JavaFX

JavaFX Interoperability with SWT

Release 2.2

E24175-03

August 2012

Learn how to add a JavaFX scene graph to a Standard Widget Toolkit (SWT) application, and how to make SWT and JavaFX controls interoperate.

ORACLE

JavaFX/JavaFX Interoperability with SWT, Release 2.2

E24175-03

Primary Author: Steve Northover

Contributing Author: Nancy Hildebrandt

Contributor:

Contents

1 JavaFX Interoperability with SWT

1

JavaFX Interoperability with SWT

This article shows how to add a JavaFX scene graph to a Standard Widget Toolkit (SWT) application, and how to make SWT and JavaFX controls interoperate.

Introduction

If you develop SWT applications, you know that SWT uses the native operating system controls and cannot easily be configured to use advanced GUI features, such as animation. You can quickly add sparkle to an SWT application by integrating JavaFX with SWT. All you need is the `FXCanvas` class, which is located in the `javafx.embed.swt` package. FXCanvas is a regular SWT canvas that can be used anywhere that an SWT canvas can appear. It's that simple.

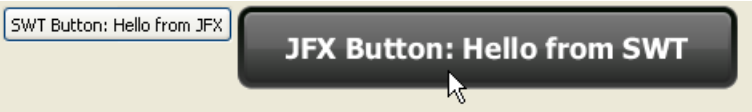
In this article, you will see how to create an interactive SWT button and JavaFX button, shown in Figure 1–1.

Figure 1–1 SWT Button on Left, JavaFX Button on Right

When the user clicks either button, the text is changed in the other button, as shown in Figure 1–2 and Figure 1–3. This example shows how the SWT code and JavaFX code can interoperate.

Figure 1–2 Clicking the SWT Button Changes the JavaFX Button Label

Figure 1–3 Clicking the JavaFX Button Changes the SWT Button Label



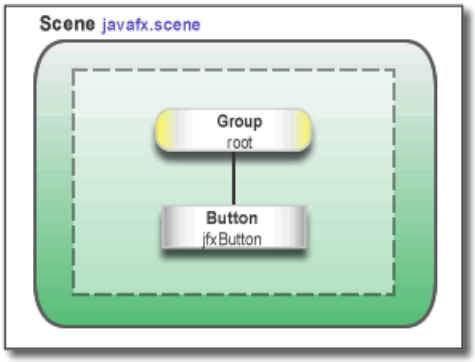
Adding JavaFX Content to an SWT Component

In JavaFX, the Java code that creates and manipulates JavaFX classes runs in the JavaFX User thread. In SWT, code that creates and manipulates SWT widgets runs in the event loop thread. When JavaFX is embedded in SWT, these two threads are the same. This means that there are no restrictions when calling methods defined in one toolkit from the other.

Example 1–1 shows the code to create the SWT button and JavaFX button shown in Figure 1–1. As shown in the code, you set JavaFX content into an FXCanvas with the `setScene()` method in the `FXCanvas` class. To force SWT to lay out the canvas based on the new JavaFX content, resize the JavaFX content first. To do this, get the JavaFX Window that contains the JavaFX content and call `sizeToScene()`. When JavaFX is embedded in SWT, a new preferred size is set for `FXCanvas`, enabling SWT to resize the embedded JFX content in the same manner as other SWT controls.

JavaFX constructs content in terms of a hierarchical scene graph, placed inside a scene. The code in Example 1–1 places the JavaFX button into a scene with the scene graph shown in Figure 1–4 and described in comments in the code example.

