Macoratti.net Xamarin Android - Criando Interfaces com o usuário via código e via XML

Neste artigo vou apresentar os conceitos básicos sobre Layouts relacionados com a interface do usuário (UI) em aplicações **Android** usando o Visual Studio com Xamarin e a linguagem C#.

Todas as aplicações para dispositivos móveis requerem uma interface com o usuário, a famosa UI (*User Interface*), de forma a interagir com o usuário recebendo e exibindo informações.

As Views e as ViewsGroups

Tudo o que você vê em um aplicativo Android é uma **View** : botões, etiquetas, caixas de texto e botões de rádio são todos exemplos de **views**.

As **Views** são organizadas em uma hierarquia utilizando diversos tipos de **ViewGroups**. Uma **ViewGroup** é um tipo especial de **View** que é usado para organizar (*layout*) as outras views na tela.

As **Views e as ViewGroups** podem ser criadas usando dois métodos diferentes: **programaticamente ou declarativamente.**

- Ao usar uma abordagem **programática**, o desenvolvedor faz chamadas de API para criar e posicionar cada **View** individual na interface.
- Ao usar uma abordagem **declarativa**, o desenvolvedor cria arquivos de layout XML que especificam como as **Views** devem organizadas.

Criar interfaces com o usuário usando o Visual Studio com Xamarin ou o Xamarin Studio é bem simples e você pode realizar essa tarefa de duas maneiras :

1. Criar a interface com o usuário, via programação, usando código C#;
2. Criar a interface com o usuário usando a abordagem declarativa usando XML;

Vejamos cada uma dessas abordagens.

Recursos usados:

- **Visual Studio Community 2015 ou Xamarin Studio**
- **Xamarin**
- **Emulador Android virtual ou físico (veja como emular usando o Vysor)**

Nota: Baixe e use a versão Community 2015 do VS ela é grátis e é equivalente a versão Professional.

1- Criando uma interface com o usuário via código C#

Abra o VS Community 2015 e clique em **New Project**;

Selecione a linguagem Visual C# e o template **Android -> Blank App (Android);**

Informe o nome **App.Layouts1** e clique no botão OK:

A seguir abra o arquivo **MainActivity.cs** criado e altere o seu código substituindo o código gerado pelo código abaixo:

```
using Android.App;
using Android.OS;
using Android.Views;
using Android.Widget;

namespace App.Layouts1
{
   [Activity(Label = "App.Layouts1", MainLauncher = true, Icon = "@drawable/icon")]
   public class MainActivity : Activity
   {
      protected override void OnCreate(Bundle bundle)
      {
         base.OnCreate(bundle);

         //cria uma instância do Layout LinearLayout
         LinearLayout layout = new LinearLayout(this);
          //define a cor de fundo do layout
         layout.SetBackgroundColor(Android.Graphics.Color.Gray);
         //define a orientação do leiaute
         layout.Orientation = Orientation.Vertical;
         //define um TextView
         TextView lblNome = new TextView(this);
         lblNome.Text = "Informe o nome";
         //define um EditText
         EditText txtNome = new EditText(this);
         //define um button
         Button button = new Button(this);
         //atribui um texto ao button
```

```
            button.Text = "Macoratti .net!";
            //define a cor de fundo do botão
             button.SetBackgroundColor(Android.Graphics.Color.Red);
            //inclui o TextView, EditText e o Button no leiaute
            layout.AddView(lblNome, ViewGroup.LayoutParams.MatchParent, ViewGroup.LayoutParams.WrapContent);
            layout.AddView(txtNome, ViewGroup.LayoutParams.MatchParent, ViewGroup.LayoutParams.WrapContent);
            layout.AddView(button, ViewGroup.LayoutParams.MatchParent, ViewGroup.LayoutParams.WrapContent);
            //define o layout como a view principal
            SetContentView(layout);
        }
    }
}
```

Neste código estamos definindo a interface do usuário criando objetos de cada classe do componente que desejamos usar. No exemplo criamos:

- Um objeto LinearLayout - layout
- Um objeto TextView - lblNome
- Um objeto EditText - txtNome
- Um objeto Button - button

Definimos valores para algumas propriedades como texto e cor de fundo e incluímos os objetos no container layout.

O resultado ao executar a aplicação pode ser visto na figura ao lado do código.

Assim criamos uma interface totalmente via código.

Se você abrir o arquivo **Main.axml** na pasta **Resources\layout** verá que ele permanece inalterado e que não esta sendo usado no projeto.

