

## 39. Enlace a datos

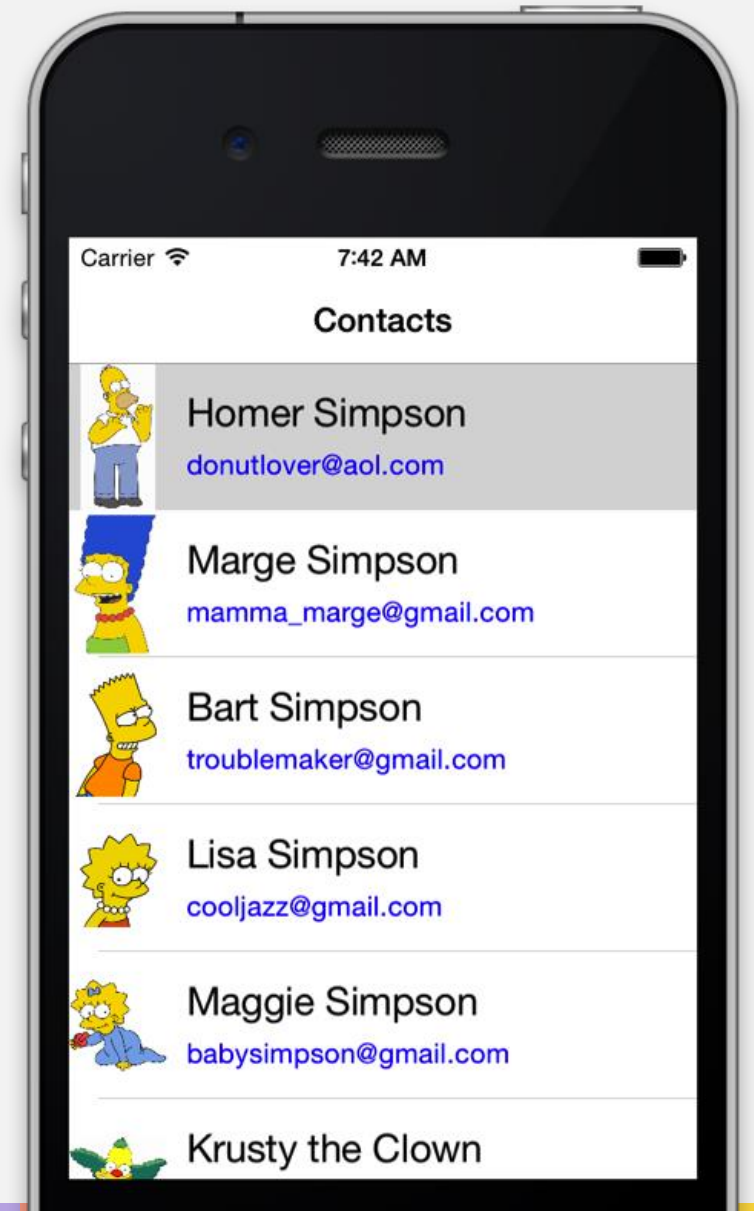
# Las aplicaciones son impulsadas por datos

❖ La mayoría de las aplicaciones muestran y manipulan datos de alguna forma.

- generado internamente
- leer de una fuente externa

❖ Las clases creadas para representar datos a menudo se denominan modelos.

- también puede referirse a objetos de tipo "entidad"



