

# Chapter 1: Fonts

You can include fonts in your Adobe® Flex® applications. Although it is easier to use the default device fonts, you can embed other fonts so that you can apply special effects to text-based controls, such as rotating and fading.

## About fonts

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

[<asdoclink>asdoc\\_%Flex20%\\_mx.core.FontAsset](#)

When you compile a Flex application, the application stores the names of the fonts that you used to create the text. Adobe® Flash® Player uses the font names to locate identical or similar fonts on the user's system when the Flex application runs. You can also embed fonts in the Flex application so that the exact font is used, regardless of whether the client's system has that font.

You define the font that appears in each of your components by using the `fontFamily` style property. You can set this property in an external style sheet, an `<fx:Style>` block, or inline. This property can take a list of fonts, as the following example shows:

```
.myClass {
    fontFamily: Arial, Helvetica;
    color: Red;
    fontSize: 22;
    fontWeight: bold;
}
```

If the client's system does not have the first font in the list, Flash Player attempts to find the second, and so on, until it finds a font that matches. If no fonts match, Flash Player makes a best guess to determine which font the client uses.

Fonts are inheritable style properties. So, if you set a font style on a container, all controls inside that container inherit that style, as the following example shows:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/InheritableExample.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
<fx:Style>
  @namespace s "library://ns.adobe.com/flex/spark";

  s|VGroup {
    fontFamily: Helvetica;
    fontSize: 13pt;
  }

  s|HGroup {
    fontFamily: Times;
    fontSize: 13pt;
  }
</fx:Style>
<s:Panel title="Styles Inherited from VGroup Type Selector">
  <s:VGroup>
    <s:Button label="This Button uses Helvetica"/>
    <s:Label text="This Label control is in Helvetica."/>
    <s:RichText height="75" width="200">
      <s:text>
        The text in this RichText control uses the Helvetica font
        because it is inherited from the VGroup style.
      </s:text>
    </s:RichText>
  </s:VGroup>
</s:Panel>
<s:Panel title="Styles Inherited from HGroup Type Selector">
  <s:HGroup>
    <s:Button label="This Button uses Times"/>
    <s:Label text="This Label control is in Times."/>
    <s:RichText height="75" width="200">
      <s:text>
        The text in this RichText control uses the Times font
        because it is inherited from the HGroup style.
      </s:text>
    </s:RichText>
  </s:HGroup>
</s:Panel>
</s:Application>
```

This example defines the HGroup and VGroup type selectors' `fontSize` and `fontFamily` properties. Flex applies these styles to all components in the container that support those properties; in these cases, the Button, Label, and RichText controls.

## Using device fonts

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

You can specify any font for the `fontFamily` property. However, not all systems have all font faces, which can result in an unexpected appearance of your controls. The safest course when specifying font faces is to include a device font as a default at the end of the font list. *Device fonts* do not export font outline information and are not embedded in the SWF file. Instead, Flash Player uses whatever font on the client's local computer most closely resembles the device font.

The following example specifies the device font `_sans` to use if Flash Player cannot find either of the other fonts on the client machine:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/DeviceFont.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>

  <fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    .myClass {
      fontFamily: Arial, Helvetica, "_sans";
      color: Red;
      fontSize: 12;
      fontWeight: bold;
    }
  </fx:Style>
  <s:Panel title="myClass Class Selector with Device Font">
    <s:VGroup styleName="myClass">
      <s:Button label="Click Me"/>
      <s:Label text="This is a Label control."/>
      <s:RichText width="200">
        <s:text>
          The text in the RichText control uses the myClass class selector.
        </s:text>
      </s:RichText>
    </s:VGroup>
  </s:Panel>
</s:Application>
```

**Note:** You must surround device font names with quotation marks when defining them with style declarations.

Flash Player supports three device fonts. The following table describes these fonts:

Font name	Description
<code>_sans</code>	The <code>_sans</code> device font is a sans-serif typeface; for example, Helvetica or Arial.
<code>_serif</code>	The <code>_serif</code> device font is a serif typeface; for example, Times Roman.
<code>_typewriter</code>	The <code>_typewriter</code> device font is a monospace font; for example, Courier.

Using device fonts does not affect the size of the SWF file because the fonts reside on the client. However, using device fonts can affect performance of the application because it requires that Flash Player interact with the local operating system. Also, if you use only device fonts, your selection is limited to three fonts.

## Using embedded fonts

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

`<asdoclink>asdoc_%Flex20%_mx.core.FontAsset`

Rather than rely on a client machine to have the fonts you specify, you can embed TrueType font (TTF) or OpenType font (OTF) families in your Flex application. This means that the font is always available to Flash Player when the application is running, and you do not have to consider the implications of a missing font.

Embedded fonts have the following benefits:

- Client environment does not need the font to be installed.
- Embedded fonts are anti-aliased, which means that their edges are smoothed for easier readability. This is especially apparent when the text size is large.
- Embedded fonts provide smoother playback when zooming.
- Text appears exactly as you expect when you use embedded fonts.
- When you embed a font, you can use the advanced anti-aliasing information that provides clear, high-quality text rendering in SWF files. Using advanced anti-aliasing greatly improves the readability of text, particularly when it is rendered at smaller font sizes. For more information about advanced anti-aliasing, see “[Using advanced anti-aliasing with non-CFF based fonts](#)” on page 9.

Using embedded fonts is not always the best solution, however. Embedded fonts have the following limitations and drawbacks:

- Embed only TrueType or OpenType fonts. To embed other font types such as Type 1 PostScript fonts, embed that font in a SWF file that you create in Flash, and then embed that SWF file in your Flex application. For more information, see [Embedding fonts from SWF files](#).
- Embedded fonts increase the file size of your application, because the document must contain font outlines for the text. This can result in longer download times for your users.
- Embedded fonts, in general, decrease the legibility of the text at sizes smaller than 10 points. All embedded fonts use anti-aliasing to render the font information on the client screen. As a result, fonts might appear fuzzy or illegible at small sizes.
- In some cases, the text that is rendered by embedded fonts can be truncated when they are used in visual components. In these cases, you might be required to change the padding properties of the component by using style properties or subclassing it. This only occurs with some fonts.
- If you use Halo controls in a Flex 4 application, you might have to add additional code to make the Halo control use the embedded font, or embed the font twice: once for the Halo control and once for the Spark control. For more information, see “[Embedding fonts with Halo components](#)” on page 24.

You typically use Cascading Style Sheets (CSS) syntax for embedding fonts in Flex applications. You use the `@font-face` “at-rule” declaration to specify the source of the embedded font and then define the name of the font by using the `fontFamily` property. You typically specify the `@font-face` declaration for each face of the font for the same family that you use (for example, plain, bold, and italic).

You can also embed fonts in ActionScript by using the `[Embed]` metadata tag. As with the `@font-face` declaration, you must specify a separate `[Embed]` tag for each font face.

*Note: Check your font licenses before embedding any font files in your Flex applications. Fonts might have licensing restrictions that preclude them from being stored as vector information.*

If you attempt to embed a font that the Flex compiler cannot find, Flex throws an error and your application does not compile.

