Chương 3

Thiết kế giao diện người dùng (UI) sử dụng bố cục (Layout)

Một Layout là một bệ chứa các điều khiển, hình ảnh, nội dung, and những Layout khác. Trọng tâm trong việc tạo UI di động, bố cục giúp chúng tôi thiết kế các trang của mình bằng cách tạo điều kiện cho việc đặt các chế độ xem và bố cục lồng nhau (để có được giao cái nhìn trực quan hơn). Nếu bạn đã làm quen với phần tử HTML <div>, <table>, hoặc <form> , thì sẽ có thể hình dung dễ hơn, quen thuộc hơn. Mục đích của layout là chỉ ra ra vị trí và kích thước của từng thành phần con của nó. Điều này thường được thực hiện theo ba cách: liên quan đến các điều khiển riêng lẻ trong bố cục, liên quan đến nguồn gốc của bố cục hoặc sử dụng cấu trúc phủ như lưới. Mỗi kiểu bố trí có một cơ chế để đặt các khung nhìn con bên trong nó, chỉ định kích thước và vị trí của từng khung nhìn và tạo không gian giữa và xung quanh các khung nhìn.

Trong chương này, bạn sẽ xây dựng các dự án nhỏ để làm việc với từng loại bố cục và tính năng của chúng. Trước tiên, bạn sẽ tìm hiểu về các loại bố cục khác nhau và khám phá các điều khiển tùy chỉnh. Dưới đây là tổng quan về các loại.

**Xamarin.Forms Layouts**

Bố cục Xamarin.Forms kế thừa từ lớp View và có thể chứa các khung nhìn hoặc các bố cục khác. Bố cục Xamarin.Forms bao gồm:

* StackLayout: Xếp chồng các khung nhìn con theo chiều dọc hoặc chiều ngang
* FlexLayout: Bao lấy hoặc sắp xếp các khung nhìn con với sự biện minh, căn chỉnh,
* tăng trưởng theo tỷ lệ và đặt hàng
* Grid: Tạo một thùng chứa giống như bảng với các hàng và cột để giữ chế độ xem
* RelativeLayout: Sử dụng các ràng buộc tạo mối quan hệ giữa các yếu tố để xác định vị trí và kích thước của phần tử con.
* AbsoluteLayout: Đặt vị trí và kích thước khung nhìn con bằng cách sử dụng các hình chữ nhật hoặc tỷ lệ giới hạn cho bố cục tổng thể
* Frame: Vẽ đường viền giống như khung xung quanh container

**Sử dụng Xamarin.Forms Layouts**

Bố cục trong Xamarin.Forms là các thùng chứa và định dạng các khung nhìn. Mỗi bố cục

có các ràng buộc và hành vi riêng để phù hợp với một loạt các nhu cầu thiết kế. Bạn có thể định dạng các trang đơn giản với một vài điều khiển nhanh chóng và dễ dàng bằng cách sử dụng *StackLayout*.

Hãy thử *FlexLayout* khi bạn cần bố trí chất lỏng với quyền kiểm soát gói, thứ tự và

việc mở rộng và căn chỉnh các khung nhìn. *RelativeLayout* rất hữu ích khi bạn biết mối quan hệ phối hợp giữa các điều khiển. Sử dụng *AbsoluteLayout* khi bạn chỉ biết trong đó góc phần tư và khu vực của trang mà các điều khiển của bạn sẽ xuất hiện và khi bạn cần xếp lớp. *Grid* cung cấp một thùng chứa giống như bảng. *ContentView* là lớp cơ sở để xây dựng chế độ xem bố cục tùy chỉnh, chẳng hạn như điều khiển tùy chỉnh, có thể chứa nhiều bố cục

và các quan điểm khác, hữu ích như là một thành phần có thể tái sử dụng. Bố cục khung cung cấp một khung hình chữ nhật có thể nhìn thấy xung quanh nội dung của nó.

Bố cục Xamarin.Forms đơn giản nhất là *StackLayout*

**StackLayout**

Các khung nhìn trong StackLayout được xếp chồng lên nhau theo chiều dọc trừ khi vị trí nằm ngang được chỉ định. StackLayout là một bố cục nhanh, lỏng, hữu ích cho việc tạo mẫu và màn hình đơn giản. Bạn thêm khung hình nhỏ vào khung hình cha và sắp xếp chúng bằng cách sử dụng HorizontalOptions và VerticalOptions, cũng có thể được sử dụng để mở rộng chế độ xem và cung cấp khoảng cách giữa các chế độ xem. Hữu ích cho tất cả các bố cục Xamarin.Forms, thuộc tính Padding tạo không gian xung quanh các cạnh của toàn bộ bố cục.

Thêm cặp thẻ StackLayout vào ContentPage của bạn như thế này:

<ContentPage>

<ContentPage.Content>

<StackLayout >

<!-- Add Views Here -->

</StackLayout>

</ContentPage.Content>

</ContentPage>

Liệt kê 3-1 là định nghĩa về các khung nhìn con trong StackLayout mà không có khoảng cách giữa các khung nhìn. Hướng mặc định là Dọc, nghĩa là các khung nhìn xếp chồng lên nhau. Thuộc tính Spaces tạo phần đệm có kích thước được chỉ định giữa mỗi chế độ xem.

Liệt kê 3-1. StackLayoutHTHER.xaml trong dự án LayoutExample.Xaml

<StackLayout Spacing="0">

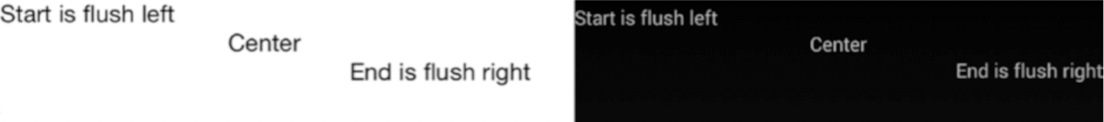
<Label Text="Start is flush left" HorizontalOptions="Start" />

<Label Text="Center" HorizontalOptions="Center" />

<Label Text="End is flush right" HorizontalOptions="End" />

</StackLayout>

Trong Hình 3-1, lưu ý HorizontalOptions cho LayoutOptions.Start, Center và End.



Hình 3-1. *StackLayout HorizontalOptions*

**Padding xung quanh tổng thể Layout**

Giống như padding trang, thuôc tính của ContentPage’s Padding tạo khoảng trống xung quanh tổng thế layout:

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="LayoutExample.Xaml.Views.StackLayoutHorizontal" Padding="10,10,10,5">

Đây là các tên tham số Padding:

Padding =” trái, tên phải , dưới”

Ví dụ dưới đây padding ở bên trái, phải, dưới nhưng không có padding ở phiá trên:

Padding=”10,0,10,5”

Ví dụ này thì khoảng trống của tất cả các phía đều bằng nhau:

Padding =”10”

Pad bên trái/phải hoặc bên trên/dưới sẽ bằng nhau với cú pháp như sau :

Padding=”10, 5”

**Xếp chồng theo hướng dọc**

Xếp chồng dọc, là hướng mặc định, đặt từng các phần tử theo thứ tự phần tử đến sau xếp chồng lên phần tử đến tr.. Khai báo VerticalOptions sử dụng FillAndExpand đệm phần cuối của bố cục với khoảng trắng, đẩy các khung nhìn khác xuống cuối trang. Ngoài ra, có bốn vị trí nằm ngang: Start, Center, End, và Fill Đây là các trường của lớp LayoutOptions.

Hãy làm cho hướng dọc mặc định rõ ràng, để bạn có thể nhìn thấy nó và thêm một vài khung nhìn vào ví dụ đầu tiên (Liệt kê 3-2).

Listing 3-2. StactLayoutVertical.xaml

<StackLayout Spacing="0" Orientation="Vertical" VerticalOptions="

FillAndExpand">

<Label Text="Start is flush left" HorizontalOptions="Start" />

<Label Text="Start 2" HorizontalOptions="Start" />

<Label Text="Center" HorizontalOptions="Center" />

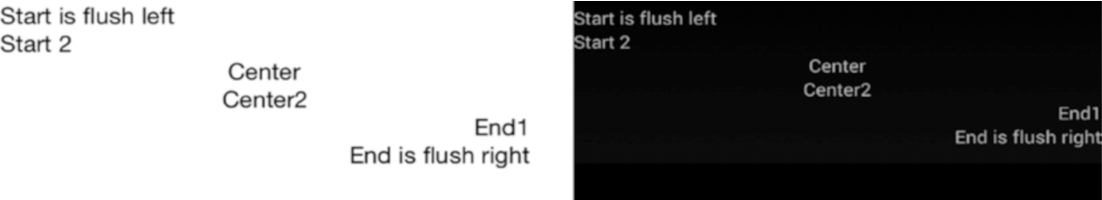
<Label Text="Center 2" HorizontalOptions="Center" />

<Label Text="End 1" HorizontalOptions="End" />

<Label Text="End is flush right" HorizontalOptions="End" />

</StackLayout>

Hình 3-2 chỉ ra cách mỗi khung hình được đặt thấp hơn sẽ có mối quan hệ theo chiều dọc and cách mà mỗi cái được cấu hình về bên lề sử dụng HorizontalOptions



***Hình 3-2.*** *Xếp chồng trên xuống dưới theo chiều dọc*

Có 4 vị trí lề: Fill. Đây là nguyên nhân khung hình thể hiện khu vực có sẵn từ trái sang phải, bắt đầu đến kết thúc:

HorizontalOptions = "Fill"

Sau đó trong phần này, chúng tôi sẽ trình bày các tùy chọn bố trí Mở rộng (như FillAndExpand), khiến cho các khung nhìn mở rộng và đệm khu vực có sẵn xung quanh khung nhìn với khoảng trống.Nếu bạn có nhiều hơn ba khung hình được định vị theo chiều ngang, thì nên sử dụng hướng ngang.

