Project Report
Operating Systems

Touch Pad Simulation in Mobile

Controlling PC cursor through mobile devices

Shashank Kumar Tekriwal (72672)
Cluster Innovation Centre
University of Delhi
9th – Dec, 2014

Contents

The Application

With this application you can control your PC/Laptop's cursor using your mobile. This is a simple *web application* that provides a touchpad on the client side for the users to control the cursor. The client interface runs on a touchscreen mobile in a browser that has *WebSocket* (1) support. The server side script should run on the PC whose cursor the user wants to control.

The application provides full mouse pad control that has the following features:

- Cursor Movement
- Left Click
- Right Click
- Scrolling
- Double finger tap right click
- Double finger scrolling

Server side script is in Java on the JavaEE 7 Platform.

The client side script is in HTML5 and Javascript.

The Client Side

The client interface has been written in Javascript. I am using touch based *event handlers* to detect motion on screen with single and double fingers. The events are detected and the corresponding messages are continuously send at the server for processing. The messages are being sent over the WebSocket protocol.

WebSockets

The WebSocket specification—developed as part of the HTML5 initiative—introduced the WebSocket JavaScript interface, which defines a full-duplex single socket connection over which messages can be sent between client and server. The WebSocket standard simplifies much of the complexity around bi-directional web communication and connection management. WebSocket represents the next evolutionary step in web communication compared to Comet and Ajax.

To establish a WebSocket connection, the client and server upgrade from the HTTP protocol to the WebSocket protocol during their initial handshake, as shown in the following example (2):

The browser sends a request to the server, indicating that it wants to switch protocols from HTTP to WebSocket. The client expresses its desire through the Upgrade header:

```
GET ws://echo.websocket.org/?encoding=text HTTP/1.1
Origin: http://websocket.org Cookie: __utma=99as
Connection: Upgrade
Host: echo.websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Upgrade: websocket
Sec-WebSocket-Version: 13
```

If the server understands the WebSocket protocol, it agrees to the protocol switch through the Upgrade header.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Fri, 10 Feb 2012 17:38:18 GMT
Connection: Upgrade
Server: Kaazing Gateway
Upgrade: WebSocket
Access-Control-Allow-Origin: http://websocket.org
Access-Control-Allow-Credentials: true
Sec-WebSocket-Accept: rLHCkw/SKsO9GAH/ZSFhBATDKrU=
Access-Control-Allow-Headers: content-type
```

At this point the HTTP connection breaks down and is replaced by the WebSocket connection over the same underlying TCP/IP connection. The WebSocket connection uses the same ports as HTTP (80) and HTTPS (443), by default.

To connect to an end-point, just create a new WebSocket instance, providing the new object with a URL that represents the end-point to which you wish to connect, as shown in the following example. Note that a ws:// and wss:// prefix are proposed to indicate a WebSocket and a secure WebSocket connection, respectively.

```
var myWebSocket = new WebSocket("ws://www.websockets.org");
```

The Server Side

The server side code is in Java. The Java API for WebSocket (JSR 356) provides support for creating WebSocket Java components, initiating and intercepting WebSocket events and creating and consuming WebSocket text and binary messages. The Java API for WebSocket is part of the Java EE 7 platform. The WebSocket API in Java is implemented through the *javax.websocket* package.

The Server side script receives messages from the client which it then parses and performs actions like click, cursor movement, scrolling etc. using the Robot API (3) of Java.

WebSocket API in Java

To implement a Websocket server listener a Websocket Endpoint (4) needs to be created. The Web Socket Endpoint represents an object that can handle websocket conversations. The Endpoint class holds lifecycle methods that may be overridden to intercept websocket open, error and close events. By implementing the onOpen method, the programmatic endpoint gains access to the Session object, to which the developer may add MessageHandler implementations in order to intercept incoming websocket messages. Each instance of a websocket endpoint is guaranteed not to be called by more than one thread at a time per active connection.

A Websocket endpoint example class:

```
import javax.websocket.OnClose;
import javax.websocket.OnError;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint("/actions")
public class DeviceWebSocketServer {

    @OnOpen
        public void open(Session session) {
    }

    @OnClose
        public void close(Session session) {
    }

    @OnError
        public void onError(Throwable error) {
    }

    @OnMessage
        public void handleMessage(String message, Session session) {
    }
}
```

Java Robots

This class is used to generate native system input events for the purposes of test automation, self-running demos, and other applications where control of the mouse and keyboard is needed. The primary purpose of Robot is to facilitate automated testing of Java platform implementations.

Java.awt.Robot class is used to take the control of mouse and keyboard. Once you get the control, you can do any type of operation related to mouse and keyboard through your java code. Java Robot Class is a part of the *java.awt* package. Using the class to generate input events differs from posting events to the AWT event queue or AWT components in that the events are generated in the platform's native input queue. For example, *Robot.mouseMove* will actually move the mouse cursor instead of just generating mouse move events.

Some basic functions that Robot class provides for making native system calls:

```
public void mouseMove(int x,int y)
public void mousePress(int buttons)
public void mouseWheel(int wheelAmt)
```

Message Parsing

In my code an example message from the client that is received on the server end looks like – *"Me"*, *"Mf", "Mm1*3" etc.*

These messages are parsed on the server. Example: *"Me"* would mean – *M* stands for a message for mouse related event + *e* stands for left click. Thus, this message would trigger a left click event.

```
robot.mousePress(InputEvent.Button1_Mask);
```

Similarly, message *"Mm1*3"* would mean to move the cursor from its current location to 1px in positive x-axis direction and 3px in the negative y-axis direction.

Bibliography

1. The Web Sockets API. *w3.org.* [Online] http://www.w3.org/TR/2009/WD-websockets-20091222/.

2. *websocket.org.* [Online] https://www.websocket.org/aboutwebsocket.html.

3. Robot. *docs.oracle.com.* [Online] https://docs.oracle.com/javase/7/docs/api/java/awt/Robot.html.

4. Endpoint. *docs.oracle.com.* [Online] http://docs.oracle.com/javaee/7/api/javax/websocket/Endpoint.html.