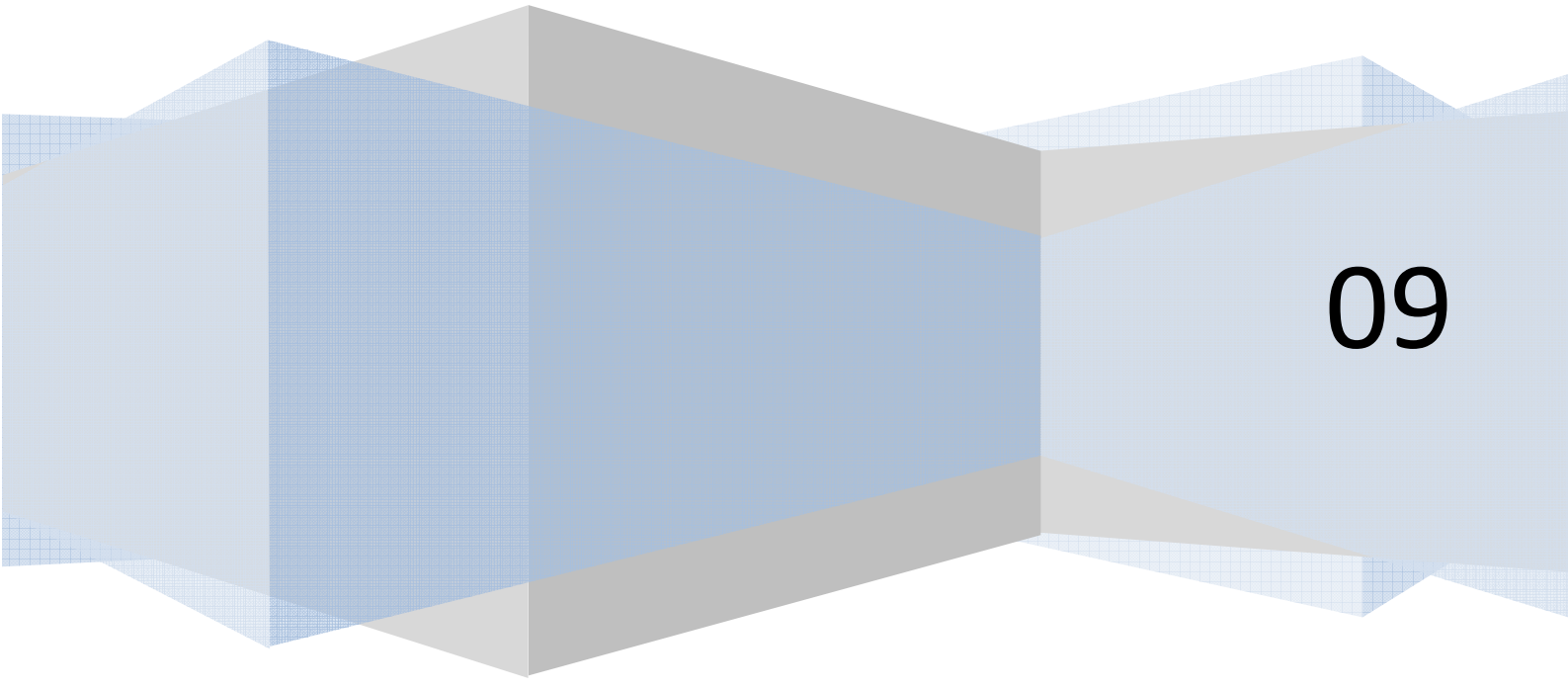
Research In Motion

A13 – Storing Persistent Data

For BlackBerry SmartPhones

Andre Fabris



09

Contents

A13 Storing Persistent Data

This tutorial will show you how to create, save and retrieve data on the device.

We are going to implement the Persistable interface to create an application which will store some data. (Figure 1).

Figure 1

Introduction

If we want to save an object and be able to retrieve it even after the handheld is restarted, or the battery is removed, that object has to implement the Persistable interface.

Sub-classes of that object do not inherit this implementation automatically.

Just a note if you plan to build applications which might have upgrades in the future: configuration of instance data in the class and its superclasses must not change – otherwise you will not be able to access old saved data. Adding or changing the number of classes might also make saved data unusable.

Please note that you can always find more information about the APIs we are using in BlackBerry API reference document which is the part of any JDE Component Pack (Figure 2). You can find it on your computer under Start / Programs / Research in Motion / BlackBerry JDE 4.x.

