Developing a 2D game in Java ME - Part 7

This article shows how to use HTTP connection to send high scores data to a central server and how to use the SMS functionality to invite other friends to play the Arkanoid-clone game. This is the seventh (and last) in a series of articles that covers all the basics of developing an application for mobile devices using Java ME, learning the main base libraries, classes, and methods available in Java ME.

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons: hamishwillee
(18 Jun 2013)
Original comment by r2j7 in Sept 2008 still valid - server side implementation required. On 20130618 Mishrapk12 - posted that the pads are not moving in SDK1.0 nor sdk 2.0

Introduction

24 Aug 2008

Now that the Arkanoid single-player experience is complete, the game can be connected to the rest of the world. Besides showing high scores to friends, the results can be shared in the Internet.

One of the main features of mobile devices is the ability to be connected almost anywhere and anytime. With Java ME, you can easily access the following communication features:

- HTTP
- Socket
- SMS
- Bluetooth

This article explains how to use HTTP connection to send high scores data to a central server and how to use the SMS functionality to invite other friends to play the Arkanoid game.

Connector and connections

The main entry point for network functionality is the factory class Connector. This class allows creating Connection objects trough the following static methods:

- open(String name)
- open(String name, int mode)

The name parameter is a URL string that identifies the connection you want to make. Depending on the protocol in the URL, a different kind of Connection object is returned. See the following list of URLs and the corresponding Connection objects:

- "http://www.server.com/": returns a HTTP connection using the class HttpConnection.
- "socket://server.com:8080": returns a TCP/IP connection.
- "btsp://2343434d3434": returns a Bluetooth connection.
- "sms://+351910000000": returns an SMS connection to the phone number +351910000000.

Of the above-mentioned protocols, only HTTP is mandatory on all devices supporting Java ME. All other protocols are device dependent; however, most new phones support all these protocols.

Note that when you try to open any kind of connection, the user of the application is notified by the Application Management System and given the option to accept or decline the connection request. If the connection request is declined, the application receives a SecurityException and the connection is not established.

HTTP

HTTP connections are among the most commonly used types of connections. To create an HTTP connection:

```
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;

public class Network {

  public byte[] httpConnection(String url, byte[] dataToSend)
    throws IOException
  {
    HttpConnection hc = null;
    // Prepare Connection
    hc = (HttpConnection) Connector.open(url, Connector.READ_WRITE);
    [...]
  }
}
```

HTTP connections are represented by the HttpConnection class. This class has three states:

- **Setup** where the request parameters can be defined.
- **Connected** where you can send and receive data. You need to send data before you can receive an answer.

(C) Copyright Nokia 2013. All rights reserved.

- **Closed** after you read the data and the connection is ended.

When you receive the HttpConnection object from the Connector, it is in the Setup state. You can then configure the request parameters. One of the main options is the HTTP request method (POST, GET, or HEAD).

```
  public byte[] httpConnection(String url, byte[] dataToSend) throws IOException {
    [..]
    if (dataToSend == null){
      hc.setRequestMethod( HttpConnection.GET );
    } else {
      hc.setRequestMethod( HttpConnection.POST );
    }
    [...]
  }
```

You can also configure the request properties, such as the type of the content you are sending in case of POST.

```
    hc.setRequestProperty("Content-type", "application/octet-stream" );
```

After you have configured the connection, you can start sending data. When you start this process, you cannot change the request parameters. If the connection is established, you start receiving data.

```
  public byte[] httpConnection(String url, byte[] dataToSend) throws IOException {
    [...]
    if (dataToSend != null){
      // Write Data
      OutputStream os = hc.openOutputStream();
      os.write( dataToSend );
      os.close();
    }
    // gets answer from  server
    int rc = hc.getResponseCode();
    // check http response
    if (rc == HttpConnection.HTTP_OK){
      // Read Data
      InputStream in = hc.openInputStream();
      ByteArrayOutputStream tmp = new ByteArrayOutputStream();
      int ch;
      while ((ch = in.read()) != -1) {
          tmp.write(ch);
      }
      data = tmp.toByteArray();
      in.close();
      hc.close();
    }

    return data;
  }
```

The `httpConnection()` method is now complete, and next, it is implemented in the game to send the high scores to a server. Start by adding a new option to the high scores screen:

```
public Displayable initScoreForm() {
  [...]
  cmdSendHighScore = new Command("Send to Server", Command.ITEM, 1);
  highScoreForm.addCommand(cmdSendHighScore);
  [...]
}
```

Next, create a method to use it when this Command is selected:

```
  public String sendScore(String user, int score) {
    String result = "No Answer";
    // server to send data
    String url = "http://www.sergioestevao.com/midpAdventures/post.php";
    // prepare http request
    String urlTotal = url + "?user=" + user + "&score=" + score;
    byte[] data = null;
    try {
      data = network.httpConnection(urlTotal, null);
    } catch (IOException e) {
      result = "Communication Problems";
      e.printStackTrace();
    } catch (SecurityException s) {
      // user denied access to communication
      result = "You need to allow communications in order to send the highscore to server.";
      s.printStackTrace();
    }
    // check data return.
    if (data != null) {
      result = new String(data);
    }
    return result;
  }
```

And call it when the Command is used:

```
   if (cmd == comInviteSend) {
        result = sendScore(scores[0].name, scores[0].value);
        display(new Alert("Result", result, null, AlertType.INFO));
   }
```

If you test this code, the following warning is displayed:

Warning: To avoid potential deadlock, operations that may block, such as

(C) Copyright Nokia 2013. All rights reserved.

networking, should be performed in a different thread than the commandAction() handler.

