Macoratti.net Xamarin Android - Tratando arquivos e arquivos do sistema (FileSystem)

Neste artigo vou mostrar como tratar arquivos e arquivos do sistema no Xamarin Android usando o Visual Studio 2015 e a linguagem C#.

Muitas vezes precisamos tratar arquivos e acessar o sistema de arquivos para realizar operações de leitura e gravação com arquivos, sejam eles arquivos textos, banco de dados, etc.

Há duas áreas principais para armazenar arquivos quanto trabalhamos com dispositivos móveis:

1- Armazenamento interno
2- Armazenamento externo

Usando o Armazenamento Interno

Usando o armazenamento interno ou app sandbox :

1. Escrever arquivos no sistema de arquivos é muito simples, e, tudo o que é necessário é o caminho para o **Sandbox** para o nosso aplicativo:

```
String sandbox = FilesDir.AbsolutePath;
```

2. Uma vez que temos esse caminho, podemos usar os tipos da plataforma .NET para manipular os arquivos:

```
string arquivo = Path.Combine (sandbox, "meuArquivo.txt");
Bool existe = Arquivo.Existe (arquivo);
File.WriteAllText (arquivo, "Macoratti .net - Android");
String valor = File.ReadAllText (arquivo);
```

3. Às vezes, só precisamos armazenar arquivos temporariamente. Nesses casos, podemos usa o cache local

```
String cache = CacheDir.AbsolutePath
```

Usando o Armazenamento Externo

Usar o armazenamento externo tem apenas alguns requisitos extras. Normalmente, utilizar o armazenamento externo é quase igual a usar o armazenamento interno, exceto pelo fato que temos uma pasta raiz diferente.

Se desejamos escrever para o armazenamento externo precisamos solicitar permissão para fazer isso.

```
[assembly: UsesPermission(Manifest.Permission.WriteExternalStorage)]
[assembly: UsesPermission(Manifest.Permission.ReadExternalStorage)]
```

Uma vez que temos a permissão para acessar o sistema de arquivos podemos verificar que a media externa esta anexada e gravável.

```
bool gravavel = Android.OS.Environment.ExternalStorageState == Android.OS.Environment.MediaMounted;
bool legivel = gravavel || Android.OS.Environment.ExternalStorageState == Android.OS.Environment.MediaMountedReadOnly
```

Podemos também obter o caminho da localização externa e então trabalhar como desejamos com os arquivos do sistema do armazenamento interno.

```
string external = GetExternalFilesDir(null).AbsolutePath;
string externalCache = ExternalCacheDir.AbsolutePath;
```

Existem também alguns locais externos extras, como os diretórios públicos. E, é aqui onde o Android armazena e lê arquivos como arquivos de música, filmes e downloads.

```
string pastaDownloads =
Android.OS.Environment.GetExternalStoragePublicDirectory(Android.OS.Environment.DirectoryDownloads).AbsolutePath;
```

Podemos também ver quanto espaço ou espaço livre esta disponível consultando a localização do arquivos:

```
long espacoLivre = FilesDir.FreeSpace;
long espacoTotal = GetExternalFilesDir(null).TotalSpace;
```

Assim, trabalhar com o sistema de arquivos Android é semelhante ao uso dos sistemas de arquivos em qualquer plataforma. Podemos fazer uso de todos os recursos no .NET, bem como muitos dos recursos do Java.

No entanto, o Android não permite o acesso a todos os locais do dispositivo, especialmente a localização do sistema ou outros locais protegidas.

Normalmente, um aplicativo acessará seu próprio local de sandbox interno obtido a partir de **FilesDir** e possivelmente o local de cache do aplicativo obtido do **CacheDir**. Arquivos salvos no armazenamento interno são privados para o nosso aplicativo e nem outros aplicativos nem o usuário podem acessá-los.

Quando o usuário desinstala nosso aplicativo, os arquivos internos são removidos automaticamente. Além disso, o usuário pode removê-los através das telas de configurações do dispositivo. Como resultado, precisamos verificar se um arquivo existe antes de tentar ler o arquivo.

Vamos por tudo isso em prática criando um projeto Xamarin Android para realizar as operações com arquivos.

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.

