JavaFX: New Kid on RIA Block

Sang Shin
Sun Microsystems
javapassion.com
Topics

- JavaFX Scripting language overview (Sang)
- Animation (Jim)
- Effects & Transforms (Jim)
- Events (Jim)
- Production Suite (Jim)
- Scene graph (Sang)
- Custom node (Sang)
- Media (Sang)
- Deployment (Sang)
- Parsing and Accessing RESTful Web services (Sang)
- Cross-domain network access (Sang)
- JavaFX Mobile (Sang)
JavaFX Script
Overview
What is JavaFX Script?

- JavaFX Script source files are called “scripts”
- Everything in JavaFX script is an expression
  - All blocks are expressions
  - The last line is the result
Features of JavaFX Script

• Declarative, statically-typed scripting language
• Facilitates rapid GUI development
• Runs on Virtual Machine for the Java™ platform
• Deployment options same as Java programs
• Fully utilizes Java class libraries behind the scenes
• For content designers and Media engineers
• Cool, interesting language features for building RIA apps
  > Object literal, Sequence, Data binding, Animation, Media
Class Definition

- Address class definition: The Address class declares street, city, state, and zip instance variables all of type String

```java
class Address {
    var street: String;
    var city: String;
    var state: String;
    var zip: String;
}
```
Declaring an Object Literal

- In the JavaFX Script programming language, an object instance can be created with an object literal (unlike Java).

- Example: The first word (Address) specifies the type of object, class, that you are creating.

```java
Address {
    street: "1 Main Street"; // separated by semi colons
    city: "Santa Clara";
    state: "CA";
    zip: "95050";
}
```
Declaring Object Literals

- When declaring an object literal, the instance variables may be separated by commas or whitespace, as well as the semi-colon.
- The following declaration is also correct:

```javascript
Address {
    street: "1 Main Street"  // separated by whitespace
    city: "Santa Clara"      // separated by whitespace
}
```

```javascript
Address {
    street: "200 Pine Street",  // separated by comma
    city: "San Francisco",      // separated by comma
}
```
Nesting an Object inside Another Object

• Nesting `Address` object inside `Customer` object

```python
def customer = Customer {
    firstName: "John";
    lastName: "Doe";
    phoneNum: "(408) 555-1212";
    address: Address {
        street: "1 Main Street";
        city: "Santa Clara";
        state: "CA";
        zip: "95050";
    }
}
```
What is a Sequence?

• In addition to the five basic data types, the JavaFX Script programming language also provides data structures known as sequences.

• A Sequence represents an ordered list of objects; the objects of a sequence are called items.
Creating Sequences

• One way to create a sequence is to explicitly list its items.
• Each element is separated by a comma and the list is enclosed in square brackets [ and ]

> For example, the following code declares a sequence and assigns it to a variable named weekDays
  > var weekDays = ["Mon", "Tue", "Wed", "Thu", "Fri"];
> The compiler knows that we intend to create a "sequence of strings" because the individual items are all declared as String literals
Specifying Sequence's Type Explicitly

• You can also explicitly specify a sequence's type by modifying its variable declaration to include the name of the type followed by "[]":

> var weekDays: String[] = ["Mon","Tue","Wed","Thu","Fri"];

> This tells the compiler that the weekDays variable will hold a sequence of Strings (as opposed to a single String).
Sequences with Shorthand Notation

• There is also a shorthand notation that makes it easier to create sequences that form an arithmetic series.

• To create a sequence consisting of the numbers 1 through 100, use the following:

  > var nums = [1..100];
Creating Sequences with Predicate

• You can use a boolean expression, or a predicate, to declare a new sequence that is a subset of an existing sequence. For example, consider the following:

```javascript
var nums = [1,2,3,4,5];
```

• To create a second sequence (based on items found in this first sequence) but containing only numbers greater than 2, use the following code:

```javascript
var numsGreaterThanTwo = nums[n | n > 2];
```

```javascript
// Select all items from the num sequence where the value of an item is greater than 2 and assign those items to a new sequence called numsGreaterThanTwo.
```
Binding

