Storing user preferences in Java ME

This article shows how to use the Java ME RMS (Record Management Store) framework to quickly save user preferences as key-value.

20 Oct 2013

Introduction

This article demonstrates a way simple framework to manage key-value pairs using the Java ME RMS framework. The framework consists of a `Preference` class to represent the key-value pair and a `PreferenceManager` for saving and restoring preferences from RMS.

While the main use case for the framework is saving application preferences, the framework is suitable for any use-case requiring permanent storage of key-value information.

Preference

First we need a class to represent the key-value pair. Let's call it `Preference` and its code is below.

```
public class Preference
{
    public int id;
    public String name;
    public String value;

    public static String sep="~$v#*^~";


    public Preference(String name, String value)
    {
        this.name=name;
        this.value=value;
    }

    public Preference(String stringData)
    {
        String[] tokens=StringUtils.split(stringData,sep);
        id=Integer.parseInt(tokens[0]);
        name=tokens[1];
        value=tokens[2];
    }

    public String toString()
    {
        String result=String.valueOf(id);
        result+=sep+name;
        result+=sep+value;
        return result;
    }
}
```

We need three variables for the `Preference` object.

1. `id`: An integer to uniquely identify the preference in the record store.
2. `name`: A string for the unique name of the preference.
3. `value`: The actual value of the preference.

The way a `Preference` object will be stored in RMS is by its string representation. We use the `toString` function for that purpose. What it does is to concatenate the three variables injecting a separator between them (the `sep` variable). We need a separator that is very unlikely to be found in the name or the value of a preference given the context of our application.

The first constructor takes two string variables and sets their values to the according object variables. The second takes a string representation of a `Preference`, splits it by the `sep` variable and sets the object values by the output of the splitting. You can find the code that spits the string in the included source code.

The `id` variable of the `Preference` object is handled by the helper class found below and the developer should not change its value.

PreferenceManager

```
import java.util.Vector;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;


public class PreferenceManager
{
    public static final String preferenceStoreName="PreferenceStore";


    public static RecordStore openRecordStore()
```


(C) Copyright Nokia 2013. All rights reserved.

```
    {
        try
        {
            RecordStore store=null;
            try
            {
                store=RecordStore.openRecordStore(preferenceStoreName, true);
            }
            catch(RecordStoreException rse)
            {
                RecordStore.deleteRecordStore(preferenceStoreName);
                store=RecordStore.openRecordStore(preferenceStoreName,true);
                store.closeRecordStore();
                store=RecordStore.openRecordStore(preferenceStoreName,true);
            }
            return store;
        }
        catch (Exception ex)
        {
         return null;
        }
    }


    public static Vector getPreferences()
    {
        try
        {
            RecordStore store=openRecordStore();

            RecordEnumeration re = store.enumerateRecords(null, null, false);
            Vector prefs=new Vector();
            for (int i=0;i<store.getNumRecords();i++)
            {
                prefs.addElement(new Preference(new String(re.nextRecord())));
            }
            store.closeRecordStore();
            return prefs;
        }
        catch (Exception ex)
        {
            return null;
        }
    }


    public static Preference getPreference(String name)
    {
        try
        {
            RecordStore store=openRecordStore();

            RecordEnumeration re = store.enumerateRecords(null, null, false);
            for (int i=0;i<store.getNumRecords();i++)
            {
                try
                {
                 Preference pref=new Preference(new String(re.nextRecord()));
                 if (pref.name.toLowerCase().equals(name.toLowerCase()))
                 {
                  store.closeRecordStore();
                  return pref;
                 }
                }
                catch (Exception ex1)
                {

                }
            }
            return null;
        }
        catch (Exception ex)
        {
            return null;
        }
    }


    public static void saveNewPreference(Preference newPreference)
    {
        try
        {
            RecordStore store=openRecordStore();

            byte[] bytes = (newPreference.toString()).getBytes();
            int newID=store.addRecord(bytes,0,bytes.length);
            newPreference.id=newID;
            bytes = (newPreference.toString()).getBytes();
            store.setRecord(newID,bytes,0,bytes.length);
            store.closeRecordStore();
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }


    public static void updatePreference(Preference pref)
    {
        try
        {
            RecordStore store=openRecordStore();

            byte[] bytes = (pref.toString()).getBytes();
            store.setRecord(pref.id,bytes,0,bytes.length);
            store.closeRecordStore();
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }


    public static void deletePreference(Preference pref)
    {
        try
        {
```