Figure 1–4 JavaFX Scene Graph in SWT Application



Example 1–1 Java Code for Plain SWT and JavaFX Buttons

```
import javafx.embed.swt.FXCanvas;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.paint.Color;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.layout.RowLayout;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
```

```
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Shell;

public class TwoButtons {

    public static void main(String[] args) {
        final Display display = new Display();
        final Shell shell = new Shell(display);
        final RowLayout layout = new RowLayout();
        shell.setLayout(layout);

        /* Create the SWT button */
        final org.eclipse.swt.widgets.Button swtButton =
                new org.eclipse.swt.widgets.Button(shell, SWT.PUSH);
        swtButton.setText("SWT Button");

        /* Create an FXCanvas */
        final FXCanvas fxCanvas = new FXCanvas(shell, SWT.NONE) {
            public Point computeSize(int wHint, int hHint, boolean changed) {
                getScene().getWindow().sizeToScene();
                int width = (int) getScene().getWidth();
                int height = (int) getScene().getHeight();
                return new Point(width, height);
            }
        };
        /* Create a JavaFX Group node */
        Group group = new Group();
        /* Create a JavaFX button */
        final Button jfxButton = new Button("JFX Button");
        /* Assign the CSS ID ipad-dark-grey */
        jfxButton.setId("ipad-dark-grey");
        /* Add the button as a child of the Group node */
        group.getChildren().add(jfxButton);
        /* Create the Scene instance and set the group node as root */
        Scene scene = new Scene(group, Color.rgb(
                shell.getBackground().getRed(),
                shell.getBackground().getGreen(),
                shell.getBackground().getBlue()));
        /* Attach an external stylesheet */
        scene.getStylesheets().add("twobuttons/Buttons.css");
        fxCanvas.setScene(scene);

        /* Add Listeners */
        swtButton.addListener(SWT.Selection, new Listener() {

            public void handleEvent(Event event) {
                jfxButton.setText("JFX Button: Hello from SWT");
                shell.layout();
            }
        });
        jfxButton.setOnAction(new EventHandler<ActionEvent>() {

            public void handle(ActionEvent event) {
                swtButton.setText("SWT Button: Hello from JFX");
                shell.layout();
            }
        });

        shell.open();
        while (!shell.isDisposed()) {
```

```
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }
}
```

The button style is based on a blog by Jasper Potts at the following location:
http://fxexperience.com/2011/12/styling-fx-buttons-with-css/

Creating SWT-JavaFX Applications in an IDE

Creating an SWT-JavaFX application in an IDE is simply a matter of adding the following libraries to your project:

- swt.jar, from an SWT zip download, available at http://eclipse.org/swt
- jfxrt.jar, from one of the following locations:
  - If JavaFX is bundled with the JDK (JDK 7u6 and later), the *JDK_HOME*/jre/lib directory. For example, for a default JDK installation on Windows, the full path is:
    C:\Program Files\Java\jdk1.7.0_06\jre\lib
  - If you are using a standalone installation of the JavaFX SDK, the *JAVAFX_SDK_HOME*/lib directory. For example, for a default Windows installation, the full path is:
    C:\Program Files\Oracle\JavaFX 2.0 SDK

  The jfxrt.jar library is required to validate your JavaFX code in the IDE and for compiling.

Note: Ensure that all JAR files are either 32 bit or 64 bit, as required for your environment.

Packaging SWT-JavaFX Applications

How you package your SWT-JavaFX application depends on whether JavaFX is bundled with the JDK (7u6 and later) or installed in a different location (for releases prior to JDK 7u6).

Packaging the Application when JavaFX is Bundled with the JDK

If you use NetBeans IDE 7.2 or later, no special handling is required to package your application, provided you have added the libraries as described in Creating SWT-JavaFX Applications in an IDE. You can simply do a Clean and Build, which produces a double-clickable JAR file in the /dist directory of the project.

Packaging the Application with a Standalone JavaFX Installation

When an SWT-JavaFX application is built, the JAR file must be packaged as a JavaFX application so the application on startup will look for the standalone JavaFX Runtime on the user's system. The SWT library (swt.jar) must be included as a resource (32-bit or 64-bit to match the target system).

The JavaFX SDK includes JavaFX Ant tasks to build JavaFX applications. The ant-javafx.jar file is required to load the JavaFX Ant task definitions. It is located in the *javafx-sdk-home*\lib directory of the JavaFX SDK. You must also declare the fx: namespace to load the JavaFX Ant task definitions. For more information about JavaFX Ant tasks, see http://docs.oracle.com/javafx/2/deployment/javafx_ant_task_reference.htm

If you used NetBeans IDE to create a Java application for your SWT-JavaFX code, you can build the application with two extra steps:

- Add ant-javafx.jar to as a Compile library to the NetBeans project properties. The default location is *JAVAFX_SDK_HOME*/lib.

  The ant-javafx.jar file contains a set of Ant tasks for packaging JavaFX applications.

- Override some of the default Ant build tasks in order to build a JavaFX application and include the SWT library as a resource. Do this by modifying the build.xml file in the NetBeans project directory, as described in the following example.

Example 1–2 shows a custom build.xml script for NetBeans IDE, containing those overrides, for installation of the JavaFX 2.1 SDK. This build.xml script is included in the TwoButtons sample application. After you do a Clean and Build in NetBeans IDE, you can run the application outside NetBeans IDE by double-clicking the JAR file.

