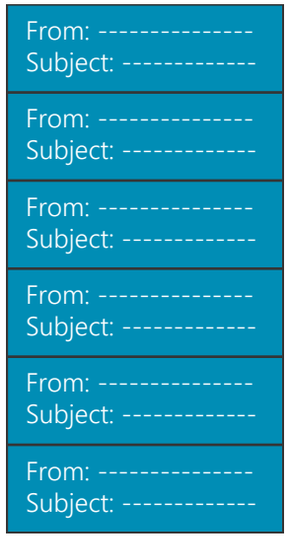
Tasks

1. Create a container page by deriving from `MasterDetailPage`
2. Create a master page using a list
3. Create one detail page
4. Update the detail page when the user selects an item in the master

Motivation

Apps often need to display a collection of homogenous data and let the user examine the details of each item

Contacts

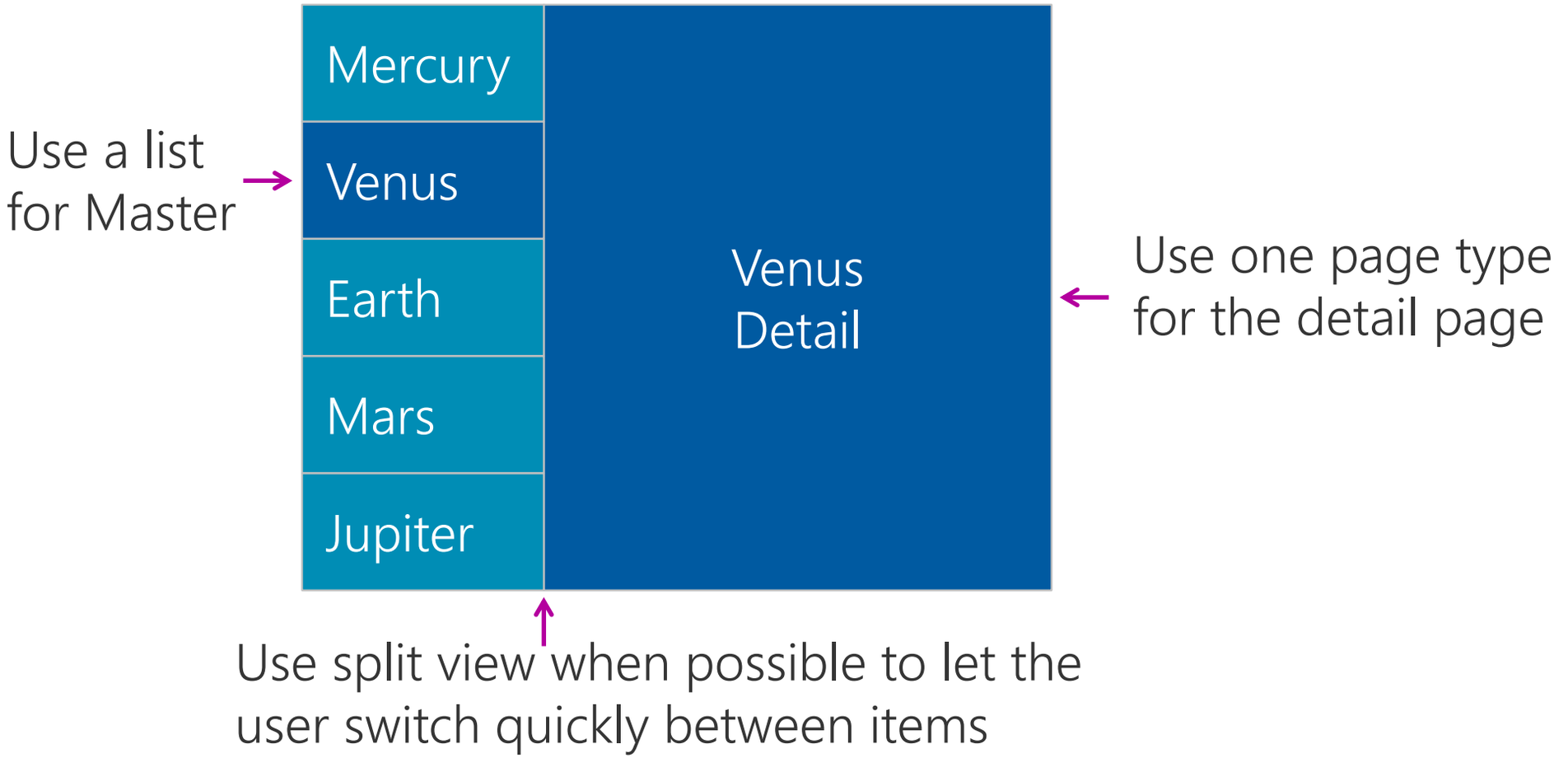