**Xếp chồng theo chiều ngangg**

Các khung nhìn có thể được xếp chồng lên nhau theo chiều ngang bằng cách đặt thuộc tính Định hướng thành Ngang, như trong Liệt kê 3-3. Tất cả các khung nhìn nằm trên cùng một trục ngang.

***Listing 3-3.*** StackLayoutHorizontal.xaml Continued

<StackLayout Spacing="0" Orientation="Horizontal">

<Label Text="Start------" />

<Label Text="------Center------" HorizontalOptions="CenterAndExpand" /> <Label Text="------End" />

</StackLayout>

Bởi vì hướng nằm ngang, nên chế độ xem đầu tiên và cuối cùng đã không yêu cầu tham số ngang.

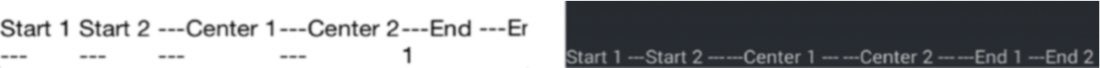
Hình 3-3 chỉ ra cách mỗi khung hình được đặt đúng vào vị trí của chúng



*Hình 3-3. Sắp xếp Từ trái sang phải theo chiều ngang*

Đệm ngang từ Bố cục mở rộng ngăn cách các khung nhìn. Đặt chế độ xem trung tâm Xếp ngang ngang dọc thành CenterAndExpand, tham số đầy đủ là LayoutOptions.CenterAndExpand, cung cấp không gian ở bên trái và bên phải của chế độ xem ở giữa.

Bạn có thể sắp xếp các chế độ xem theo chiều ngang bằng cách đặt Định hướng thành Ngang, mặc dù vị trí chính xác là không thể. Các khung nhìn được xếp chồng sang trái sang phải theo thứ tự được thêm vào bố cục, với các tín hiệu từ Ngang ngang.Hình 3-4 cho thấy StackLayout trông như thế nào nếu chúng ta thêm một vài khung nhìn vào bên phải của các khung nhìn trước đó.



Liệt kê 3-4 là mã với các khung nhìn bổ sung. Trong các ví dụ mã trực tuyến, di chuyển qua lại giữa StackLayoutHTHER.xaml, chứa các ví dụ đơn giản hơn và StackLayoutV vertical.xaml, có thêm các chế độ xem bổ sung.

***Liệt kê 3-4.*** StackLayoutV vertical.xaml với Chế độ xem bằng cách sử dụng ngang

<StackLayout Spacing="0" Orientation="Horizontal">  
<Label Text="Start 1 ---" />  
<Label Text="Start 2 ---" />  
<Label Text="---Center 1 ---" HorizontalOptions="CenterAndExpand" /> <Label Text="---Center 2 ---" HorizontalOptions="CenterAndExpand" /> <Label Text="---End 1 " />

<Label Text="---End 2 " />

</StackLayout>

Nếu như bạn muốn liên kết layout con với layout cha , nên xem xét cách tổ chức các Layouts.

**Tổ chức Layouts**

Layouts. Có thể chứa nhưng layout khác với thuộc tính Children.

Một trang phức tạp với nhiều hàng lượt xem theo chiều ngang được hoàn thành

với StackLayouts lồng nhau:

<StackLayout>

<StackLayout>

<!-- Your Views -->

</StackLayout>

<StackLayout>

<!—More of Your Views -->

</StackLayout>

</StackLayout>

Viêc điểu chỉnh kích thước khung hình trong layout and khoảng cách giữa chúng là hiệu chỉnh quan trọng nhất.

**Expanding Mở rộng và đệm khung hình bằng cách sử dụng LayoutOptions**

Sử dụng tùy chọn Layout mở rộng để khiến khung hình mở rộng hoặc đệm khu vực có sẵn với không gian. FillAndExpand khiến các khung hùnh tăng lên mà không tạo không gian đệm xung quanh chúng. Tất cả các tùy chọn mở rộng khác xung quanh khung nhìn với không gian

Bên dưới là một số lực chọn định dạng từ trái sang phải theo chiều ngang:

* **FillAndExpand** expands the view to the right:

HorizontalOptions = "FillAndExpand"

* **StartAndExpand** pads to the right with space:

HorizontalOptions = "StartAndExpand"

* **EndAndExpand** pads to the left with space:

HorizontalOptions = "EndAndExpand"

* **CenterAndExpand** pads to the left and right with space:

HorizontalOptions = "CenterAndExpand"

Bên dưới là các lựa chọn định dạng có sẵn từ trên xuống dưới theo chiều dọc

* **FillAndExpand** expands the view to the bottom:

VerticalOptions = "FillAndExpand"

* **StartAndExpand** pads to the bottom with space:

VerticalOptions = "StartAndExpand"

* **EndAndExpand** pads to the top with space:

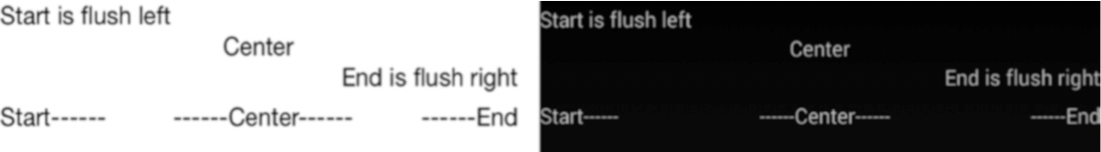
VerticalOptions = "EndAndExpand"

* **CenterAndExpand** pads to the top and bottom with space:

VerticalOptions = "CenterAndExpand"

**Code Hoàn thiện: StackLayout**

Liệt kê 3-5 hiển thị ví dụ StackLayout đầy đủ của chúng tôi với bố cục dọc và ngang, sử dụng ngang dọc và tùy chọn bố trí mở rộng, như trong Hình 3-5.



***Hình 3-5.*** *Hai StackLayouts: một theo chiều dọc một theo chiều ngang*

**Listing 3-5.** Code hoàn thiện StactLayoutHorizontal.xaml

<?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="LayoutExample.Xaml.Views.StackLayoutHorizontal"

Padding="10,10,10,5">

<StackLayout>  
<StackLayout Spacing="0" Orientation="Vertical" VerticalOptions= "FillAndExpand">

<Label Text="Start is flush left" HorizontalOptions="Start" /> <Label Text="Center" HorizontalOptions="Center" />  
<Label Text="End is flush right" HorizontalOptions="End" />

</StackLayout>

<StackLayout Spacing="0" Orientation="Horizontal">

<Label Text="Start------" />

<Label Text="------Center------" HorizontalOptions=

"CenterAndExpand" />

<Label Text="------End" />

</StackLayout>

</StackLayout>

</ContentPage>

Bây giờ chúng ta đã hình dung được, bây hãy chuyển sang một bố cục mạnh mẽ và linh hoạt hơn: FlexLayout

**FlexLayout**

Một trong những bổ sung mới nhất cho hộp công cụ bố trí Xamarin, FlexLayout là con dao quân đội của bố cục. Xuất phát từ Mô-đun bố trí hộp linh hoạt CSS, FlexLayout hứa hẹn sự dễ dàng của StackLayout, độ chính xác của bố cục Grid và khả năng phản hồi của RelativeLayout. Được lấy cảm hứng từ CSS, nó giúp các điều khiển của bạn được sử dụng một cách linh hoạt và mượt mà trên toàn bộ màn hình của thiết bị. Cách thêm FlexLayout vào ContentPage:

<ContentPage>

<ContentPage.Content>

<FlexLayout Direction="Column"

AlignItems="Center"

JustifyContent="SpaceEvenly">

<!-- Add Views Here -->

</FlexLayout>

</ContentPage>

Vị trí khing hình trong một FlexLayout bắt đầu với trục.

**Vị trí khung hình sử dụng Axes**

Có hai trục giúp xác định hành vi của các chế độ xem trên FlexLayout: trục chính và trục chéo. Bạn đặt trục chính bằng thuộc tính Direction, theo chiều ngang (Hàng) theo mặc định trừ khi được chỉ định vị trí ngang (Cột). Khi trục chính của bạn được đặt, bạn có thể định vị các chế độ xem của mình bằng JustifyContent và AlignItems:

* **JustifyContent** sử dụng trục chính để đặt nơi bắt đầu các chế độ xem, như Bắt đầu, Trung tâm và Kết thúc hoặc chỉ đơn giản là cung cấp khoảng cách giữa các chế độ xem.
* **AlignItems** sử dụng trục chéo để sắp xếp các đỉnh, đáy hoặc tất cả các mặt của chế độ xem của bạn.

