

## Macoratti.net Xamarin Android - Apresentando e usando Fragments - I

---



Neste artigo vou apresentar o conceito de **Fragments** e sua utilização em aplicações Xamarin Android no Visual Studio com a linguagem C#.

Desenvolver a interface com o usuário para diferentes tamanhos de tela em dispositivos como *smartphones* e *tablets* não é uma tarefa trivial e pode consumir muito tempo.

Os tamanhos de tela maiores encontradas na maioria dos *tablets* adicionou uma camada extra de complexidade para o desenvolvimento das interfaces, um layout Android projetado para a tela pequena não necessariamente funciona bem para telas maiores, e vice-versa.

A fim de reduzir o número de complicações que este cenário projeta, a partir da versão do [Android 3.0](#) temos disponível duas funcionalidades : **Fragments e Support Packages**.

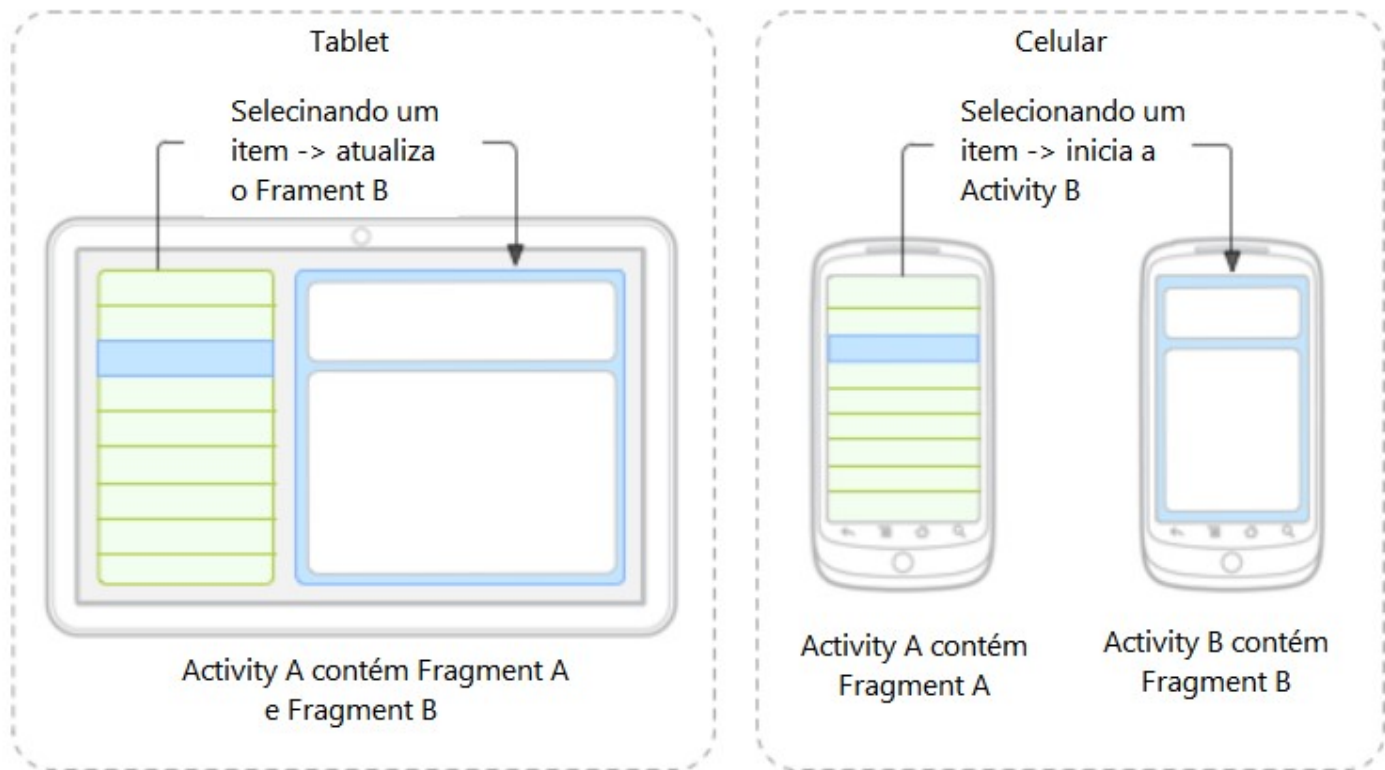
Os **Support Packages** ou **Compatibility Libraries** permitem que os **Fragments** sejam utilizados em dispositivos que executem em versões do Android anteriores a versão 3.0.(*API Nível 11 e Xamarin Android 4.0 ou superior*).

Podemos pensar nos **Fragments** como módulos de interface. Eles permitem que o desenvolvedor divida a interface do usuário em partes isoladas, reutilizáveis que podem ser executadas em atividades separadas. Em tempo de execução, as atividades em si vão decidir quais **Fragments** usar.