• Cause and effect – responding to changes
• *bind* operator allows dynamic content to be expressed declaratively
• Dependency based evaluation of any expression
• Automated by the JavaFX runtime rather than manually wired by the programmer
• Eliminates the listener pattern
Binding to a Simple Expression

```javascript
var x = 0;

// Bind variable x to variable y. Whenever the value of x changes, 
// the value of variable y automatically changes as well.
var y = bind x + 10;

x = 1;
println("----y after x is changed to 1 = {y}"); // y now equals 11

x = 47;
println("----y after x is changed to 47 = {y}"); // y now equals 57
```
Definition of a Bound Function

var scale = 1.0;

// makePoint is a bound function. It will be invoked even when a value of
// non-function-argument such as scale changes. If you remove bound
// keyword, then, the change of the value of scale does not invoke the
// function.
bound function makePoint(xPos : Number, yPos : Number) : Point {
    Point {
        x: xPos * scale
        y: yPos * scale
    }
}

class Point {
    var x : Number;
    var y : Number;
}
Invocation of a Bound Function

// Code in the previous slide

// The bind keyword, placed just before the invocation of makePoint, binds the newly created Point object (pt) to the outcome of the makePoint function.
var myX = 3.0;
var myY = 3.0;
def pt = bind makePoint(myX, myY);
println(pt.x);  // 3.0

myX = 10.0;
println(pt.x);  // 10.0

scale = 2.0;
println(pt.x);  // 20.0
Demo: Binding
Scene Graph
Why Declarative Syntax for Building GUI?

- Because the **structure of declared objects** in the code reflects the **visual structure of the scene graph**, and this enables you to understand and maintain the code easily.

- The order of elements you declare in the code matches the order in which they appear in the application.
What is Scene Graph?

- Scene Graph enables declarative GUI programming
- The scene graph is a tree-like data structure which defines a hierarchy of graphical objects in a scene.
- Each item in the scene graph tree is called a Node.

  > Branch nodes are either of type `Group` or of type `CustomNode`, or a subclass of thereof.

  > Leaf nodes are classes such as `javafx.scene.shape.Rectangle`, `javafx.scene.text.Text`, `javafx.scene.image.ImageView`, `javafx.scene.media.MediaView`, or other such leaf nodes.
Node – Coordinate System

• The Node class defines a traditional computer graphics "local" coordinate system in which the x axis increases to the right and the y axis increases downwards.

• The concrete node classes for shapes provide variables for configuring the geometry and location of the shape within this local coordinate space.

  > For example, javafx.scene.shape.Rectangle provides x, y, width, height variables while javafx.scene.shape.Circle provides centerX, centerY, and radius.
Node – Transformations, Effects, Animation

• Any Node can have transformations applied to it.
  > Translation, Rotation, Scaling, Shearing

• Any Node can have effects applied to it
  > DropShadow, etc

• Any Node can have animation applied to it
  > Attributes can be typically bound with values that can be changed through animation
JavaFX Architecture

- JavaFX Script Software
- Scene Graph
- Effects
- Java 2D
- Graphics hardware

Models a JavaFX GUI
Scene Graph: Group

Group {
    transforms: Translate {
        x: 15, y: 15
    }
    content: [
        Text {
            x: 10, y: 50
            font: Font: {
                size: 50
            }
            content: "Hello World"
        }
        Circle {
            centerX: 100, centerY: 100
            radius: 40
            fill: Color.BLACK
        }
    ]
}
CustomNode
CustomNode

- Primary means of Scene Graph encapsulation
  > All other nodes are not extendable
- Use it when extending existing GUI class is not enough for your task
- Simply override the `create()` method, which returns a `Node` object
- Transformation, Effects, Animation can be built in the class or can be applied later to the object instance (just like any other Scene Graph node)
Simple CustomNode Example

- Extend CustomNode class and override create()

class Bars extends CustomNode {
    override function create():Node {
        return Group {
            content: for(x in [0..4]) {
                Rectangle {
                    y: indexof x * 20
                    width: 100
                    height: 10
                    fill: Color.RED
                }
            }
        };
    }
}

// Bars object literal
Bars { }
CustomNode with Animation Built-in

```javascript
var animation_length: Number = 30;

Timeline {
    ...
}.playFromStart();;

class Bars extends CustomNode {
    var height = 10;
    override function create():Node {
        return Group {
            content: for(x in [0..4]) {
                Rectangle {
                    y: indexof x * 20
                    width: bind animation_length - (x * 5)
                    height: bind height
                    fill: Color.RED
                }
            } // content
        } // Group
    }
```
Demo:

Creating New CustomNode With Built-in Animation

Demo:

Building “Under the Sea” Step by Step

http://www.javapassion.com/handsonlabs/javafx_animation/index.html#4.1
Demo:

Building “Picture Display” Step by Step

Media
Motivation and Goals

• Video and audio are ubiquitous on the Net
• Java support is spotty at best – JMF
  > Need to work “out of the box”
• Top grade media support
  > Simple to deploy and program
  > Zero configuration, support whatever the native platform supports
  > Integration with JavaFX platform – scenegraph
FX Media API Overview

- Simple model-view-controller design
- All classes in `javafx.scene.media` package.
- `MediaView` takes all cool features of SceneGraph node, such as effects, transforming, clip, opacity, etc...
Media Classes

• Media – represents the media source
  > Tracks – audio, video and subtitles currently supported
  > Duration, size, etc
  > Metadata information

• MediaPlayer – controls for playing media
• MediaView – display for MediaPlayer
Media Format Support

• Platform specific media types are supported by native platform
  > DirectShow on Windows (wmv, wav, avi, asf, etc...)
  > Quicktime on Mac (mov, mp4, etc)

• FXM is the cross platform format that can be played across desktop computers and mobile phones.

• FXM is an FLV subset which currently consists of
  > VP6 Video Encoding
  > MP3 Audio Encoding

• On2 FLX tool available for transcoding.
Example of Creating a Media Player

```javascript
var video: Media = Media {
    source: "http://..."
};

var player: MediaPlayer = MediaPlayer {
    media: video
    rate: 1.0
    volume: 0.4
};

var view: MediaView = MediaView {
    mediaPlayer: player
    x: 200
    y: 200
};

Stage {
    title: "Media Player"
    width: 700
    height: 700
    scene: Scene {
        content: [view]
    }
}
```
Demo: Media
Deployment
Deployment Options

• JavaFX 1.0 applications can be deployed using the two standard Java deployment technologies
  > Java Plugin: A tool used for deploying Java applets that run inside a web browser
  > Java Web Start: A tool used for deploying stand-alone Java applications on the desktop, using JNLP (Java Network Launching Protocol).

• Or using mobile emulation
  > JavaFX 1.0 Mobile Emulator Beta Release: A tool provided with the JavaFX 1.0 SDK, which displays your applications as they would look on a typical mobile device.
New Plugin Architecture

- Live outside the browser
  - Cannot crash the browser
- Applets can run in separate JVM
  - Choose the version they want
  - Configurable per applet basis
  - Live beyond the browser
- Unify Applet and JavaWeb Start model
- Works on FF3 and IE7 only
Execution Models

- Standard, Web Start, Applet, Mobile emulator
Java Deployment Kit

• A Javascript library that detects JRE, download missing software, create applet tags, etc
• Enables easier Applet deployment
• Works without any JRE installed in the browser
• http://java.com/js/deployJava.js (for Java applets)
  > Creates deployJava Javascript object
• http://dl.javafx.com/1.1/dtfx.js (for JavaFX applets)
  > Extended with JavaFX
  > Includes http://java.com/js/deployJava.js
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>PhotoFlip</title>
</head>
<body>
<h1>PhotoFlip</h1>
<script src="http://dl.javafx.com/1.1/dtfx.js"></script>
<script>
javafx(
    {
        archive: "PhotoFlip.jar",
        draggable: true,
        width: 300,
        height: 340,
        code: "fliptransition.Main",
        name: "PhotoFlip"
    }
);
</script>
</body>
<html>
Demo:

Draggable Applet
Parsing & Accessing RESTful Web Services
Pull Parser

- The pull parser supplies a sequence of events as it process the document under application control
- XML and JSON are the two data formats currently supported.
  - `documentType: PullParser.XML`
- Java InputStream object needs to be provided
  - `input: aInputStreamObject`
Usage Modes of Pull Parser

- The parser can be used in two different modes
  - in 'callback' mode
    - Use it with `onEvent` handler
  - in 'linear' mode
    - Use it with `forward()` and `seek()`
    - Pulls events, discarding or skipping over those it is not interested in
PullParser with onEvent Handler

```java
var total;
var title;
def parser = PullParser {
    documentType: PullParser.XML;
    input: anInputStreamThatContainsXML;
    onEvent: function(event: Event) {
        if (event.type == PullParser.START_ELEMENT) {
            if (event.qname.name == "ResultSet" and
                event.level == 0) {
                total = event.getAttributeValue(QName{name:"totalResultsAvailable"});
            }
        } else if (event.type == PullParser.END_ELEMENT) {
            if (event.qname.name == "Title" and
                event.level == 2) {
                title = event.text;
            }
        }
    }
}
parser.parse();
parser.input.close();
println("results: \{total\}, title: \{title\}");
```
PullParser with forward() and seek()

```javascript
parser.onEvent = function(event: Event) {
    println(event)
}

parser.forward();
parser.forward(2);
parser.seek(QName{name:"child"});
parser.seek(QName{name:"child"}, 2);
```