## Embedded font syntax

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

To embed TrueType or OpenType fonts, you use the following syntax in your style sheet or `<fx:Style>` tag:

```
@font-face {
    src: url("location");
    fontFamily: alias;
    fontStyle: normal | italic | oblique ;
    fontWeight: normal | bold | heavy ;
    advancedAntiAliasing: true | false;
    embedAsCFF:true|false ;
}
```

The `src` property specifies the location and filename of the font.

The `fontFamily` property sets the alias for the font that you use to apply the font in style sheets. This property is required. If you embed a font with a family name that matches the family name of a system font, the Flex compiler gives you a warning. You can disable this warning by setting the `show-shadows-system-font-warnings` compiler option to `false`.

The `fontStyle` and `fontWeight` properties set the type face values for the font. These properties are optional, unless you are embedding a face that requires them. The default values are `normal`.

The `advancedAntiAliasing` property determines whether to include the advanced anti-aliasing information when embedding the font. This property is optional. This property is ignored if you embed a font with the `embedAsCFF` property set to `true`. You cannot use this option when embedding fonts from a SWF file (see Embedding fonts from SWF files). For more information on using advanced anti-aliasing, see [“Using advanced anti-aliasing with non-CFF based fonts”](#) on page 9.

The `embedAsCFF` (Compact Font Format) property indicates whether to embed an FTE-enabled font for components. Flash Text Engine (FTE) is a library that provides text controls with a rich set of formatting options.

If you set the `embedAsCFF` property to `true`, then the embedded font will let you use the advanced formatting features of FTE such as bidirectional text, kerning, and ligatures. If you set the value of `embedAsCFF` to `false`, then the embedded font will not support FTE, and will work only with the Halo text components. If you use Halo text components in Flex 4 application, you might need to embed the same font multiple times (once with `embedAsCFF` set to `true` and once with `embedAsCFF` set to `false`). The default value is `true`. Alternatively, you can use FTE-based classes for text rendering in your Halo text controls. For more information, see [“Embedding fonts with Halo components”](#) on page 24.

The following example embeds the `MyriadWebPro.ttf` font file:

```
@font-face {
    src: url("../assets/MyriadWebPro.ttf");
    fontFamily: myFontFamily;
    embedAsCFF: true;
}
```

After you embed a font with an `@font-face` declaration, you can use the value of the `fontFamily` property, or *alias*, in a type or class selector. The following example uses `myFontFamily`, the value of the `fontFamily` property, as the font in the `VGroup` type selector:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontFace.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myFontFamily;
      advancedAntiAliasing: true;
      embedAsCFF: true;
    }

    s|VGroup {
      fontFamily: myFontFamily;
      fontSize: 15;
    }
  </fx:Style>
  <s:Panel title="Embedded Font Applied With Type Selector">
    <s:VGroup>
      <!-- This Halo button uses a system font. -->
      <mx:Button label="Click Me"/>
      <!-- This Spark button uses the font of the VGroup container. -->
      <s:Button label="Click Me"/>
      <s:Label text="This is a Label control."/>
      <s:RichText width="250">
        <s:text>
          The text in this RichText control uses the
          font set on the VGroup.
        </s:text>
      </s:RichText>
    </s:VGroup>
  </s:Panel>
  <!-- This button uses the default font because it is not in the VGroup. -->
  <s:Button label="Click Me"/>
</s:Application>
```

When you run this example, you might notice that the Halo Button (in the `mx` namespace) control's label disappears. This is because the default style of a Halo Button control's label uses a bold typeface. However, the embedded font's typeface (Myriad Web Pro) does not contain a definition for the bold typeface. To have the Halo Button control's label use the proper typeface, you can:

- Add `fontWeight:bold` to the `@font-face` rule. This will render the Button label's text, but with a device font.
- Embed a bold typeface so that the label of a Halo Button control is rendered with the correct font.

- Change the Button control's typeface to be non-bold. The Spark Button (in the s namespace) control's label renders with the embedded font because it does not require a bold faced font

For information on embedding bold typefaces, see “Using multiple typefaces” on page 15.

You can also apply the embedded font inline by specifying the alias as the value of the control's `fontFamily` property, as the following example shows:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontFaceInline.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <fx:Style>
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myFontFamily;
      advancedAntiAliasing: true;
    }
  </fx:Style>
  <s:Panel title="Embedded Font Applied Inline">
    <s:VGroup fontFamily="myFontFamily">
      <s:Button label="Click Me"/>
      <s:Label text="This is a label."/>
      <s:RichText width="200">
        <s:text>
          The text in the RichText control is Myriad Web Pro.
        </s:text>
      </s:RichText>
    </s:VGroup>
  </s:Panel>
</s:Application>
```

## Locating embedded fonts

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

The `src` property in the `@font-face` declaration specifies the location of the font family. You use the `src` property to embed a TrueType or OpenType font by location by specifying a valid URI to the font. The URI can be relative (for example, `../fontfolder/akbar.ttf`) or absolute (for example, `c:/myfonts/akbar.ttf`). The URI can also point to a SWF file that has embedded fonts within it.

You must specify the `url` of the `src` property in the `@font-face` declaration. All other properties are optional.

Do not mix embedded and nonembedded fonts in the same `fontFamily` property.

## Embedding fonts in ActionScript

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

You can embed TrueType or OTF font files or system fonts by location or by name by using the `[Embed]` metadata tag in ActionScript. To embed a font by location, you use the `source` property in the `[Embed]` metadata tag. To embed a font by name, you use the `systemFont` property in the `[Embed]` metadata tag.

The `[Embed]` metadata tag takes the same properties that you set as the `@font-face` rule. You separate them with commas. For the list of properties, see “[Embedded font syntax](#)” on page 5

The following examples embed fonts by location by using the `[Embed]` tag syntax:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontFaceActionScriptByLocation.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <fx:Style>
    .mystyle1 {
      fontFamily:myMyriadFont;
      fontSize: 32pt;
    }
    .mystyle2 {
      fontFamily:myBoldMyriadFont;
      fontSize: 32pt;
      fontWeight: bold;
    }
  </fx:Style>

  <fx:Script>
    /*
     * Embed a font by location.
     */
    [Embed(source='../assets/MyriadWebPro.ttf',
      fontFamily='myMyriadFont',
      mimeType='application/x-font',
      cff='true'
    )]
    // You do not use this variable directly. It exists so that
    // the compiler will link in the font.
    private var font1:Class;

    /*
     * Embed a font with bold typeface by location.
     */
    [Embed(source='../assets/MyriadWebPro-Bold.ttf',
      fontWeight='bold',
      fontFamily='myBoldMyriadFont',
      mimeType='application/x-font',
      advancedAntiAliasing='true',
      cff='true'
    )]
  </fx:Script>
</s:Application>
```

```
private var font2:Class;