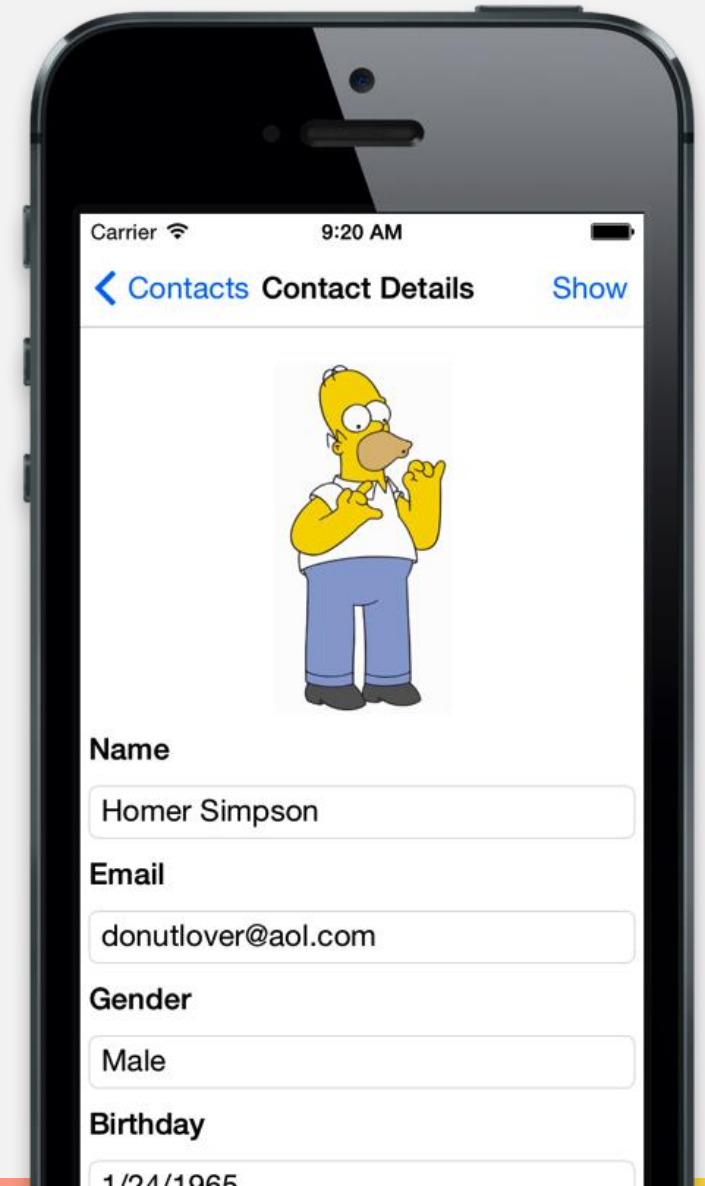
# Data > Views

Usamos código para mostrar datos internos en nuestras páginas.

```
headshot.Source = ...;  
nameEntry.Text = person.Name;  
emailEntry.Text = person.Email;  
birthday.Date = person.Dob;  
...
```

... y eventos para proporcionar interactividad/comportamiento.

```
nameEntry.TextChanged += (sender, e) =>  
    person.Name = nameEntry.Text;  
emailEntry.TextChanged += (sender, e) =>  
    person.Email = emailEntry.Text;
```

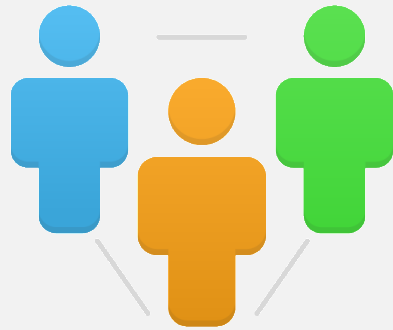


# Data > Views en código

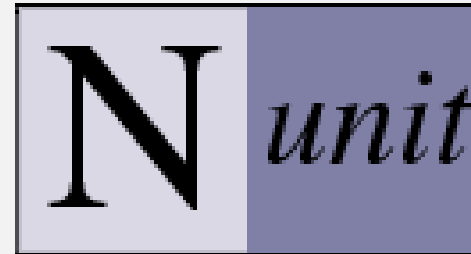
Este enfoque funciona, y para aplicaciones pequeñas es perfectamente adecuado, pero tiene desventajas a medida que la aplicación crece en complejidad.



Las actualizaciones de los datos no están centralizadas



Las relaciones en los datos o el comportamiento de la interfaz de usuario son más difíciles de administrar



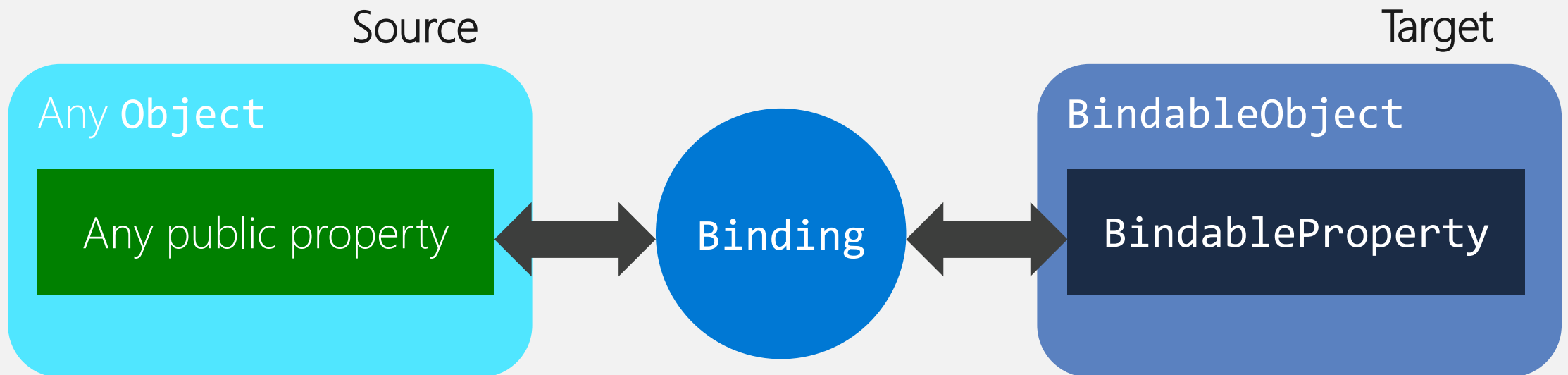
Difícil de testear



La interfaz de usuario está estrechamente acoplada al código detrás de la lógica, los cambios se transmiten a través del código