HttpRequest

- Used to make asynchronous HTTP requests.
- Can also be used to invoke RESTful Web Services.
- The location and the HTTP method are specified as attributes
  > GET method is default
- Lifecycle event handlers can be specified
  > onStarted, onConnecting, onWriting, onReading, onDone, etc
  > onInput
- The `enqueue()` method is then used to start request asynchronously
Performing a GET Request

def getRequest: HttpRequest = HttpRequest {
    location: "http://www.wikipedia.org";

    onStarted: function() {
        println("onStarted - started
            performing method:{getRequest.method}
            on location:{getRequest.location}");
    }

    onConnecting: function() { println("onConnecting") }
    onDoneConnect: function() { println("onDoneConnect") }
    onReadingHeaders: function() { ... }
    onResponseCode: function(code:Integer) { ... }
    onResponseMessage: function(msg:String) { ... }
    onResponseHeaders: function(headerNames: String[]) { ,,,}
    onReading: function() { ... }
    onToRead: function(bytes: Integer) { ... }
    onRead: function(bytes: Integer) { ... }
    onInput: function(is: java.io.InputStream) { ... }
    onException: function(ex: java.lang.Exception) { ... }
    onDoneRead: function() { println("onDoneRead") }
    onDone: function() { println("onDone") }
}

getRequest.enqueue();
Performing a POST Request

def testContent: String = "test content";
def testContentSize: Integer = testContent.getBytes().length;

class PostRequest extends HttpRequest {
    override public var method = HttpRequest.POST;

    // override the enqueue function in order to make sure
    // the Content-Type and Content-Length headers are provided
    override function enqueue(): Integer {
        setHeader("Content-Type", "somecontent/type");
        setHeader("Content-Length", "{testContentSize}");
        return super.enqueue();
    }
}

def postRequest: HttpRequest = PostRequest {
    location: "http://localhost:8080/TestServlet/";
    onStarted: function() { ... }
};

postRequest.enqueue();
Demo:

Yahoo CoffeeShop Search
Cross-Domain Network Access
Let's Go Over Sandbox Security Model

• The applet security model, known as the sandbox, only lets applets connect to the webserver they were loaded from.
  > They cannot connect to anywhere else unless they are signed.
  > Designed to prevent “denial of service” attack

• Signing is great when you need access to more than what is allowed inside the sandbox, but it has two problems:
  > Your user will receive an ugly warning dialog about the applet, and the applet will have full access to the user's computer.
  > Full access is overkill when all you want to do is talk to a webservice on another server
Cross-domain Network Access

• Ordinarily, unsigned JavaFX Scripts may only connect back to the web server from which they originated.

• Web servers now have facility to selectively open up their data to some clients via the use of a policy file called crossdomain.xml.

• The ability to check this file has been added to the JRE Version 6 Update 10 and also to JavaFX Script Runtime version 1.1
crossdomain.xml policy file

• If the server hosting a webservice has special xml file on it then the applet plugin will allow connections to that server.

• This special file is called a crossdomain.xml file and it must be present on the exact subdomain hosting the webservice.

• The crossdomain.xml mechanism was originally designed for Flash applications, but with JavaSE 6 update 10 now Java apps can take advantage of these services too!
Example: crossdomain.xml file for the Flickr server

```xml
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```
JavaFX Mobile
JavaFX Platform

Applications and Services

App Framework

Mobile Runtime
- Mobile Extensions

Desktop Runtime
- Desktop Extensions

TV Runtime
- TV Extensions

Common Platform

Java ME

Java SE

Authoring Tools
- Developer Tools
- Designer Tools
Mobile vs. Desktop

- Common profile
- Different implementation
- Platform specific APIs
  - Performance
  - Capabilities
Demo:
JavaFX Mobile

http://javafx.com/samples/SimpleVideoPlayerMobile/index.html
JavaFX Technology Overview