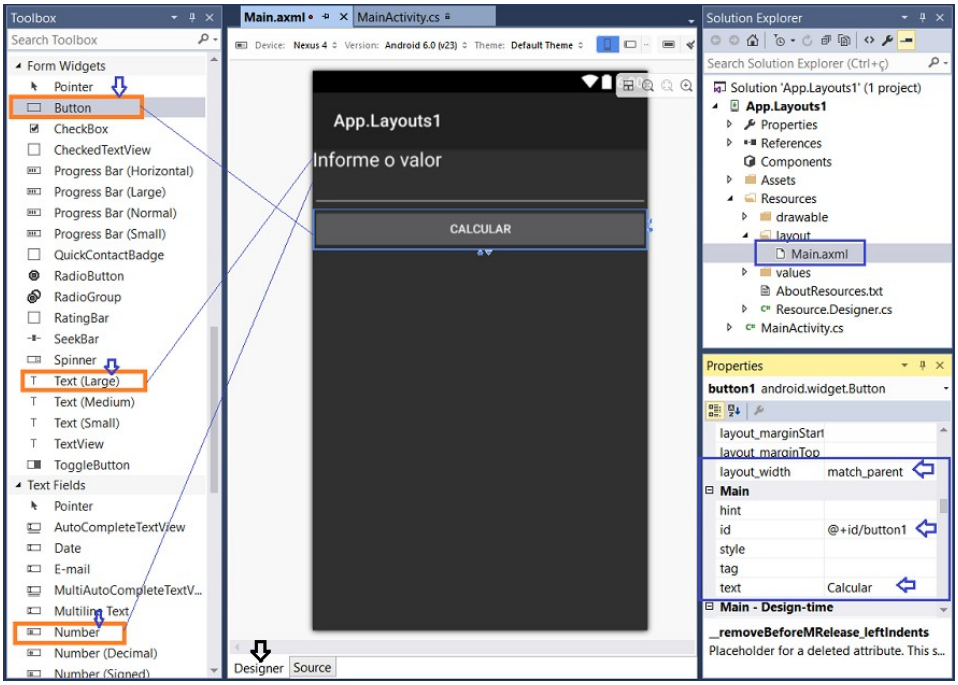
2- Criando uma interface com o usuário via XML

Talvez a maneira mais simples e intuitiva de criar a interface com o usuário seja via XML com a ajuda do designer do Xamarin.

Esta abordagem permite usar os controles disponíveis na **ToolBox**, arrastando cada controle para a interface e assim teremos o código XML gerado automaticamente.

Abra o arquivo **Main.axml** e no modo Designer inclua a partir da **ToolBox** os controles:

- Text (Large) - text= Informe o valor
- Number
- Button - text= Calcular



Na figura vemos a interface sendo montada e as propriedades principais sendo definidas via janela **Properties**.

***Nota**: A propriedade **id** define no arquivo **Resource.Designer.cs** cada um dos componentes atribuindo a cada um número único de identificação.*

O arquivo **Resource.designer.cs** é um arquivo C# na pasta Resources que é gerado pelo Xamarin.Android e contém definições de ID para todos os recursos na App.

Se você clicar no modo **Source** verá o código XML gerado:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:minWidth="25px"
   android:minHeight="25px">
   <TextView
      android:text="Informe o valor"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/textView1" />
   <EditText
      android:inputType="number"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/editText1" />
   <Button
      android:text="Calcular"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/button1" />
</LinearLayout>
```

A estrutura de layout Android geralmente segue a estrutura do tipo, com o nome do elemento correspondente ao nome tipo e os nomes das propriedades que correspondem aos nomes de atributos.

Por exemplo, o tipo **Button** tem uma propriedade **Text**, e assim, o XML terá um elemento : **<Button android:text = "..." />.**

Um dos atributos mais importantes no layout é o atributo id. Este atributo é usado para identificar de forma única a view.

Note que temos dois atributos usados em todos os componentes :

- **match_parent** e **wrap_content**

Qual o papel desses atributos ?

Essas atributos permitem definir como o componente vai atuar na interface.

Assim temos que :

- **wrap_content** - informa ao componente/view para ocupar o espaço que ele vai necessitar *(altura e/ou largura)* para exibir suas informações no layout.
- **match_parent** - informa ao componente/view para ocupar o mesmo espaço da sua view pai, ou seja, ele vai preencher todo o conteúdo de seu layout pai.

Nota : *Você também pode encontrar o atributo **fill_parent** em versões anteriores á versão 2.3. Esse atributo atua da mesma forma que **match_parent**.*

Dessa forma o arquivo de Layout é uma forma fácil de criar uma UI separada do código, e, da mesma forma, os arquivos de recursos podem ser usados para separar o texto localizado no layout.

Podemos conseguir isso colocando strings no arquivo de recurso e então referenciando cada string a partir do layout.

Vamos supor que temos o um **Button** com a propriedade **Text** igual a : "Macoratti .net".

No arquivo de layout XML teríamos o elemento : **<Button android:text ="Macoratti .net" />**

Podemos extrair o valor do texto e colocá-lo em um arquivo XML na pasta **Resources/values.**

Vamos então criar o arquivo **buttonTexto.xml** na pasta **Resources/values** com o seguinte conteúdo:

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
  <string name="buttonTexto">Macoratti .net</string>
</resources>
```