# Introducción: Data Binding

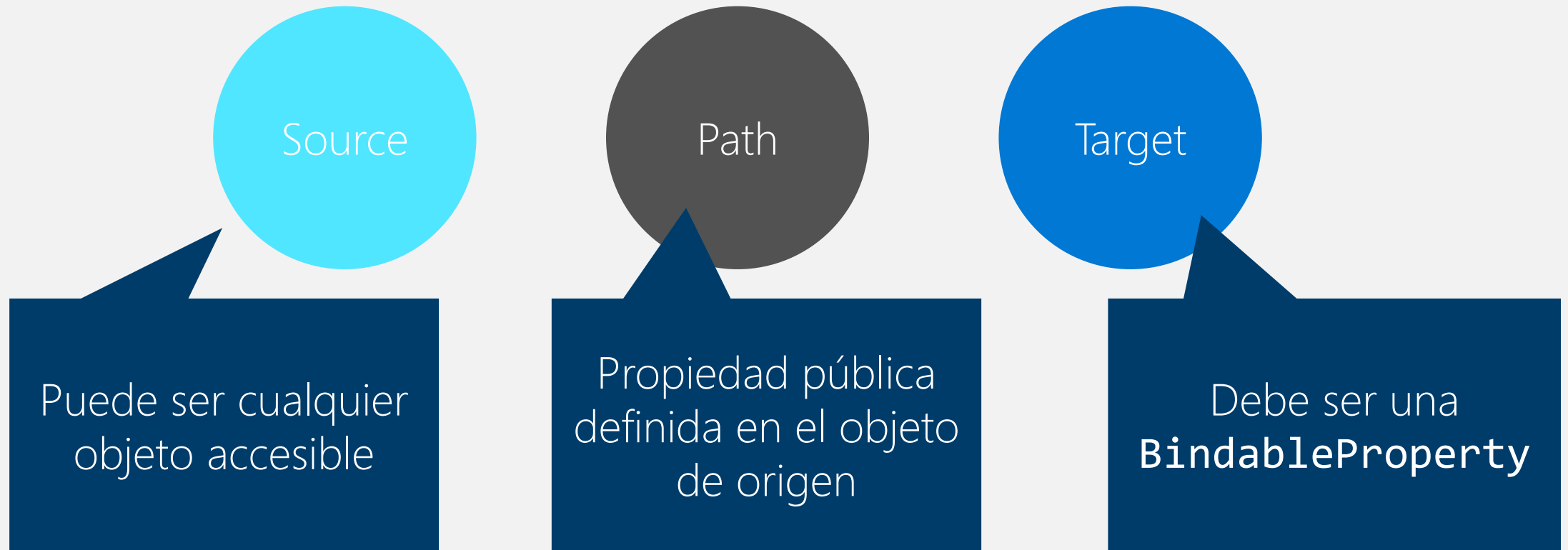
El enlace de datos implica crear una relación flexible entre una propiedad de origen (Source) y una propiedad de destino (Target) para que el origen y el destino no se conozcan entre sí.



El enlace actúa como intermediario: mueve los datos entre el origen y el destino

# Creando Bindings en .NET MAUI

Los enlaces requieren **tres piezas** de información



# Creando bindings [Source]

```
Person person = new Person() { Name = "Homer Simpson", ... };
```

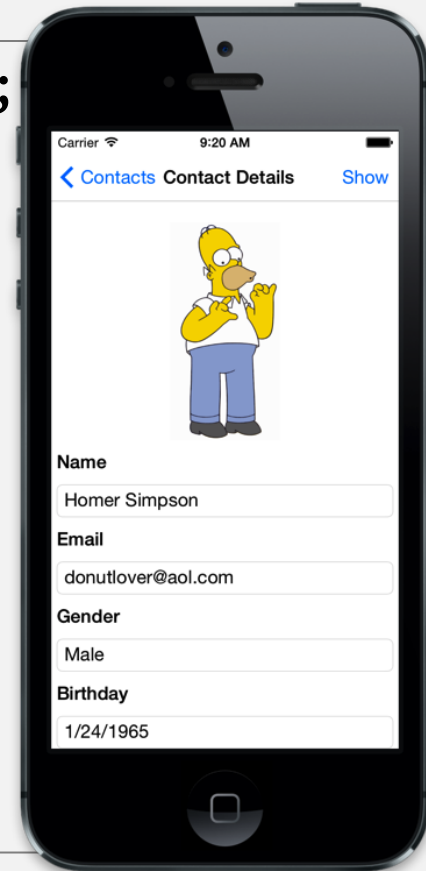
```
Entry nameEntry = new Entry();
```

1 

```
Binding nameBinding = new Binding();  
nameBinding.Source = person;
```

...

**Binding** identifica la fuente de los datos: aquí es de donde provienen los datos, en este caso es una sola persona definida en nuestra aplicación.



