Fonts

Setting Fonts in Xamarin.Forms

This article describes how Xamarin.Forms lets you specify font attributes (including weight and size) on controls that display text. Font information can be specified in code or specified in Xaml. It is also possible to use a custom font.

Setting Font in Code

Use the three font-related properties of any controls that display text:

- **FontFamily** – the `string` font name.
- **FontSize** – the font size as a `double`.
- **FontAttributes** – a string specifying style information like *Italic* and **Bold** (using the `FontAttributes` enumeration in C#).

This code shows how to create a label and specify the font size and weight to display:

```
var about = new Label {
    FontSize = Device.GetNamedSize (NamedSize.Medium,
typeof(Label)),
    FontAttributes = FontAttributes.Bold,
    Text = "Medium Bold Font"
};
```

Font Size

The `FontSize` property can be set to a double value, for instance:

```
label.FontSize = 24;
```

You can also use the `NamedSize` enumeration which has four built-in options; Xamarin.Forms chooses the best size for each platform.

- **Micro**
- **Small**
- **Medium**
- **Large**

The `NamedSize` enumeration can be used wherever a `FontSize` can be specified using the `Device.GetNamedSize` method to convert the value to a `double`:

```
label.FontSize = Device.GetNamedSize(NamedSize.Small,
typeof(Label));
```

Font Attributes

Font styles such as **bold** and *italic* can be set on the `FontAttributes` property. The following values are currently supported:

- **None**
- **Bold**
- **Italic**

The `FontAttribute` enumeration can be used as follows (you can specify a single attribute or `OR` them together):

```
label.FontAttributes = FontAttributes.Bold | FontAttributes.Italic;
```

FormattedString

Some Xamarin.Forms controls (such as `Label`) also support different font attributes within a string using the `FormattedString` class. A `FormattedString` consists of one-or-more `Span`s, each of which can have its own formatting attributes.

The `Span` class has the following attributes:

- **Text** – the value to display
- **FontFamily** – the font name
- **FontSize** – the font size
- **FontAttributes** – style information like *italic* and **bold**
- **ForegroundColor** – text color
- **BackgroundColor** – background color

An example of creating and displaying a `FormattedString` is shown below – note that it is assigned to the labels' `FormattedText` property and not the `Text` property.

```
var labelFormatted = new Label ();
var fs = new FormattedString ();
fs.Spans.Add (new Span { Text="Red, ", ForegroundColor = Color.Red,
FontSize = 20, FontAttributes = FontAttributes.Italic });
fs.Spans.Add (new Span { Text=" blue, ", ForegroundColor =
Color.Blue, FontSize = 32 });
fs.Spans.Add (new Span { Text=" and green!", ForegroundColor =
Color.Green, FontSize = 12 });
labelFormatted.FormattedText = fs;
```

Setting Font Info Per Platform

Alternatively, the `Device.OnPlatform` method can be used to easily set different font names on each platform, as demonstrated in this code:

```
label.FontFamily = Device.OnPlatform (
    iOS:      "MarkerFelt-Thin"
    Android:  "Droid Sans Mono"
    WinPhone: "Comic Sans MS"
);
label.FontSize = Device.OnPlatform (
    24,
    Device.GetNamedSize (NamedSize.Medium, label),
```

```
    Device.GetNamedSize (NamedSize.Large, label)
);
```

A good source of font information for iOS is iosfonts.com.

Setting the Font in Xaml

Xamarin.Forms controls that display text all have a `Font` property that can be set in Xaml. The simplest way to set the font in Xaml is to use the named size enumeration values, as shown in this example:

```
<Label Text="Login" FontSize="Large"/>
<Label Text="Instructions" FontSize="Small"/>
```

There is a built-in converter for the `Font` property that allows all font settings to be expressed as a string value in Xaml. The following examples show how you can specify font attributes and sizes in Xaml:

```
<Label Text="Italics are supported" FontAttributes="Italic" />
<Label Text="Biggest NamedSize" FontSize="Large" />
<Label Text="Use size 72" FontSize="72" />
```

To specify mulitple `Font` settings, combine the required settings into a single font attribute string. The font attribute string should be formatted as `"[font-face],[attributes],[size]"`. The order of the parameters is important, all parameters are optional, and mulitple `attributes` can be specified, for example:

```
<Label Text="Small bold text" FontAttributes="Bold" FontSize="Micro"
/>
<Label Text="Really big italic text" FontAttributes="Italic"
FontSize="72" />
```