Remember that you are executing this network code as an answer to a user event that is being executed on the UI thread. If this code blocks or takes a long time to execute, the application will get stuck.

To avoid this issue, use a thread to do the network-related tasks. Start by declaring a thread and some state variables to control the actions of the thread.

```
private static final int ACTION_SEND_HIGHSCORE = 0;
public Thread workThread;
public boolean active = false;
```

Whenever there is a time-consuming event, a method that prepares the thread and starts the action is called.

```
public void doAction(int action) {
  // stores action to do
  this.action = action;
  // check if thread is already created
  if (workThread == null) {
    workThread = new Thread(this);
    workThread.start();
    active = true;
  }
  // wakes up thread to work
  synchronized (this) {
    notify();
  }
}
```

The new thread executes the run method and does what needs to be done:

```
public void run() {
    while (active) {
      // check what action to do
      switch (action) {
        case (ACTION_SEND_HIGHSCORE):
          // send the first score to the server
          result = sendScore(scores[0].name, scores[0].value);
          commandAction(cmdReceiveHighScore, highScoreForm);
        break;
      }
      // waits for action to do.
      synchronized (this) {
        try {
          wait();
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }
  }
```

Declare the Runnable interface in the MIDlet:

```
public class MyMidlet extends MIDlet implements CommandListener, Runnable {
```

Now call the `doAction()` method when the user selects the 'Send high score' option. Note that the `run()` method invokes a command called `cmdReceiveHighScore`. This is used to show the results of the communication to the user.

```
      if (cmd == cmdSendHighScore) {
        doAction(ACTION_SEND_HIGHSCORE);
      }
      if (cmd == cmdReceiveHighScore) {
        display(new Alert("Result", result, null, AlertType.INFO));
      }
```

SMS

Another use case for communication in Java ME is to invite friends to play the game. You can also have a competition in the high-scores tables.

First, implement the method that sends the SMS in the network class.

```
public boolean sendSms(String number, String message){
  boolean result = true;
  try {
    //sets address to send message
    String addr = "sms://"+number;
    // opens connection
    MessageConnection conn = (MessageConnection) Connector.open(addr);
    // prepares text message
    TextMessage msg =
    (TextMessage)conn.newMessage(MessageConnection.TEXT_MESSAGE);
    //set text
    msg.setPayloadText(message);
    // send message
    conn.send(msg);
    } catch (Exception e) {
      result = false;
    }
    return result;
}
```


(C) Copyright Nokia 2013. All rights reserved.

As you can see, you just need to use the `open()` method of the Connector class with a URL of the format "sms://number". Then create a TextMessage and send it trough Connection. The size of the message is limited to around 160 characters.

Next, create a form to receive the name and phone number from the user.

```
  public Displayable initInviteForm() {
    if (inviteForm == null) {
      inviteForm = new Form("Invite");
      inviteForm.setCommandListener(this);
      inviteForm.addCommand(initBackCommand());

      inviteName = new TextField("Name:", "", 20, TextField.ANY);
      inviteNumber = new TextField("Number:", "", 20, TextField.PHONENUMBER);
      inviteForm.append(inviteName);
      inviteForm.append(inviteNumber);
      comInviteSend = new Command("Invite", Command.ITEM, 1);
      inviteForm.addCommand(comInviteSend);
    }

    return inviteForm;
  }
```

Implement the command's action to show the form and send the message using the worker thread.

```
public void commandAction(Command cmd, Displayable display) {
   [...]
   else if (display == inviteForm) {
      if (cmd == comBack) {
        display(initMainForm());
      }
      if (cmd == comInviteSend) {
        doAction(ACTION_SEND_INVITE);
      }
    }
  [...]
}
```

Next, implement the new action in the run method:

```
public void run() {
  [...]
  case (ACTION_SEND_INVITE):
    // invite another player to play
    String inviteMessage = " invites you to play Transnoid!";
    network.sendSms(inviteNumber.getString(), inviteName.getString()
      + inviteMessage);
    break;
  }
  [..]
}
```

Now run the game and start sending invitations to friends.

Downloads

- Full Source Code
- Jad File
- Jar File

Developing a 2D game in Java ME - Part 1

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.