Storing application settings in Java ME

Overview

This code snippet demonstrates how to store application settings by using RMS (Record Management System).

To store application settings in an RMS database, the following steps should be performed:

1. Open or create an RMS database for the application using the `RecordStore.openRecordStore` method. This method returns the RecordStore object, which represents the application database.
2. Serialize the app-defined data structure for setting to array of byte.
3. Execute `RecordStore.addRecord` method for the database object and pass the array of bytes as parameter.
4. Close RMS database.

This midlet consists of three source files:

1. StoringAppSettings.java - Contains the midlet class.
2. AppSetting.java - Contains the application setting class. This class stores the application setting values and has methods for serializing and deserializing.
3. AddSettingForm.java - Contains a form that lets the user enter a name and value for the new application setting.

Source file: StoringAppSettings.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.List;

import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordListener;

import java.util.Vector;
import javax.microedition.lcdui.Alert;


public class StoringAppSettings extends MIDlet
        implements CommandListener, RecordListener {

    private Display display;
    // List for displaying settings of application.
    private List recordList;
    // Command for adding new setting.
    private Command cmdAdd;
    // Command for deleteing all records from record store.
    private Command cmdDeleteAll;
    // Command for exiting from midlet.
    private Command cmdExit;

    // Name of record store of application.
    private final String RECORD_STORE_NAME = "StoringAppSettingsStore";
    // Record store of application.
    private RecordStore recordStore;
    // Array of settings deserialized from records in record store.
    private Vector appSettings;

    // Form for adding new setting.
    private AddSettingForm addSettingForm;

    // Alert to be displayed if error occurs.
    private Alert alert;

    /**
     * Constructor
     */
    public StoringAppSettings() {
        initializeComponents();
    }

    /**
     * Initializes components of midlet
     */
    private void initializeComponents() {
        // Get display
        display = Display.getDisplay(this);

        // Create recordList
        recordList = new List("App settings", List.IMPLICIT);
        cmdAdd = new Command("Add setting", Command.OK, 0);
        recordList.addCommand(cmdAdd);
        cmdDeleteAll = new Command("Delete all settings", Command.OK, 0);
        recordList.addCommand(cmdDeleteAll);
        cmdExit = new Command("Exit", Command.EXIT, 0);
        recordList.addCommand(cmdExit);
        recordList.setCommandListener(this);

        // Create form for adding new settings.
        addSettingForm = new AddSettingForm();
        addSettingForm.setCommandListener(this);

        // Create array of app settings.
        appSettings = new Vector();
```


(C) Copyright Nokia 2013. All rights reserved.

```
        // Open record store, read settings from it and display settings.
        try {
            openRecordStore();
            getAppSettings();
            addSettingsToForm();
        } catch(RecordStoreException rsExc) {
            showAlert("Record store error", rsExc.getMessage(), null);
        } catch(SecurityException secExc) {
            showAlert("Security error", secExc.getMessage(), null);
        }
    }

    /**
     * Opens named record store of application. Record store will be created
     * if it is not exists.
     */
    private void openRecordStore() throws RecordStoreException, SecurityException {
        recordStore = RecordStore.openRecordStore(RECORD_STORE_NAME, true);
        recordStore.addRecordListener(this);
    }

    /**
     * Closes record store.
     */
    private void closeRecordStore(){
        try {
            recordStore.closeRecordStore();
        } catch (RecordStoreException ex) {
            // Do nothing.
        }
    }

    /**
     * Reads settings from record store and adds it to array of settings.
     * @throws RecordStoreException if error occurs on working with record store.
     */
    private void getAppSettings() throws RecordStoreException {
        appSettings.removeAllElements();

        RecordEnumeration recEnum = recordStore.enumerateRecords(null,
                null, false);
        while(recEnum.hasNextElement() == true) {
            int settingId = recEnum.nextRecordId();
            byte[] data = recordStore.getRecord(settingId);
            AppSetting setting = AppSetting.deserialize(data);
            if(setting != null) {
                appSettings.addElement(setting);
            }
        }
        recEnum.destroy();
    }