Inbox

Astronomy

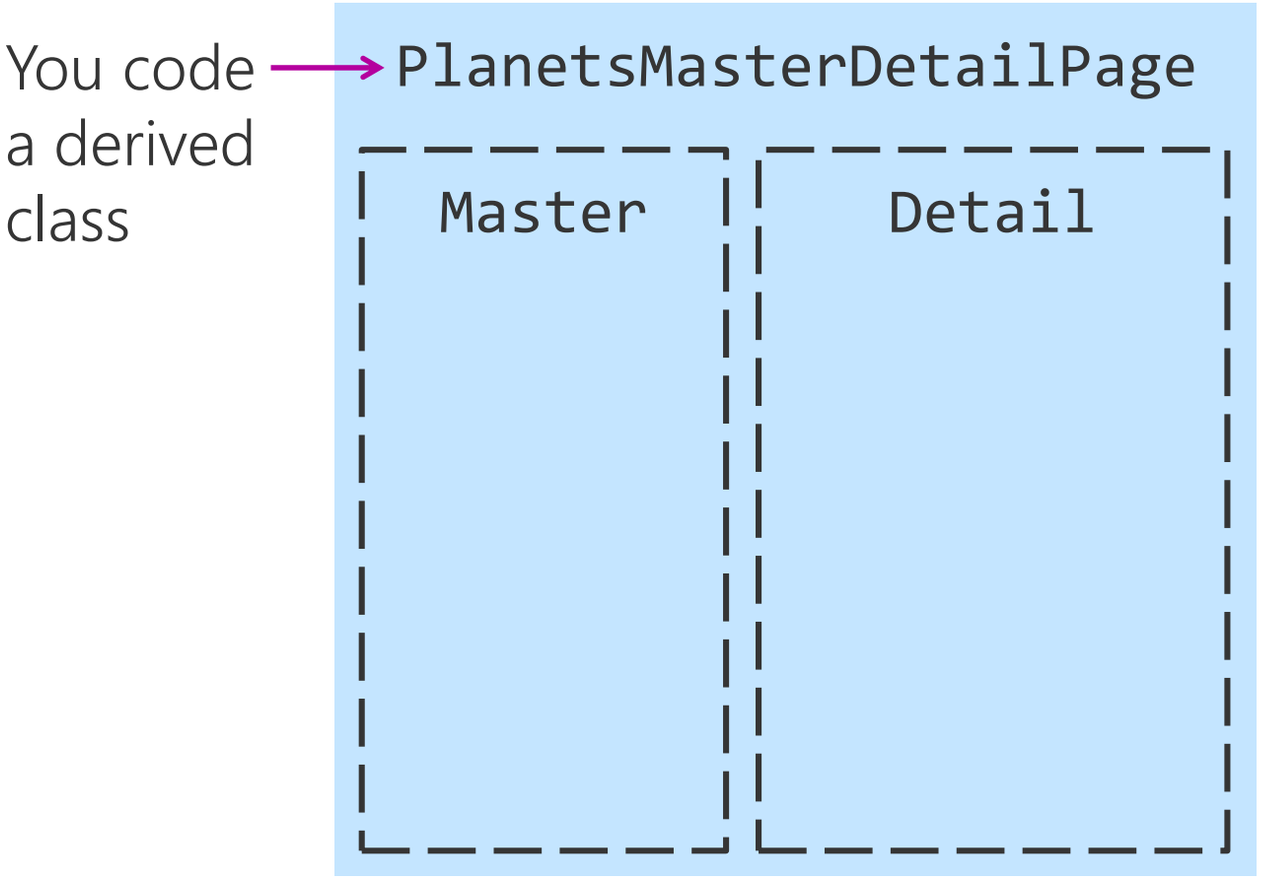
Characteristics of master-detail

Master-detail display is optimized to let the user browse a collection



Master-detail using MasterDetailPage

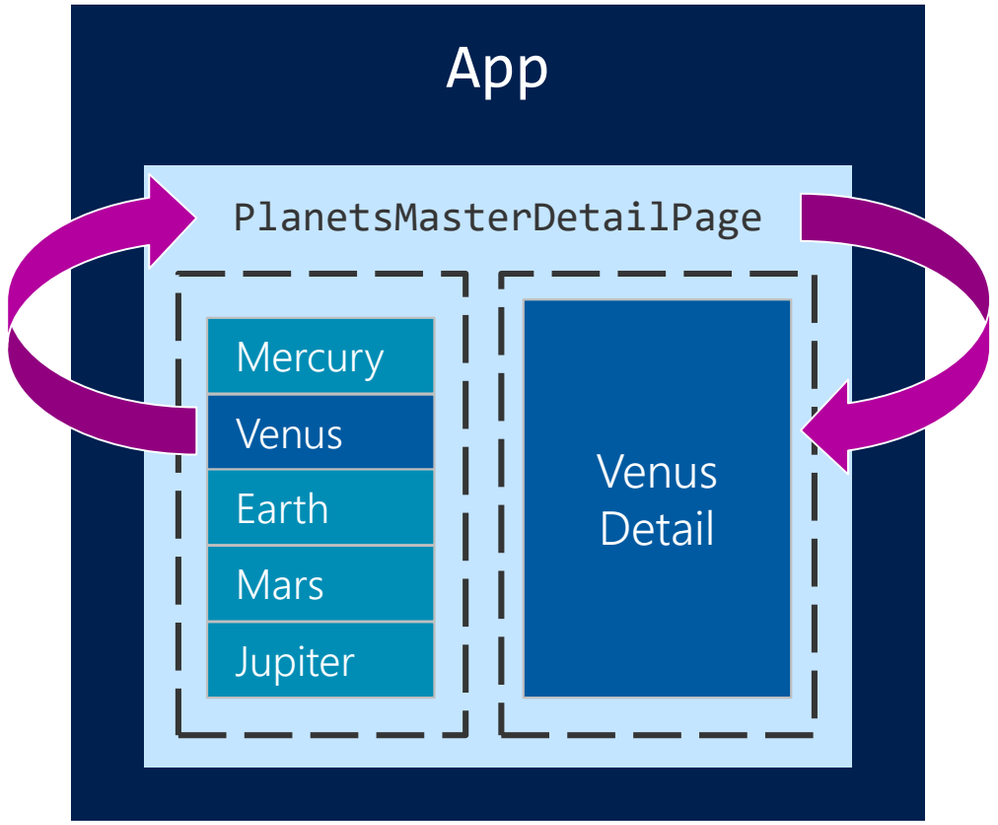
It is common to derive from **`MasterDetailPage`** to implement master-detail which provides a convenient location for your UI-update logic