Criando o projeto no VS Community 2015

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

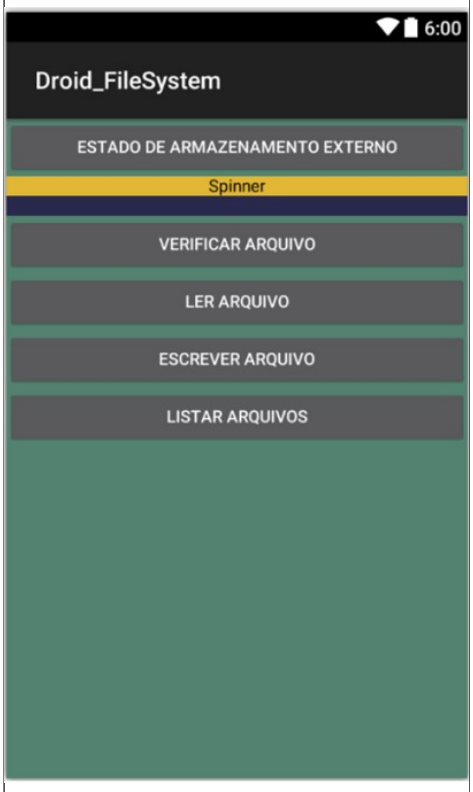
Selecione a linguagem Visual C# e o template **Android -> Blank App(Android)**

Informe o nome um nome adequado ao seu projeto, eu vou usar o nome **Droid_FileSystem**, e clique no botão **OK**;

Abra o arquivo **Main.axml** na pasta **Resources/layout** no modo **Designer** e inclua a partir da ToolBox os seguintes controles:

- **1 Button**
- **1 Spinner**
- **1 TextView**
- **4 Buttons**
- **1 TextView**

Abaixo vemos o leiaute no emulador do Xamarin e ao lado o respectivo 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:background="#538370">
  <Button
    android:text="@string/EstadoArmazenamento"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/EstadoArmazenagem" />
  <Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#e1b735"
    android:id="@+id/spinner" />
  <TextView
    android:text=" "
    android:background="#28294c"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/labelCaminhoBase" />
  <Button
    android:text="@string/verificaArquivo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/verificaArquivo" />
  <Button
    android:text="@string/leArquivo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/leArquivo" />
  <Button
    android:text="@string/escreveArquivo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/escreveArquivo" />
  <Button
    android:text="@string/listaArquivos"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
    android:id="@+id/listaArquivos" />
  <TextView
    android:text=" "
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:id="@+id/lblMensagem" />
</LinearLayout>
```

A seguir abra o arquivo **Strings.xml** na pasta **Resources/values** e altere o seu conteúdo com o seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="verificaArquivo">Verificar Arquivo</string>
  <string name="leArquivo">Ler Arquivo</string>
  <string name="escreveArquivo">Escrever Arquivo</string>
  <string name="listaArquivos">Listar Arquivos</string>
  <string name="EstadoArmazenagem">Estado de Armazenamento Externo</string>
  <string name="ApplicationName">FileSystem</string>
</resources>
```