(C) Copyright Nokia 2013. All rights reserved.

```
            RecordStore store=openRecordStore();

            store.deleteRecord(pref.id);
            store.closeRecordStore();
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }


}
```

Let's see what each function does.

- `preferenceStoreName` : a unique name for the preferences store in RMS. Must be at most 32 chars long.
- `openRecordStore`: Opens and returns the record store which is now ready for interaction.
- `getPreferences`: Return all `Preferences` found in the record store.
- `getPreference`: Return the `Preference` with the unique name `name`. If none is found in the store then `null` is returned.
- `saveNewPreference`: Takes a new `Preference` and saves it into the record store. `newPreference` doesn't require the `id` variable set.
- `updatePreference`: Updates the specific `Preference` into the store. The `id` is preserved.
- `deletePreference`: Deletes the `Preference`.

Extensions

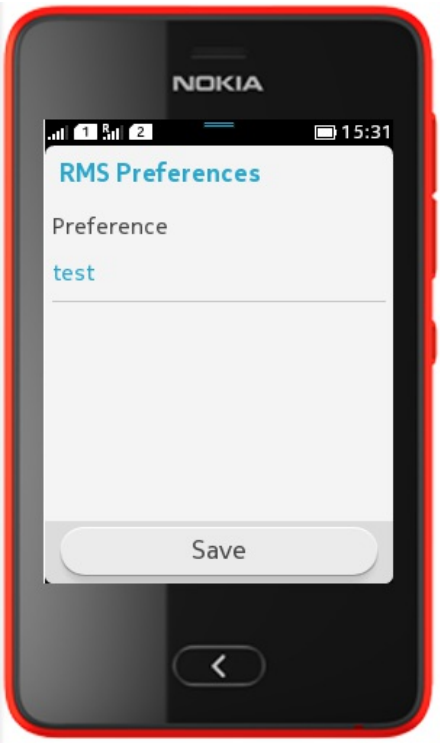
The reader can alter or extend the above code to his liking. For example a handy addition would be an overload of the `updatePreference` function that will update the `Preference` if it is found in the record store and add it if it is not.

```
    public static void updatePreference(String name, String newValue)
    {
        Preference pref=getPreference(name);
        if (pref==null)
            saveNewPreference(new Preference(name,newValue));
        else
        {
            pref.value=newValue;
            updatePreference(pref);
        }
    }
```

Other extensions would be to create wrapper methods like `saveInt`,`getInt`,`saveFloat`,`getFloat`,etc that would convert the string found in the value of the `Preference` to the appropriate type while saving or retrieving it.

Example

In the source included in this article you will find a Form that shows how easy the above code is to use.




```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
```


(C) Copyright Nokia 2013. All rights reserved.

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


public class MainForm extends Form implements CommandListener
{
    TextField prefText;
    Command saveCommand;


    public MainForm(String title)
    {
        super(title);

        Preference pref=PreferenceManager.getPreference("pref1");

        if (pref==null)
        {
            PreferenceManager.updatePreference("pref1", "test");
        }

        prefText=new TextField("Preference", pref.value, 20, TextField.ANY);
        append(prefText);

        saveCommand=new Command("Save", Command.OK, 0);
        addCommand(saveCommand);

        setCommandListener(this);
    }


    public void commandAction(Command c, Displayable d)
    {
        if (c==saveCommand)
        {
            PreferenceManager.updatePreference("pref1", prefText.getString());
        }
    }

}
```

Related sources

A good old article from IBM library about J2ME RMS: J2ME record management store

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.