Usando o recurso **Fragments** podemos dividir nossa tela e dar comportamento a cada pedaço (*fragmento*) de uma view com um **Fragment**.

Os **Fragments** podem ser vistos também como *mini-activites* que executam dentro de uma **Activity** principal, a *Activity host*, possuindo um ciclo de vida bem definido a saber : **create/pasuse/resume destroy**.

**Por exemplo :** A imagem ilustra uma aplicação de **Fragments** durante a variação da forma do dispositivo:



O **Fragment A** contém uma lista, enquanto o **Fragment B** contém detalhes de um item selecionado na lista.

Quando o aplicativo é executado em um **Tablet**, ele pode exibir ambos os fragmentos na mesma atividade.

Quando o mesmo aplicativo é executado em um **celular** (*com o seu tamanho de tela menor*), os fragmentos são hospedados em duas atividades separadas.

Os **Fragment A e Fragment B** são os mesmos em ambos os fatores de forma, mas as atividades que os hospedam são diferentes.

Para criar um fragmento, é preciso criar uma subclasse de **Fragment** (ou uma respectiva subclasse existente).

A classe **Fragment** tem um código que é muito parecido com o de uma **Activity**. Ela contém métodos de retorno de chamada semelhantes aos de uma atividade, como [onCreate\(\)](#), [onStart\(\)](#), [onPause\(\)](#) e [onStop\(\)](#).

Na verdade, caso esteja convertendo um aplicativo do Android existente para usar os fragmentos, basta mover o código dos métodos de retorno de chamada da atividade para os respectivos métodos de retorno de chamada do fragmento.

Geralmente, deve-se implementar pelo menos os seguintes métodos de ciclo de vida:

### onCreate()

O sistema o chama ao criar o fragmento. Dentro da implementação, deve-se inicializar os componentes essenciais do fragmento que deseja-se reter quando o fragmento for pausado ou interrompido e, em seguida, retomado.

### onCreateView()

O sistema chama isto quando é o momento de o fragmento desenhar a interface do usuário pela primeira vez. Para desenhar uma IU para o fragmento, você deve retornar uma [View](#) deste método, que é a raiz do layout do fragmento. É possível retornar como nulo se o fragmento não fornecer uma IU.

### onPause()

O sistema chama esse método como o primeiro indício de que o usuário está saindo do fragmento (embora não seja sempre uma indicação de que o fragmento está sendo destruído). É quando geralmente deve-se confirmar qualquer alteração que deva se manter além da sessão atual do usuário (porque o usuário pode não retornar).

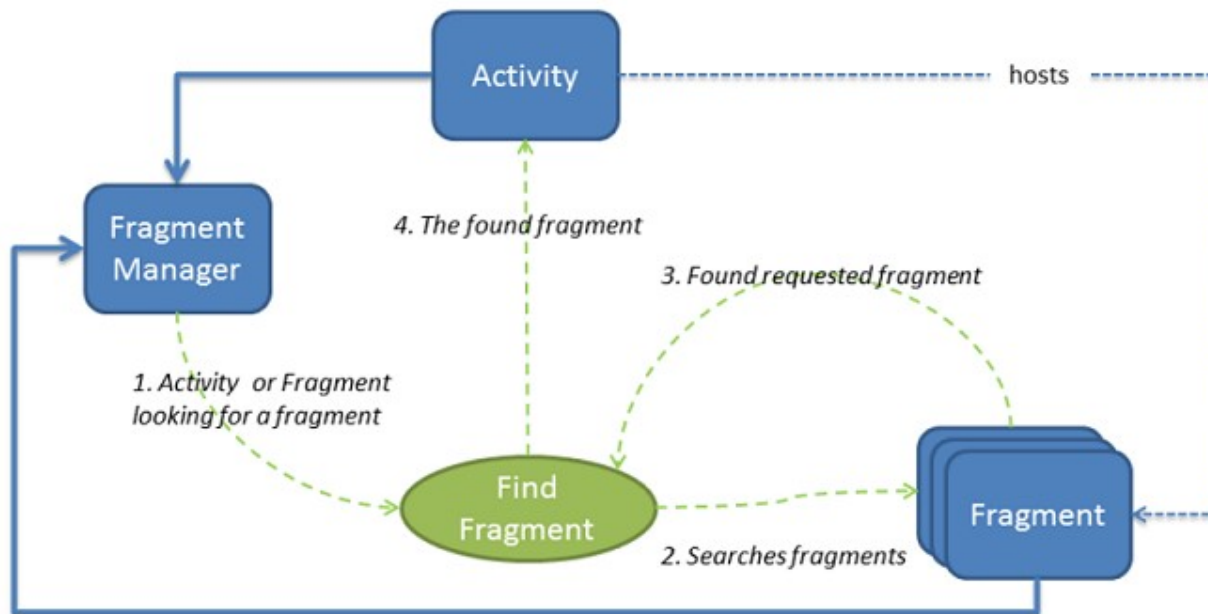
A maioria dos aplicativos deve implementar pelo menos três destes métodos para cada fragmento, mas há vários outros métodos de retorno de chamada que você deve usar para lidar com diversos estágios do ciclo de vida do fragmento.

