

ADOBE® FLEX® Gumbo Preview Release Features and Migration Guide



© 2009 Adobe Systems Incorporated. All rights reserved.

Adobe® Flex® 4 Features and Migration Guide. Prerelease version.

This prerelease version of the Software may not contain trademark and copyright notices that will appear in the commercially available version of the Software.

Adobe, the Adobe logo, Flash, Flex, Flex Builder and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. ActiveX and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple and Macintosh are trademarks of Apple Inc., registered in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Solaris is a registered trademark or trademark of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

This Work is licensed under the Creative Commons Attribution Non-Commercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA.

Contents

Chapter 1: New SDK Features

Namespaces	1
Language tags	4
Gumbonents	7
Component architecture	9
FXG	10
States	37
Layouts	43
Effects	45
Advanced CSS	52
HTML wrappers	61
Deferred instantiation	63
Two-way data binding	67
Animated filters	69
Pixel Bender integration	70
Text primitives	75
DataGroup	77
ASDoc	79
Conditional compilation	82

Chapter 2: New Flash Builder Features

Flex Package Explorer	83
Project Configuration	84
Flash Builder Command Line Build	85
Refactor enhancements	92
Support for Flex Projects created with Flash Catalyst	93
Generating accessor functions	94
Customizing file templates	94
Generating event handlers	98
ASDoc support	101
Applying Flex themes	101
Creating and editing Flash components	105
Debugger enhancements	106
Network Monitor	110
Profiler enhancements	112
Flex unit test integration	113

Chapter 3: Working with data and services in Flex

Workflow for accessing data services	115
Connecting to remote data	115
Building the client application	121
Managing the access of data from the server	123
Coding access to data services	125

Debugging applications for remote services 125

Data services in Flash Builder 127

Tutorial: Creating an application accessing remote data services 127

Chapter 4: Migrating Flex 3 applications

Namespaces 136

Declarations 137

Default properties for custom components 138

Loading SWF files 138

Charting 138

States 139

HTML wrapper 139

Layouts 139

Binding 140

Filters 141

Component rendering 141

Styles 141

Fonts 143

Globals 143

Graphics 144

Ant tasks 144

Chapter 1: New SDK Features

The following new features for the SDK are available with the Adobe MAX Gumbo Preview:

- “[Namespaces](#)” on page 1
- “[Language tags](#)” on page 4
- “[Gumbonents](#)” on page 7
- “[Component architecture](#)” on page 9
- “[FXG](#)” on page 10
- “[States](#)” on page 37
- “[Layouts](#)” on page 43
- “[Effects](#)” on page 45
- “[Advanced CSS](#)” on page 52
- “[HTML wrappers](#)” on page 61
- “[Deferred instantiation](#)” on page 63
- “[Two-way data binding](#)” on page 67
- “[Animated filters](#)” on page 69
- “[Pixel Bender integration](#)” on page 70
- “[Text primitives](#)” on page 75
- “[DataGroup](#)” on page 77
- “[ASDoc](#)” on page 79
- “[Conditional compilation](#)” on page 82

Namespaces

The namespaces for Flex 4 applications are as follows:

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:mx="library://ns.adobe.com/flex/halo"  
xmlns:s="library://ns.adobe.com/flex/spark"
```

For example, the `xmlns` properties in the following `<s:Application>` tag indicates that tags corresponding to the Spark component set use the prefix `s:`.