# Creando bindings [Path]

```
Person person = new Person() { Name = "Homer Simpson", ... };
```

```
Entry nameEntry = new Entry();
```

```
Binding nameBinding = new Binding();
```

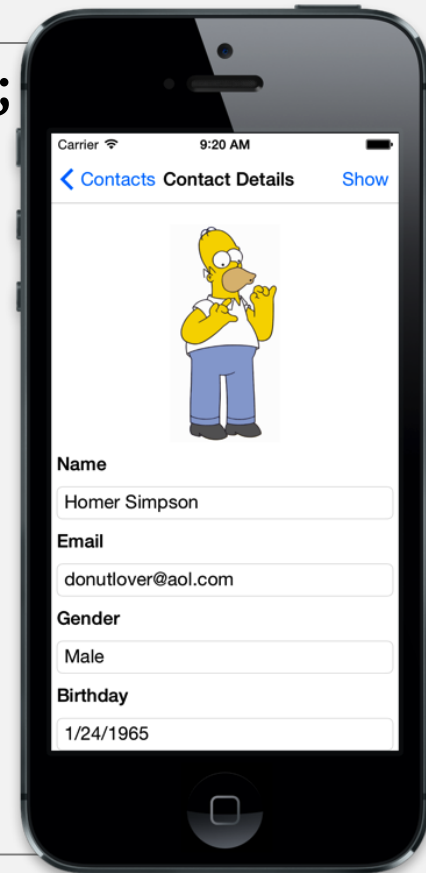
```
nameBinding.Source = person;
```

2

```
nameBinding.Path = "Name";
```

...

**Binding** identifica la ruta de la propiedad que identifica una propiedad en la fuente de donde obtener los datos, en este caso queremos obtener el valor de la propiedad **Person.Name**



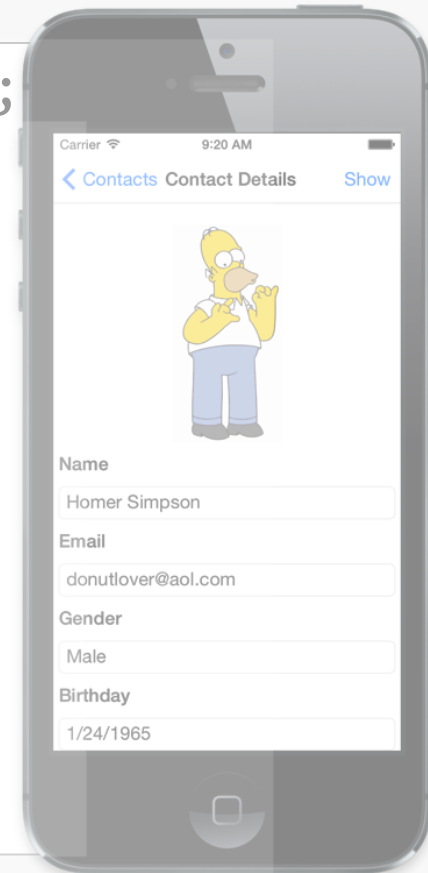


# Creando bindings [Path]

```
Person person ... };  
Entry nameEnt ...  
Binding nameB ...  
nameBinding.S ...  
nameBinding.P ...  
...
```

## More Path Examples

```
new Binding("Property")  
new Binding("Property.Child")  
new Binding("Property[Key]")  
new Binding("Property[1]")  
new Binding("Item[Key]")  
new Binding(".")
```



# Creando bindings [Target]

```
Person person = new Person() { Name = "Homer Simpson", ... };
```

```
Entry nameEntry = new Entry();
```

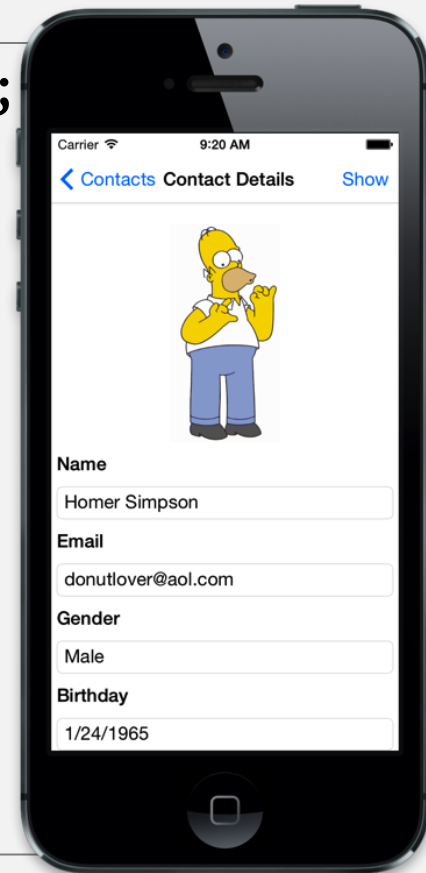
```
Binding nameBinding = new Binding();
```

```
nameBinding.Source = person;
```

```
nameBinding.Path = "Name";
```

3 `nameEntry.SetBinding(Entry.TextProperty, nameBinding);`

Binding asocia a la propiedad de destino mediante el método `BindableObject.SetBinding`



# Creando bindings [Target]

```
Person person = new Person() { Name = "Homer Simpson", ... };
```

```
Entry nameEntry = new Entry();
```