    /**
     * Adds settings to settings list control from array of settings.
     */
    private void addSettingsToForm() {
        recordList.deleteAll();

        for(int index = 0; index < appSettings.size(); ++index) {
            AppSetting setting = (AppSetting)appSettings.elementAt(index);
            recordList.append(setting.toString(), null);
        }
    }

    /**
     * Adds setting to record store.
     * @param name - name of new record.
     * @param value - value of new record.
     */
    private void addSetting(String name, String value) {
        try {
            AppSetting setting = new AppSetting(name, value);
            byte[] data = setting.serialize();
            recordStore.addRecord(data, 0, data.length);
        } catch(RecordStoreException rsExc) {
            showAlert("Record store error", rsExc.getMessage(), recordList);
        }
    }

    /**
     * Deletes record store and creates it again.
     */
    private void deleteAllSettings() {
        try {
            recordStore.closeRecordStore();
            RecordStore.deleteRecordStore(RECORD_STORE_NAME);

            openRecordStore();

        } catch(RecordStoreException rsExc) {
            showAlert("Record store error", rsExc.getMessage(), recordList);
        }
    }

    /**
     * From MIDlet.
     * Signals the MIDlet that it has entered the Active state.
     */
    public void startApp() {
        display.setCurrent(recordList);
    }

    /**
     * From MIDlet.
     * Signals the MIDlet to enter the Paused state.
     */
    public void pauseApp() {
        // No implementation required
    }

    /**
     * From MIDlet.
     * Signals the MIDlet to terminate and enter the Destroyed state.
     */
    public void destroyApp(boolean unconditional) {
        closeRecordStore();
    }

    /**
```


(C) Copyright Nokia 2013. All rights reserved.

```
     * Performs exit from midlet.
     */
    private void exitMIDlet() {
        notifyDestroyed();
    }

    /**
     * Shows alert with specified title and text. If next displayable is not
     * specified then application will be closed after alert closing.
     * @param title - Title of alert.
     * @param message - text of alert.
     * @param nextDisp - next displayable. Can be null.
     */
    private void showAlert(String title, String message, Displayable nextDisp) {
        alert = new Alert(title);
        alert.setString(message);
        alert.setTimeout(Alert.FOREVER);

        if(nextDisp != null) {
            display.setCurrent(alert, nextDisp);
        } else {
            display.setCurrent(alert);
            alert.setCommandListener(this);
        }
    }

    /**
     * From CommandListener.
     * Indicates that a command event has occurred on Displayable d.
     * @param cmd - a Command object identifying the command.
     * @param d - the Displayable on which this event has occurred.
     */
    public void commandAction(Command cmd, Displayable d) {
        if(d == recordList) {
            if(cmd == cmdAdd) {
                addSettingForm.clear();
                display.setCurrent(addSettingForm);
            }
            if(cmd == cmdDeleteAll) {
                deleteAllSettings();
                appSettings.removeAllElements();
                addSettingsToForm();
            }
            if(cmd == cmdExit) {
                exitMIDlet();
            }
        }
        if(d == addSettingForm) {
            if(cmd == addSettingForm.getSaveCommand()) {
                addSetting(addSettingForm.getName(), addSettingForm.getValue());
                display.setCurrent(recordList);
            }
            if(cmd == addSettingForm.getBackCommand()) {
                display.setCurrent(recordList);
            }
        }
        if(d == alert) {
            // Handle "ok" command from alert
            exitMIDlet();
        }
    }

    /**
     * From RecordListener.
     * Called when a record has been added to a record store.
     * @param store - the RecordStore in which the record is stored
     * @param recordId - the recordId of the record that has been added
     */
    public void recordAdded(RecordStore store, int recordId) {
        try {
            getAppSettings();
            addSettingsToForm();
        } catch(RecordStoreException rsExc ) {
            showAlert("Record store error", rsExc.getMessage(), recordList);
        }
    }

    /**
     * From RecordListener.
     * Called after a record in a record store has been changed.
     * @param store - the RecordStore in which the record is stored
     * @param recordId - the recordId of the record that has been added
     */
    public void recordChanged(RecordStore store, int recordId) {
        // No implementation required.
    }