```
<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"  
>
```

You can still use the 2006 namespace, but you cannot use Flex 4 features (such as the new components and layout schemes) with it.

Flex defines the following Universal Resource Identifiers (URI) for the Flex namespaces:

- `xmlns:fx="http://ns.adobe.com/mxml/2009"`

The MXML language namespace URI. This namespace includes the top-level ActionScript language elements, such as Object, Number, Boolean, and Array. For a complete list of the top-level elements, see the Top Level package in the *Adobe Flex Language Reference*.

This namespace also includes the tags built into the MXML compiler, such as `<fx:Script>`, `<fx:Declarations>`, and `<fx:Style>` tags. For a list of the compiler elements, see the MXML Only Tags appendix in the *Adobe Flex Language Reference*.

This namespace does not include the Halo or Spark component sets.

The complete list of top-level ActionScript language elements included in this namespace is defined by the `frameworks\mxml-2009-manifest.xml` manifest file in your Flex SDK installation directory. Note that this file does not list the MXML compiler tags because they are built into the MXML compiler.

- `xmlns:mx="library://ns.adobe.com/flex/halo"`

The Halo component set namespace URI. This namespace includes all of the components in the Flex `mx.*` packages, the Flex charting components, and the Flex data visualization components.

The complete list of elements included in this namespace is defined by the `frameworks\halo-manifest.xml` manifest file in your Flex SDK installation directory.

- `xmlns:s="library://ns.adobe.com/flex/spark"`

The Spark component set namespace URI. This namespace includes all of the components in the Flex `spark.*` packages and the text framework classes in the `flashx.*` packages.

This namespace includes the RPC classes for the WebService, HTTPService, and RemoteObject components and additional classes to support the RPC components. These classes are included in the `mx:` namespace, but are provided as a convenience so that you can also reference them by using the `s:` namespace.

This namespace also includes several graphics, effect, and state classes from the `mx.*` packages. These classes are included in the `mx:` namespace, but are provided as a convenience so that you can also reference them by using the `s:` namespace.

The complete list of elements included in this namespace is defined by the `frameworks\spark-manifest.xml` manifest file in your Flex SDK installation directory.

The following table lists the classes from the `mx.*` packages included in this namespace:

Category	Class
RPC classes	mx.messaging.channels.AMFChannel mx.rpc.CallResponder mx.messaging.ChannelSet mx.messaging.Consumer mx.messaging.channels.HTTPChannel mx.rpc.http.mxml.HTTPService mx.messaging.Producer mx.rpc.remoting.mxml.RemoteObject mx.rpc.remoting.mxml.Operation mx.messaging.channels.RTMPChannel mx.messaging.channels.SecureAMFChannel mx.messaging.channels.SecureStreamingAMFChannel mx.messaging.channels.SecureHTTPChannel mx.messaging.channels.SecureStreamingHTTPChannel mx.messaging.channels.SecureRTMPChannel mx.messaging.channels.StreamingAMFChannel mx.messaging.channels.StreamingHTTPChannel mx.rpc.soap.mxml.WebService mx.rpc.soap.mxml.Operation mx.data.mxml.DataService
Graphics classes	mx.graphics.BitmapFill mx.geom.CompoundTransform mx.graphics.GradientEntry mx.graphics.LinearGradient mx.graphics.LinearGradientStroke mx.graphics.RadialGradient mx.graphics.RadialGradientStroke mx.graphics.SolidColor mx.graphics.SolidColorStroke mx.graphics.Stroke mx.geom.Transform
Effect classes	mx.effects.Parallel mx.effects.Sequence mx.states.Transition mx.effects.Wait
States classes	mx.states.State mx.states.AddItems

See also[MXML 2009 specification](#)

“Namespaces” on page 136

Language tags

Flex 4 introduces new language tags that you can use in your MXML files.

For additional information about the new language tags, see the MXML 2009 specification.

Declarations

Use the `<Declarations>` tag to declare non-default, non-visual properties of the current class.

This tag is used for MXML-based custom components that declare default properties.

The following example defines two style properties as child tags of the root tag. It also defines the text property in the `<Declarations>` tag:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- language/MyButton.mxml -->
<s:Button
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  xmlns:s="library://ns.adobe.com/flex/spark"
>

  <!-- Properties of the class can still be declared as child tags. -->
  <s:color>Red</s:color>
  <s:fontSize>16</s:fontSize>

  <!-- Custom, non-default, non-visual properties must be contained in a Declarations tag.
-->
  <fx:Declarations>
    <fx:String id="text">Click Me</fx:String>
  </fx:Declarations>

</s:Button>
```