Coordinating page

Your master-detail page coordinates between **`Master`** and **`Detail`**

`Master` notifies your page of the user's selection

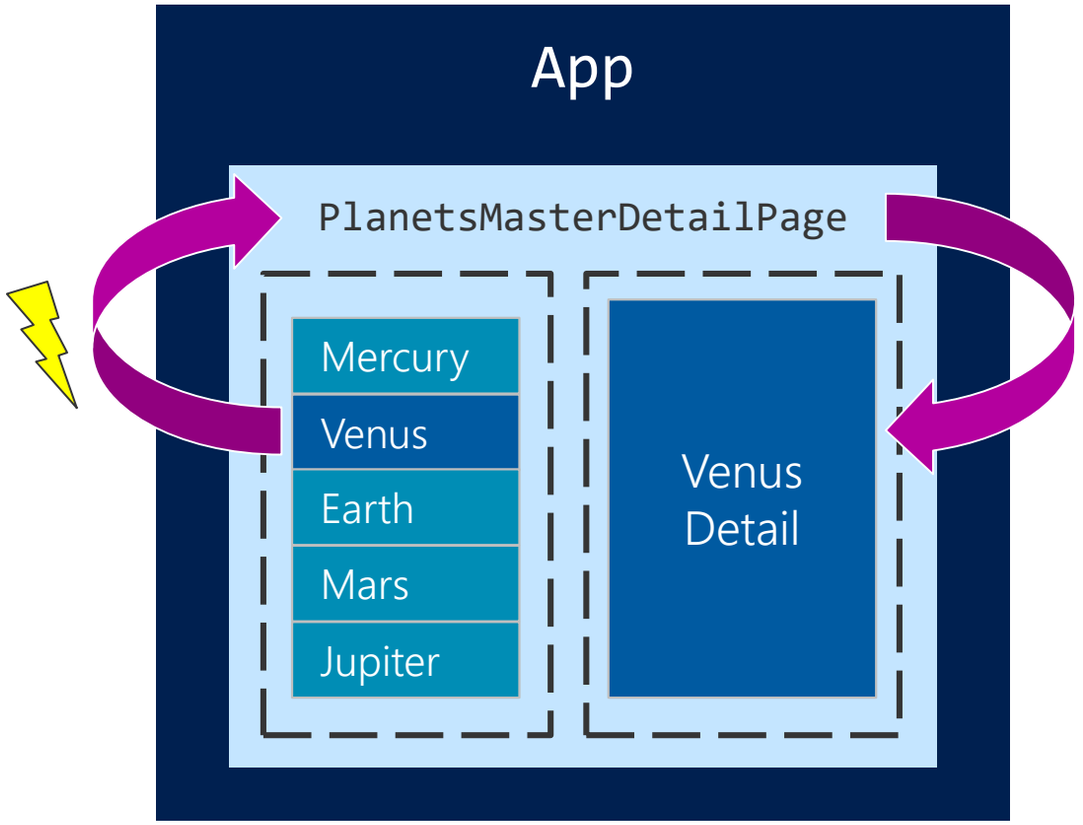


Your page updates the item shown in the **`Detail`** area

Minimize coupling

It is good practice to use an event to so your master-detail page does not know about the internal workings of your Master page

