Blu-ray Disc Application Development with Java ME, Part 2: Responding to User Input

Print-friendly Version

By Bruce Hopkins, January 2009

Contents

Introduction

Java technology is a critical part of the new high-definition video standard: the Blu-ray Disc standard. This article is part 2 in a two-part series of articles that introduce developers to the Blu-ray Disc Java (BD-J) APIs. Using the BD-J APIs, developers can create Java ME applications for all Blu-ray disc players, including the Sony PlayStation 3 gaming console.

In Part 1, we provided an introduction to the BD-J platform and discussed the significant differences between the system requirements for developing, compared to playing, Blu-ray content. Additionally, we learned that the BD-J platform comprises various other supporting APIs, including GEM/MHP (Globally Executable Multi Home Platform) and Java TV. Furthermore, we were introduced to the application lifecycle of BD-J Xlets. We wrapped things up with a simple example application that draws text on the screen.

In this article, we introduce you to the Java ME SDK 3.0, which is the perfect tool for all Java ME development — whether you're doing Blu-ray application development or mobile phone application development. The Java ME SDK 3.0 provides substantial enhancements to its predecessor, the Wireless Toolkit for CLDC 2.5, mainly due to the fact that it allows developers the ability to author, edit, and compile all Java ME applications — especially of course, BD-J applications.

This article extends the code that was used in the previous article ,so that the application can respond to user input. Of course, since a Blu-ray player is a set-top device, you can't expect users to interact with it with a keyboard and mouse. So we're going to look at the APIs involved in responding to input from a remote control, regardless of whether the user used an infrared remote control or Bluetooth remote control (such as the one used on the PS3 gaming console).

Setting Up Your Development Environment with the Java ME SDK 3.0

Let's get started by getting our development environment setup so that we can compile our first BD-J application. The Java ME SDK 3.0 (which we'll refer to as the SDK from now on) isn't capable of compiling BD-J applications out-of-the-box due to the licensing restrictions on the BD-J libraries. What does this mean to you? It simply means that a `bdj.jar` file is **not** included in the SDK, and you'll need to come up with your own. The good news is that you can obtain one easily using one of the following two mechanisms:

1. Apply for the BD-J stubs for free from the Blu-ray Disc Association website.
2. Use the bdj.jar file that included with your PC Blu-ray Disc Player software installation.
   - You can only pursue this option if you have a PC-based Blu-ray player, such as Cyberlink PowerDVD or ArcSoft Media Player.
   - If you're not sure where this file is, then perform a search on your file system to find it. Don't worry: your PC player **must** have this file in order to play Blu-ray movies since the Java standard is a part of the Blu-ray disc specification.

Now after you've obtained a `bdj.jar` file, then all you need to do is to configure the BD-J platform in the SDK. This is accomplished by going to the Tools > Options menu items to configure the SDK. This operation is also reflected in Figure 1.

Figure 1: Setting the Path to the BD-J Stub Files

Now that you've configured the BD-J platform in the SDK, let's look at the steps involved to create and build a BD-J project.

Building and Creating Your First Project

The Java ME SDK 3.0 is the first freely available, commercial development kit that allows developers to author, build, and package BD-J applications. Years ago, the tools that enabled developers to author Blu-ray discs were few and far between, and required several thousands of dollars in licensing fees. This equated to the fact that, historically, only movie studios could afford to purchase the tools required to create BD-J movies and games. Now, the SDK enables any developer to create BD-J applications. This can be accomplished in three (3) easy steps:

1. Choose the Appropriate Project Type
2. Provide a Name and Location for Your Project
3. Select the Appropriate Platform

Step 1: Choose the Appropriate Project Type
Remember that the SDK is intended to be used not only for BD-J application development, but also for all Java ME platforms and configurations. When you create your new project, therefore, you need to specify the appropriate project type, which is CDC (Connected Device Configuration). The following figure depicts the appropriate selection for Step 1:

Figure 2: Creating a New CDC Project

Step 2: Provide a Name and Location for Your Project

In this step, you get the option to specify various details of your BD-J project. Here, you can provide the project name, folder path, package name, and Main Class name. This is the name of the Java class that will extend `javax.tv.xlet.Xlet`. In this example, we're going to write a basic BD-J Xlet that accepts user input from a remote control, so the fully qualified name of this class with be `firstbdjapp.UserInputXlet`. Figure 3 shows the settings that we used for this example.

Figure 3: Specifying the Name and Details for the Project

Step 3: Select the Appropriate Platform
The final step is to select the appropriate Java platform for your CDC project, which would be `External BD-J` instead of the default option. In this step, you can select the device you want to target and the profile that you want to use. Figure 4 shows the final step involved in creating a basic BD-J project.

Figure 4: Selecting the Appropriate Platform for Your Project

So, there you have it! The time it takes to create a simple BD-J project with the Java ME SDK is about 3 minutes. Now that that's out of the way, let's take a look at the source code that you can now compile using your newly created BD-J project.