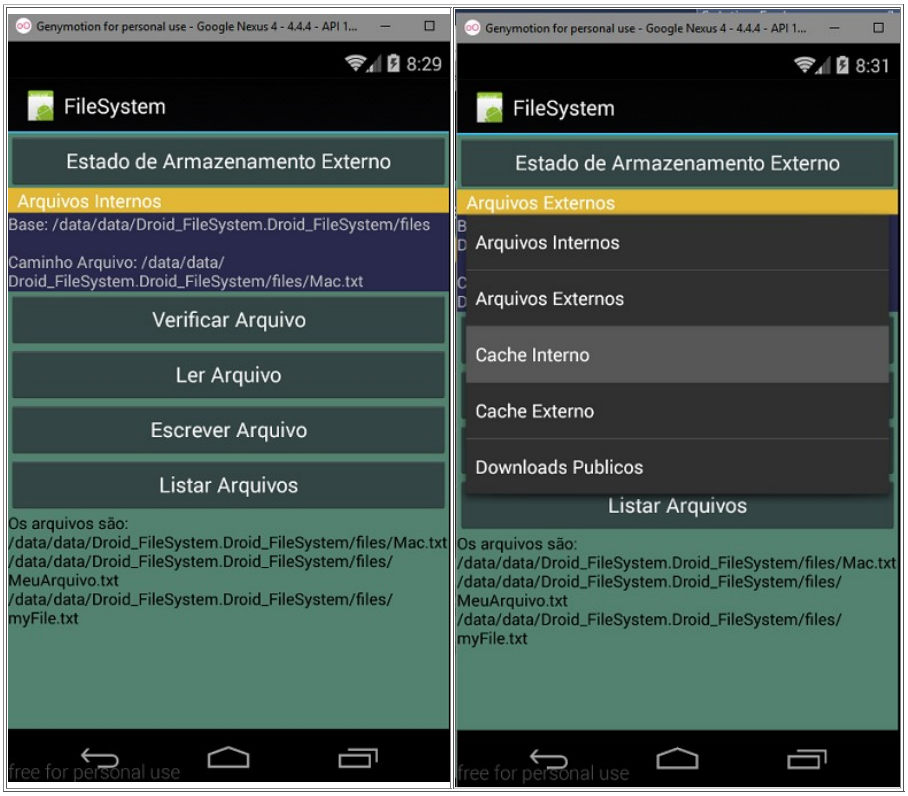
Vamos agora abrir o arquivo **MainActivity** na raiz do projeto e incluir o código abaixo neste arquivo.

```
using Android.App;
using Android.Widget;
using Android.OS;
using Android;
using System.IO;
using System;
using System.Collections.Generic;
using System.Linq;

[assembly: UsesPermission(Manifest.Permission.WriteExternalStorage)]
[assembly: UsesPermission(Manifest.Permission.ReadExternalStorage)]

namespace Droid_FileSystem
{
  [Activity(Label = "FileSystem", MainLauncher = true, Icon = "@drawable/icon")]
  public class MainActivity : Activity
  {
    private const string MeuArquivo = "Mac.txt";
    private const string ValorTexto = "Macoratti .net - Android";

    private string caminhoBase = "";
    private string nomeArquivoBase = "";

    protected override void OnCreate(Bundle bundle)
    {
      base.OnCreate(bundle);

      SetContentView (Resource.Layout.Main);

      // Permite selecionar o local
      var spinner = FindViewById<Spinner>(Resource.Id.spinner);
      var labelCaminhoBase = FindViewById<TextView>(Resource.Id.labelCaminhoBase);
      var dic = new Dictionary<string, string> {
        { "Arquivos Internos", FilesDir.AbsolutePath },
        { "Arquivos Externos", GetExternalFilesDir (null).AbsolutePath },
        { "Cache Interno", CacheDir.AbsolutePath },
        { "Cache Externo", ExternalCacheDir.AbsolutePath },
        { "Downloads Publicos", Android.OS.Environment.GetExternalStoragePublicDirectory (Android.OS.Environment.DirectoryDownloads).AbsolutePath },
      };

      string[] chaves = dic.Keys.ToArray();
      var adapter = new ArrayAdapter(this, Android.Resource.Layout.SimpleSpinnerItem, chaves);
      adapter.SetDropDownViewResource(Android.Resource.Layout.SimpleSpinnerDropDownItem);
      spinner.Adapter = adapter;

      spinner.ItemSelected += (sender, e) => {
        string chave = chaves[e.Position];
        caminhoBase = dic[chave];
        nomeArquivoBase = Path.Combine(caminhoBase, MeuArquivo);
        // atualiza aUI
        labelCaminhoBase.Text = string.Format("Base: {0}\n\nCaminho Arquivo: {1}", caminhoBase, nomeArquivoBase);
      };

      //obtém a referencia aos controles
      var verificaArquivo = FindViewById<Button>(Resource.Id.verificaArquivo);
      var LeArquivo = FindViewById<Button>(Resource.Id.leArquivo);
```

```
        var EscreveArquivo = FindViewById<Button>(Resource.Id.escreveArquivo);
        var ListaArquivos = FindViewById<Button>(Resource.Id.listaArquivos);
        var EstadoArmazenagem = FindViewById<Button>(Resource.Id.EstadoArmazenagem);
        var labelMensagem = FindViewById<TextView>(Resource.Id.lblMensagem);

        // define as operações
        verificaArquivo.Click += delegate {
            try
            {
                // verifica se o arquivo existe
                var arqExiste = File.Exists(nomeArquivoBase);
                // atualiza a UI
                labelMensagem.Text = "O Arquivo Existe: " + arqExiste;
            }
            catch (Exception ex)
            {
                // exibe mensagens de erros na  UI
                labelMensagem.Text = ex.Message;
            }
        };

        LeArquivo.Click += delegate {
            try
            {
                // le o valor do arquivo
                string valor = File.ReadAllText(nomeArquivoBase);
                // atualiza a UI
                labelMensagem.Text = "O valor é : " + valor;
            }
            catch (Exception ex)
            {
                // exibe mensagens de erros na  UI
                // uma exceção ocorre se o arquivo não existe
                labelMensagem.Text = ex.Message;
            }
        };

        EscreveArquivo.Click += delegate {
            // verifica os erros
            try
            {
                // escreve o valor no arquivo
                File.WriteAllText(nomeArquivoBase, ValorTexto);
                //atualiza a UI
                labelMensagem.Text = "O texto foi escrito no arquivo.";
            }
            catch (Exception ex)
            {
                // exibe mensagens de erros na  UI
                labelMensagem.Text = ex.Message;
            }
        };

        ListaArquivos.Click += delegate {
            try
            {
                // lista os arquivos
                var arquivos = Directory.GetFiles(caminhoBase);
                var valor = string.Join(System.Environment.NewLine, arquivos);
                labelMensagem.Text = "Os arquivos são: \n" + valor;
            }
            catch (Exception ex)
            {
                labelMensagem.Text = ex.Message;
            }
        };

        EstadoArmazenagem.Click += delegate {
            var gravavel = Android.OS.Environment.ExternalStorageState == Android.OS.Environment.MediaMounted;
            var legivel = gravavel || Android.OS.Environment.ExternalStorageState == Android.OS.Environment.MediaMountedReadOnly;
            var arquivo = new Java.IO.File(caminhoBase);

            const int bytesEmMB = 1000 * 1000;

            labelMensagem.Text = string.Format(
                "Legível: {0}\nGravável: {1}\nEspaço Livre: {2} MB\nEspaço Total: {3} MB\n",
                legivel, gravavel, arquivo.FreeSpace / bytesInMB, arquivo.TotalSpace / bytesEmMB);
        };
    }
}
```