(fonte <https://developer.android.com/guide/components/fragments.html> acessado em julho de 2016)

Para ajudar uma [Atividade](#) coordenar e administrar todos esses fragmentos, o Android introduziu uma nova classe chamada a **FragmentManager**.

Cada [Atividade](#) tem sua própria instância de um **FragmentManager** para adicionar, excluir e encontrar fragmentos hospedados.

O diagrama a seguir ilustra a relação entre os fragmentos e Atividades:



(Origem : [https://developer.xamarin.com/guides/android/platform\\_features/fragments/](https://developer.xamarin.com/guides/android/platform_features/fragments/))

Para usar **Fragments** em uma aplicação Xamarin Android, é preciso criar um novo tipo de fragmento, e, a seguir, quer seja no layout ou via código, podemos inserir os fragments em seus locais apropriados na interface do usuário.

**Nota :** A versão do Android deve ser a 3.0 ou superior; para versões abaixo disso, é preciso ter instalado o pacote **Xamarin.Android.Support.v4**.

Um fragmento é geralmente usado como parte de uma interface do usuário da atividade e contribui para a atividade com seu próprio layout.

Para criar um **Fragment**, temos que criar uma classe que deve herdar de **Android.App.Fragment** e, em seguida, substituir o método **OnCreateView**.

Assim, para fornecer um layout para um fragmento, você deve implementar o método de retorno de chamada [onCreateView\(\)](#), que o sistema Android chama no momento em que o fragmento deve desenhar o layout. A implementação deste método deve retornar uma [View](#), que é a raiz do layout do fragmento.

Para retornar um layout de [onCreateView\(\)](#), é possível inflá-lo a partir de um [recurso de layout](#) definido no XML. Para ajudar a fazer isto, o [onCreateView\(\)](#) fornece um objeto [LayoutInflater](#).

O método **OnCreateView** será chamado pela Atividade de hospedagem quando for hora de colocar o fragmento na tela, e vai retornar uma View.

Uma implementação típica de **OnCreateView** vai criar esta **View** inflando um arquivo

de layout e, em seguida, vai anexá-lo a um contêiner pai.

As características do contêiner são importantes visto que o Android vai aplicar os parâmetros do layout do pai na interface do usuário do **Fragment**.

Podemos incluir um **Fragment** em uma **Activity** de forma declarativa, usando um arquivo **.axml**, e usando a tag **<Fragment>**, ou via código usando a classe **FragmentManager**.

Na forma declarativa a tag **<Fragment>** pode usar o atributo **class** :

```
<?xml version="1.0" encoding="utf-8"?>
<fragment class="com.xamarin.sample.fragments.TitlesFragment"
    android:id="@+id/titles_fragment"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

ou o atributo **android:name** :

```
<?xml version="1.0" encoding="utf-8"?>
<fragment android:name="com.xamarin.sample.fragments.TitlesFragment"
    android:id="@+id/titles_fragment"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

As duas abordagem identificam uma classe **Fragment**.

A cada **Fragment** deve ser atribuído um identificador exclusivo, e para isso podemos usar:

- **android: id** – Como com outros elementos de interface do usuário em um arquivo de layout, este é um identificador exclusivo;
- **android: tag** - Este atributo é uma string exclusiva;

Se nenhum dos dois métodos acima forem usados, então o **Fragment** irá assumir o **ID** da view Contêiner.

Cada atividade tem uma instância de **Android.App.FragmentManager** que vai encontrar ou dinamicamente alterar seus fragmentos. Cada conjunto dessas mudanças é conhecida como uma transação, e é realizada usando uma das APIs contidas na classe **Android.App.FragmentTransation** classe, que é gerida pela classe **FragmentManager**.

A classe **FragmentManager** localiza os **Fragments** anexados a uma Activity usando

os métodos **FindFragmentById** ou **FindFragmentByTag**.

