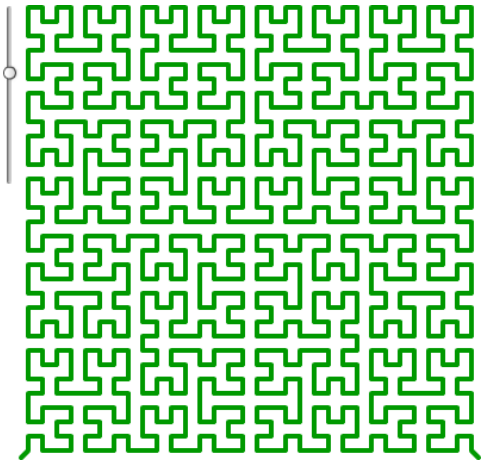
Simple Yet Demonstrative Example of Flex Capabilities: Interactive Hilbert Curve

A. Alekseeva

The present text shows how to make a Flex interactive widget drawing the Hilbert space-filling curve up to order n which is controlled by a slider.

Flex is an open web-oriented environment maintained by Adobe Systems, Inc., known for their open PDF format for printable documents which eventually became the world standard for open document interchange. Flex is also an free and open technology: sources are written in ActionScript (a dialect of ECMAScript) and MXML (a dialect of XML) are compiled by Flex SDK (can be freely downloaded from Adobe website for various operation systems) into SWF files, which can be included into web pages or PDF documents, or viewed separately.

The adoption of SWF format approaches the adoption of PDF format for printable documents. SWF players are available for Linux, Microsoft Windows, MacOS X and a variety of more exotic operation systems. SWF players are either installed by default or can be installed within one click on all Linux desktop distributions of last 2-3 years. Due to the necessity of SWF player for video- or audiostreaming services such as YOUTUBE or LAST.FM, SWF player is already installed on waste majority of computers. As consequence, Flex widgets can be used "out of the box" without installing additional software or even saving any files on the local data storage; in particular, in all libraries, CIP-pools and Internet cafés, where software installation by user is usually prohibited. Additionally, Flex widgets are compatible with mobile devices, such as netbooks, tablets, PDAs and smartphones. For those few who have not yet installed SWF player on their computer, one can also compile standalone platform-specific executables which can be started without installing *any* libraries whatsoever.



1 Internals of the widget

The Hilbert curve widget took about three hours of working time; its source code consists of six small files comprising only 3.8 KB in 118 lines.

```
./hilbert-curve/src/
  hilbert/
    HilbertCurve.as      – Methods for drawing Hilbert curve
    HilbertCurvePlot.as  – Hilbert curve UI-Component class
    Parameters.as        – Plot parameters' defaults and ranges
    Default.css          – Default plot style (color & size)
  Hilbert.mxml           – The widget outline
  Hilbert.css            – Widget style
```

Hilbert.mxml is the "entry point" of the widget.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                xmlns:hl="hilbert.*">
    <mx:Style source="Hilbert.css"/>
    <mx:HBox styleName="mainBox">
        <mx:VSlider id="nSlider"
                    value = "{hilbert.Parameters.nDefault}"
                    minimum = "{hilbert.Parameters.nMin}"
                    maximum = "{hilbert.Parameters.nMax}"
                    snapInterval = "1"
                    tickInterval = "1"
                    liveDragging = "false"/>
        <hl:HilbertCurvePlot id="plot" order="{nSlider.value}"/>
    </mx:HBox>
</mx:Application>
```

The first line is the mandatory XML declaration, which is followed by the root node `mx:Application`. The application uses two namespaces: `mx` for the predefined Flex components and `hl` for our custom Hilbert curve plot component. It contains a horizontal box with a vertical slider setting the Hilbert curve order and the Hilbert curve plot itself as it is defined in `HilbertCurvePlot.as`. The `order` parameter of the plot is bound to the value of the slider. Thus every time the slider value changes, the `HilbertCurvePlot` will be notified automatically.

The range and the default value for the slider are taken from `Parameters.as`. The step and the tick interval are set to 1 since only integer values are allowed for the order. The `liveDragging` feature is disabled, which means that the Hilbert curve will be redrawn only after the user releases the slider (otherwise it would be redrawn also while the user is dragging the slider).

Parameters.as is almost trivial:

```
package hilbert {
  /**
   * Ranges and initial values for Hilbert curve parameters
   */
  public class Parameters extends Object {
    public static const nDefault:uint = 2;  // The initial value
    public static const nMin:uint = 0;      // and the range of
    public static const nMax:uint = 8;      // Hilbert curve order
  }
}
```

HilbertCurve.as contains the methods for drawing the Hilbert curve.