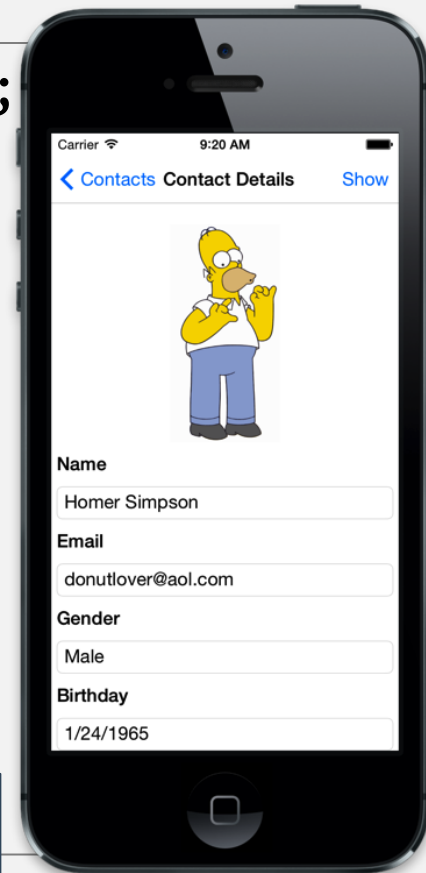
```
Binding nameBinding = new Binding();
```

```
nameBinding.Source = person;
```

```
nameBinding.Path = "Name";
```

3 nameEntry.SetBinding(Entry.TextProperty, nameBinding);

Esto se pasa a la propiedad de destino específica con la que funcionará el enlace; debe ser **BindableProperty**



# Creando bindings [Target]

```
Person person = new Person() { Name = "Homer Simpson", ... };
```

```
Entry nameEntry = new Entry();
```

```
Binding nameBinding = new Binding();
```

```
nameBinding.Source = person;
```

```
nameBinding.Path = "Name";
```

3 `nameEntry.SetBinding(Entry.TextProperty, nameBinding);`

... y el enlace que identifica la fuente y la propiedad en la fuente para aplicar



# Creando bindings [XAML]

Crear enlaces en XAML con la extensión de marcado **{Binding}**

```
<StackLayout Padding="20" Spacing="20">
  <StackLayout.Resources>
    <ResourceDictionary>
      <Person x:Key="homer" Name="Homer Simpson" .../>
    </ResourceDictionary>
  </StackLayout.Resources>
  <Entry Text="{Binding Name,
    Source={StaticResource homer}}" />
  ...
</StackLayout>
```

Asignado a la propiedad Target

{Binding} toma Path  
como el primer  
argumento sin nombre

Source suministrado a  
través del recurso

# Data binding source

- ❖ Las páginas a menudo muestran propiedades de una pequeña cantidad de objetos de datos
- ❖ Puede establecer la fuente de enlace a datos en cada vinculación por separado, o usar `BindingContext` como la fuente de vinculación predeterminada

```
public class Person
{
    public string Name { get; set; }
    public string Email { get; set; }
    public Gender Gender { get; set; }
}
```

**Name**

Homer Simpson

**Email**

donutlover@aol.com

**Gender**

Male



# Multiple Bindings

**BindingContext** proporciona el origen de cualquier enlace asociado con una vista cuando la propiedad **Binding.Source** no está establecida

```
Person person = new Person() { Name = "Homer Simpson", ... };  
...  
nameEntry.BindingContext = person;  
nameEntry.SetBinding<Person>(Entry.TextProperty,  
                             data => data.Name, BindingMode.TwoWay);
```

# BindingContext



# BindingContext

**BindingContext** se hereda automáticamente de padre a hijo: puede configurarlo una vez en la vista raíz y se usará para todos los hijos.

```
public partial class PersonDetailsPage : ContentPage
{
    public PersonDetailsPage (Person person)
    {
        BindingContext = person;
        InitializeComponent ();
    }
}
```

The diagram illustrates the data binding process. A box in the code snippet highlights the line `BindingContext = person;`. Five purple arrows originate from this box and point to the corresponding UI controls in the form on the right: the Name text box, the Email text box, the Gender text box, the Birthday text box, and the Favorite toggle switch.


<b>Name</b>	<input type="text" value="Homer Simpson"/>
<b>Email</b>	<input type="text" value="donutlover@aol.com"/>
<b>Gender</b>	<input type="text" value="Male"/>
<b>Birthday</b>	<input type="text" value="1/24/1965"/>
<b>Favorite</b>	<input checked="" type="checkbox"/>

# BindingContext inheritance

**BindingContext** se hereda automáticamente de padre a hijo: puede configurarlo una vez en la vista raíz y se usará para todos los hijos.

```
BindingContext = person;
```

```
<StackLayout Padding="20" Spacing="20">  
  <Entry Text="{Binding Name}" />  
  <Entry Text="{Binding Email}" />  
  <Label Text="{Binding Gender}" />  
  <Label Text="{Binding Dob}" />  
</StackLayout>
```