</fx:Script>
<s:Panel title="Embedded Fonts Using ActionScript">
  <s:VGroup>
    <s:RichText
      width="100%"
      height="75"
      styleName="mystyle1"
      text="This text uses the MyriadWebPro font."
    />
    <s:RichText
      width="100%"
      height="75"
      styleName="mystyle2"
      text="This text uses the MyriadWebPro-Bold font."
    />
  </s:VGroup>
</s:Panel>
</s:Application>
```

You use the value of the `fontName` property that you set in the `[Embed]` tag as the alias (`fontFamily`) in your style definition.

To embed a font with a different typeface (such as bold or italic), you specify the `fontWeight` or `fontStyle` properties in the `[Embed]` statement and in the style definition. For more information on embedding different typefaces, see [“Using multiple typefaces”](#) on page 15.

You can specify a subset of the font’s character range by specifying the `unicodeRange` parameter in the `[Embed]` metadata tag or the `@font-face` declaration. Embedding a range of characters rather than using the default of all characters can reduce the size of the embedded font and, therefore, reduce the final output size of your SWF file. For more information, see [“Setting character ranges”](#) on page 19.

## Using advanced anti-aliasing with non-CFF based fonts

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

When you embed non-CFF fonts (with the `embedAsCFF` property set to `false`), you can use advanced anti-aliasing to provide those fonts with additional information about the font. Non-CFF embedded fonts that use the advanced anti-aliasing information are typically clearer and appear sharper at smaller font sizes. CFF fonts have this information by default.

By default, non-CFF fonts that you embed in Flex applications use the advanced anti-aliasing information. This default is set by the `fonts.advanced-anti-aliasing` compiler option in the `flex-config.xml` file (the default value is `true`). You can override this default value by setting the value in your style sheets or changing it in the configuration file. To disable advanced anti-aliasing in style sheets, you set the `advancedAntiAliasing` style property to `false` in your `@font-face` rule, as the following example shows:

```
@font-face {
  src:url("../assets/MyriadWebPro.ttf");
  fontFamily: myFontFamily;
  advancedAntiAliasing: false;
  embedAsCFF: false;
}
```

Using advanced anti-aliasing can degrade the performance of your compiler. This is not a run-time concern, but can be noticeable if you compile your applications frequently or use the web-tier compiler. Using advanced anti-aliasing can also cause a slight delay when you load SWF files. You notice this delay especially if you are using several different character sets, so be aware of the number of fonts that you use. The presence of advanced anti-aliasing information may also cause an increase in the memory usage in Flash Player and Adobe® AIR™. Using four or five fonts, for example, can increase memory usage by approximately 4 MB.

When you embed non-CFF fonts that use advanced anti-aliasing in your Flex applications, the fonts function exactly as other embedded fonts. They are anti-aliased, you can rotate them, and you can make them partially or wholly transparent.

Font definitions that use advanced anti-aliasing support several additional styles properties: `fontAntiAliasType`, `fontGridFitType`, `fontSharpness`, and `fontThickness`. These properties are all inheriting styles.

Because the advanced anti-aliasing-related style properties are CSS styles, you can use them in the same way that you use standard style properties, such as `fontFamily` and `fontSize`. For example, a text-based component could use subpixel-fitted advanced anti-aliasing of New Century 14 at sharpness 50 and thickness -35, while all Button controls could use pixel-fitted advanced anti-aliasing of Tahoma 10 at sharpness 0 and thickness 0. These styles apply to all the text in a TextField control; you cannot apply them to some characters and not others.

The default values for the advanced anti-aliasing styles properties are defined in the `defaults.css` file. If you replace this file or use another style sheet that overrides these properties, Flash Player and AIR use the standard font renderer to render the fonts that use advanced anti-aliasing. If you embed fonts that use advanced anti-aliasing, you must set the `fontAntiAliasType` property to `advanced`, or you lose the benefits of the advanced anti-aliasing information.

The following table describes these properties:

Style property	Description
<code>fontAntiAliasType</code>	<p>Sets the <code>antiAliasType</code> property of internal TextField controls. The valid values are <code>normal</code> and <code>advanced</code>. The default value is <code>advanced</code>, which enables advanced anti-aliasing for the font.</p> <p>Set this property to <code>normal</code> to prevent the compiler from using advanced anti-aliasing.</p> <p>This style has no effect for system fonts or fonts embedded without the advanced anti-aliasing information.</p>

Style property	Description
<code>fontGridFitType</code>	<p>Sets the <code>gridFitType</code> property of internal <code>TextField</code> controls. The valid values are <code>none</code>, <code>pixel</code>, and <code>subpixel</code>. The default value is <code>pixel</code>. For more information, see the <a href="#">TextField</a> and <a href="#">GridFitType</a> classes in the <i>Adobe Flex Language Reference</i>.</p> <p>This property has the same effect as the <code>gridFitType</code> style property of the <code>TextField</code> control for system fonts, only it applies when you embed fonts with advanced anti-aliasing.</p> <p>Changing the value of this property has no effect unless the <code>fontAntiAliasType</code> property is set to <code>advanced</code>.</p>
<code>fontSharpness</code>	<p>Sets the <code>sharpness</code> property of internal <code>TextField</code> controls. The valid values are numbers from -400 to 400. The default value is 0.</p> <p>This property has the same effect as the <code>fontSharpness</code> style property on the <code>TextField</code> control for system fonts, only it applies when you embed fonts with advanced anti-aliasing.</p> <p>Changing the value of this property has no effect unless the <code>fontAntiAliasType</code> property is set to <code>advanced</code>.</p>
<code>fontThickness</code>	<p>Sets the <code>thickness</code> property of internal <code>TextField</code> controls. The valid values are numbers from -200 to 200. The default value is 0.</p> <p>This property has the same effect as the <code>fontThickness</code> style property on the <code>TextField</code> control for system fonts, only it applies when you embed fonts with advanced anti-aliasing.</p> <p>Changing the value of this property has no effect unless the <code>fontAntiAliasType</code> property is set to <code>advanced</code>.</p>

To use functionality similar to advanced anti-aliasing with CFF based fonts, you use the functionality of FTE that is built into Spark's text-based controls. For more information, see [Formatting text](#).

## Detecting embedded fonts

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

You can use the [SystemManager](#) class's `isFontFaceEmbedded()` method to determine whether the font is embedded or whether it has been registered globally with the `register()` method of the [Font](#) class. The `isFontFaceEmbedded()` method takes a single argument—the object that describes the font's [TextFormat](#)—and returns a Boolean value that indicates whether the font family you specify is embedded, as the following example shows:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/DetectingEmbeddedFonts.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark"
  creationComplete="determineIfFontFaceIsEmbedded()">

  <s:layout>
    <s:VerticalLayout/>
  </s:layout>

  <fx:Style>
    @font-face {
      src: url(../assets/MyriadWebPro.ttf);
      fontFamily: myPlainFont;
      advancedAntiAliasing: true;
      embedAsCFF: true;
    }

    .myStyle1 {
      fontFamily: myPlainFont;
      fontSize:12pt
    }
  </fx:Style>

  <fx:Script><![CDATA[
    import mx.managers.SystemManager;
    import mx.core.FlexGlobals;
    import flash.text.TextFormat;

    [Bindable]
    private var b1:Boolean;
    [Bindable]
    private var b2:Boolean;

    public function determineIfFontFaceIsEmbedded():void {
      var tf1:TextFormat = new TextFormat();
```

```

        tf1.font = "myPlainFont";

        var tf2:TextFormat = new TextFormat();
        tf2.font = "Arial";

        b1 = FlexGlobals.topLevelApplication.systemManager.
            isFontFaceEmbedded(tf1);
        b2 = FlexGlobals.topLevelApplication.systemManager.
            isFontFaceEmbedded(tf2);
    }
]]></fx:Script>
<mx:Form>
    <mx:FormItem label="isFontFaceEmbedded (myPlainFont):">
        <mx:Label id="l1" text="{b1}"/>
    </mx:FormItem>
    <mx:FormItem label="isFontFaceEmbedded (Arial):">
        <mx:Label id="l2" text="{b2}"/>
    </mx:FormItem>
</mx:Form>
</s:Application>

```

In this example, the font identified by the myPlainFont family name is embedded, but the Arial font is not.