Example 1–2 Custom build.xml Script to Build the Application JAR File

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Declare the fx: namespace, necessary for JavaFX Ant task definitions -->
<project name="TwoButtons" default="default" basedir="."
         xmlns:fx="javafx:com.sun.javafx.tools.ant">
    <description>Builds, tests, and runs the project TwoButtons.</description>

    <import file="nbproject/build-impl.xml"/>

    <!-- Try to find the JavaFX SDK -->
    <target name="find-javafx" unless="javafx.sdk">
        <property environment="env" />
        <condition property="javafx.sdk"
                   value="${env.ProgramFiles(x86)}/Oracle/JavaFX 2.1 SDK/"
                   else="${env.ProgramFiles}/Oracle/JavaFX 2.1 SDK/">
            <and>
                <contains string="${os.arch}" substring="x86"/>
                <available file="${env.ProgramFiles(x86)}/Oracle/JavaFX 2.1 SDK/
rt/lib/jfxrt.jar"/>
            </and>
        </condition>
    </target>

    <!-- Check if the jfxrt.jar library exists in the specified JavaFX SDK
         directory-->
    <target name="check-javafx">
        <available file="${javafx.sdk}/rt/lib/jfxrt.jar"
                   property="found-javafx"/>
    </target>

    <!-- If the JavaFX SDK cannot be found -->
    <target name="javafx-missing" unless="found-javafx">
        <fail>.
            Ant could not find the JavaFX 2.1 SDK. Please set [javafx.sdk]
```

```
            on the command line. For example:
            ant -Djavafx.sdk="C:\Program Files\Oracle\JavaFX 2.1 SDK"
            or ant -Djavafx.sdk="C:\Program Files (x86)\Oracle\JavaFX 2.1 SDK"
        </fail>
    </target>

    <!-- When not running in NetBeans IDE, try to locate the JavaFX SDK -->
    <target name="-pre-init" depends="find-javafx, check-javafx"
            unless="netbeans-home">
        <echo message="Using JavaFX SDK: ${javafx.sdk}"/>
        <property name="javafx.tools.ant.jar"
                  value="${javafx.sdk}/lib/ant-javafx.jar"/>
        <property name="file.reference.jfxrt.jar"
                  value="${javafx.sdk}/rt/lib/jfxrt.jar"/>
        <property name="file.reference.ant-javafx.jar"
                  value="${javafx.sdk}/lib/ant-javafx.jar"/>
    </target>

    <target name="-pre-compile" depends="javafx-missing">
    </target>

    <target name="-pre-jar" depends="javafx-missing">
    </target>

    <target name="-post-jar">
        <taskdef resource="com/sun/javafx/tools/ant/antlib.xml"
                 uri="javafx:com.sun.javafx.tools.ant"
                 classpath="${javafx.tools.ant.jar}"/>

        <!-- Remove the JavaFX libraries from /dist/lib because the app
             will use the installed JavaFX Runtime -->
        <delete file="${dist.dir}/lib/jfxrt.jar"/>
        <delete file="${dist.dir}/lib/ant-javafx.jar"/>

        <!-- Package the JAR file so that it has the code to
             find the installed JavaFX Runtime -->
        <fx:jar destfile="${dist.jar}">
            <fx:application mainClass="${main.class}"/>
            <fileset dir="${build.classes.dir}"/>

            <!-- Add the SWT library -->
            <fx:resources>
                <fx:fileset dir="dist" includes="lib/swt.jar"/>
            </fx:resources>

            <!-- Add information for the manifest -->
            <manifest>
                <attribute name="Implementation-Vendor"
                           value="${application.vendor}"/>
                <attribute name="Implementation-Title"
                           value="${application.title}"/>
                <attribute name="Implementation-Version"
                           value="1.0"/>
            </manifest>
        </fx:jar>
    </target>
</project>
```

Special VM Option for Mac

On Mac, in order for SWT applications to run, the `-XstartOnFirstThread` VM option must be specified. SWT applications run their event loop in `main()`, unlike JavaFX and AWT/Swing, and the Mac needs to be given this information.

In the NetBeans IDE for Mac, you must add `-XstartOnFirstThread` in the VM Options field of the Run category in Project Properties to run SWT applications.

In the Eclipse IDE for Mac, when a Java program references SWT, the IDE automatically adds the VM option `-XstartOnFirstThread`. In most cases, this automatic addition is helpful. However, there is one case when adding this VM option causes a problem, namely with an Eclipse project for an SWT application that also includes one or more "pure" JavaFX classes that do not interoperate with the SWT classes. A "pure" JavaFX application that is launched from such an Eclipse project will hang because it does not expect `-XstartOnFirstThread`.

The following issue has been reported related to this issue due to automatic insertion of the `XstartOnFirstThread` VM option in Eclipse on Mac:
https://bugs.eclipse.org/bugs/show_bug.cgi?id=211625

As a workaround, for SWT applications that contain pure JavaFX classes, you can create an Eclipse "Standard VM" instead of using the default. The Standard VM that you create points to the same Java, but the Eclipse IDE does not add the `-XstartOnFirstThread` VM option.