Publish an event, pass an ID to identify the user's selection



Use the ID to decide which item to display in the detail area

How to code master-detail [overview]

Several steps are required to implement master-detail UI and behavior

1. Create the Master UI
2. Code the Master selection event
3. Create the Detail UI
4. Code the Detail data-loading
5. Derive from `MasterDetailPage`
6. Choose split `MasterBehavior`
7. Set your app's `MainPage`

Create the Master UI [step 1]

The master view is a **`Page`** that shows an overview of the collection

`Title` must be set or a runtime exception

```
<?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="Minutes.PlanetsMasterPage"
             Title="Planets">

   <ListView x:Name="masterList" />

</ContentPage>
```

Common to use a **`ListView`** to show the collection

Code the master selection event [step 2]

The master raises an event when the user selects an item

Define an event

```
public partial class PlanetsMasterPage : ContentPage
{
    public event EventHandler<int> MasterItemSelected;

    public PlanetsMasterPage()
    {  ...
       masterList.ItemsSource = PlanetData.Planets;

       masterList.ItemTapped += OnMasterItemTapped;
    }

    void OnMasterItemTapped(object sender, ItemTappedEventArgs e)
    {
       MasterItemSelected?.Invoke(this, ((Planet)e.Item).Id);
    }
}
```

Raise the event

Pass identity of selected item

Create the Detail UI [step 3]

The detail view is a **Page** that shows an expanded view of a single item

```
<?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="Minutes.PlanetsDetailPage">

   <StackLayout>
      <Label x:Name="nameLabel" />
      <Label x:Name="diameterLabel" />
   </StackLayout>

</ContentPage>
```

Show multiple properties of one item

Code the detail data-loading [step 4]

The detail **Page** code-behind populates the UI with item data

```
public partial class PlanetsDetailPage : ContentPage
{
    public PlanetsDetailPage(int id)
    {   ...
        var planet = PlanetData.GetById(id);

        nameLabel.Text = planet.Name;
        diameterLabel.Text = planet.Diameter + " km";
    }
}
```

Use the identifier to lookup the item to display

Show item data in the UI

Derive from MasterDetailPage [step 5]

Your **`MasterDetailPage-`**derived class coordinates between **`Master`** and **`Detail`**

```
class PlanetsMasterDetailPage : MasterDetailPage
{
    public PlanetsMasterDetailPage()
    {
        var master = new PlanetsMasterPage();
        master.MasterItemSelected += OnMasterItemSelected;

        this.Master = master;
        this.Detail = new PlanetsDetailPage(0);

        this.MasterBehavior = MasterBehavior.Split;
    }

    void OnMasterItemSelected(object sender, int id)
    {
        this.Detail = new PlanetsDetailPage(id);
    }
}
```

Choose split master behavior [step 6]

Choose one of the split options for your master-detail display

These are most common for master-detail

	Portrait	Landscape
Default	popover	split
Popover	popover	popover
Split	split	split
SplitOnLandscape	popover	split
SplitOnPortrait	split	popover

Set your App's MainPage [step 7]

Use your **`MasterDetailPage`** derived type as your app's **`MainPage`**

```
public partial class App : Application
{
    public App()
    {  ...
       MainPage = new PlanetsMasterDetailPage();
    }
}
```

Flash Quiz

Flash Quiz

1. When using split UI behavior with `MasterDetailPage`, we're required to wrap the detail page in a `NavigationPage`
   a) True
   b) False

Flash Quiz

1. When using split UI behavior with `MasterDetailPage`, we're required to wrap the detail page in a `NavigationPage`
   a) True
   b) False

Flash Quiz

2. If our application implements **all** supported Xamarin.Forms head project types, we only need to provide a menu icon for the iOS project
   a) True
   b) False

Flash Quiz

2. If our application implements **all** supported Xamarin.Forms head project types, we only need to provide a menu icon for the iOS project
   a) True
   b) False

Exercise

Display a master-detail split view of a collection

Summary

1. Choose between *split* and *popover* for your master view
2. Display a collection using master-detail split view
3. Switch between pages using popover drawer navigation

Microsoft

Thank You!

Please complete the class survey in your profile:

university.xamarin.com/profile