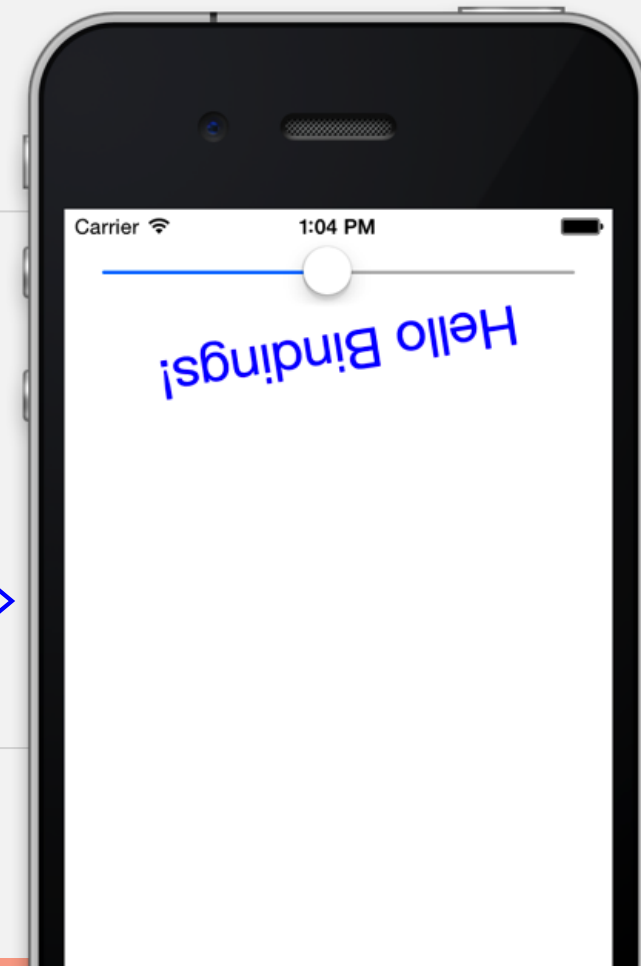


Al establecer el contexto de vinculación a **Person**, no se necesita una fuente explícita en XAML.

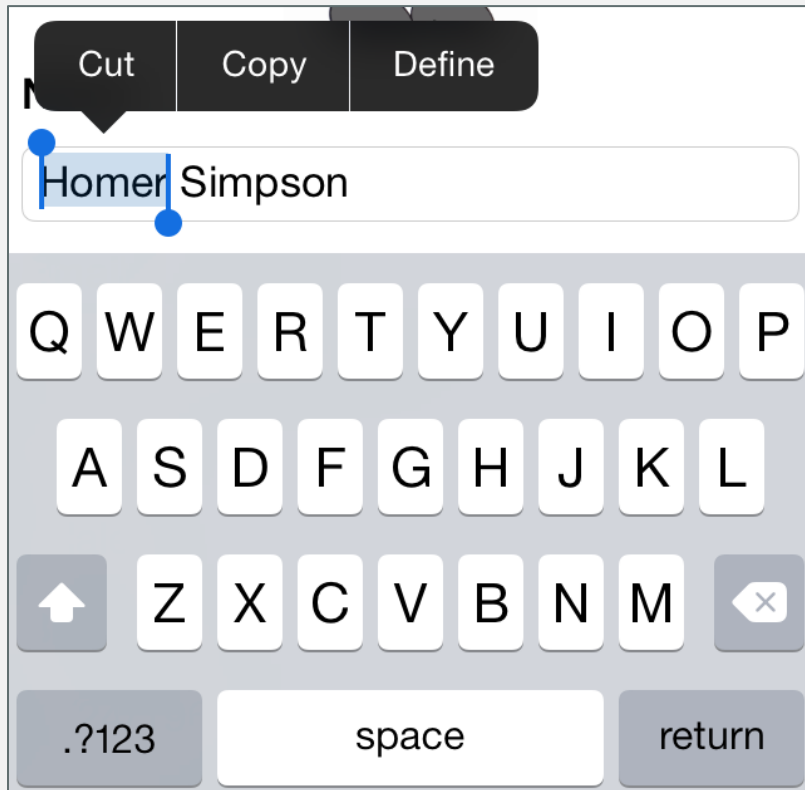
# View-to-View Bindings

- {x:Reference} identifica elementos con nombre en la misma página XAML; puede usar esto para proporcionar una fuente a un enlace.

```
<StackLayout Padding="20" Spacing="20">  
    <Label Text="Hello, Bindings" TextColor="Blue" ...  
        Rotation="{Binding Source={x:Reference slider},  
            Path=Value}" />  
    ...  
    <Slider x:Name="slider" Minimum="0" Maximum="360" />  
</StackLayout>
```



# Creando two-way bindings



Por lo general, desea que los datos sean bidireccionales

- `source > target` (siempre ocurre)
- `target > source` (opcional)

```
nameEntry.TextChanged += (sender, e)
    => person.Name = nameEntry.Text;
```

# Binding Mode

Binding **Mode** controla la dirección de la transferencia de datos, se puede establecer en "TwoWay" para habilitar los enlaces bidireccionales.

```
name.SetBinding(Entry.TextProperty,  
    new Binding("Name") {  
        Mode = BindingMode.TwoWay  
    });
```

Gestionado a través de la propiedad  
Binding.Mode



La propiedad **Source** debe  
tener un setter público



```
<Entry  
    Text="{Binding Name, Mode=TwoWay}" />
```



# Default Binding Mode

El modo de enlace predeterminado es específico de cada propiedad, la mayoría son unidireccionales de forma predeterminada con algunas excepciones que por defecto son bidireccionales