Tạo một trang XAML mới và đặt tên là FlexLayoutExample. Thêm một FlexLayout

với Hướng được đặt thành Cột hoặc dọc. Đặt AlignItems thành Center để các chế độ xem sẽ được tập trung từ trái sang phải và đối với JustifyContent, hãy sử dụng SpaceEvenly, hoạt động giống như FillAndExpand để phân phối các chế độ xem của bạn dọc theo trục chính (dọc ngay bây giờ) với kích thước và khoảng cách bằng nhau. Thêm một vài nút, như trong Liệt kê 3-6.

***Listing 3-6.*** Basic FlexLayout in FlexLayoutExample.xaml

<FlexLayout Direction="Column"

AlignItems="Center"

JustifyContent="SpaceEvenly">

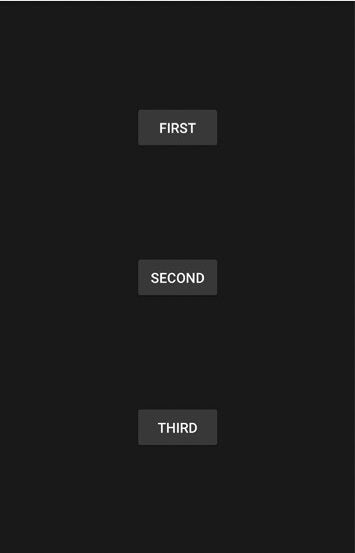
<Button Text="First" />

< Button Text="Second" />

< Button Text="Third" />

</FlexLayout>

Sau khi chạy bạn sẽ nhận thấy được các thành phần sắp xếp như hình 3-6



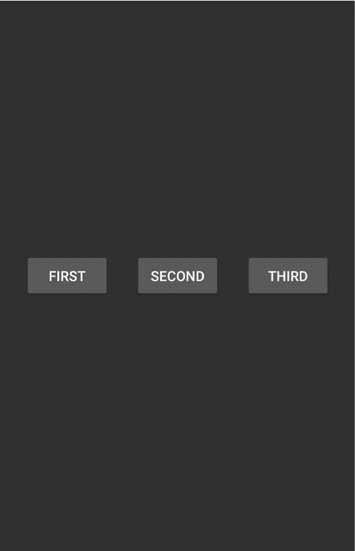
***Hình 3-6.*** *FlexLayout với Direction cài đặt theo chiều dọc*

Khung hình theo chiều dọc được cài đặt bằng việc thay đổi thuộc tính Direction của dòng:

<FlexLayout Direction="Row" AlignItems="Center"

JustifyContent="SpaceEvenly">

Từ trái sang phải theo trục chính được chỉ ra ở hình 3-7.



***Hình 3-6.*** *FlexLayout với Direction cài đặt theo chiều ngang.*

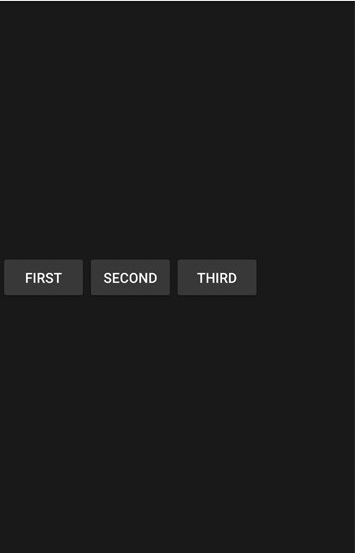
Đặt khung hình của bạn vào vị trí bắt đầu bằng việc sử dụng thuộc tính JustifyContent :

<FlexLayout Direction="Row"

AlignItems="Center"

JustifyContent="Start">

Với việc cài đặt Row, cài đặt JustifyContent là Start thì tất cả các khung hình sẽ chuyển sang bên trái, như hình 3-8.



***Hình 3-6.*** *FlexLayout với Direction cài đặt vị trí bắt đầu.*

Dịch chuyển chúng qua phải bằng việc cài đặt JustifyContent là End, hoặc căn giữa bằng giá trị Center. Từ khi AlightItems làm việc với trục chéo, nó được mặc định là dọc, bạn có thể điều chỉnh nó bắt đầu từ phía trên khung hình với khoẳng trống giữa chúng (hình 3-9)

<FlexLayout Direction="Row"

AlignItems="Start"

JustifyContent="Start">



***Hình 3-9.*** *FlexLayout với AlignItems cài đặt vị trí bắt đầu..*

**FlexLayout Patterns**

FlexLayout rất linh hoạt. Dưới đây là một số cách bạn có thể sử dụng nó:

* **Stack**: Tạo một ngăn xếp cơ bản, dọc hoặc ngang, giống như StackLayout. Trong Liệt kê 3-6, chúng ta đặt Direction thành Cột và xếp chồng một số điều khiển theo chiều dọc. Điều khiển hướng cho ngăn xếp hàng theo chiều ngang.
* **Wrap**: Định dạng nhiều mục như hình ảnh bằng cách gói chúng khi chúng chạm tới mép màn hình. Sử dụng thuộc tính Wrap của FlexLayout, đặt nó thành Wrap (NoWrap là mặc định). Đặt FlexLayout trong ScrollView để người dùng có thể cuộn qua các mục được bọc.
* **Catalog**: Tạo các bảng đóng gói dữ liệu tĩnh để cuộn qua và chọn. Đặt FlexLayout trong ScrollView và điền nó với Khung (xem phần sau trong chương này) có chứa chi tiết và có thể là hình ảnh. Danh sách khung tĩnh này sẽ cuộn theo bất kỳ Hướng nào bạn đặt nhưng không thể liên kết như ListView.
* **Page**: Lồng hai FlexLayouts để tạo một trang có tiêu đề, chân trang, nội dung và lề bên. Mở rộng phần nội dung theo chiều dọc bằng cách sử dụng thuộc tính FlexLayout.Grow (đặt thành 1) trên chế độ xem nội dung của bạn, hoạt động giống như thông số sao trong bố cục Lưới. Trong chế độ xem lề bên của bạn, hãy đặt FlexLayout.Bocation thành chiều rộng pixel mong muốn (ví dụ: 20). Lưu ý rằng Phát triển và Cơ sở được đặt trên các chế độ xem, không phải trên FlexLayout (ví dụ: <Lable Text = "Content" FlexLayout.Grow = "1" />).

**Grid (lưới)**

Lưới chứa các khung nhìn giống như bảng. Nó được tổ chức thành các hàng và cột, mỗi cột có chiều cao và chiều rộng, được đặt tại các tọa độ hàng / cột cụ thể được gọi là các ô. GridUnitType cung cấp các tùy chọn để định cỡ các hàng và cột, trong khi phương thức Grid.Children.Add cho phép cả các chế độ xem một ô và đa ô. ColumnSpaces và RowSpaces cung cấp phần đệm giữa các ô.

Tạo một đối tượng Grid và định nghĩa từng dòng và cột , được chỉ ra như hình 3-7.

***Listing 3-7.*** Starting GridExample1.xaml

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

</Grid>

Chiều cao cụ thể từng RowDefinition, và chiều rộng mỗi ColumnDefinition. Tự động thay đổi kích thước dòng cho mỗi chiều cao và chiều rộng .

Thêm một khung hình tại dòng và cột 0, chỉ một ô trong bảng:

<Label Text="I'm at 0,0" FontSize="30" FontAttributes="Bold" />

Bây giờ hãy chia bẳng này thành 4 dòng và 3 cột:

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="Auto" />

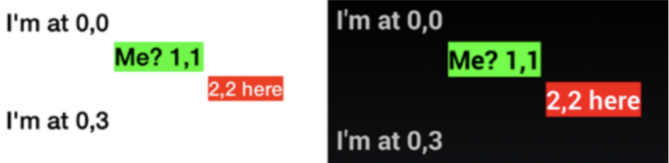
</Grid.ColumnDefinitions>

</Grid>

Sau đó thêm một khung hình vào (1,1),(2,2) và (0,3). Thêm một Lable để làm nó trở nên thú vị hơn:

<Label Text="I'm at 0,0" FontSize="30" FontAttributes="Bold" />  
<Label Text="Me? 1,1" FontSize="30" FontAttributes="Bold" TextColor="Black" BackgroundColor="Lime" Grid.Row="1" Grid.Column="1" /> <Label Text="2,2 here" FontSize="30" FontAttributes="Bold" TextColor="White" BackgroundColor="Red" Grid.Row="2" Grid.Column="2" /> <Label Text="I'm at 0,3" FontSize="30" FontAttributes="Bold" Grid. Row="3" Grid.Column="0" />

Figure 3-10 shows our Grid with four labels, completing the code in GridExample1.cs.



***Hình 3-10****. Grid chứa 4 khung hình*.

Nó sẽ tự động thay đổi độ rộng cột và chiều cao hàng cho chúng ta, phù hợp với nội dung bên trong.

Để làm theo cùng với các ví dụ trực tuyến, hãy lưu ví dụ hiện tại của bạn, GridExample1. cs, và sau đó tạo một lớp trang XAML được gọi là GridExample2. Sao chép toàn bộ Lưới của bạn từ GridExample1.xaml vào GridExample2.xaml (chú ý không sao chép dòng đầu tiên của

khai báo ContentPage) và tiếp tục làm việc với GridExample2. Nhớ lại

để cập nhật lớp ứng dụng của bạn (chẳng hạn như App.cs) với tham chiếu MainPage mới cho GridExample2.