    /**
     * From RecordListener.
     * Called after a record has been deleted from a record store.
     * @param store - the RecordStore in which the record is stored
     * @param recordId - the recordId of the record that has been added
     */
    public void recordDeleted(RecordStore store, int recordId) {
        // No implementation required.
    }
}
```

Source file: AppSetting.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

/**
 * Setting of application. Consist of pair "name - value"
 */
public class AppSetting {
    // Name of setting.
    private String name;
    // Value of setting.
    private String value;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    /**
     * Constructor.
     * @param name - name of new setting.
     * @param value - value of new setting.
     */
    public AppSetting(String name, String value) {
        this.name = name;
        this.value = value;
    }

    /**
     * Deserializes app setting from data buffer and returns it.
     * @param data - data for creating new setting.
     * @return new setting.
     */
    public static AppSetting deserialize(byte[] data) {
        String name = null;
        String value = null;

        try {
            ByteArrayInputStream byteInStream = new ByteArrayInputStream(data);
            DataInputStream dataInStream = new DataInputStream(byteInStream);

            name = dataInStream.readUTF();
            value = dataInStream.readUTF();

            dataInStream.close();
            byteInStream.close();

        } catch(IOException exc) {
            return null;
        }

        return new AppSetting(name, value);
    }

    /**
     * Serilizes setting to array of bytes.
     * @return array of bytes representing serialized setting.
     */
    public byte[] serialize() {
        byte[] data = null;

        try {
            ByteArrayOutputStream byteOutStream = new ByteArrayOutputStream();
            DataOutputStream dataOutStream = new DataOutputStream(byteOutStream);

            dataOutStream.writeUTF(name);
            dataOutStream.writeUTF(value);

            dataOutStream.flush();
            data = byteOutStream.toByteArray();

            dataOutStream.close();
            byteOutStream.close();

        } catch(IOException exc) {
            return null;
        }

        return data;
    }

    /**
     * Gets name of setting.
     * @return name of setting.
     */
    public String getName() {
        return name;
    }

    /**
     * Gets value of setting.
     * @return value of setting.
     */
    public String getValue() {
        return value;
    }

    /**
     * Returns textual representation of setting.
     * @return textual representation of setting.
     */
    public String toString() {
        return name + ": " + value;
    }
}
```

Source file: AddSettingForm.java

```
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;
import javax.microedition.lcdui.Command;

/**
 * Form for receiving from user name, and value of the new setting.
 */
public class AddSettingForm extends Form {
    // Text field for entering name of new setting.
    private TextField nameField;
    // Text field for entering value of new setting.
    private TextField valueField;

    // Command for saving new setting in record store.
    private Command cmdSave;
    // Command for altering saving new setting.
    private Command cmdBack;

    /**
     * Constructor.
     */
    public AddSettingForm() {
        super("Add setting");

        nameField = new TextField("Name", null, 30, 0);
```


(C) Copyright Nokia 2013. All rights reserved.

```
        append(nameField);
        valueField = new TextField("Value", null, 30, 0);
        append(valueField);

        cmdSave = new Command("Save", Command.SCREEN, 0);
        addCommand(cmdSave);
        cmdBack = new Command("Back", Command.BACK, 0);
        addCommand(cmdBack);
    }

    /**
     * Clears antered values in text fields of form.
     */
    public void clear() {
       nameField.setString(null);
       valueField.setString(null);
    }

    /**
     * Gets from name text field name of new setting and returns it.
     * @return name of new setting.
     */
    public String getName() {
        return nameField.getString();
    }

    /**
     * Gets from value text field value of new setting and returns it.
     * @return value of new setting.
     */
    public String getValue() {
        return valueField.getString();
    }

    /**
     * @return command for saving new setting in record store.
     */
    public Command getSaveCommand() {
        return cmdSave;
    }

    /**
     * @return command for altering saving new setting.
     */
    public Command getBackCommand() {
        return cmdBack;
    }
}
```

Postconditions

List of application settings is shown on display.

By selecting the "Add setting" menu command, the user can create a new application setting. By selecting the "Delete all settings" menu command, the user can delete all application settings.

Supplementary material

Executables and source files can be found in Media:Storing application settings.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.