The following main application uses the custom Button from the previous example. The two MXML files are in the same directory:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- language/MainApp.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"
  xmlns:myComps="*"
>
  <myComps:MyButton id="b1" label="{b1.text}"/>
</s:Application>
```

You can declare visual children inside a `<Declarations>` tag, but they are instantiated as if they were non-visual. You must manually add them to the display list with a call to the container's `addChild()` method.

Definition

Use one or more `<Definition>` tags inside a `<Library>` tag to define graphical children that you can then use in other parts of the application file.

An element in the `<Definition>` tag is not instantiated or added to the display list until it is added as a tag outside of the `<Library>` tag.

The `<Definition>` element must define a `name` attribute. You use this attribute as the tag name when instantiating the element.

A `Library` tag can have any number of `<Definition>` tags as children.

The following example defines the `MyCircle` and `MySquare` graphics with the `Definition` tags. It then instantiates several instances of these in the application file:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- language/DefinitionExample.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"
>
  <fx:Library>
    <fx:Definition name="MySquare">
      <s:Group>
        <s:Rect width="100%" height="100%">
          <s:stroke>
            <s:SolidColorStroke color="red"/>
          </s:stroke>
        </s:Rect>
      </s:Group>
    </fx:Definition>
    <fx:Definition name="MyCircle">
      <s:Group>
        <s:Ellipse width="100%" height="100%">
          <s:stroke>
            <s:SolidColorStroke color="blue"/>
          </s:stroke>
        </s:Ellipse>
      </s:Group>
    </fx:Definition>
  </fx:Library>
  <mx:Canvas>
    <fx:MySquare x="0" y="0" height="20" width="20"/>
    <fx:MySquare x="25" y="0" height="20" width="20"/>
    <fx:MyCircle x="50" y="0" height="20" width="20"/>
    <fx:MyCircle x="0" y="25" height="20" width="20"/>
    <fx:MySquare x="25" y="25" height="20" width="20"/>
    <fx:MySquare x="50" y="25" height="20" width="20"/>
    <fx:MyCircle x="0" y="50" height="20" width="20"/>
    <fx:MyCircle x="25" y="50" height="20" width="20"/>
    <fx:MySquare x="50" y="50" height="20" width="20"/>
  </mx:Canvas>
</s:Application>
```

Each Definition in the Library tag is compiled into a separate ActionScript class. that is a subclass of the type represented by the first node in the definition. In the previous example, the new class is a subclass of `mx.graphics.Group`. This scope of this class is limited to the document. It should be treated as a private ActionScript class.

For more information about the Library tag, see “[Library](#)” on page 6.

DesignLayer

The `<DesignLayer>` tag adds support for layers and layer groups.

Currently, this specification for the `<DesignLayer>` tag is not public.

Library

Use the `<Library>` tag to define zero or more named graphic `<Definition>` children. The definition itself in a library is not an instance of that graphic, but it lets you reference that definition any number of times in the document as an instance.

The Library tag must be the first child of the document’s root tag. You can only have one Library tag per document.

The following example defines a single graphic in the `<Library>` tag, and then uses it three times in the application:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- language/LibraryExample.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"
>
  <fx:Library>
    <fx:Definition name="MyTextGraphic">
      <s:Group>
        <s:RichText width="75">
          <s:content>Hello World!</s:content>
        </s:RichText>
        <s:Rect width="100%" height="100%">
          <s:stroke>
            <s:SolidColorStroke color="red"/>
          </s:stroke>
        </s:Rect>
      </s:Group>
    </fx:Definition>
  </fx:Library>

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

  <fx:MyTextGraphic/>
  <fx:MyTextGraphic/>
  <fx:MyTextGraphic/>
</s:Application>
```

For more information, see “[Definition](#)” on page 5.

Private

The `<Private>` tag provides meta information about the MXML or FXG document.

It must be a child of the root document tag, and it must be the last tag in the file.

The compiler ignores all content of the `<Private>` tag, although it must be valid XML. The XML can be empty, contain arbitrary tags, or contain a string of characters.

The following example adds information about the author and date to the MXML file:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- language/PrivateExample.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"
  xmlns:myComps="*"
>
  <mx:Canvas top="0" bottom="0" left="0" right="0">
    <s:Graphic>
      <s:RichText x="0" y="0">
        <s:content>Hello World!</s:content>
      </s:RichText>
    </s:Graphic>
  </mx:Canvas>
  <fx:Private>
    <fx:Date>10/22/2008</fx:Date>
    <fx:Author>Nick Danger</fx:Author>
  </fx:Private>
</s:Application>
```

Reparent

The `<Reparent>` language tag lets you specify an alternate parent for a given document node, in the context of a specific state.