```
package hilbert {
 import flash.display.CapsStyle;
 import flash.display.Graphics;
 import flash.geom.Rectangle;

 /**
  * This class contains drawing methods for the Hilbert curve.
  */
 public class HilbertCurve {
   /**
    * Paints the Hilbert curve.
    *
    * @param g Graphic context to draw in
    * @param rect Rectangle to fill
    * @param order Hilbert curve order
    * @param color Color to use
    */
   public static function draw(g:Graphics, order:uint,
         rect:Rectangle, color:uint):void {
     var thickness:Number = Math.min(rect.width, rect.height) / (3 <<
          order);

     g.clear();
     g.lineStyle(thickness, color, 1, false, "normal",
         CapsStyle.ROUND);

     g.moveTo(thickness/2, rect.height-thickness/2);
     drawRec(g, order, rect, 1, 3);
     g.lineTo(rect.width-thickness/2, rect.height-thickness/2);
   }

   /* Fills rectangle with a segment of Hilbert curve recursively.
    *
    * @param g Graphic context to draw in
    * @param order Hilbert curve order
    * @param rect Rectangle to fill
    * @param start Number of the corner the curve starts in
    * @param end Number of the corner the curve ends in
    */
   private static function drawRec(g:Graphics, order:uint,
       rect:Rectangle, start:uint, end:uint):void {
     if (order == 0) {
       g.lineTo(rect.x + rect.width/2, rect.y + rect.height/2);
     } else {
       drawRec(g, order-1, quadrant(rect, start), start, ~end);
       drawRec(g, order-1, quadrant(rect, ~end), start, end);
       drawRec(g, order-1, quadrant(rect, ~start), start, end);
       drawRec(g, order-1, quadrant(rect, end), ~start, end);
     }
   }
```

```
    /* Divides a rectangle into 4 quadrants and returns
     * the n'th of them.
     *
     * @param rect Rectangle to quadrisect
     * @param n Number of quadrant to return
     * @return n'th quadrant as a Rectangle
     */
    private static function quadrant(rect:Rectangle, n:uint):Rectangle
         {
      var quadrantWidth:Number = rect.width/2;
      var quadrantHeight:Number = rect.height/2;

      var shiftY:uint = n & 1;
      var shiftX:uint = (n >> 1) & 1;
      return new Rectangle(
        rect.x + shiftX*quadrantWidth,
        rect.y + shiftY*quadrantWidth,
        quadrantWidth, quadrantHeight
      );
    }
  }
}
```

HilbertCurvePlot.as contains mostly boilerplate code which is necessary to define a new UI-Component rendering the Hilbert curve.

```
package hilbert {
  import flash.geom.Rectangle;
  import mx.core.UIComponent;

  /**
   * Hilbert curve plot widget.
   */
  [Style(name="size", type="Number", format="Length", inherit="yes")]
  [Style(name="color", type="uint", format="Color", inherit="yes")]
  public class HilbertCurvePlot extends UIComponent {
    // Internal variables:
    private var _order:uint = Parameters.nDefault;
        // Hilbert curve order

    /* Getter and setter for the "order" property.
     */
    public function get order():uint {return _order; }
    public function set order(v:uint):void {
      if (v != _order) {
        _order = v;
        invalidateProperties(); // This marks component to be redrawn
      }
    }

    /* This method is called whenever some style property changes.
     */
    override public function styleChanged(styleProp:String):void {
      super.styleChanged(styleProp);
      invalidateProperties();  // This marks component to be redrawn
    }
```