**Sizing Rows and Columns**

Kích thước của các hàng và cột được xác định bởi GridLpm. Bạn có thể tự động hóa, mở rộng hoặc đặt chiều cao và chiều rộng cụ thể trên các hàng hoặc cột. GridLemgth được định nghĩa bởi GridUnitType, trong đó có ba:

* **Auto** kích thước kích thước của một hàng hoặc cột theo nội dung của nó.
* **Absolute** chỉ ra kích thước số của hàng hoặc cột. Sử dụng XAML, GridUnitType.Absolute được ngụ ý bằng cách sử dụng một giá trị số.
* **Star** (“ \* “) là cài đặt mặc định, mở rộng kích thước của một hàng hoặc cột vào khoảng trống còn lại sau khi Tuyệt đối và Ô tô đã được phân bổ.

Gán một đối tượng GridLength tuyệt đối cho Chiều cao trong RowDefinition:

<RowDefinition Height="200" />

or a Star to Width in ColumnDefinition:

<ColumnDefinition Width="1\*" />

**Sizing to Fit Views**

Giá trị tự động của GridUnitType kích thước hàng hoặc cột theo kích thước của các khung nhìn được chứa. Ví dụ Grid của chúng tôi được tạo thành hoàn toàn từ các hàng có kích thước Tự động:

<RowDefinition Height="Auto" />

Bây giờ thêm Absolute vào Stat.

**Setting Exact Size**

Giá trịn Absolute của GridUnitType thiết đặt chiều cao và độ rộng chỉnh xác cho dòng và cột. thay đổi giá trị chiều cao ở dòng thứ 2 trở thành kích thước tuyệt dối 200:

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

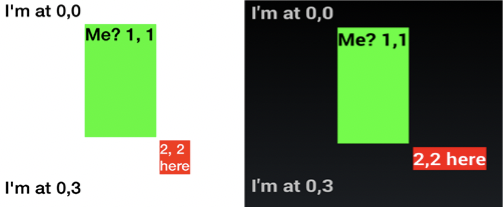
**<RowDefinition Height="200" />**

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

Dòng thứ 2 được cài đặt một chiều cao tuyệt đối, mô tả theo chiều cao, hình 3-11 chỉ ra điều này. Đoạn code này ở GridExample2.xaml.



Hình 3-11. Sử dụng GridUnitType.Absolute để thiết đặt chiều cao dòng cột tại vị trí (1,1)

Width assigned using GridUnitType.Absolute works in a manner similar to the

Height assignment:  
<ColumnDefinition Width="200" />

**Mở rộng khung hình vừa với khoảng có sẵn**

GridUnitType.Star, hoặc “\*”, cài đặt mặc định cho cả chiều cao và chiều rộng. Mở rộng một khung hình với các dòng hoặc các cột để làm đầy khoảng trống còn lại. Điều này rất hữu ích để lấp đầy màn hình theo chiều ngang với các cột hoặc theo chiều dọc với các hàng đến cạnh màn hình, đặc biệt là khi các chế độ xem nhỏ. Nó hoạt động tương tự như tùy chọn bố trí FillAndExpand, chèn không gian đệm vào hàng hoặc cột được chỉ định.

Mở rộng theo chiều dọc bằng cách đặt Chiều cao của RowDefDef. Thay đổi RowDefDef thứ ba để sử dụng Star.

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

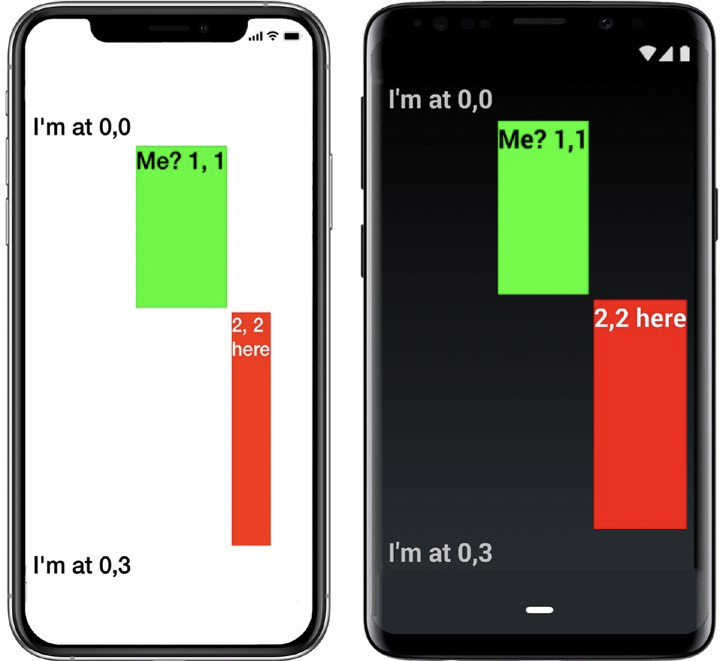
<RowDefinition Height="200" />

**<RowDefinition Height="1\*" />**

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

GridUnitType.star mở rộng để đẩy hàng bên dưới xuống hết cỡ của màn hình, như trong Hình 3-12. Hãy nhớ rằng trong nhiều ví dụ này, thuộc tính Padding đang được sử dụng để tạo khoảng trống xung quanh các cạnh bên ngoài của trang (xem Liệt kê 3-8). Điều này hoàn thành mã trong GridExample2.xaml.



***Hình 3-12****. Lấp đầy không gian dọc có sẵn bằng cách sử dụng GridUnitType.star trong Chiều cao*

Mở rộng theo chiều ngang bằng cách đặt chiều rộng của Cột xác định. Chiều rộng được gán bằng GridUnitType.star hoạt động theo cách tương tự như gán Chiều cao, mở rộng cột sang bên phải để tiêu thụ khoảng trống còn lại ở cạnh phải của màn hình:

<ColumnDefinition Width="1\*" />

**Mở rộng khung hình theo tỉ lệ**

Bạn có thể kiểm soát tỷ lệ kích thước ô với nhau trong các ô GridUnitType.Star bằng cách sử dụng tham số đầu tiên trong GridLength. Kỹ thuật này đặc biệt hữu ích với các góc nhìn nhỏ.Trong ví dụ đầu tiên này, các tham số đầu tiên trong tất cả các hàng cộng tối đa 3 (1 + 2), chia hàng thành ba phần bằng nhau. Điều này dẫn đến hàng đầu tiên mở rộng đến một phần ba không gian và hàng thứ hai mở rộng đến hai phần ba không gian:

<RowDefinition Height="1\*" />

<RowDefinition Height="2\*" />

Trong ví dụ tiếp theo này, các tham số đầu tiên trong tất cả các hàng cộng tối đa 4 (1 + 3), chia hàng thành bốn phần bằng nhau. Điều này dẫn đến hàng đầu tiên mở rộng đến một phần tư không gian và hàng thứ hai mở rộng đến ba phần tư không gian:

<RowDefinition Height="1\*" />

<RowDefinition Height="3\*" />

Tham số đầu tiên biểu thị một phần của tổng không gian giữa các hàng hoặc cột Sao. Nếu tất cả các tham số này được chỉ định là 1, không gian sẽ được chia đều.

**Tạo khung hình với nhiều ô**

Các chế độ xem đơn có thể được định kích thước để trải rộng nhiều ô trong lưới bằng cách sử dụng RowSpan và thuộc tính ColumnSpan. Các thuộc tính này đặt số lượng ô mà một chế độ xem có thể chiếm.

**Spanning Columns**

Mở rộng chế độ xem từ trái sang phải qua nhiều cột bằng cách sử dụng tham số thứ hai và thứ ba của phương thức thêm (Add), indexLeftColumn và indexRightColumn, để chỉ định các cột cần mở rộng. Để làm theo cùng với các ví dụ trực tuyến, hãy lưu ví dụ hiện tại của bạn, GridExample2, sau đó tạo một trang XAML mới có tên là GridExample3. Sao chép XAML từ GridExample2.xaml vào GridExample3.xaml và tiếp tục với GridExample3.xaml. Hãy nhớ cập nhật App.cs với tham chiếu MainPage mới lên GridExample3.

Hãy để mở rộng khung nhìn (1,1) của chúng tôi vào cột bên phải. Thêm một chiều rộng được đánh dấu sao vào cột thứ hai để nó sẽ mở rộng theo chiều ngang:

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

**<ColumnDefinition Width="1\*" />**

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

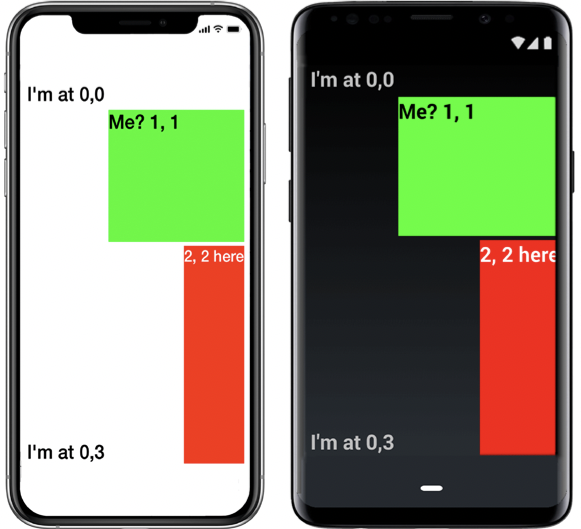
Bắt đầu với khung hình (1,1). Thay đổi khung hình label để sử dụng RowSpan và ColumnSpan, điều này bạn sẽ thiết đặt để chiếm 2 cột:

<Label Text="Me? 1,1" FontSize="30" FontAttributes="Bold" TextColor="Black" BackgroundColor="Lime" Grid.Row="1" Grid.Column="1"

**Grid.RowSpan="1" Grid.ColumnSpan="2" />**

Từ trái sang phải, chế độ xem này kéo dài từ bên trái của cột 1 sang bên trái của

cột 3 – một khoảng cách hai cột. (Vâng, nó hơi lạ một chút là không có cột 3 có thể nhìn thấy, nhưng ký hiệu yêu cầu điểm cuối và đó là điểm bắt đầu của cột hoặc hàng mà chúng tôi đang mở rộng.) Xem kết quả trong Hình 3-13.