A seguir basta atualizar o arquivo de Layout **Main.axml** para referenciar o valor **buttonTexto** no componente **Button**:

...

```
 <Button
        android:text="@string/buttonTexto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/button1" />
```

...

Usando este padrão estamos aptos a não somente extrair strings mas qualquer valor para qualquer atributo, incluindo informações do layout.

Após definir o código da interface no arquivo **Main.axml** você tem que indicar para **Activity** qual arquivo de layout será usado, e , isso é feito invocando o método **SetContentView**() no método **OnCreate**() da Activity.

Para o nosso exemplo podemos alterar o código do arquivo **MainActivity.cs** conforme abaixo:

```
using Android.App;
using Android.OS;
using Android.Widget;

namespace App.Layouts1
{
   [Activity(Label = "App.Layouts1", MainLauncher = true, Icon = "@drawable/icon")]
   public class MainActivity : Activity
   {
      protected override void OnCreate(Bundle bundle)
      {
         base.OnCreate(bundle);

         int contador = 1;
         SetContentView(Resource.Layout.Main);

         Button button = FindViewById<Button>(Resource.Id.button1);
         button.Click += delegate
         {
            button.Text = string.Format("{0} cliques !", contador++);
         };
      }
   }
}
```

Veja que definimos no método **SetContentView**() o arquivo **Main.axml** que esta na pasta **Resources/layout** como sendo a view principal da aplicação.

Assim, não importando se o layout foi criado via código ou via XML, podemos acessar os vários controles de forma similar.

No exemplo estamos acessando o controle **Button** em tempo de execução usando o método **FindViewById** para a seguir definir o evento **Click** no botão usando um delegate.

Assim podemos concluir que a abordagem declarativa possui a seguintes vantagens:

• Proporciona uma melhor separação do design visual de um aplicativo da sua lógica de processamento;
• Permite que vários layouts sejam criados para apoiar vários dispositivos ou configurações de dispositivos com uma única base de código;
• Ferramentas de desenvolvimento, tais como Android Studio, Xamarin Studio e o plugin Android para Eclipse, permitem visualizar a interface do usuário, sem a necessidade de compilar e executar o aplicativo após cada alteração;

Pegue o projeto completo aqui : **App.Layouts1.zip** (*sem as referências*)

E esta é a mensagem que dele (Jesus) ouvimos, e vos anunciamos: que Deus é luz, e não há nele trevas nenhumas.
Se dissermos que temos comunhão com ele, e andarmos em trevas, mentimos, e não praticamos a verdade.
1 João 1:5,6

Referências:

- Seção VB .NET do Site Macoratti.net
- Super DVD .NET - A sua porta de entrada na plataforma .NET
- Super DVD Vídeo Aulas - Vídeo Aula sobre VB .NET, ASP .NET e C#
- Super DVD C# - Recursos de aprendizagens e vídeo aulas para C#
- Seção C# do site Macoratti.net
- Seção ASP .NET do site Macoratti .net
- Curso Básico VB .NET - Vídeo Aulas
- Curso C# Básico - Vídeo Aulas
- Curso Fundamentos da Programação Orientada a Objetos com VB .NET NEW
- **Macoratti .net | Facebook**
- **macoratti - YouTube**
- **Jose C Macoratti (@macorati) | Twitter**
- Xamarim - Desenvolvimento Multiplataforma com C# ... - Macoratti.net
- Xamarin - Apresentando Xamarin.Forms - Macoratti.net
- Xamarin.Forms - Olá Mundo - Criando sua primeira ... - Macoratti.net
- Xamarin.Forms - Olá Mundo - Anatomia da aplicação - Macoratti.net
- https://developer.xamarin.com/api/type/Android.Widget.ListView/
- https://developer.xamarin.com/api/property/Android.Widget.ListView.Adapter/

José Carlos Macoratti