}

Executando o projeto usando o emulador **Genymotion** iremos obter o seguinte resultado:



Nesta aplicação podemos selecionar o tipo de armazenamento com o qual desejamos tratar usando o controle **Spinner**. Assim acessamos o armazenamento interno e externo.

O acesso ao armazenamento externo não é diferente, mas requer permissões de aplicativo. Começando com Android 4.4, essas permissões não são necessárias se o único acesso externo for feito através do método **GetExternalFilesDir**() e a propriedade **ExternalCacheDir**.

Os arquivos externos são visíveis ao usuário, especialmente se eles se conectam com um computador via USB. Arquivos salvos no cache, seja interno via **CacheDir** ou externo via **ExternalCacheDir**, são destinados a serem temporários.

O Android pode apagar estes arquivos quando o dispositivo começar a ficar sem espaço livre ou se o aplicativo for desinstalado. No entanto, não devemos confiar no sistema para limpar esses arquivos e manter o cache.

É muito importante lembrar que o armazenamento externo pode ficar indisponível se o usuário montar o armazenamento externo em um computador ou remover a mídia. Como resultado, é essencial verificar a disponibilidade do armazenamento externo antes do uso.

No namespace **Android.OS**, o tipo **Environment** contém as propriedades, métodos e outros tipos usados para determinar o estado do armazenamento externo. A propriedade **ExternalStorageState** pode ser **MediaMounted**, que é gravável, ou **MediaMountedReadOnly**, que é somente leitura.

Existem locais públicos especiais que são usados pelo Android para armazenar os downloads do usuário como Fotos, música e outros arquivos. Esses locais são obtidos passando o diretório desejado para o método **GetExternalStoragePublicDirectory**() no ambiente.

Podemos ver quanto espaço livre está disponível usando as propriedades **FreeSpace** ou **TotalSpace** nos objetos **File** do Java, como o resultado da propriedade **FilesDir**.

O valores retornados não são o espaço exato disponível, mas uma representação de quanto espaço livre pode estar disponível. Se houver alguns MB extra sobre o tamanho a ser salvo, então é provavelmente que podemos continuar.

A aplicação mostra como usar esses recursos na prática em uma aplicação Android básica.

Pegue o projeto aqui : **Droid_FileSystem.zip** (sem as referências)

"Portanto nós também, pois que estamos rodeados de uma tão grande nuvem de testemunhas, deixemos todo o embaraço, e o pecado que tão de perto nos rodeia, e corramos com paciência a carreira que nos está proposta,"
Hebreus 12:1

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**
- VB.NET 2005 - Controles - Macoratti.net
- Seção de Jogos do site Macoratti .net
- 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.App.AlertDialog/
- Xamarin Android - Tratando eventos de forma declarativa
- Seção Mobile/Xamarin do site Macoratti .net
- https://developer.android.com/reference/android/app/Activity.html
- https://developer.xamarin.com/api/type/Android.Widget.ProgressBar/

José Carlos Macoratti