Creating the UserInputXlet.java Application

One of the most critical features of your BD-J movie, game, or application is its ability to respond to user input. The source of the user input can originate from a variety of mechanisms, including the following:

- An infrared remote control
- A Bluetooth remote control
- A keyboard attached to a PC-based Blu-ray player
- Hardware buttons that are physically attached to a set-top Blu-ray player

As you may recall from Part 1 in this series, the BD-J APIs depend upon several other preexisting Java ME specifications including Java TV and GEM/MHP. The good news for us is that GEM/MHP APIs provide an abstraction layer between your code and the device or mechanism that was used to send the input commands to the Blu-ray player. This is great news for developers because it allow us to focus on *what* commands the user has sent, and not *how* the user sent it.

Now, the traditional Java paradigm for event handling is to use the observer pattern, where you create an object that implements a particular interface. That object "subscribes" to events of a certain type, and then receives callbacks on the method that implemented the interface. If you've done any MIDP, AWT, or Swing event handling, then this paradigm should be very familiar to you. Listing 1 shows the entire contents of our `initXlet()` method and how we can set up our application to listen to user-input events.

Listing 1: The `initXlet()` Method of `UserInputXlet.java`

```
public void initXlet(XletContext context) {
    // the code to get the key events
```

```
    UserEventRepository userEventRepository = new UserEventRepository("Blu-ray events");
    userEventRepository.addAllArrowKeys();
    userEventRepository.addKey(HRcEvent.VK_ENTER);
    userEventRepository.addAllColourKeys();

    userEventListener = new UserEventListener() {

        public void userEventReceived(UserEvent userEvent) {
           if (userEvent.getType() == HRcEvent.KEY_PRESSED){
               if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_0){
                  message = "key pressed: Colored Key 0";
                  backgroundColor = new Color(255, 10, 10); //red
               } else

               if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_1){
                  message = "key pressed: Colored Key 1";
                  backgroundColor = new Color(10, 255, 10); //green
               } else

               if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_2){
                  message = "key pressed: Colored Key 2";
                  backgroundColor = new Color(255, 255, 10); //yellow
               } else

               if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_3){
                  message = "key pressed: Colored Key 3";
                  backgroundColor = new Color(10, 10, 255); //blue
               }
               gui.repaint();
           }
        }
    };

    EventManager.getInstance().addUserEventListener(userEventListener, userEventRepository);

    scene = HSceneFactory.getInstance().getDefaultHScene();

    backgroundColor = new Color(10, 10, 10);  // black
    gui = new TextContainer();

    gui.setSize(1920, 1080);  // BD screen size
    scene.add(gui, BorderLayout.CENTER);
    scene.validate();
}
```

First of all, we need to create an instance of the `UserEventRepository` class and to specify all the events that we're interested in. As you can see, in this case we're only interested in arrow key events, Enter key (also known as the OK key) events, or any of the color key events. **Note:** If you have a Blu-ray player, then you should notice that the remote control has four color keys, in addition to normal keys, that allow you to control the player. The next step therefore, is to create an instance of the `UserEventListener` class and to implement the `userEventReceived()` method with the code that we want to be executed when the user executes the key.

The default background color of the application is black, but when the user presses one of the color keys, we're going to repaint the screen, tell the user which button was pushed, and change the background color of the screen. You should note that the event handling code presented in Listing 1 will definitely let you know if `VK_COLORED_KEY_0` was pressed; but, depending upon your country's locale, that key may not be the same color as in other locales. In other words, in the United States and North America, `VK_COLORED_KEY_0` is the red color key, but you can't count on red to be `VK_COLORED_KEY_0` in every country.

Now let's take a look at the code that draws simple text strings on the screen, which is our inner class, `TextContainer`. Listing 2 is the code for TextContainer.

Listing 2: The Inner Class `TextContainer` of `UserInputXlet.java`

```
class TextContainer extends Container {

    public void paint(Graphics g) {
        g.setFont(font);
        g.setColor(backgroundColor);
        g.fillRect(20, 20, getWidth() - 40, getHeight() - 40);
        g.setColor(new Color(245, 245, 245));
        int message_width = g.getFontMetrics().stringWidth(message);
        g.drawString(message, (getWidth() - message_width) / 2, 500);

    }
```

```
    }
```

As you can see, TextContainer is a simple inner class that extends `java.awt.Container`. Since it is a container, it has its own graphics object which we can access via the `paint()` method which we overrode. This allows us to draw (or redraw) any part of the screen that I want. So, each time the `repaint()` method is called on this object, then the `paint()` method is called "as soon as possible." As you can see from Listing 1, after we receive a key event from the user, we call the `repaint()`method to redraw the entire screen and display which user event was received by the application. The following figure depicts how my PC-based Blu-ray player responds when I press the red color key button.

Figure 5: Success! We Can Now Capture and Respond to User Input Events

Understanding the Basic File Structure and Burning to a Blu-ray Disc

All right, so we are now at the point where we have a complete working example. We have a simple foundational application that can receive and respond to user input. This basic application can be extended to receive user input for any Blu-ray movie, game, or application that you want.