***Hình 3-13****: Ô tại vị trí 1,1 là cột kéo dài và 2,2 hàng kéo dài*

Từ đỉnh xuống đay, khùn hình này kéo dài từ đinh cua hàng 1 đến đỉnh của hàng 2

**Spanning Rows**

Mở rộng chế độ xem từ một ô xuống qua nhiều hàng bằng cách chỉ định các hàng để mở rộng trong thuộc tính Grid.RowSpan trên chế độ xem. Bây giờ thay đổi chế độ xem tại (2,2) để chiếm hai hàng:

<Label Text="2,2 here" FontSize="30" FontAttributes="Bold" TextColor="White" BackgroundColor="Red" Grid.Row="2" Grid.Column="2

**Grid.RowSpan="2" Grid.ColumnSpan="1" />**

Từ trái sang phải, chế độ xem này kéo dài từ bên trái của cột 2 sang bên trái của cột 3. Từ trên xuống dưới, chế độ xem này kéo dài từ đầu hàng 2 đến đầu hàng 4 (khoảng cách hai hàng) và mang lại những gì bạn thấy trong hình 3-13. (Có một hàng 4 không nhìn thấy được; nó chỉ là một điểm cuối.) Điều này hoàn thành mã trong GridExample3.cs.

**Padding Between Cells**

Bạn có thể thêm không gian giữa các ô bằng cách sử dụng các thuộc tính RowSpaces và CộtSpaces của bố cục Lưới của bạn. RowSpaces cung cấp phần đệm giữa các hàng, trong khi Cột không gian cung cấp không gian giữa các cột. Dưới đây, một ví dụ cung cấp 20 đơn vị đệm cho mỗi:

<Grid ColumnSpacing="20" RowSpacing="20">

**CODE COMPLETE: Grid**

Liệt kê 3-8 là ví dụ Grid của chúng tôi được hiển thị trước đây trong Hình 3-12 thể hiện bốn nhãn, hai trong số đó trải dài trên nhiều hàng hoặc cột.

***Listing 3-8.*** GridExample2.xaml Code Complete

<?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="LayoutExample.Xaml.Views.

GridExample2" Padding="10,10,10,5">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="200" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<Label Text="I'm at 0,0" FontSize="30" FontAttributes="Bold" />

<Label Text="Me? 1,1" FontSize="30" FontAttributes="Bold"

TextColor="Black" BackgroundColor="Lime" Grid.Row="1" Grid.

Column="1" />

<Label Text="2,2 here" FontSize="30" FontAttributes="Bold"

TextColor="White" BackgroundColor="Red" Grid.Row="2" Grid.

Column="2" />

<Label Text="I'm at 0,3" FontSize="30" FontAttributes="Bold" Grid.

Row="3" Grid.Column="0" />

</Grid>

</ContentPage>

**RelativeLayout**

RelativeLayout tự động chia tỷ lệ các thành phần của nó thành các kích thước màn hình khác nhau. Được tạo thành từ khung hình bố cục chính và các khung nhìn con của nó, bố cục này được xác định bởi các mối quan hệ giữa các khung nhìn. Mỗi khung nhìn con được gắn với các khung nhìn anh chị em của nó hoặc với khung nhìn bố trí cha mẹ bằng cách sử dụng các ràng buộc. Một ràng buộc có thể liên kết các vị trí và kích thước của khung nhìn: tọa độ x / y và kích thước chiều rộng / chiều cao. RelativeLayout cho phép chúng ta tạo ra một mạng lưới các khung nhìn được kết nối với nhau như dải cao su để vừa với màn hình, cung cấp thiết kế phản hồi tích hợp hoặc bố cục tự động.

Hãy bắt đầu với một trang XAML mới, tạo một cá thể RelativeLayout và đặt nhãn ở 0,0 ở góc trên bên trái của bố cục, như trong Liệt kê 3-9.

***Listing 3-9.*** Starting RelativeLayoutExample.xaml

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="LayoutExample.Xaml.Views.RelativeLayoutExample"

x:Name="Self" Padding="10,10,10,5">

<RelativeLayout x:Name="LayoutContainer">

<Label x:Name="UpperLeft" Text="Upper Left" FontSize="20"

RelativeLayout.XConstraint="{ConstraintExpression

Type=Constant, Constant=0}"

RelativeLayout.YConstraint="{ConstraintExpression

Type=Constant, Constant=0}"/>

</RelativeLayout>

</ContentPage>

Nhãn UpperLeft được thêm vào với một ràng buộc vị trí đối với bố trí chính; sử dụng Constant = 0 cho cả x và y đặt nhãn ở góc trên bên trái, tại gốc: 0,0. Tiếp theo, chúng tôi muốn thêm nhiều lượt xem liên quan đến lượt xem cha mẹ và con hiện có.

**Cài đặt khung hình vị trí và kích thước**

Mỗi lần chúng ta thêm khung hình vào RelativeLayout, chúng ta hỏi: liệu chúng ta có muốn đặt vị trí của khung hình, kích thước của khung hình hay cả hai?

Chỉ định vị trí với XConstraint và Yconstraint:

RelativeLayout.XConstraint="{ConstraintExpression Type=Constant,

Constant=10}"

RelativeLayout.YConstraint="{ConstraintExpression Type=Constant,

Constant=25}"

Chỉ định kích thớpc với XConstraint và Yconstraint:

RelativeLayout.WidthConstraint="{ConstraintExpression Type=Constant,

Constant=50}"

RelativeLayout.HeightConstraint="{ConstraintExpression Type=Constant,

Constant=200}"

Tất cả các tọa độ x / y, chiều rộng và chiều cao này cuối cùng trở thành giá trị tuyệt đối. Tuy nhiên, việc gõ dữ liệu hạn chế chúng ta sử dụng các lớp ràng buộc. Điều này khuyến khích các tính toán dựa trên các giá trị của quan điểm anh chị em và cha mẹ, giữ cho mọi thứ tương đối.

**Using Constraints**

Kích thước và vị trí được chỉ định bằng cách sử dụng các ràng buộc. Đối tượng ràng buộc có ba bảng liệt kê:

• **Constant** , đối với các chỉ định x / y tuyệt đối về vị trí và / hoặc kích thước

• **RelativeToParent**, để tính toán tương đối x / y của vị trí và / hoặc kích thước để bố trí cha

• **RelativeToView**, để tính toán tương đối x / y về vị trí và / hoặc kích thước giữa các khung hình ( liên quan)

Các phần sau thảo luận từng chi tiết hơn.

**Kích thước và vị trí tuyệt đối**

Constant được dùng cho kích thước và vin trí tuyệt đối

Đây là ví dụ về vị trí, đặt nhãn UpperLeft ở tọa độ 0,0 trong bố cục:

<Label x:Name="UpperLeft" Text="Upper Left" FontSize="20"

RelativeLayout.XConstraint="{ConstraintExpression Type=Constant,

Constant=0}"

RelativeLayout.YConstraint="{ConstraintExpression Type=Constant,

Constant=0}" />

Đây là ví dụ về kích thước, tại một khung hình kích thước 100,100 với kích thước rộng 50 đơn vị và cao 200 đơn vị:

<Label Text="Constants are Absolute" FontSize="20" RelativeLayout.

XConstraint="{ConstraintExpression Type=Constant, Constant=100}"

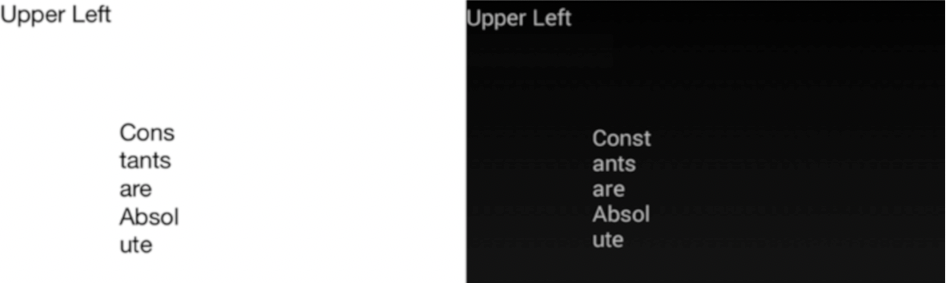
RelativeLayout.YConstraint="{ConstraintExpression Type=Constant,

Constant=100}" RelativeLayout.WidthConstraint="{ConstraintExpression

Type=Constant, Constant=50}" RelativeLayout.HeightConstraint=

"{ConstraintExpression Type=Constant, Constant=200}" />

Nhãn mới này được chỉ ra ở hình 3-14, với chữ viết trong 50 đơn vị



***Hình 3-14****. Label với một giá trị không đổi*

**Sự ràng buộc tương đối**

Ràng buộc RelativeToParent liên kết một vị trí / kích thước của chế độ xem với RelativeLayout gốc. Điều này hữu ích cho việc đặt và định cỡ các khung nhìn liên quan đến toàn bộ trang hoặc phần.