You can use the [Font](#) class's `enumerateFonts()` method to output information about device or embedded fonts. The following example lists embedded fonts:

```

<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/EnumerateFonts.mxml -->
<s:Application
    xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:mx="library://ns.adobe.com/flex/halo"
    xmlns:s="library://ns.adobe.com/flex/spark"
    creationComplete="listFonts()">

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
<fx:Style>
    @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFont;
        advancedAntiAliasing: true;
        embedAsCFF: true;
    }
    @font-face {
        src:url("../assets/MyriadWebPro-Bold.ttf");
        fontFamily: myFont;
        fontWeight: bold;
        advancedAntiAliasing: true;
        embedAsCFF: true;
    }
    @font-face {
        src:url("../assets/MyriadWebPro-Italic.ttf");
        fontFamily: myFont;
        fontStyle: italic;
        advancedAntiAliasing: true;

```

```

        embedAsCFF: true;
    }

    .myPlainStyle {
        fontSize: 20;
        fontFamily: myFont;
    }

    .myBoldStyle {
        fontSize: 20;
        fontFamily: myFont;
        fontWeight: bold;
    }

    .myItalicStyle {
        fontSize: 20;
        fontFamily: myFont;
        fontStyle: italic;
    }
</fx:Style>
<fx:Script><![CDATA[
    private function listFonts():void {
        var fontArray:Array = Font.enumerateFonts(true);
        tal.text += "Fonts: \n";
        for(var i:int = 0; i < fontArray.length; i++) {
            var thisFont:Font = fontArray[i];
            tal.text += "FONT " + i + " :: name: " + thisFont.fontName + "; typeface: " +
                thisFont.fontStyle + "; type: " + thisFont.fontType;
            if (thisFont.fontType == "embeddedCFF" || thisFont.fontType == "embedded") {
                tal.text += "*";
            }
            tal.text += "\n";
        }
    }
}]></fx:Script>
<s:VGroup>
    <s:RichText text="Plain Label" styleName="myPlainStyle"/>
    <s:RichText text="Bold Label" styleName="myBoldStyle"/>
    <s:RichText text="Italic Label" styleName="myItalicStyle"/>
    <s:TextArea id="tal" height="200" width="400"/>
    <s:RichText text="* Embedded" styleName="myItalicStyle"/>
</s:VGroup>
</s:Application>

```

The following list shows the first few lines of sample output. This list will vary depending on the client's system.

```

FONT 0 :: name: myFont; typeface: regular; type: embeddedCFF*
FONT 1 :: name: myFont; typeface: bold; type: embeddedCFF*
FONT 2 :: name: myFont; typeface: italic; type: embeddedCFF*
FONT 3 :: name: Marlett; typeface: regular; type: device
FONT 4 :: name: Arial; typeface: regular; type: device
FONT 5 :: name: Arial CE; typeface: regular; type: device

```

The `enumerateFonts()` method takes a single Boolean argument: `enumerateDeviceFonts`. The default value of the `enumerateDeviceFonts` property is `false`, which means it returns an Array of embedded fonts by default.

If you set the `enumerateDeviceFonts` argument to `true`, the `enumerateFonts()` method returns an array of available device fonts on the client system, but only if the client's `mms.cfg` file sets the `DisableDeviceFontEnumeration` property to 0, the default value. If you set the `DisableDeviceFontEnumeration` property to 1, Flash Player cannot list device fonts on a client computer unless you explicitly configure the client to allow it. For more information about configuring the client with the `mms.cfg` file, see the Flash Player documentation.

## Using multiple typefaces

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

Most fonts have four typeface styles: plain, bold, italic, and bold-italic. You can embed any number of typeface styles in your Flex applications. If you embed only the bold typeface in your application, you cannot use the normal (or plain) typeface unless you also embed that typeface. For each typeface that you use, you must add a new `@font-face` declaration to your style sheet.

**Note:** Some Flex controls, such as `Button`, use the bold typeface style by default, rather than the plain style. If you use an embedded font for a `Button` label, you must either embed the bold font style for that font, or set the default typeface for the `Button` label to match a typeface that you embed.

The following example embeds the bold, italic, and plain typefaces of the Myriad Web Pro font. After you define the font face, you define selectors for the font by using the same alias as the `FontFamily`. You define one for the bold, one for the italic, and one for the plain face. To apply the font styles, this example applies the class selectors to the `Label` controls inline:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/MultipleFaces.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
<fx:Style>
  @font-face {
    src:url("../assets/MyriadWebPro.ttf");
    fontFamily: myFont;
    advancedAntiAliasing: true;
    embedAsCFF: true;
  }
  @font-face {
    /* Note the different filename for boldface. */
    src:url("../assets/MyriadWebPro-Bold.ttf");
    fontFamily: myFont; /* Notice that this is the same alias. */
    fontWeight: bold;
    advancedAntiAliasing: true;
    embedAsCFF: true;
  }
  @font-face {
    /* Note the different filename for italic face. */
    src:url("../assets/MyriadWebPro-Italic.ttf");
    fontFamily: myFont; /* Notice that this is the same alias. */
    fontStyle: italic;
```

```

        advancedAntiAliasing: true;
        embedAsCFF: true;
    }

    .myPlainStyle {
        fontSize: 32;
        fontFamily: myFont;
    }

    .myBoldStyle {
        fontSize: 32;
        fontFamily: myFont;
        fontWeight: bold;
    }

    .myItalicStyle {
        fontSize: 32;
        fontFamily: myFont;
        fontStyle: italic;
    }
</fx:Style>
<s:VGroup>
    <s:Label text="Plain Text" styleName="myPlainStyle"/>
    <s:Label text="Italic Text" styleName="myItalicStyle"/>
    <s:Label text="Bold Text" styleName="myBoldStyle"/>
</s:VGroup>
</s:Application>

```

Optionally, you can apply the bold or italic type to controls inline, as the following example shows:

```

<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/MultipleFacesAppliedInline.mxml -->
<s:Application
    xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:mx="library://ns.adobe.com/flex/halo"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
<fx:Style>
    @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFont;
        advancedAntiAliasing: true;
        embedAsCFF: true;
    }
    @font-face {
        src:url("../assets/MyriadWebPro-Bold.ttf");
        fontFamily: myFont;
        fontWeight: bold;
        advancedAntiAliasing: true;
        embedAsCFF: true;
    }

```

```

    }
    @font-face {
        src:url("../assets/MyriadWebPro-Italic.ttf");
        fontFamily: myFont;
        fontStyle: italic;
        advancedAntiAliasing: true;
        embedAsCFF: true;
    }

    .myStyle1 {
        fontSize: 32;
        fontFamily: myFont;
    }
</fx:Style>
<s:VGroup styleName="myStyle1">
    <s:Label text="Plain Text"/>
    <s:Label text="Italic Text" fontStyle="italic"/>
    <s:Label text="Bold Text" fontWeight="bold"/>
</s:VGroup>
</s:Application>

```

If you use a bold-italic font, the font must have a separate typeface for that font. You specify both properties (`fontWeight` and `fontStyle`) in the `@font-face` and selector blocks, as the following example shows:

```

@font-face {
    src:url("../assets/KNIZIA-BI.TTF");
    fontStyle: italic;
    fontWeight: bold;
    fontFamily: myFont;
    embedAsCFF: true;
}
.myBoldItalicStyle {
    fontFamily:myFont;
    fontWeight:bold;
    fontStyle:italic;
    fontSize: 32;
}

```

In the `@font-face` definition, you can specify whether the font is bold or italic by using the `fontWeight` and `fontStyle` properties. For a bold font, you can set `fontWeight` to `bold` or an integer greater than or equal to 700. You can specify the `fontWeight` as `plain` or `normal` for a nonboldface font. For an italic font, you can set `fontStyle` to `italic` or `oblique`. You can specify the `fontStyle` as `plain` or `normal` for a nonitalic face. If you do not specify a `fontWeight` or `fontStyle`, Flex assumes you embedded the plain or regular font face.

Flex does not require that bold or italic styles require a bold or italic font to be embedded. You can embed any font and use it on a control that uses bold or italic. The results might be less desirable than if you embedded a font with a bold or italic font face, but the text still renders.

You can also add any other properties for the embedded font, such as `fontSize`, to the selector, as you would with any class or type selector.

By default, Flex includes the entire font definition for each embedded font in the application, so you should limit the number of fonts that you use to reduce the size of the application. You can limit the size of the font definition by defining the character range of the font. For more information, see [“Setting character ranges”](#) on page 19.

## About the font managers

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

Flex includes several font managers to handle embedded fonts. The font managers take embedded font definitions and draw each character in Flash Player. This process is known as *transcoding*. The font managers are Batik, JRE, AFE (Adobe Font Engine), and CFF, represented by the `BatikFontManager`, `JREFontManager`, `AFEFontManager`, and `CFFFontManager` classes, respectively.

The Batik and JRE font managers can transcode non-CFF TrueType fonts. The AFE font manager adds support for non-CFF OpenType fonts. The CFF font manager can transcode TrueType and OpenType CFF fonts.

The Batik font manager transcodes only TrueType URL fonts (embedded by using `src:url`). It does not transcode system fonts. If you specify the font location when you embed the font, the compiler will use the Batik font manager. In general, the Batik font manager provides smoother rendering and more accurate line metrics (which affect multiline text and line-length calculations) than the JRE font manager.

The JRE font manager transcodes TrueType system fonts, but the quality of output is generally not as good as the Batik font manager. If you install the font on your system, the compiler will use the JRE font manager because the Batik font manager does not support system fonts.

The AFE font manager is the only font manager that you can use to transcode OpenType fonts for non-CFF fonts. It can also transcode TrueType fonts, but the fonts can only be URL fonts, not system fonts. If you embed an OpenType font, the compiler will use the AFE font manager to transcode the font because the other font managers do not support OpenType fonts, unless that OpenType font is a system font, in which case, the compiler will throw an error. None of the font managers can transcode OpenType fonts that are embedded as system fonts.

The CFF font manager supports both TrueType and OpenType fonts. It also supports URL and system fonts. Use this manager for all CFF fonts.

The following table shows which fonts are supported by which font managers:

	<b>Batik</b>	<b>AFE</b>	<b>JRE</b>
Font type	TrueType	TrueType, OpenType	TrueType
Method of embedding	URL	URL	System

You determine which font managers the compiler can use in the `flex-config.xml` file. The default setting is to use all of them, as the following example shows:

```
<font>
  <managers>
    <manager-class>flash.fonts.JREFontManager</manager-class>
    <manager-class>flash.fonts.AFEFontManager</manager-class>
    <manager-class>flash.fonts.BatikFontManager</manager-class>
    <manager-class>flash.fonts.CFFFontManager</manager-class>
  </managers>
</font>
```

The preference of `<manager>` elements is in reverse order. This means that by default the CFF font manager is the preferred font manager; the compiler checks to see if a font can be transcoded using it first. If not, then the compiler checks to see whether the font can be transcoded using the Batik font manager and then the AFE font manager. Finally, if the other font managers fail, the compiler checks to see whether the JRE font manager can transcode the font.

If you experience compilation or transcoding errors related to fonts, you can try changing the order of the font managers in the `flex-config.xml` file or by using the command-line compiler arguments.

## Setting character ranges

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

By specifying a range of symbols that compose the face of an embedded font, you reduce the size of an embedded font. Each character in a font that you use must be described; removing some of these characters reduces the overall size of the description information that Flex must include for each embedded font.

You can set the range of glyphs in the `flex-config.xml` file or in the `@font-face` declaration. You specify individual characters or ranges of characters using the Unicode values for the characters, and you can set multiple ranges for each font declaration.

The syntax for setting a character range is as follows:

```
U+[beginning of range] - [end of range];
```

For example:

```
U+0041-005A
```

If you use a character that is outside of the declared range, Flex displays a device font for that character. For more information on setting character ranges in Flex applications, see the CSS-2 Fonts specification at [www.w3.org/TR/1998/REC-CSS2-19980512/fonts.html#descdef-unicode-range](http://www.w3.org/TR/1998/REC-CSS2-19980512/fonts.html#descdef-unicode-range).

If you embed a font from a SWF file, you can restrict the character range in Flash. For more information, see [Embedding fonts from SWF files](#).

## Setting ranges in font-face declarations

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

You can set the range of allowable characters in an MXML file by using the `unicodeRange` property of the `@font-face` declaration. The following example embeds the Myriad Web Pro font and defines the range of characters for the font in the `<fx:Style>` tag:

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontCharacterRange.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myFontFamily;
      advancedAntiAliasing: true;
      embedAsCFF: true;
      unicodeRange:
        U+0041-005A, /* Upper-Case [A..Z] */
        U+0061-007A, /* Lower-Case a-z */
        U+0030-0039, /* Numbers [0..9] */
        U+002E-002E; /* Period [.] */
    }
    s|RichText {
      fontFamily: myFontFamily;
      fontSize: 32;
    }
  </fx:Style>
  <s:Panel title="Embedded Font Character Range">
    <s:RichText width="400" height="150">
      <s:text>
        The Text Uses Only Some of Available Characters
        0 1 2 3 4 5 6 7 8 9.
      </s:text>
    </s:RichText>
  </s:Panel>
</s:Application>
```

## Setting ranges in flex-config.xml

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

You can specify the language and character range for embedded fonts in the flex-config.xml file by using the `<language-range>` child tag. This lets you define the range once and use it across multiple `@font-face` blocks.

The following example creates an `englishRange` and an `otherRange` named ranges in the flex-config.xml file:

```
<font>  
  <languages>  
    <language-range>  
      <lang>englishRange</lang>  
      <range>U+0020-007E</range>  
    </language-range>  
    <language-range>  
      <lang>otherRange</lang>  
      <range>U+00??</range>  
    </language-range>  
  </languages>  
</font>
```

In your MXML file, you point to the defined ranges by using the `unicodeRange` property of the `@font-face` declaration, as the following example shows:

```
@font-face {  
  fontFamily: myPlainFont;  
  src: url("../assets/MyriadWebPro.ttf");  
  unicodeRange: "englishRange";  
  embedAsCFF: true;  
}
```

Flex includes a file that lists convenient mappings of the Flash UnicodeTable.xml character ranges for use in the Flex configuration file. For Adobe LiveCycle Data Services ES, the file is located at `flex_app_root/WEB-INF/flex/flash-unicode-table.xml`; for Adobe Flex SDK, the file is located at `flex_install_dir/frameworks/flash-unicode-table.xml`.