`DatePicker.Date`

`Entry.Text`

`ListView.SelectedItem`

`MultiPage<T>.SelectedItem`

`Picker.SelectedIndex`

`SearchBar.Text`

`Stepper.Value`

`Switch.IsToggled`

`TimePicker.Time`

# Notificando cambios a la UI

Los enlaces unidireccionales y bidireccionales siempre actualizan la interfaz de usuario cuando se cambia la propiedad de origen

```
person.Email = "homer@dunkin.com";
```

```
public class Person
{
    public string Name { get; set; }
    public string Email { get; set; }
    public Gender Gender { get; set; }
    public DateTime Dob { get; set; }
    public bool IsFavorite { get; set; }
}
```

**Q:** Cómo sabe  
.NET MAUI que la  
propiedad ha  
cambiado?



<b>Name</b>	<input type="text" value="Homer Simpson"/>
<b>Email</b>	<input type="text" value="donutlover@aol.com"/>
<b>Gender</b>	<input type="text" value="Male"/>
<b>Birthday</b>	<input type="text" value="1/24/1965"/>
<b>Favorite</b>	<input checked="" type="checkbox"/>

# INotifyPropertyChanged

INotifyPropertyChanged proporciona un contrato de notificación de cambio, debe ser implementado por cualquier objeto de modelo modificable al que se vincule.

```
namespace System.ComponentModel
{
    public interface INotifyPropertyChanged
    {
        event PropertyChangedEventHandler PropertyChanged;
    }
}
```



# Implementing INotifyPropertyChanged

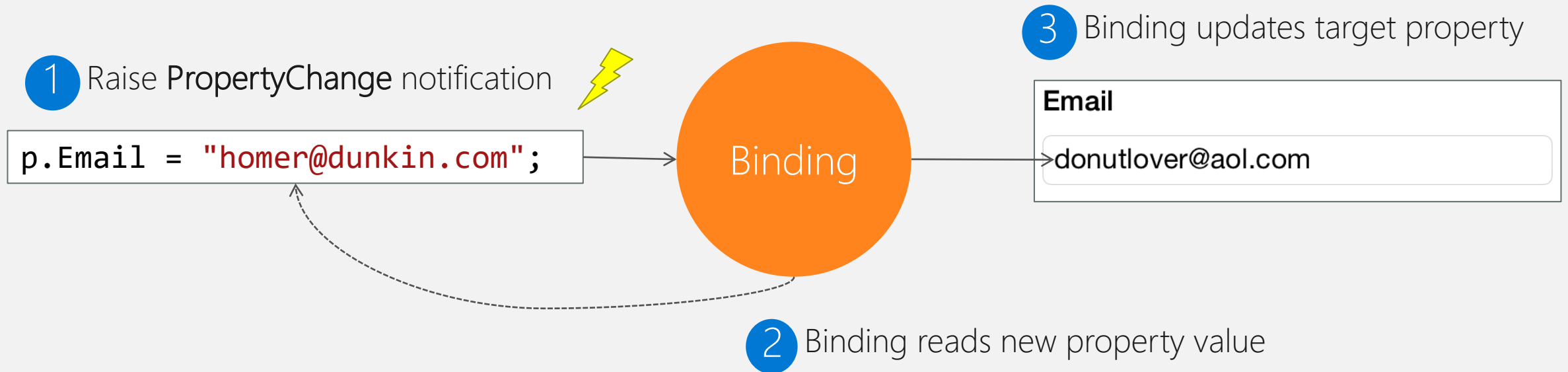
```
public class Person : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged = delegate {};

    string email;
    public string Email {
        get { return email; }
        set {
            if (email != value) {
                email = value;
                PropertyChanged(this,
                    new PropertyChangedEventArgs("Email"));
            }
        }
    }
}
```

Debe generar el evento **PropertyChanged** cuando se cambia cualquier propiedad; de lo contrario, la interfaz de usuario no se actualizará.

# INPC + Bindings

- Binding will subscribe to the PropertyChanged event and update the target property when it sees the source property notification



# Conversión de tipos

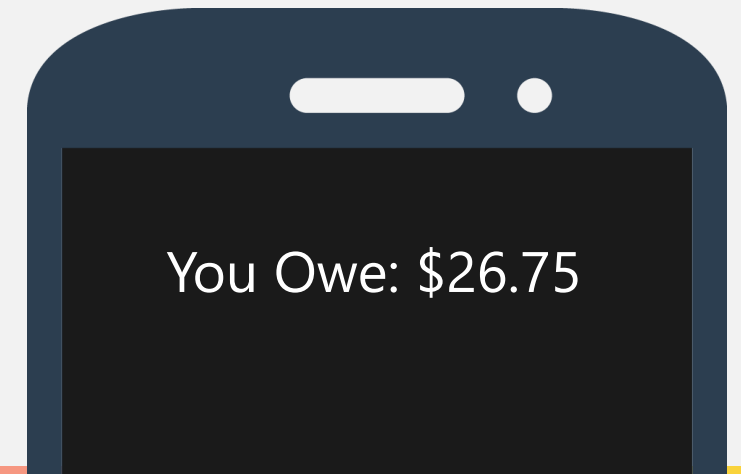
# Simple Textual Conversions

El enlace puede usar un formato de tipo cadena cuando se va desde Fuente > Destino

```
public double BillAmount { get; set; }
```

```
<Label Text="{Binding BillAmount,  
StringFormat='You Owe: {0:C}'}"/>
```

Binding llama a **String.Format** pasando la cadena de formato especificada y el valor de origen antes de asignarlo al destino



# Going beyond textual formatting

Los enlaces intentan coaccionar automáticamente los datos cuando C# lo permitiría, pero a veces los datos disponibles no son exactamente lo que la interfaz de usuario necesita mostrar.