Khởi tạo một khung nhìn con khác, chẳng hạn như Lab và thêm nó vào bộ sưu tập con bằng cách sử dụng ràng buộc RelativeToParent. Ví dụ này đặt vị trí của chế độ xem con mới xuống một nửa chiều dài và chiều rộng của bố trí cha mẹ bằng cách đặt Hệ số thành 0,5:

<Label Text="Halfway down and across" FontSize="15"

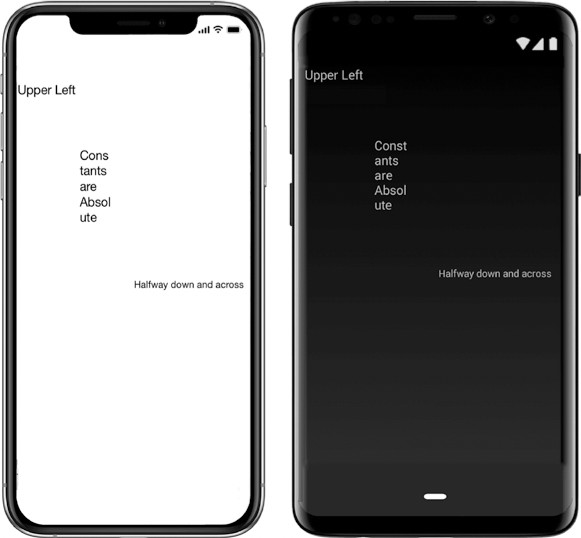
RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToParent,

Property=Width, Factor=0.5}"

RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent,

Property=Height, Factor=0.5}" />

Các phép gán Loại này cho RelativeToParent sử dụng khung nhìn cha, RelativeLayout, để trả về tọa độ x bằng một nửa chiều rộng của bố trí cha và tọa độ y bằng một nửa chiều cao của Layout cha (xem Hình 3-15).



***Hình 3-15****. RelativeToParent với một phép tính chiều cao và chiều rộng*

Tạo một BoxView một nửa trang có chiều cao bằng một nửa chiều rộng và một nửa chiều rộng của chế độ xem chính bằng cách chuyển các phép tính RelativeToParent vào các tham số Add:

<BoxView Color="Accent" HorizontalOptions="Start" VerticalOptions="Start

AndExpand"

RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToParent,

Property=Width, Factor=0}"

RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent,

Property=Height, Factor=0.5}"

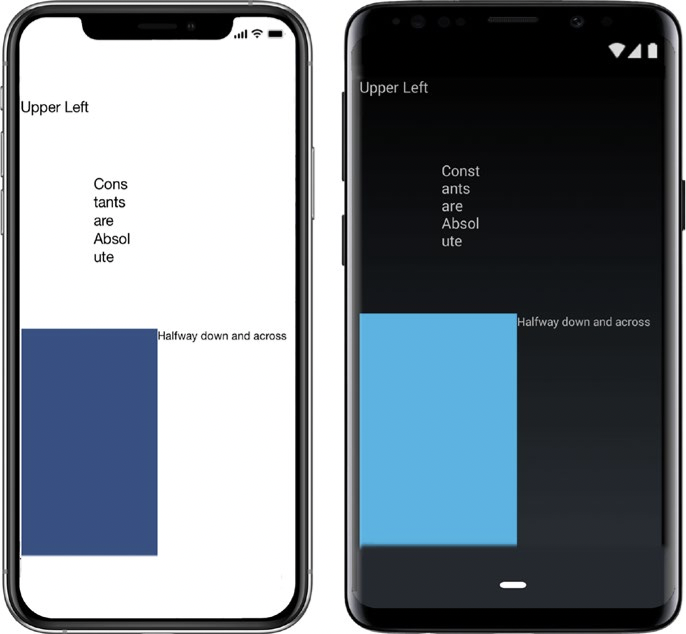
RelativeLayout.HeightConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Height, Factor=0.5}"

RelativeLayout.WidthConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Width, Factor=0.5}" />

Kết quả xem hình 3-16



***Hình 3-16****. Sử dụng RelativeToParent đặt một BoxView*

Ràng buộc RelativeToView

RelativeToView ràng buộc một chế độ xem Vị trí / kích thước của chế độ xem so với chế độ xem khác. Điều này thường được sử dụng để bù đắp số lượt xem bằng một vài pixel bằng thuộc tính Constant. Tuy nhiên, các vị trí kề nhau phức tạp hơn thường được yêu cầu, chẳng hạn như cài đặt các điều khiển bên dưới hoặc bên cạnh một vị trí khác. Mặc dù có thể đạt được một số ràng buộc phức tạp khi sử dụng XAML, nhưng nó thường dễ sử dụng C # hơn và đó là trường hợp ở đây. (Liệt kê 3-10 có thể được thực thi bằng cách sử dụng tiện ích mở rộng đánh dấu XAML, Ví dụ này đặt vị trí của chế độ xem con mới bên dưới chế độ xem anh chị em, nhưng chúng tôi làm việc trong mã C # phía sau

Liệt kê 3-10. Ràng buộc RelativeToView trong RelativeLayoutExample.cs

Label below = new Label

{

Text = "Below Upper Left",

FontSize = 15

};

LayoutContainer.Children.Add(below,

Constraint.Constant(0),

Constraint.RelativeToView(UpperLeft, (parent, sibling) =>

{

return sibling.Y + sibling.Height;

}) );

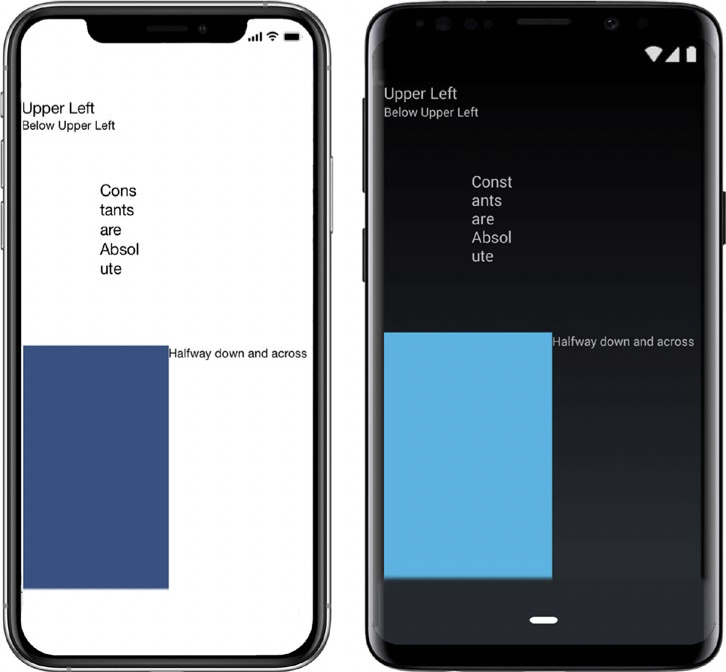
Khung hình BelowUpperLeft được đặt bên dưới khung hình UpperLeft, được chỉ ra tròn hình 3-17 (cho android).



***Hình 3-17****. Đặt một nhãn bên dưới một nhãn khác bằng việc sử dụng RelativeToView*

**CODE COMPLETE: RelativeLayout**

Liệt kê 3-11 là ví dụ XAML đầy đủ cho RelativeLayout sử dụng Ràng buộc: Constant, RelativeToParent, và RelativeToView (xem hình 3-18)



***Hình 3-18****. RelativeLayoutExample.cs sử dụng tất cả các loại ràng buộc*

Liệt kê 3-11. Code hoàn thiện RelativeLayoutExample.xaml

<?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="LayoutExample.Xaml.Views.RelativeLayoutExample" x:Name="Self"

Padding="10,10,10,5">

<RelativeLayout x:Name="LayoutContainer">  
<Label x:Name="UpperLeft" Text="Upper Left" FontSize="20" RelativeLayout.XConstraint="{ConstraintExpression Type=Constant, Constant=0}"

RelativeLayout.YConstraint="{ConstraintExpression Type=Constant,

Constant=0}" />

<Label Text="Constants are Absolute" FontSize="20" RelativeLayout.XConstraint="{ConstraintExpression Type=Constant, Constant=100}" RelativeLayout.YConstraint="{ConstraintExpression Type=Constant, Constant=100}" RelativeLayout.WidthConstraint="{ConstraintExpression Type=Constant, Constant=50}" RelativeLayout.HeightConstraint="{ConstraintExpression Type=Constant, Constant=200}" />

<Label Text="Halfway down and across" FontSize="15"

RelativeLayout.XConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Width, Factor=0.5}"

RelativeLayout.YConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Height, Factor=0.5}" />

<BoxView Color="Accent" HorizontalOptions="Start" Vertical

Options="StartAndExpand"

RelativeLayout.XConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Width, Factor=0}"

RelativeLayout.YConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Height, Factor=0.5}"

RelativeLayout.HeightConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Height, Factor=0.5}"

RelativeLayout.WidthConstraint="{ConstraintExpression

Type=RelativeToParent, Property=Width, Factor=0.5}" />

</RelativeLayout>

</ContentPage>

**AbsoluteLayout**

AbsoluteLayout là một tập hợp các khung nhìn được đặt ở tọa độ x / y từ 0 đến 1 và giới hạn về kích thước. Các vị trí xem thường không tuyệt đối, vì chúng tôi hiếm khi sử dụng tọa độ x hoặc y phụ thuộc vào thiết bị. Vị trí thường liên quan đến 0 ở điểm gốc và 1 tại điểm xa nhất dọc theo một trục. Bố cục tuyệt đối ở chỗ các khung nhìn sẽ đi chính xác nơi bạn đặt chúng, thậm chí trên đầu các khung nhìn khác, vì vậy điều này có thể hữu ích cho việc xếp lớp.