```
    /* This obligatory method sets UI-Component measures
     */
    protected override function measure():void {
      measuredWidth = measuredHeight = getStyle("size");
    }

    /* This method is called by environment every time
     * the properties are changed. We override it in order
     * to redraw the curve after such changes.
     */
    protected override function commitProperties():void {
        draw()
    }

    /* Draws the Hilbert curve.
     */
    private function draw():void {
      var size:Number = getStyle("size");
      var color:uint = getStyle("color");
      var square:Rectangle = new Rectangle(0, 0, size, size);
      HilbertCurve.draw(graphics, order, square, color);
    }
  }
```

Default.css must be provided for the case `HilbertCurvePlot` would be used with color and size unspecified.

```
@namespace "hilbert.*";

HilbertCurvePlot {
  color: green;
  size: 384;
}
```

Hilbert.css defines the appearance of the widget.

```
@namespace "http://www.adobe.com/2006/mxml";

Application {
  backgroundColor: silver;
}

Box.mainBox {
  backgroundColor: white;
  borderStyle: solid;  borderColor: darkBlue;
  paddingLeft: 10;     paddingRight: 10;
  paddingTop: 10;      paddingBottom: 10;
  horizontalGap: 10;   verticalGap: 10;
}

#nSlider {
    right: 10;  top: 10;
    dataTipPrecision: 0;
}

#plot {
  left: 10;    top: 10;
  size: 384;   color: green;
}
```

2 Localization and Internationalization

Let's add a title line to our widget, say, "`The Hilbert curve of order {0}`" in English and "`Hilbert-Kurve {0}. Ordnung`" in German. We should add a directory `locale` to our source directory and add two new files, both called `texts.properties` into subdirectories called `en_US` and `de_DE` (corresponding to the locales' names) of `src/locale`. Additionally, we'll include a package called `i18n` (common shorthand for "internationalization") containing the language selector component. That's what our project now looks like:

```
./hilbert-curve/src/
  locale/
    en_US/
      texts.properties
    de_DE/
      texts.properties
  i18n/
    LanguageSelector.mxml
  hilbert/
    HilbertCurve.as
    HilbertCurvePlot.as
    Parameters.as
    Default.css
  Hilbert.mxml
  Hilbert.css
```

Contents of `locale/en_US/texts.properties`:

```
title_text=The Hilbert curve of order {0}
```

Contents of `locale/de_DE/texts.properties`:

```
title_text=Hilbert-Kurve {0}. Ordnung
```

Now let's add the title string and language selector to `Hilbert.mxml`:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                xmlns:hl="hilbert.*" xmlns:i18n="i18n.*">
  <mx:Style source="Hilbert.css"/>
  <mx:Metadata>[ResourceBundle("texts")]</mx:Metadata>
  <mx:VBox>
    <i18n:LanguageSelector/>
    <mx:HBox styleName="mainBox">
      <mx:VSlider id="nSlider"
                  value = "{hilbert.Parameters.nDefault}"
                  minimum = "{hilbert.Parameters.nMin}"
                  maximum = "{hilbert.Parameters.nMax}"
                  snapInterval = "1"
                  tickInterval = "1"
                  liveDragging = "false"/>
      <hl:HilbertCurvePlot id="plot" order="{nSlider.value}"/>
    </mx:HBox>
    <mx:Text id="title" text="{resourceManager.getString('texts',
                                  'title_text', [nSlider.value])}"/>
  </mx:VBox>
</mx:Application>
```

That's all. Everything works now.

3 Conclusion

As you can see, Flex provides a convenient and well-structured environment for development of interactive and dynamical applications, which is also easy to learn. Adobe provides comprehensive online and offline documentation and an optional non-free IDE called Flash Builder (former Flex Builder) with such handy features as boilerplate code autogeneration and visual editor for MXML files.

The sources presented above compile to a small 75 KB standalone cross-platform SWF and can be optionally compiled into platform-specific standalone executables. As of today, no other technology provides the same degree of platform-independent accessibility.