The following example shows the predefined range Latin 1:

```
<language-range>  
  <lang>Latin I</lang>  
  <range>U+0020,U+00A1-00FF,U+2000-206F,U+20A0-20CF,U+2100-2183</range>  
</language-range>
```

To make ranges listed in the `flash-unicode-table.xml` file available in your Flex applications, copy the ranges from this file and add them to the `flex-config.xml` files.

## Detecting available ranges

[Chunk: No] [Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

You can use the `Font` class to detect the available characters in an embedded font. You do this with the `hasGlyphs()` method.

The following example embeds the same font twice, each time restricting the font to different character ranges. The first font includes support only for the letters A and B. The second font family includes all 128 glyphs in the Basic Latin block.

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- charts/CharacterRangeDetection.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark"
  creationComplete="checkCharacterSupport();"
>
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <fx:Style>
    @font-face {
      font-family: myABFont;
      advancedAntiAliasing: true;
      src:url("../assets/MyriadWebPro.ttf");
      /*
       * Limit range to the letters A and B.
       */
      unicodeRange: U+0041-0042;
      embedAsCFF: true;
    }
    @font-face {
      font-family: myWideRangeFont;
      advancedAntiAliasing: true;
      src:url("../assets/MyriadWebPro.ttf");
      /*
       * Set range to the 128 characters in
       * the Basic Latin block.
       */
      unicodeRange: U+0041-007F;
      embedAsCFF: true;
    }
  </fx:Style>
  <fx:Script><![CDATA[
    public function checkCharacterSupport():void {
      var fontArray:Array = Font.enumerateFonts(false);
      for(var i:int = 0; i < fontArray.length; i++) {
        var thisFont:Font = fontArray[i];
        if (thisFont.hasGlyphs("DHARMA")) {
          ta1.text += "The font '" + thisFont.fontName +
            "' supports these glyphs.\n";
        } else {
          ta1.text += "The font '" + thisFont.fontName +
```

```

        "' does not support these glyphs.\n";
    }
}
]]></fx:Script>
<s:VGroup>
    <s:RichText>
        <s:text>
            myABFont unicodeRange: U+0041-0042 (letters A and B)
        </s:text>
    </s:RichText>
    <s:RichText>
        <s:text>
            myWideRangeFont unicodeRange: U+0041-007F (Basic Latin chars)
        </s:text>
    </s:RichText>
    <s:Label text="Glyphs: DHARMA"/>
    <s:RichText id="tal" height="150" width="300"/>
</s:VGroup>
</s:Application>

```

## Embedding double-byte fonts

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

When using double-byte fonts in Flex, you should embed the smallest possible set of characters. If you embed a font's entire character set, the size of your application's SWF file can be very large. You can define sets of Unicode character ranges in the `flex-config.xml` file and then reference the name of that range in your style's `@font-face` declaration.

Flex provides predefined character ranges for common double-byte languages such as Thai, Kanji, Hangul, and Hebrew in the `flash-unicode-table.xml` file. This file is not processed by Flex, but is included to provide you with ready definitions for various character ranges. For example, the following character range for Thai is listed in the `flash-unicode-table.xml` file:

```

<language-range>
    <lang>Thai</lang>
    <range>U+0E01-0E5B</range>
</language-range>

```

To use this language in your Flex application, copy the character range to the `flex-config.xml` file or pass it on the command line by using the `fonts.languages.language-range` option. Add the full definition as a child tag to the `<languages>` tag, as the following example shows:

```
<flex-config>
  <compiler>
    <font>
      <languages>
        <language-range>
          <lang>thai</lang>
          <range>U+0E01-0E5B</range>
        </language-range>
      </languages>
    </font>
  </compiler>
  ...
</flex-config>
```

You can change the value of the `<lang>` element to anything you want. When you embed the font by using CSS, you refer to the language by using this value in the `unicodeRange` property of the `@font-face` declaration, as the following example shows:

```
@font-face {
  fontFamily: "Thai_font";
  src: url("../assets/THAN.TTF"); /* Embed from file */
  unicodeRange: "thai";
  embedAsCFF: true;
}
```

## Embedding fonts with Halo components

[Output: IPH, Print, Web] [Revision Control: Changing]

The `embedAsCFF` (Compact Font Format) property indicates whether to embed a font that supports the advanced text layout features used by the Flash Text Engine (FTE). This is sometimes referred to as DefineFont4. If you set the value of the `embedAsCFF` property to `true`, then you can only use that font with controls that support FTE. If you set the value of the `embedAsCFF` property to `false`, then the embedded font does not support FTE and you can only use that font with controls that do not have FTE support.

The implications of this appear when you try to use Halo controls with a font that was embedded with FTE support. In those cases, the text does not appear, as the following example shows.

In the first panel, the embedded CFF font is applied to the Spark control and the embedded non-CFF font is applied to the Halo control. The result is that the Button labels show correctly.

In the second Panel, both Button labels use the CFF font. The result is that no text appears for the Halo control in the second Panel because the Halo control does not support CFF.

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/CFFTest.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">
  <s:layout>
    <s:VerticalLayout/>
  </s:layout>
  <fx:Style>
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myCFFFont;
      advancedAntiAliasing: false;
      embedAsCFF: true;
    }
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myFontNoCFF;
      advancedAntiAliasing: true;
      embedAsCFF: false;
    }

    .myCFFStyle {
      fontSize: 32;
      fontFamily: myCFFFont;
    }
    .myStyleNoCFF {
      fontSize: 32;
      fontFamily: myFontNoCFF;
    }
  </fx:Style>
  <s:Panel title="Using Correct Fonts">
    <s:VGroup>
      <s:Label text="Spark Label" styleName="myCFFStyle"/>
      <mx:Label text="Halo Label" styleName="myStyleNoCFF"/>
    </s:VGroup>
  </s:Panel>
  <s:Panel title="Using Incorrect Fonts">
    <s:VGroup>
      <s:Label text="Spark Label" styleName="myCFFStyle"/>
      <!-- The label text will not show up because it's a Halo control that
      is attempting to use a Spark-compatible embedded font. -->
      <mx:Label text="Halo Label" styleName="myCFFStyle"/>
    </s:VGroup>
  </s:Panel>
</s:Application>
```

As this example illustrates, if you mix Halo and Spark controls inside the same application, you might not be able to use the same embedded font.

There are two possible remedies to this situation:

- Specify that the Halo controls use FTE classes style to render text, rather than their default text renderers.
- Embed both the non-CFF version of the font in addition to the CFF version of the font.

The following sections describe each of these solutions.

## More Help topics

Halo text controls

## Using FTE in Halo controls

[Output: IPH, Print, Web] [Revision Control: Changing]

The controls that support FTE include all Spark components in the `spark.components` package. This includes the Spark text controls such as `Label`, `RichText`, and `RichEditableText`. This also includes Spark versions of the `TextInput` and `TextArea` controls. This does not include Halo controls in the `mx.controls` package.

The reason that Halo controls do not support FTE is that by default they use the `UITextField` subcomponent to render text. The `UITextField` subcomponent does not support FTE. Spark controls, on the other hand, use FTE-compatible classes to render text.

The Flex SDK provides the `mx.core.UITLFFextField` and `mx.controls.TLFFTextInput` classes that support FTE for Halo text controls. You can use these classes in some Halo controls so that those controls can use CFF versions of embedded fonts. As a result, those controls can use the same embedded fonts that you also use with the Spark controls. You do this by setting the `textFieldClass` and `textInputClass` styles to use these classes.

The easiest way to use the `TLFFTextInput` and `UITLFFextField` classes with your Halo text controls is to apply the `TLFFText.css` theme file to your application. This theme applies the `TLFFTextInput` and `UITLFFextField` classes to your Halo controls. The `TLFFText.css` theme file is a convenience theme that is set up to apply only FTE-supporting classes to Halo controls. The following excerpt from the `TLFFText.css` theme file shows that the `textInputClass` and `textFieldClass` style properties are set to classes that support FTE:

```
DateField {
    textInputClass: ClassReference("mx.controls.TLFFTextInput");
}
Label {
    textFieldClass: ClassReference("mx.core.UITLFFextField");
}
```

On the command line, you specify the `TLFFText.css` theme file by using the `theme` compiler option, as the following example shows:

```
mxmlc -theme=themes/TLFFText.css MyApp.mxml
```

In Flash Builder, you can add a theme file by selecting `Project > Properties`. Select `Flex Compiler`, and add `-theme=themes/TLFFText.css` to the `Additional Compiler Arguments` option.

The following example is compiled with the `TLFFText.css` theme. Because this theme is used, the Halo text controls can use the same embedded font as the Spark embedded font. If you compile this example without the `theme` option, the label for the Halo Button does not render.

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/UseTLFTextTheme.mxml -->
<!-- Compile this example by setting theme=TLFText.css for a compiler argument. -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">

  <s:layout>
    <s:VerticalLayout/>
  </s:layout>

  <fx:Style>
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myCFFFont;
      embedAsCFF: true;
    }

    .myCFFStyle {
      fontSize: 32;
      fontFamily: myCFFFont;
    }
  </fx:Style>

  <s:Panel title="Spark and Halo Buttons">
    <s:VGroup>
      <s:Button label="Spark Button" styleName="myCFFStyle"/>
      <mx:Button label="Halo Button" styleName="myCFFStyle"/>
    </s:VGroup>
  </s:Panel>

</s:Application>
```

The Halo DataGrid control has a special class, `TLFDataGridItemRenderer`, that you can use for custom item renderers. The `TLFText.css` file specifies it as follows:

```
defaultDataGridItemRenderer:
ClassReference("mx.controls.dataGridClasses.TLFDataGridItemRenderer");
```

Some controls are not affected by the `TLFText.css` theme. This is because some Halo controls have Spark equivalents. As a result, you should use the Spark version of the control instead of the Halo version where possible.

Rather than use the `TLFText.css` theme file to add FTE support to your Halo controls, you can manually replace the non-FTE classes with the FTE classes for text rendering on a Halo control. You do this by setting the value of the `textFieldClass` or `textInputClass` style properties to the `UITLFTextField` or `TLFTextInput` classes, as the following example shows:

```

<codeblock> Resolved code-reference.
<?xml version="1.0" encoding="utf-8"?>
<!-- fonts/TextFieldClassExample.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo">
  <s:layout>
    <s:VerticalLayout
      paddingLeft="20" paddingRight="20"
      paddingTop="20" paddingBottom="20" />
  </s:layout>
  <fx:Style>
    @namespace mx "library://ns.adobe.com/flex/halo";
    @namespace s "library://ns.adobe.com/flex/spark";
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: cffFont;
      embedAsCFF: true;
    }
    mx|Label, s|Label {
      fontFamily: cffFont;
      color: haloBlue;
      fontSize: 22;
      textFieldClass:ClassReference("mx.core.UITLFTextField");
    }
  </fx:Style>
  <mx:Label text="Hello World 1234567890 [Halo Label]" width="100%"/>
  <s:Label text="Hello World 1234567890 [Spark Label]" width="100%"/>
</s:Application>

```

You can also specify the value of the `textFieldClass` or `textInputClass` style properties inline, as the following example shows:

```

<mx:Label text="Hello World 1234567890 [Label]" width="100%"
  textFieldClass="mx.core.UITLFTextField"/>

```

You can also use the `setStyle()` method to specify the value of these style properties in ActionScript, as the following example shows:

```

import mx.core.UITLFTextField;
myLabel.setStyle("textFieldClass", mx.core.UITLFTextField);

```

The `UITLFTextField` and `TLFTTextInput` classes do not support editing, scrolling, selection, linking, and rich text. As a result, you can only use these classes on Halo controls that do not use these features.

The following Halo controls support the `UITLFTextField` and `TLFTTextInput` classes, which means that you can use CFF fonts with them:

- Accordion
- Alert (the text is not selectable)
- Button
- ButtonBar
- CheckBox
- DateChooser

- FileSystemComboBox
- FileSystemHistoryButton
- FormHeading
- FormItem
- FormHeading
- FormItem
- HSlider (the labels follow the same rules as the Halo Label control)
- LinkBar
- LinkButton
- Menu
- MenuBar
- Panel
- PopUpButton
- PopUpMenuButton
- PrintDataGrid
- ProgressBar
- RadioButton
- TabBar
- TabNavigator
- TitleWindow
- ToggleButtonBar
- ToolTip
- VSlider (the labels follow the same rules as the Halo Label control)

Other Halo controls have some limitations when it comes to using the `UITLFTextField` and `TLFTextInput` classes. In some cases, you should use the Spark equivalents. In other cases, you can use the `UITLFTextField` and `TLFTextInput` subcomponents with the Halo control, as long as you avoid using advanced text features, such as editability or the `htmlText` property. The following table describes these limitations:

Halo control	Description
List-based components (such as List, FileSystemList, HorizontalList, TileList, DataGrid, and Tree)	Some list-based components have Spark equivalents (including List, HorizontalList, and TileList). You can use the <code>UITLFTextField</code> and <code>TLFTextInput</code> classes in the other components if you do not use selection, editability, HTML links, or scrolling. Otherwise, you should embed a non-CFF version of the font to support these controls.
Label and Text	Use Spark equivalents such as Label, RichText, and RichEditableText. You can use <code>UITLFTextField</code> with the Label and Text controls if the text is not selectable or you do not use the <code>htmlText</code> property to specify the content of the controls.  Otherwise, you should embed a non-CFF version of the font to support these controls.
TextInput and TextArea	Use the Spark equivalents. Otherwise, you should embed a non-CFF version of the font to support these controls.

Halo control	Description
RichTextEditor	There is no equivalent class. In this case, you should embed a non-CFF version of the font to support this control.
ColorPicker	There is no equivalent class. In this case, you should embed a non-CFF version of the font to support this control. However, the ColorPicker control only displays a color value with its font, so in some cases, using an embedded font might not be necessary.
ComboBox	Use the Spark equivalent. If your ComboBox's text does not need to be editable, you can use the TLFTTextInput class. Otherwise, you should embed a non-CFF version of the font to support this control.
DateField	If you do not use editability, then you can use the TLFTTextInput class. Otherwise, you should embed a non-CFF version of the font to support this control.
NumericStepper	If you do not use editability, then you can use the TLFTTextInput class. Otherwise, you should embed a non-CFF version of the font to support this control.

## Embedding non-CFF versions of fonts for Halo components

[Output: IPH, Print, Web] [Revision Control: Changing]

To use Halo controls with embedded fonts, you can embed the non-CFF version of the font instead of changing the properties of the control to use CFF fonts. You should do this only if your Halo control:

- Has no Spark equivalent
- Does not support using the UITLTextField and TLFTTextInput classes for text rendering
- Must use selection, scrolling, or HTML text
- Is editable

If you embed a non-CFF version of a font in addition to a CFF version of the font in your application, your SWF file will be larger than if you embedded only a single version of the font. As a result, only do this when absolutely necessary.

If you compile an application with the `compatibility-version` compiler option set to 3.x, then the non-CFF version of the font is embedded automatically.

The following example embeds both a non-CFF version and a CFF version of the font so that the Spark and Halo Labels use embedded fonts. The reason this is required is that the Label is selectable in this example.

```
<codeblock> Resolved code-reference.
<?xml version="1.0"?>
<!-- fonts/EmbedBoth.mxml -->
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark">

  <s:layout>
    <s:VerticalLayout/>
  </s:layout>

  <fx:Style>
    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myCFFFont;
      embedAsCFF: true;
    }

    @font-face {
      src:url("../assets/MyriadWebPro.ttf");
      fontFamily: myFontNoCFF;
      advancedAntiAliasing: true;
      embedAsCFF: false;
    }

    .myCFFStyle {
      fontSize: 32;
      fontFamily: myCFFFont;
    }

    .myStyleNoCFF {
      fontSize: 32;
      fontFamily: myFontNoCFF;
    }
  </fx:Style>

  <s:Panel title="Using Two Different Embedded Fonts">
    <s:VGroup>
      <s:Label text="Spark Label" styleName="myCFFStyle"/>
      <mx:Label text="Halo Label" styleName="myStyleNoCFF" selectable="true"/>
    </s:VGroup>
  </s:Panel>

</s:Application>
```

Note that when embedding a non-CFF font, you have the option of specifying the `advancedAntiAliasing` property. With CFF fonts, this property is not ignored. The advanced anti-aliasing functionality is provided with FTE.

## Troubleshooting fonts in Flex applications

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

There are some techniques that you can use to successfully embed fonts into your Flex applications.

In many cases, you can resolve an error by changing the font manager. This is set in the flex-config.xml configuration file; for example:

```
<font>
  <managers>
    <manager-class>flash.fonts.JREFontManager</manager-class>
    <manager-class>flash.fonts.AFEFontManager</manager-class>
    <manager-class>flash.fonts.BatikFontManager</manager-class>
  </managers>
</font>
```

You can try changing the order of font managers, as the following example shows:

```
<font>
  <managers>
    <manager-class>flash.fonts.AFEFontManager</manager-class>
    <manager-class>flash.fonts.BatikFontManager</manager-class>
    <manager-class>flash.fonts.JREFontManager</manager-class>
  </managers>
</font>
```

## Resolving compiler errors

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

The following table describes common compiler errors and their solutions:

Error	Solution
Unable to resolve 'swf_file_name' for transcoding	Indicates that the font was not found by the compiler. Ensure that the path to the font is correct in the @font-face declaration or the [Embed] tag and that the path is accessible by the compiler.
Font 'font_name' with style_description not found	Indicates that the fontName property used in the [Embed] statement might not match the name of the font.  For fonts in SWF files, ensure that the spelling and word spacing of the font name in the list of available fonts in Flash is the same as the fontName property in your [Embed] statement and the fontFamily property that you use in your style definitions.  This error can also mean that the font's style was not properly embedded in Flash. Open the FLA file and ensure that there is a text area with the font and style described, that the text is dynamic, and that you selected a character range for that text.

## Resolving run-time errors

[Output: IPH, Print, Web] [EditorialStatus: Preliminary Review]

To determine if your fonts are embedded properly, you can use the isFontFaceEmbedded() method of the SystemManager, as described in “Detecting embedded fonts” on page 11.

To properly embed your fonts, try the following techniques:

- If one type of control is not correctly displaying its text, ensure that you are embedding the appropriate typeface. For example, the Halo Button control's text labels require the bold typeface. If you do not embed the bold typeface, the Halo Button control does not display the embedded font.
- In your Flex application, ensure that you set all properties for each font typeface in the @font-face declaration or [Embed] statement. To embed a bold typeface, you must set the fontWeight property to bold, as the following example shows:

```
@font-face {
    src: url(../assets/MyriadWebProEmbed.ttf);
    fontFamily: "Myriad Web Pro";
    fontWeight: bold;
    embedAsCFF: true;
}
```

You also must set the `fontWeight` style property in your style definition:

```
.myStyle2 {
    fontFamily:"Myriad Web Pro";
    fontWeight:bold;
    fontSize:12pt;
}
```

If you use the `[Embed]` statement, you must set the `fontWeight` property to `bold` as the following example shows:

```
[Embed(source="MyriadWebProEmbed.ttf", fontName="Myriad Web Pro",fontWeight="bold")]
```

- For fonts that are embedded in SWF files that you import, open the FLA file in Flash and ensure that all of the typefaces were added properly. Select each text area and do the following:
  - Check that the font name is correct. Ensure that the spelling and word spacing of the font name in the list of available fonts in Flash is the same as the `fontFamily` property in the `@font-face` declaration or the `fontName` property in your `[Embed]` statement. This value must also match the `fontFamily` property that you use in your style definitions.

If you did not select an anti-aliasing option for the font in Flash 8 (for example, you chose Bitmap Text (no anti-alias)), you might need to change the value of the font name to a format that matches `fontName_fontSizept_st` (for example, "Wingdings\_8pt\_st"). In the CSS for that bitmap font, be sure to set `fontAntiAliasType` to `normal`.
  - To determine the exact font name exported by Flash (which you must match as the value of the `fontFamily` property in your Flex application), open the SWF file in Flash 8 and select `Debug > Variables`.
- Check that the style is properly applied. For example, select the bold text area and check that the typeface really is bold.
- Click the Embed button and ensure that the range of embedded characters includes the characters that you use in your Flex application.
- Check that each text area is set to Dynamic Text and not Static Text or Input Text. The type of text is indicated by the first drop-down box in the text's Properties tab.
- Unless the SWF file was compiled with CFF, you must set the value of the `embedAsCFF` property to `false` for the imported font.
- For fonts in SWF files, ensure that you are using the latest SWF file that contains your fonts and was generated in Flash. Regenerate the SWF file in Flash if necessary.
- Ensure that the value of the `embedAsCFF` property is correct for the way that you are using a font. For Spark controls, set `embedAsCFF` to `true`. For fonts that are used by Halo controls, try setting `embedAsCFF` to `false`. The following example sets the `embedAsCFF` property to `false` in the `@font-face` declaration:

```
@font-face {
    src: url(../assets/MyriadWebProEmbed.ttf);
    fontFamily: "Myriad Web Pro";
    fontWeight: bold;
    embedAsCFF: false;
}
```