10

Converters

Transcrição

O layout da aplicação ainda está estranho, com os botões "Salvar" e "Editar" sendo exibidos ao mesmo tempo. Queremos que quando o perfil estiver aberto para edição, exista apenas a opção de "Salvar" e, quando os campos estiverem travados para edição, que somente o botão "Editar" esteja habilitado.

Para que ambos os botões nunca estejam habilitados ao mesmo tempo, iremos ao código XAML e, no botão "Salvar", aproveitaremos a propriedade `Editando` que já utilizamos para habilitar ou desabilitar os campos de entrada. Nosso objetivo é deixar o botão "Salvar" invisível enquanto o usuário não estiver editando.

A propriedade `IsVisible` será usada amarrada a outra (da `ViewModel`), chamada `Editando`, a partir do `Binding`.

```
<Button Text="Salvar" Command="{Binding SalvarCommand}" IsVisible="{Binding Editando}"></Button
```

Rodando a aplicação, veremos que o botão "Salvar" sumiu da segunda aba. Quando clicamos no botão "Editar", no entanto, ele volta a ser exibido e, quando apertado, somos levados à primeira aba. Neste momento, ao clicarmos em "Salvar", queremos que a app volte a travar os campos para edição. Faremos isto modificando a propriedade `Editando` no `ViewModel`. Clicando-se em "Salvar", o `this.Editando` deve ser `false`:

```
SalvarCommand = new Command(() =>
{
    this.Editando = false;
    MessagingCenter.Send<Usuario>(usuario, "SucessoSalvarUsuario");
});
```

Verificando-se a aplicação pelo emulador, veremos que, ao entrarmos no perfil de usuário e clicarmos em "Editar", os campos de edição são habilitados, e o botão "Salvar" é exibido. Porém, como já falamos, queremos que apenas este botão esteja visível, e não o "Editar".

Vamos modificar o `IsVisible` deste botão, acessando-se a propriedade `Editando` e pegando seu valor invertido: quando ele for `true`, o `IsVisible` deverá ser `false`, e vice-versa.

Então, em `MasterView.xaml`, faremos uma **conversão de valores**, de `false` para `true` e de `true` para `false`. No XAML, existem os conversores, os *converters* do Xamarin Forms. Para realizar esta "tradução" de valores, criaremos uma pasta, clicando com o lado direito do mouse em "TestDrive (Portable)" no menu lateral do programa, selecionando "*Add* > *New Folder*".

Esta nova pasta se chamará "Converters", em que adicionaremos uma nova classe, clicando com o lado direito do mouse em cima da pasta, e em "*Add* > *Class*".

A nova classe será denominada `NegativoConverter`, e ela implementará uma interface chamada `IValueConverter`. Nesta nova classe, portanto, teremos:

```
class NegativoConverter : IValueConverter
{
```

```
}
```

Clicaremos em `IValueConverter` com o lado direito do mouse, e em "*Quick Actions and Refactorings*" e "*Implement interface*". Aparecerão dois métodos novos, que recebem como argumento um objeto de nome `value`. Precisaremos simplesmente retornar, no corpo do método, este valor convertido em `boolean`, invertendo seu valor por meio do negativo, adicionando `!`.

Da mesma forma, há o método `Convert` para conversão do valor, e um método que o converte de volta (`ConvertBack`), fazendo o caminho inverso. Dentro de cada objeto, então, acrescentaremos a linha abaixo:

```
return !(bool)value;
```

Como queremos aplicar este conversor no código XAML, abriremos `MasterView.xaml`, em que faremos o `Binding` da propriedade `Visible` do botão "Editar", utilizando um conversor. No Xamarin Forms há uma sintaxe que permite o uso de conversores dentro de um `Binding`. Quando colocamos uma vírgula (`,`) dentro dele, aparece automaticamente uma opção chamada `Converter`, a ser setada de acordo com o conversor usado.

```
<Button Text="Editar" Command="{Binding EditarCommand}" IsVisible="{Binding Editando, Converter
```

No entanto, este conversor ainda não foi declarado no código XAML, portanto não poderemos usá-lo aqui, já que ele não será encontrado. Em `TabbedPage`, definiremos alguns recursos (`Resources`) para a declaração do conversor. Nele, precisaremos configurar também um dicionário de recursos, `ResourceDictionary`.

Ao tentarmos digitar `NegativoConverter`, ele não aparecerá como uma das opções do Xamarin. Isto ocorre porque seu *namespace* não é reconhecido.

Como já vimos anteriormente, para resolver isto basta declararmos o *namespace* do XML, incluindo `xmlns:converters="clr-namespace:TestDrive.Converters"` dentro da tag `TabbedPage` localizada no início do arquivo `MasterView.xaml`. A partir disto, colocaremos como prefixo a palavra `converters` que acabamos de criar.

Agora criaremos uma chave para este conversor, a ser utilizada no `Binding`, para sua identificação.

```
<TabbedPage.Resources>
  <ResourceDictionary>
      <converters:NegativoConverter x:Key="negativo"></converters:NegativoConverter>
  </ResourceDictionary>
</TabbedPage.Resources>
```

Voltando ao `Binding`, abriremos novas chaves para declarar um recurso estático da página, `StaticResource`:

```
<Button Text="Editar" Command="{Binding EditarCommand}" IsVisible="{Binding Editando, Converter
```

Rodaremos a aplicação. Clicando em "Perfil" após o login, somos direcionados à página "Editar" (a segunda aba), com os campos de preenchimento de dados bloqueado. Apertando-se "Editar", o único botão visível, os mesmos campos estarão

habilitados, e apenas o botão "Salvar" será visível. Ao clicarmos nele, os dados alterados serão salvos e vamos retornamos à aba "Usuário".

A navegação entre abas está funcionando como gostaríamos, e aprendemos a usar um conversor para a troca de valores utilizáveis no código XAML.

Assim, temos infinitas possibilidades: podemos converter valores numéricos para cores, por exemplo, data para `string`, e vice-versa. Os conversores são uma maneira muito poderosa de auxiliar o `Binding` a fazer estas conversões "malucas" de que precisamos no dia a dia.