For more information, see [“States”](#) on page 37.

Gumbonents

Flex 4 introduces a new set of components that take advantage of Flex 4 features. The components are sometimes known as Spark components, or Gumbonents.

In many cases, the differences between the Spark and Halo versions of the Flex components are not visible to you. They mainly concern the component’s interaction with the Flex 4 layouts, as well as the architecture of the skins and states.

In some cases, the Spark component is different from the Halo component. For example, the Spark Application and the Halo Application components are different in several ways: the differences include the background colors and the default layout.

The [Gumbo Component Architecture specification](#) describes each of the Gumbonents and provides some background information as to how and why changes were made from the Halo version of the component.

The following list of commonly-used components new to Flex 4 includes links to their ASDoc pages:

- [Application](#)
- [Button](#)
- [CheckBox](#)
- [List](#)
- [NumericStepper](#) (extends [Spinner](#))
- [RadioButton](#)
- [ScrollBar](#), [HScrollBar](#), and [VScrollBar](#)
- [Slider](#), [HSlider](#), and [VSlider](#)
- [TextArea](#)
- [TextInput](#)
- [ToggleButton](#)

Visual Gumbonents have additional classes that define their skins and behaviors. For example, in addition to the `Button` class, there is also a `ButtonSkin` class. The skin classes for Gumbonents are typically in the `spark.skins.*` package.

You can use Gumbonents and Halo components interchangeably in your Flex applications. The new namespace includes Gumbonents as well as the Halo components. You do this by specifying a namespace for each set of components. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- gumbonents/BasicGumbonentUsage.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:HorizontalLayout/>
    </s:layout>
    <s:Button label="Click Me"/>
    <mx:Button label="Click Me"/>
</s:Application>
```

Gumbonents and Halo components can be used within each others' containers. Gumbonents should work within Halo containers, and Halo components should work within Gumbonent containers.

Note: *The only restriction is that you cannot use a Gumbo container as a child of a Halo navigator container, such as `Accordion`. If you want to use a Gumbo container as a child of a navigator, wrap it in a Halo container, such as `HBox` or `VBox`.*

You can incrementally adopt Gumbonents in your applications, when and if they provide functionality or structure that Halo components lack. No Halo components have been removed from the Flex 4 framework.

All Gumbonents target Flash Player 10. If your application uses Gumbonents, your users must use Flash Player 10.

In some cases, you must use the Gumbonent to use new Flex 4 functionality. For example, if you want to declaratively specify `BasicLayout` for your application, you must use the `Spark Application` rather than `Halo Application` as the root tag in your Flex application.

Gumbonents are based on a new component architecture. This architecture attempts to separate the components into two parts, a part that controls the look and feel of the component, and a part that controls the behavior of the component. For more information about the new component architecture, see “[Component architecture](#)” on page 9.

Component architecture

In Flex 4, the core components are factored into separate classes that control different aspects of the component. One class, the base class, defines the core behavior of the component. This includes its events, data, subcomponents, and managing states. Another class, the skin class, manages everything related to the component’s appearance, including graphics, layout, states, transitions, and the way in which data is represented.

The skin class is associated with a component by using CSS. In the defaults.css file, each component’s type selector points to the skin class for that component. For example, the type selector in the defaults.css file for the Spark Button contains the following:

```
Button {  
    skinClass: ClassReference("mx.skins.spark.FxButtonSkin");  
}
```

The following table describes the five new base classes for Flex 4 components:

Class	Description
SkinnableComponent	The base class for all skinnable Flex 4 components. For example, Button.
Group	A base class that contains content items, an array of items that are owned and managed by the Group itself.
MXMLComponent	Defines components that are not skinnable. Most user-defined components will extend this class.
ItemsComponent	The base class for skinnable components that have content items, such as Panel.
Skin	A container class for declarative skin elements.

To support the changes to the basic component architecture, the following new skin-related lifecycle methods have been added:

Method	Description
<code>loadSkin()/unloadSkin()</code>	<p>Manage the loading and unloading of skins and re-assignment of skin elements when a style change affects a component skin. Both methods are called in the <code>commitProperties()</code> phase of the component lifecycle and are typically not invoked directly.</p> <p>The <code>loadSkin()</code> method loads the skin class for a component based on the <code>skinClass</code> style, finds the skin parts in the skin class and assigns them to properties in the component, and adds event listeners to the component.</p> <p>The <code>unloadSkin()</code> method removes references to skin parts and removes event listeners that act on the component.</p> <p>You typically do not override the <code>loadSkin()</code> and <code>unloadSkin()</code> methods when you write custom Gumbonents or extend existing Gumbonents.</p>
<code>partAdded()/partRemoved()</code>	<p>Handle the setup and teardown of skin parts.</p> <p>The <code>partAdded()</code> method attaches event listeners to newly-instantiated parts and pushes data or properties down to the skin part.</p> <p>Similarly, the <code>partRemoved()</code> method removes event listeners acting on the removed part as well as any other cleanup that is required when a part is removed.</p>
<code>commitSkinState()/invalidateSkinState()</code>	<p>Manage skin states.</p> <p>You never call the <code>commitSkinState()</code> method directly. Instead, when the component state changes, the component calls the <code>invalidateSkinState()</code> method which eventually invokes the <code>commitSkinState()</code> method to set the new state of the skin.</p>
<code>getUpdatedSkinState()</code>	<p>Returns the name of the state the skin should enter based on current interaction. This method is the main extension point when adding new states to a skin in addition to declaring the skin state in the skin class.</p>

See also

[Gumbo Component Architecture specification](#)

[Gumbo Skinning specification](#)

FXG

FXG is a declarative syntax for defining graphics in Flex applications. It can also be used as an interchange format with other Adobe tools such as Thermo. FXG very closely follows the Flash Player 10 rendering model.

FXG defines the following:

- Graphics and text primitives
- Fills, strokes, gradients, and bitmaps
- Support for filters, masks, alphas, and blend modes

In FXG, all graphic elements implement the `IGraphicElement` interface.

You can use the FXG elements either in an MXML file or as a component in a standalone document fragment. The following is a simple example of an FXG document fragment inside an MXML file:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- FXG/SimpleLineExample.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"
>
  <mx:Panel title="Line Graphic Example"
    height="75%" width="75%" layout="horizontal"
    paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">
    <mx:Canvas top="0" bottom="0" left="0" right="0">
      <s:Graphic x="0" y="0">
        <s:Line xFrom="0" xTo="100" yFrom="0" yTo="100">
          <s:stroke>
            <s:Stroke color="0x000000" weight="1"/>
          </s:stroke>
        </s:Line>
      </s:Graphic>
    </mx:Canvas>
  </mx:Panel>
</s:Application>

```

An FXG document fragment consists of a single definition and an optional library that is contained within a root `<Graphic>` element. An FXG document fragment can include zero or more containers and Graphic elements.

An FXG document fragment can also be a standalone *.mxml or *.fxg file that is used as a custom component inside a Flex application. In this case, the `<Graphic>` element must be the root of the document.

The following is an example of a separate FXG document that is loaded as a custom component inside a Flex application:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- FXG/comps/GraphicComp.mxml -->
<s:Graphic
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:Rect id="rect1" width="200" height="200" >
    <s:fill>
      <s:SolidColor color="0xFFFFCC" />
    </s:fill>
    <s:stroke>
      <s:SolidColorStroke color="0x660099" weight="2" />
    </s:stroke>
  </s:Rect>
</s:Graphic>

```

The following Flex application uses the component defined in GraphicComp.mxml:

