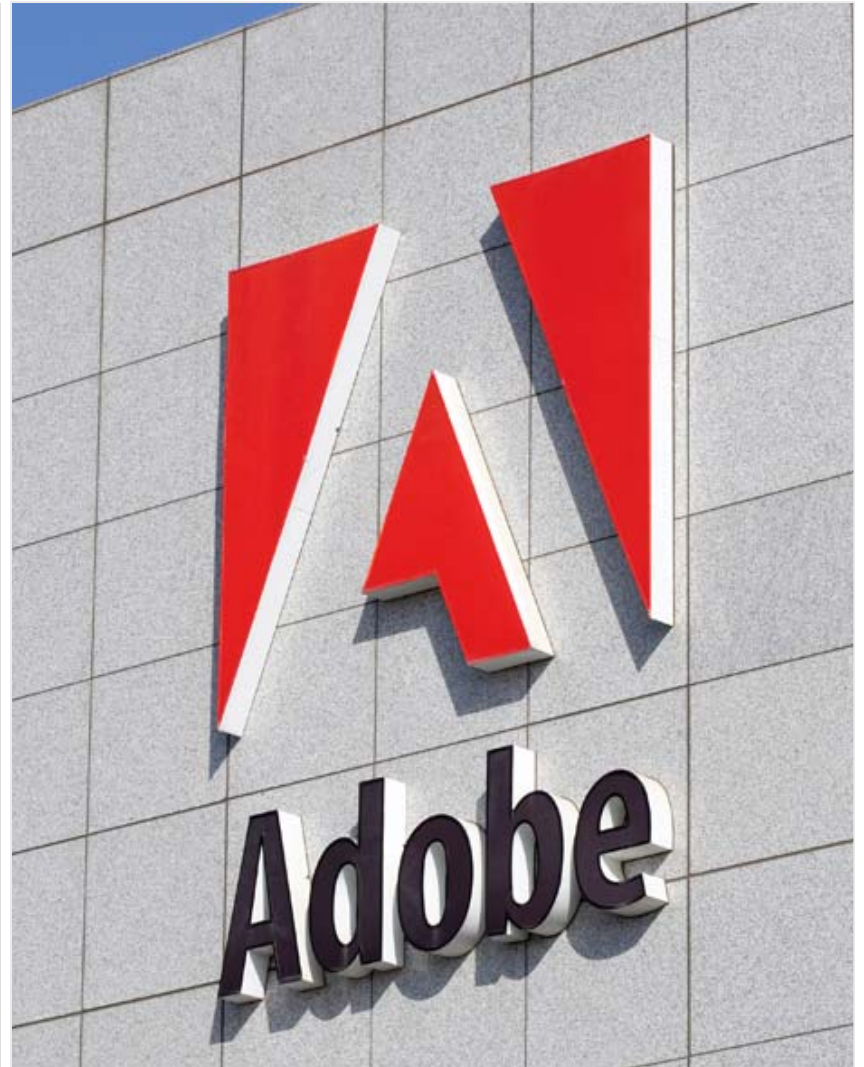


Flex Framework Internals Part 2: Layout and Styles

David George

Adobe Systems



Measurement/Layout: Definition

The process of assigning a position and size to every component

```
<mx:Application>  
  <mx:HBox>  
    <mx:Button label="1"/>  
    <mx:Button label="2"/>  
  </mx:HBox>  
  <mx:TextArea width="100%" height="100%" text="Text"/>  
</mx:Application>
```



Measurement/Layout: Description

```
<mx:Application>
  <mx:HBox>
    <mx:Button label="1"/>
    <mx:Button label="2"/>
  </mx:HBox>
  <mx:TextArea width="100%" height="100%" text="Text"/>
</mx:Application>
```

- Measurement Phase: traverse tree from bottom up
 - Buttons and TextArea compute measured sizes
 - HBox computes its measured size
 - Application computes its measured size
- Layout Phase: traverse tree from top down
 - Application sets sizes and positions of HBox and TextArea
 - HBox sets sizes and positions of Buttons
 - Buttons set sizes and positions of their borders, labels, etc.

Triggering a Measurement/Layout pass

- `invalidateProperties()`
 - Used to defer work related to a property change
 - Example: setter function for `ViewStack.selectedIndex`
- `invalidateSize()`
 - Called if we need to recalculate the component's measured sizes
 - Example: setter for `Button.label`
- `invalidateDisplayList()`
 - Called if the appearance of the component needs to change
 - Example: `Button.cornerRadius` is changed

Invalidation triggers the LayoutManager

- Invalidate functions notify the LayoutManager
- LayoutManager adds invalidated objects to dirty queues
- LayoutManager is notified after current script finishes
- LayoutManager traverses queues in three passes, calling component's methods:
 - commitProperties
 - measure
 - updateDisplayList

Measurement: the measure function

- The `measure()` function calculates and sets four properties:
 - `measuredWidth`, `measuredHeight`
 - `measuredMinWidth`, `measuredMinHeight`
- How these four values are used:

```
<mx:Button />
```

- Button's size is set equal to `measuredWidth/measuredHeight`
- `measuredMinWidth` and `measuredMinHeight` are ignored

```
<mx:HBox><mx:Button width="40%" /></mx:HBox>
```

- `measuredWidth` and `measuredHeight` are used to calculate parent's size
- `measuredMinWidth` and `measuredMinHeight` set a limit on object resizability

```
<mx:Button width="150" height="22" />
```

- The values are ignored. In fact, the `measure` function is never called.

Measurement: the measure function

```
override protected function measure():void
{
  for(var i:int = 0; i < numChildren; i++)
  {
    var child:UIComponent = UIComponent(getChildAt(i));
    measuredWidth = Math.max(child.getExplicitOrMeasuredWidth(), measuredWidth);
    measuredHeight += child.getExplicitOrMeasuredHeight() + VERTICAL_GAP;
  }
  measuredMinWidth = 10;
  measuredMinHeight = measuredHeight;
}
```

See also: the `measure()` function in `RandomWalk`

Layout: the `updateDisplayList` function

- `updateDisplayList` is passed two numbers - a width and a height
- `updateDisplayList` sets the position and size of each child
- `updateDisplayList` may use drawing API to draw on itself
- Example: the `updateDisplayList` function in `RandomWalk`

Layout: the component doesn't control its size

- The component user may set the component to any size
 - Example: `<mx:Button width="2" height="596"/>`
- The component must not draw or place children outside specified size
- Strategies:
 - Graceful degradation of content (when space gets tight, omit details)
 - Clipping
 - Scrolling
 - `UITextField.truncateToFit()`
 - Judicious zooming
 - Use of popups
 - Component-specific solutions

Layout-Related Properties and Methods

Different Kinds of Size

- **Measured size**
 - How large the component would “like” to be
 - The value calculated by the `measure()` function
- **Explicit size**
 - A value assigned by the user of the component
 - `<mx:Button width="100"/>` sets explicit size to 100 pixels
- **Percentage size**
 - Also a value assigned by the user of the component
 - `<mx:Button width="50%"/>` sets percentage size to 50
- **Actual size**
 - The actual, current size of the component
 - Value is chosen by parent’s `updateDisplayList` function

Size-related properties and methods

- Measured size `measuredWidth`
- Explicit size `explicitWidth`
- Percentage size `percentWidth`
- Actual size set using `setActualSize()`, get using “width”

- Setting “width”:
 - Sets actual width
 - Also sets `explicitWidth`
 - Exception: setting `width=“50%”` from MXML sets `percentWidth`
- `getExplicitOrMeasuredWidth()` returns `explicitWidth` if defined, else `measuredWidth`
- Getting “`unscaledWidth`” returns $(width / scaleX)$

Properties for min width and max width

- `measuredMinWidth`: calculated by `measure()`
- `explicitMinWidth`: assigned by component user
 - Example: `<mx:Button minWidth="20"/>`
- `minWidth`:
 - Getter returns `explicitMinWidth` if defined, else `measuredMinWidth`
 - Setter sets `explicitMinWidth`
- `maxWidth` and `explicitMaxWidth` are synonymous

Properties and methods related to position

- `x, y`:
 - Getter returns actual position
 - Setter sets actual position
 - Setter calls `invalidateProperties` to deliver "move" event
- `move()`
 - Sets actual position
 - Delivers the "move" event immediately

Cheat Sheet

Inside the `measure` and `updateDisplayList` functions, use the following:

- Get size of a child `UIComponent` using `getExplicitOrMeasuredWidth()`
- Set size of a child `UIComponent` using `setActualSize()`
- Get position of a child `UIComponent` using `x` and `y`
- Set position of a child `UIComponent` using `move()`
- For children that are not `UIComponents`, just use `x`, `y`, `width`, and `height`

Styles

Using Styles

- Setting styles from MXML:

```
<mx:Application fontFamily="Geneva">
  <mx:Style>
    Button { color: red }
    .myStyle { font-size: 18 }
  </mx:Style>
  <mx:Button id="btn"/>
  <mx:Label styleName="myStyle"/>
  <mx:ComboBox openDuration="1000"/>
</mx:Application>
```

- Some styles are inherited, such as fontFamily
- From ActionScript: `getStyle` and `setStyle`
 - `myButton.getStyle("color")` returns `0xFF0000`
 - `myButton.setStyle("color", 0x00FF00);`

Defining Styles in your Component

- Identify properties that should be styles

- Declare [Style] metadata

```
[Style(name="horizontalGap", type="Number", format="Length", inheriting="false")]  
public class RandomWalk extends UIComponent
```

- Set default initial value in user's application or defaults.css

```
RandomWalk {  
    horizontal-gap: 10;  
}
```

- Bulletproofing: also set default initial value in an internal wrapper property:

```
private function get horizontalGapWithDefault():Number  
{  
    var result:Number = getStyle("horizontalGap");  
    return isNaN(result) ? 10 : result;  
}
```

- Implement styleChanged() function

```
override public function styleChanged(styleProp:String):void  
{  
    super.styleChanged(styleProp);  
    if (styleProp == "horizontalGap")  
        invalidateSize();  
}
```

Attaching Styles to Sub-Components

- **Problem:**
 - Component creates child objects
 - Those child objects expose style-able properties
- **Example:**
 - RandomWalk creates a child object to render the border.
 - The border object expects styles like `borderColor` and `borderThickness`
- **Solution:**
 - Declare `[Style]` metadata for `borderColor` and `borderThickness` on `RandomWalk`
 - Set `border.styleName` property to reference the `RandomWalk` object
 - Component user can set `<RandomWalk borderColor="red"/>`

Attaching Styles to Sub-Components, Part 2

- Problem: different children need different values for same style
- Example:
 - DataGrid's children include text fields for header row and other rows
 - Want to allow user to specify different fontWeight and fontColor for header row
- Solution:
 - Declare a style that's the name of a CSS class selector
`[Style(name="headerStyleName", type="String", inherit="no")]`
 - Set styleName property of headerText child to be the name of the class selector:
`headerText.styleName = getStyle("headerStyleName")`
 - Component user does the following to make the header text red and other text blue:

```
<mx:Style>  
    .myHeader { color: red }  
</mx:Style>  
<mx:DataGrid color="blue" headerStyleName="myHeader"/>
```