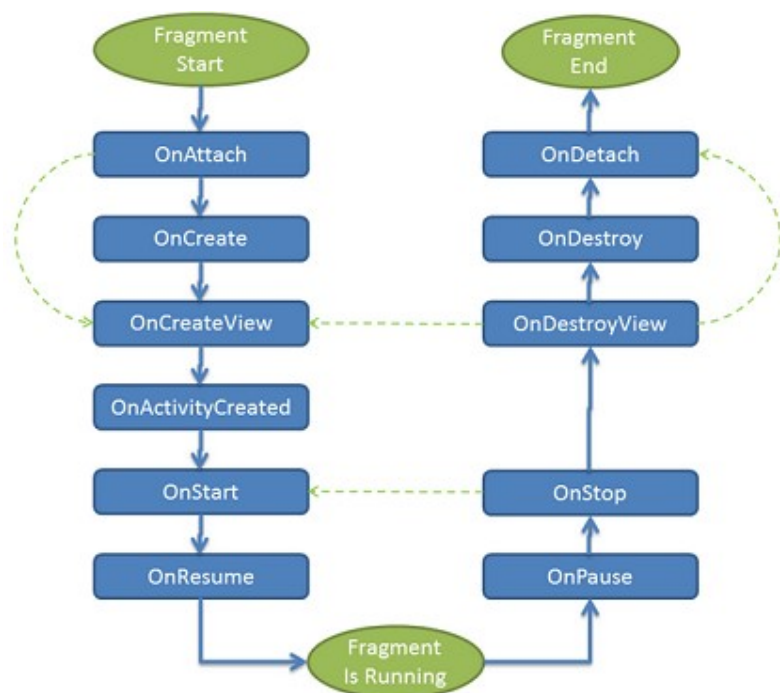
Quando a Activity está sendo criada, o Android vai instanciar cada fragmento especificado no arquivo de layout e inserir a View que é criada a partir de **OnCreateView** no lugar do elemento **Fragment**.

Os **Fragments** que são adicionados declarativamente a uma atividade são estáticos e permanecerão na atividade até que ele for destruída; Não é possível dinamicamente, substituir ou remover o fragmento durante o tempo de vida da atividade ao qual está anexado.

Os **Fragments** têm o seu próprio ciclo de vida que é um pouco independente, mas ainda é afetado pelo ciclo de vida da atividade de hospedagem.

Por exemplo, quando uma atividade pausa, todos seus fragmentos associados são pausados.

O diagrama a seguir descreve o ciclo de vida de um Fragment:



(fonte : [https://developer.xamarin.com/guides/android/platform\\_features/fragments/part\\_1\\_-\\_creating\\_a\\_fragment/](https://developer.xamarin.com/guides/android/platform_features/fragments/part_1_-_creating_a_fragment/) )

Os Fragments podem salvar e restaurar seu estado durante o ciclo de vida usando uma instância de um *Bundle*. O **Bundle** permite a um Fragment salvar dados como pares de chave/valor e é útil para dados simples que não requeem muita memória. Um Fragment pode salvar seu estado com uma chamada para **onSaveInstanceState()**.

**Nota :** Quando uma Activity salva o seu estado o Android salva o estado de quaisquer

## *Fragments hospedados.*

A **API Fragments** fornece outras subclasses que encapsulam algumas das funcionalidades mais comuns encontradas em aplicações. Estas subclasses são as seguintes:

- **ListFragment** - Este fragmento é usado para exibir uma lista de itens ligados a uma fonte de dados, como uma matriz ou um cursor;
- **DialogFragment** - Este fragmento é usado como um invólucro em torno de uma caixa de diálogo. O fragmento irá exibir o diálogo no topo da sua atividade;
- **PreferenceFragment** - Este fragmento é utilizado para mostrar objetos **Preference** como listas;

Depois dessa teoria, vamos ver então como usar **Fragments** na prática mas vou deixar isso para a [segunda parte o artigo](#).

[Veja os Destaques e novidades do SUPER DVD Visual Basic \(sempre atualizado\) : clique e confira !](#)

### Quer migrar para o VB .NET ?

- Veja mais sistemas completos para a plataforma .NET no [Super DVD .NET](#) , confira...
- [Curso Básico VB .NET - Vídeo Aulas](#)

### Quer aprender C# ??


- Chegou o [Super DVD C#](#) com exclusivo material de suporte e vídeo aulas com curso básico sobre C#.
- [Curso C# Basico - Video Aulas](#)

### Quer aprender os conceitos da Programação Orientada a objetos ?

- [Curso Fundamentos da Programação Orientada a Objetos com VB .NET](#) NEW

### Quer aprender o gerar relatórios com o

## ReportViewer no VS 2013 ?

- [Curso - Gerando Relatórios com o ReportViewer no VS 2013 - Vídeo Aulas](#) 

### 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](#) 
- [\*\*Macoratti .net | Facebook\*\*](#)
- [\*\*macoratti - YouTube\*\*](#)
- [\*\*Jose C Macoratti \(@macoratti\) | Twitter\*\*](#)
- [Xamarin - 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/>

---

[José Carlos Macoratti](#)