Sử dụng SetLayoutBound, các khung nhìn trong AbsoluteLayout được liên kết với một đối tượng giới hạn, có thể là một điểm hoặc một hình chữ nhật. Sử dụng SetLayoutFlags, các điểm giới hạn có thể xác định vị trí, trong khi các hình chữ nhật giới hạn có thể xác định vị trí và kích thước.

***Liệt kê 3-12*** cung cấp một ví dụ về AbsoluteLayout

<AbsoluteLayout VerticalOptions="FillAndExpand">

<!-- Add Views Here -->

</AbsoluteLayout>

Bây giờ chúng tôi sẽ thêm một số lượt xem. Để gán chế độ xem cho AbsoluteLayout, điều khiển được thêm vào bộ sưu tập AbsoluteLayout, sau đó LayoutFlags và LayoutBound được đặt để xác định vị trí và kích thước của chế độ xem. Các tham số của LayoutBound là "x, y, chiều rộng, chiều cao":

<AbsoluteLayout VerticalOptions="FillAndExpand">  
<Label Text="FirstLabel" AbsoluteLayout.LayoutBounds="0, 0, AutoSize, AutoSize" AbsoluteLayout.LayoutFlags="PositionProportional" />

</AbsoluteLayout>

Hình 3-19 thể hiện kết quả trên Android



***Hình 3-19****. Thêm một nhãn đến AbsoluteLayout*

**Tạo các đối tượng giới hjan với SetLayoutBound**

Lượt xem trong AbsoluteLayout có thể được đặt tại các điểm hoặc giới hạn bởi hình chữ nhật (hình vô hình). Điểm có ích khi chỉ cần một vị trí không có diện tích kích thước. Hình chữ nhật và các điểm là các đối tượng giới hạn được chỉ định trong LayoutBound.

Hãy tiếp tục với một ví dụ hình chữ nhật khác. Tạo nhãn thứ hai, thêm nó

đến bố cục, sau đó đặt cờ và giới hạn bố trí để đặt nó ở góc dưới bên trái của trang:

<Label Text="SecondLabel" AbsoluteLayout.LayoutBounds="0, 1, AutoSize,

AutoSize"

AbsoluteLayout.LayoutFlags="PositionProportional" />

Hình 3-10 thể hiện kết quả đó.



***Hình 3-20.*** *SecondLabel đặt phía dưới màn hình với toạ độ y =1*

**Thiết đặt vị trí và kích thước bằng việc sử dụng hình chữ nhật**

Hình chữ nhật cung cấp AbsoluteLayout với vị trí và kích thước của một khu vực giới hạn để đặt chế độ xem. Đây là các tham số LayoutBound của Hình chữ nhật:

     AbsoluteLayout.LayoutBound = "locationX, locationY, Chiều rộng, chiều "

Tất cả các tham số được nhân đôi từ 0 đến 1. Dưới đây là một vài ví dụ về cách các tham số LayoutBound ảnh hưởng đến vị trí:

* Nằm ở điểm gốc với chiều rộng và chiều cao tối đa: "0, 0, 1, 1"
* Nằm chính giữa trong không gian được cung cấp: ".5, 0, 1, 1"
* Căn giữa theo chiều dọc trong không gian được cung cấp: "0, .5, 1, 1"
* Theo chiều ngang và chuyển đổi theo chiều ngang: ". 5, .5, 1, 1"

Sau đây là các ví dụ về các tham số ảnh hưởng đến kích thước:

* Nằm ở điểm gốc với chiều rộng và chiều cao tối đa: "0, 0, 1, 1"
* Nằm ở điểm gốc với chiều rộng 20%: "0, 0, .2, 1"
* Nằm ở điểm gốc ở độ cao 20%: "0, 0, 1, .2"
* Nằm ở gốc tọa độ 20% chiều rộng và chiều cao: "0, 0, .2, .2"

**Đặt vị trí bằng sử dụng điểm**

Điểm có thể chỉ định vị trí của chế độ xem khi không cần kích thước. Lượt xem có thể được thêm vào AbsoluteLayout rất đơn giản bằng cách sử dụng một điểm:

     <Label Text="PointLabel" AbsoluteLayout.LayoutBounds="0, 1"

AbsoluteLayout.LayoutFlags="PositionProportional" />

Các điểm hoạt động giống như phần vị trí của hình chữ nhật, xác định vị trí x và y bằng cách sử dụng các giá trị nhân đôi từ 0 đến 1. Dưới đây là các tham số LayoutBound cho một điểm:

     AbsoluteLayout.LayoutBounds="locationX, locationY"

Các điểm và hình chữ nhật chỉ là các đối tượng hình học cho đến khi chúng được liên kết với một vị trí hoặc kích thước của chế độ xem bằng cách sử dụng SetLayoutFlags.

**Binding Location**

Liên kết vị trí x / y của đối tượng giới hạn vào chế độ xem bằng cách sử dụng các cờ này:

• **PositionProportional** liên kết vị trí hình chữ nhật hoặc điểm Vị trí x / y theo tỷ lệ với vị trí của chế độ xem, trong khi giá trị kích thước là tuyệt đối:

          AbsoluteLayout.LayoutFlags = "PositionProportional"

Nếu hình chữ nhật hoặc điểm là 0,0, thì khung nhìn sẽ ở 0,0.

• **XProportional** liên kết hình chữ nhật hoặc tọa độ điểm x theo tỷ lệ, trong khi tất cả các giá trị khác là tuyệt đối:

          AbsoluteLayout.LayoutFlags = "XProportional"

• **YProportional** liên kết tọa độ hình chữ nhật hoặc điểm y theo tỷ lệ, trong khi tất cả các giá trị khác là tuyệt đối:

          AbsoluteLayout.LayoutFlags = "YProportional"

**Binding Size**

Ràng buộc kích thước của đối tượng giới hạn vào chế độ xem bằng cách sử dụng các cờ này:

* **SizeProportional** liên kết kích thước hình chữ nhật tỷ lệ với kích thước của chế độ xem, trong khi giá trị vị trí là tuyệt đối:

AbsoluteLayout.LayoutFlags = "SizeProportional"

Nếu hình chữ nhật có kích thước .2, .5, chế độ xem sẽ có kích thước là .2, .5.

* **WidthProportional** liên kết chiều rộng hình chữ nhật tỷ lệ với chiều rộng của chế độ xem, trong khi tất cả các giá trị khác là tuyệt đối:

AbsoluteLayout.LayoutFlags = "WidthProportional"

* **HeightProportional** liên kết chiều cao hình chữ nhật tỷ lệ thuận với chiều cao của chế độ xem, trong khi tất cả các giá trị khác là tuyệt đối:

AbsoluteLayout.LayoutFlags = "heightProportional"

**Size Ràng buộc cả vị trí và kích thước**

Đặt giá trị theo tỷ lệ hoặc tuyệt đối cho cả vị trí và kích thước bằng hình chữ nhật hoặc điểm bằng cách sử dụng Tất cả hoặc Không có.

* **All** liên kết vị trí và kích thước hình chữ nhật hoặc điểm hình chữ nhật theo tỷ lệ với vị trí và kích thước của chế độ xem:

AbsoluteLayout.LayoutFlags = "Tất cả"

* **None** liên kết tất cả các giá trị là tuyệt đối:

AbsoluteLayout.LayoutFlags = "Không"

**Code hoàn : AbsoluteLayout**

Liệt kê 3-13 sử dụng AbsoluteLayout để thêm nhãn vào đầu và cuối màn hình được hiển thị trước đó trong Hình 3-20. Ví dụ này sử dụng hình chữ nhật giới hạn với một vài cách khác nhau để thêm Nhãn đầu tiên (chẳng hạn như sử dụng điểm thay vì hình chữ nhật giới hạn).

**Liệt kê 3-13**. AbsoluteLayoutExample.xaml Sử dụng hình chữ nhật

<ContentPage xmlns="http://xamarin.com/schemas/2014/

forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/

xaml" x:Class="LayoutExample.Xaml.Views.AbsoluteLayoutExample"

Padding="10,10,10,5">

<AbsoluteLayout VerticalOptions="FillAndExpand">

<Label Text="FirstLabel" AbsoluteLayout.LayoutBounds="0, 0,

AutoSize, AutoSize" AbsoluteLayout.LayoutFlags="PositionProport

ional" />

<Label Text="SecondLabel" AbsoluteLayout.LayoutBounds="0, 1,

AutoSize, AutoSize" AbsoluteLayout.LayoutFlags="PositionProport

ional" />

<Label Text="PointLabel" AbsoluteLayout.LayoutBounds="0,1”

AbsoluteLayout.LayoutFlags="PositionProportional" />

</AbsoluteLayout>

</ContentPage>

Với AbsoluteLayout trong tay, chúng tôi đã bao phủ gia đình cốt lõi của bố cục Xamarin.Forms. Bây giờ, hãy để Lọ nhìn vào một số anh em họ bố trí xa mà chúng ta sử dụng cho các mục đích cụ thể, bắt đầu với ContentView.

**ContentView**

Bố cục ContentView có thể hoạt động như một lớp container trực quan hoặc ảo, giống như một điều khiển tùy chỉnh. ContentView được thiết kế để sử dụng lại trong suốt ứng dụng của bạn. Nó cũng hữu ích để cung cấp phần đệm nhanh hoặc định dạng xung quanh chế độ xem hoặc bố cục khác.

Là một thùng chứa hình chữ nhật trực quan, ContentView cung cấp Bố cục tiêu chuẩn

các thuộc tính lớp như Padding, BackgroundColor, VerticalOptions và Options, giống như điều khiển bảng điều khiển .NET. Là một thùng chứa ảo, nó có thể chứa bố cục con chứa nhiều chế độ xem để hoán đổi trong và ngoài trang và để sử dụng trên các trang khác nhau, giống như điều khiển tùy chỉnh .NET hoặc đoạn Android.

Đây là một ContentView đơn giản (từ một lớp có tên SubContentView), một hình chữ nhật teal nhẹ nhàng với nhãn văn bản màu trắng:

<ContentView xmlns="http://xamarin.com/schemas/2014/

forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/

xaml" x:Class="LayoutExample.Xaml.Views.SubContentView"

BackgroundColor="Teal" Padding="40" HorizontalOptions="Fill">

<Label Text="a view, such as a label, a layout, or a layout of

layouts" FontSize="20" FontAttributes="Bold" TextColor="White" />

</ContentView>

ContentView can be used like any other view and placed onto a layout.  
Figure 3-21

ContentView có thể được sử dụng như bất kỳ chế độ xem nào khác và được đặt trên một bố cục.

Hình 3-21 cho thấy nhãn được giấu bên trong ContentView. ContentView này sau đó được đặt trên ContentPage:

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="LayoutExample.Xaml.Views.ContentViewExample" xmlns:Views="clr-

namespace:LayoutExample.Xaml.Views;assembly=LayoutExample.Xaml"

Padding="10,10,10,5">

<StackLayout> <Views:SubContentView />

</StackLayout>

</ContentPage>

Bây giờ ContentView được thêm màu xung quanh nhãn trong hình 3-21



***Hình 3-21****. ContentView*

**CODE COMPLETE: ContentView**

Các liệt kê 3-14 và 3-15 hiển thị ví dụ mã ContentView hoàn chỉnh được hiển thị trong Hình 3-21. Hai loại phần đệm được sử dụng ở đây: phần đệm màu bên trong ContentView và khoảng trống xung quanh rìa của ContentPage

***Listing 3-14.*** ContentViewExample.xaml

<?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="LayoutExample.Xaml.

Views.ContentViewExample" xmlns:Views="clr-namespace:LayoutExample.Xaml.

Views;assembly=LayoutExample.Xaml" Padding="10,10,10,5">

<StackLayout>

<Views:SubContentView />

</StackLayout>

</ContentPage>

***Listing 3-15.*** SubContentView.xaml

<?xml version="1.0" encoding="UTF-8"?>

<ContentView xmlns="http://xamarin.com/schemas/2014/

forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/

xaml" x:Class="LayoutExample.Xaml.Views.SubContentView"

BackgroundColor="Teal" Padding="40" HorizontalOptions="Fill">

<Label Text="a view, such as a label, a layout, or a layout of

layouts" FontSize="20" FontAttributes="Bold" TextColor="White" />

</ContentView>

**Frame (khung)**

Bố cục khung đặt một khung có thể nhìn thấy xung quanh chính nó. Thuộc tính OutlineColor chỉ định màu của khung. Xem Liệt kê 3-16.

***Listing 3-16.*** FrameExample.xaml

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://

schemas.microsoft.com/winfx/2009/xaml" x:Class="LayoutExample.Xaml.Views.

FrameExample" Padding="10,10,10,5">

<Frame OutlineColor="Red">

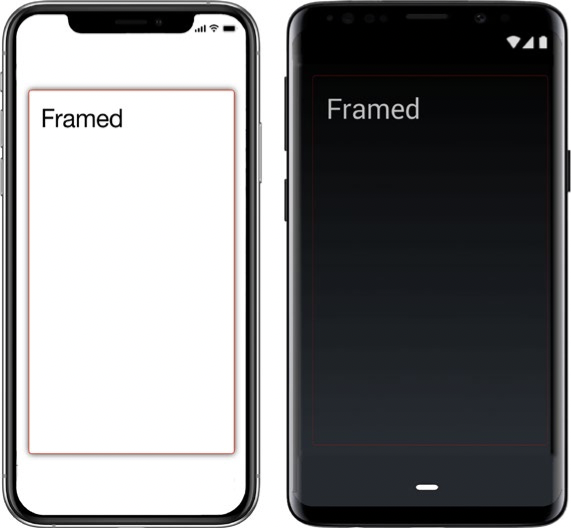
<Label Text="Framed" FontSize="40" />

</Frame>

</ContentPage>

HasShadow là một thuộc tính Boolean chỉ định hiệu ứng bóng khi nền tảng hỗ trợ nó. Giá trị Padding mặc định trên bố cục Khung là 20.

Hình 3-22 cho thấy bố cục Khung.



***Hình 3-22****. Frame Layout*

Đó là những bố cục trong Xamarin.Forms!

Khi xây dựng bố cục, một chủ đề liên quan phát sinh là việc tạo các điều khiển tùy chỉnh, được sử dụng làm thành phần để xây dựng bố cục.

**Hiểu điều khiển tuỳ chỉnh**

Các điều khiển tùy chỉnh trong Xamarin là các bố cục một phần có thể được bao gồm trong các bố cục lớn hơn trên cơ sở khi cần, có thể được tạo trên tất cả các nền tảng và có thể được thực hiện để hoạt động như điều khiển người dùng, điều khiển tùy chỉnh hoặc bảng điều khiển trong .NET. Điều khiển tùy chỉnh hầu như không được đề cập trong cuốn sách này, nhưng chủ đề được đề cập trong bối cảnh xây dựng bố cục cấp chuyên nghiệp.

Trong Xamarin.Forms, ContentView là lớp cơ sở để tạo các chế độ xem tùy chỉnh để lồng, đệm và tái sử dụng. Không nên nhầm lẫn các điều khiển tùy chỉnh với các điều khiển tùy chỉnh, thường là các chế độ xem Xamarin.Forms riêng lẻ với chức năng dành riêng cho nền tảng nâng cao được xây dựng bằng trình kết xuất tùy chỉnh (xem Chương 8). Mặc dù vậy, các nhà phát triển đôi khi sẽ đề cập đến một điều khiển tùy chỉnh duy nhất như một điều khiển tùy chỉnh. Ngoài ra, một điều khiển tùy chỉnh có khả năng chứa nhiều điều khiển và sau đó thực sự có thể trở thành điều khiển tùy chỉnh.

**Tổng kết**

Bố cục là một ví dụ điển hình về việc các nền tảng khác nhau này có thể giống nhau như thế nào. Thay đổi tên, nhưng các khái niệm không có giá trị, và để tăng tính liên tục, Xamarin cố gắng kết hợp các khía cạnh hữu ích nhất của các nền tảng khác nhau này vào Xamarin.Forms. Dưới đây là một số thuật ngữ phổ biến liên quan đến bố cục:

* *FlexLayouts* là cách mới nhất và dễ sử dụng nhất đồng thời cũng là một trong những bố cục linh hoạt nhất.
* *StackLayouts* là bố cục đơn giản nhất trong Xamarin.Forms, tuyệt vời cho các trang dễ dàng và tạo mẫu nhanh và tạo khung.
* *Rectangles* frame và kích thước của chúng và vị trí tọa độ.
* *RelativeLayouts* rất hữu ích khi tác vụ đơn giản và chính xác

tọa độ trên tay.

* *Constraint* liên kết các khung nhìn với nhau như đàn hồi xác định theo kích thước và vị trí theo ngữ cảnh. Các ràng buộc phức tạp yêu cầu C #.
* *Layout Options* xử lý căn chỉnh và định dạng với ngang dọc và dọc.
* *Custom control* vẫn tồn tại như một khái niệm trong phát triển di động, đạt được trong Xamarin.Forms với ContentView.
* *Grid layout* là cung cấp bố cục bảng Xamarin.Forms, cho chúng ta tính tương đối, độ chính xác và tính linh hoạt.

Đó là một số nguyên tắc cơ bản của bố trí màn hình điện thoại di động. Bây giờ, thời gian để làm cho những bố cục và điều khiển đẹp hơn bằng cách sử dụng kiểu dáng.