Enter Password

Invalid

Enter Password

Excellent

Quiere que el color del texto cambie según la seguridad de la contraseña

```
<Label Text="{Binding PasswordStrength}"  
       TextColor="{Binding PasswordStrength}"  
       FontSize="24" />
```

# Value Converters

- ❖ Los **Converters** de valor permiten la conversión de tipos y el formateo
- ❖ Asignado a la propiedad Converter de Binding
- ❖ Admite parámetros opcionales (Binding.ConverterParameter)

```
public interface IValueConverter
{
    object Convert(object value,
                   Type targetType,
                   object parameter,
                   CultureInfo culture);

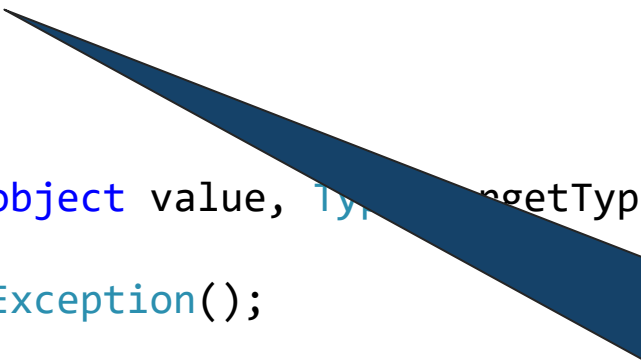
    object ConvertBack(object value,
                      Type targetType,
                      object parameter,
                      CultureInfo culture);
}
```

Convert usado para source ➡ target  
ConvertBack usado para target ➡ source

# Creando un Converter

```
public class PWStrengthConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        PasswordStrength pwdstr = (PasswordStrength) value;
        ...
        return Color.Red;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotSupportedException();
    }
}
```



El convertidor realiza cualquier traducción que sea necesaria para proporcionar datos al objetivo: ¡pueden ser conversiones simples o incluso objetos completamente diferentes!

# Creando un Converter

```
public class PWStrengthConverter :
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        PasswordStrength pwdstr =
        ...
        return Color.Red;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotSupportedException();
    }
}
```

Proporciona conversión hacia atrás para el enlace bidireccional, o puede generar una excepción si esto no es compatible; esto provocará una falla en el tiempo de ejecución

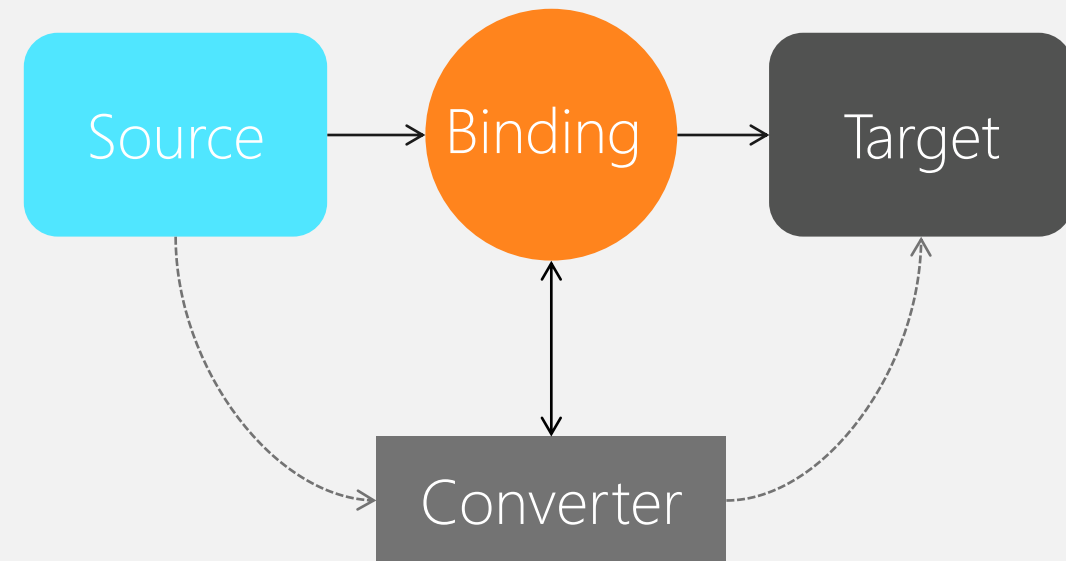


# Usando un Converter

El Converter se asigna a la propiedad de enlace **Converter**

```
var binding = new Binding("PasswordStrength"){  
    Converter = new PWStrengthConverter()  
};
```

```
<ContentPage.Resources>  
    <ResourceDictionary>  
        <c:PWStrengthConverter x:Key="pwsCvt"/>  
    </ResourceDictionary>  
</ContentPage.Resources>  
  
<Label TextColor="{Binding PasswordStrength,  
    Converter={StaticResource pwsCvt}}" />
```



El enlace pasa los valores a través del convertidor