The `FormattedString` class can also be used in XAML, as shown here:

```
<Label>
```

```
    <Label.FormattedText>
        <FormattedString>
            <FormattedString.Spans>
                <Span Text="Red, " ForegroundColor="Red"
FontAttributes="Italic" FontSize="20" />
                <Span Text=" blue, " ForegroundColor="Blue"
FontSize="32" />
                <Span Text=" and green! " ForegroundColor="Green"
FontSize="12"/>
            </FormattedString.Spans>
        </FormattedString>
    </Label.FormattedText>
</Label>
```

Device.OnPlatform can also be used in XAML to render a different font on each platform. The example below uses a custom font face on iOS (MarkerFelt-Thin) and specifies only size/attributes on the other platforms:

```
<Label Text="Login">
    <Label.FontFamily>
        <OnPlatform x:TypeArguments="x:String">
            <OnPlatform.iOS>MarkerFelt-Thin</OnPlatform.iOS>
            <OnPlatform.Android></OnPlatform.Android>
            <OnPlatform.WinPhone></OnPlatform.WinPhone>
        </OnPlatform>
    </Label.FontFamily>
</Label>
```

When specifying a custom font face, it is always a good idea to use `OnPlatform`, as it is difficult to find a font that is available on all platforms.

Using a Custom Font

Using a font other than the built-in typefaces requires some platform-specific coding. This screenshot shows the custom font **Lobster** from [Google's open-source fonts] rendered on iOS, Android and Windows Phone using Xamarin.Forms.

The steps required for each platform are outlined below. When including custom font files with an application, be sure to verify that the font's license allows for distribution.

iOS

It is possible to display a custom font by first ensuring that it is loaded, then referring to it by name using the Xamarin.Forms `Font` methods. Follow the instructions in [this blog post]:

1. Add the font file with **Build Action: BundleResource**, and
2. Update the **Info.plist** file (**Fonts provided by application**, or `UIAppFonts`, key), then
3. Refer to it by name wherever you define a font in Xamarin.Forms!

```
new Label {
    Text = "Hello, Forms!",
    FontFamily = Device.OnPlatform (
        "Lobster-Regular",
        null,
        null
    ), // set only for iOS
}
```

Android

Xamarin.Forms for Android can reference a custom font that has been added to the project by following a specific naming standard. First add the font file to the **Assets** folder in the

application project and set *Build Action: AndroidAsset*. Then use the full path and *Font Name* separated by a hash (#) as the font name in Xamarin.Forms, as the code snippet below demonstrates:

```
new Label
{
  Text = "Hello, Forms!",
  FontFamily = Device.OnPlatform(
    null,
    "Lobster-Regular.ttf#Lobster-Regular", // Android
    null
  )
}
```

Windows

Xamarin.Forms for Windows platforms can reference a custom font that has been added to the project by following a specific naming standard. First add the font file to the **/Assets/Fonts/** folder in the application project and set the Build Action:Content. Then use the full path and font filename, followed by a hash (#) and the Font Name, as the code snippet below demonstrates:

```
new Label {
    Text = "Hello, Forms!",
    FontFamily = Device.OnPlatform (
        null,
        null,
        "Assets/Fonts/Lobster-Regular.ttf#Lobster" // Windows
platforms
    )
}
```

ℹ Note that the font file name and font name may be different. To discover the font name on Windows, right-click the .ttf file and select **Preview**. The font name

can then be determined from the preview window.

The common code for the application is now complete. Platform-specific phone dialer code will now be implemented as a DependencyService.

XAML

You can also use `Device.OnPlatform` in XAML to render a custom font:

```
<Label Text="Hello Forms with XAML">
    <Label.FontFamily>
        <OnPlatform x:TypeArguments="x:String">
            <OnPlatform.iOS>Lobster-Regular</OnPlatform.iOS>
            <OnPlatform.Android>Lobster-Regular.ttf#Lobster-
Regular</OnPlatform.Android>
            <OnPlatform.WinPhone>Assets/Fonts/Lobster-
Regular.ttf#Lobster</OnPlatform.WinPhone>
        </OnPlatform>
    </Label.FontFamily>
</Label>
```

Summary

Xamarin.Forms provides simple default settings to let you size text easily for all supported platforms. It also lets you specify font face and size – even differently for each platform – when more fine-grained control is required. The `FormattedString` class can be used to construct a string containing different font specifications using the `Span` class.

Font information can also be specified in Xaml using a correctly formatted font attributes or the `FormattedString` element with `Span` children.