So how do you create an actual Blu-ray disc with the code from this article? The good news is that the process is extremely simple, thanks to the Java ME SDK 3.0. If you have a PC-based Blu-ray burner, all you need to do is copy the entire contents of your project's **deploy** folder to an empty Blu-ray disc and start the burning process. (You need to be sure that the file format is UDF 2.5).

This folder is located in the `%project_home%\build\deploy\` file path, and it contains two folders: BDMV and CERTIFICATE. The BDMV folder is the larger folder, which contains the index.bdmv file as well as these other items:

- AUXDATA (used for auxiliary data files such as fonts and sound data files)
- BACKUP (contains a backup copy of index.bdmv and other important data files)
- CLIPINF (used for storage of AV clips)
- JAR (used to keep the *.jar file(s) of your application)
- PLAYTLIST (used for storage of movie playlists)
- STREAM (used for storage of MPEG-2 transport streams)

Conclusion and Final Thoughts

In this two-part series, you were introduced to the capabilities of the Blu-ray Disc Java (BD-J) APIs and became familiar with Java ME SDK, which allow you to create BD-J applications that can be burned to actual Blu-ray discs.

The BD-J applications can be used for several purposes, including interactive menus, subtitles, PiP (picture-in-picture) and games. Note that although the PS3 is a gaming console as well as a Blu-ray player, the BD-J applications that you create will be confined to the limitations of the BD-J API for graphics rendering. This means that you'll be able to create some games using the BD-J graphics API — a good example is the Java version of Dragon's Lair. However, you don't get access to the 3D graphics engine, so you can't create games that render in 3D.

What's next? Well, there is definitely a lot of ground that we haven't covered with the BD-J APIs, such as

- Creating subtitles in your application
- Signing your code to use protected APIs on commercial Blu-ray players
- Inter-Xlet communication
- Accessing resources over the network
- Using the Virtual File System
- Initiating video playback of a video stream

As I stated in the previous article, it's a great time to be a Java developer!

Full Source Code

```
package firstbdjapp;

import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Font;
import java.awt.Graphics;

import org.havi.ui.HScene;
import org.havi.ui.HSceneFactory;

import org.dvb.event.UserEvent;
import org.dvb.event.UserEventListener;
import org.dvb.event.UserEventRepository;
import org.dvb.event.EventManager;
import org.havi.ui.event.HRcEvent;

/**
 * Just a simple xlet that draws a String in the center of the screen.
 */
public class UserInputXlet implements Xlet {

    private static Font font = new Font(null, Font.PLAIN, 48);;
    private HScene scene;
    private Container gui;
    private Color backgroundColor;
    private String message = "BD-J User Input Application";
    private UserEventListener userEventListener;

    /** Creates a new instance of UserInputXlet */
    public UserInputXlet() {
    }

    public void initXlet(XletContext context) {

        // the code to get the key events
        UserEventRepository userEventRepository = new UserEventRepository("Blu-ray events");
        userEventRepository.addAllArrowKeys();
        userEventRepository.addKey(HRcEvent.VK_ENTER);
        userEventRepository.addAllColourKeys();
        backgroundColor = new Color(10, 10, 10);  // black

        userEventListener = new UserEventListener() {

            public void userEventReceived(UserEvent userEvent) {
               if (userEvent.getType() == HRcEvent.KEY_PRESSED){
                   if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_0){
                      message = "key pressed: Colored Key 0";
                      backgroundColor = new Color(255, 10, 10); //red
                   } else

                   if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_1){
                      message = "key pressed: Colored Key 1";
                      backgroundColor = new Color(10, 255, 10); //green
```

```
                } else

                if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_2){
                   message = "key pressed: Colored Key 2";
                   backgroundColor = new Color(255, 255, 10); //yellow
                } else

                if (userEvent.getCode() == HRcEvent.VK_COLORED_KEY_3){
                   message = "key pressed: Colored Key 3";
                   backgroundColor = new Color(10, 10, 255); //blue
                }
                gui.repaint();
            }
        }
    };

    EventManager.getInstance().addUserEventListener(userEventListener, userEventRepository);

    scene = HSceneFactory.getInstance().getDefaultHScene();
    gui = new TextContainer();

    gui.setSize(1920, 1080);  // BD screen size
    scene.add(gui, BorderLayout.CENTER);
    scene.validate();
}

public void startXlet() {
    gui.setVisible(true);
    scene.setVisible(true);
}

public void pauseXlet() {
    gui.setVisible(false);
}

public void destroyXlet(boolean unconditional) {
    scene.remove(gui);
    scene = null;
    EventManager.getInstance().removeUserEventListener(userEventListener);
}


class TextContainer extends Container {

    public void paint(Graphics g) {
        g.setFont(font);
        g.setColor(backgroundColor);
        g.fillRect(20, 20, getWidth() - 40, getHeight() - 40);
        g.setColor(new Color(245, 245, 245));
        int message_width = g.getFontMetrics().stringWidth(message);
        g.drawString(message, (getWidth() - message_width) / 2, 500);

    }

}
}
```