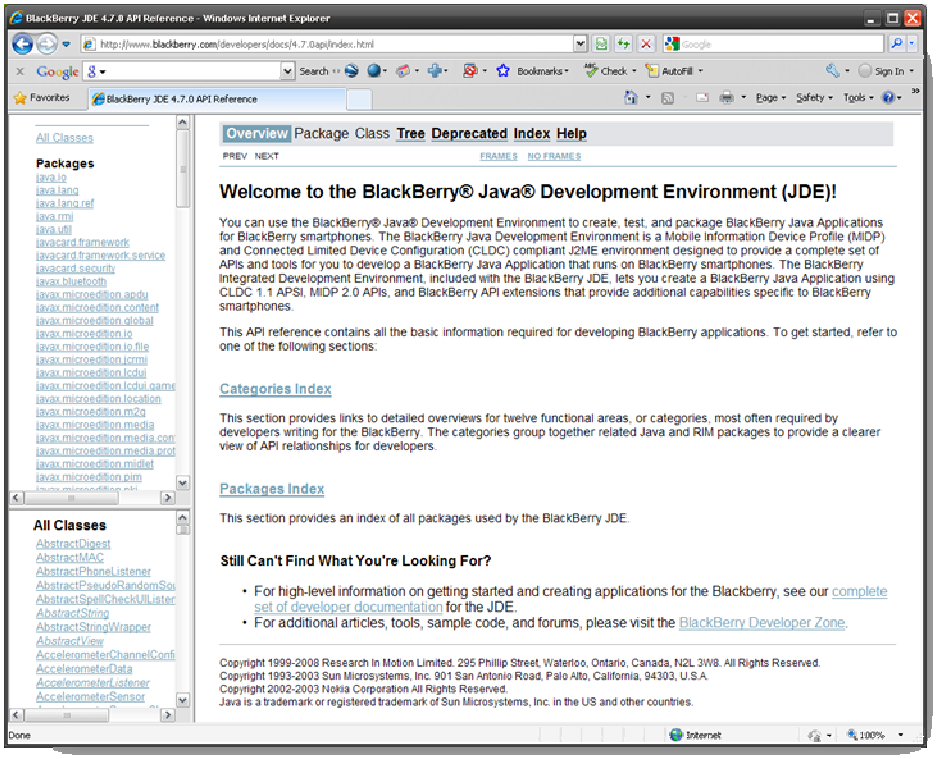


Figure 2

Setting up a New BlackBerry Project

You should already know how to setup and configure a new BlackBerry project. If you need a reminder please refer to the A10 tutorial. I will also assume you know how to create menu and display items. If not please take a look at the A11 tutorial.

Here is the beginning of our application:

```
package com.rim.samples.loadsave;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.util.*;

public class MusicStores extends UiApplication {

      private AutoTextEditField namefield;
      private AutoTextEditField addressfield;
      private EditField phonefield;
      private EditField specialtyfield;
      private static Vector _data;
      private static PersistentObject store;

      public static void main(String[] args) {
            MusicStores app = new MusicStores();
            app.enterEventDispatcher();
      }

      public MusicStores() {
            MainScreen mainScreen = new MainScreen();
            mainScreen.setTitle(new LabelField("Favourite Music Store"));
            namefield = new AutoTextEditField("Store Name: ", "");
            addressfield = new AutoTextEditField("Address: ", "");
            phonefield = new EditField("Phone Number: ", "",
Integer.MAX_VALUE, BasicEditField.FILTER_PHONE);
            specialtyfield = new EditField("Music Type: ", "",
Integer.MAX_VALUE, BasicEditField.FILTER_DEFAULT);
            mainScreen.add(namefield);
            mainScreen.add(addressfield);
            mainScreen.add(phonefield);
            mainScreen.add(specialtyfield);
            mainScreen.addMenuItem(saveItem);
            mainScreen.addMenuItem(getItem);
            pushScreen(mainScreen);
      }
```

As you probably noticed, we used 4 fields for 4 peices of information we want to input, load, save and display.

You will notice that we have a Vector, and a PeristentObject declared. I will explain their purpose shortly.

The rest of the code sets the title, adds 3 fields, and displays the screen.

Menu Items

Save Menu Item

The command "Save" does the following things: creates a new StoreInfo object called info, and retrieves the data from input fields and assigns them to the data in the info object. The StoreInfo class implements the Persistable interface and therefore can be stored. To actually save the data, we add it to the vector `_data` and then call the `store.setContents` and `store.commit` methods. We'll talk about this further in a moment. At the end we'd like to display a dialog to inform the user that the data was saved successfully, and then clear the data on the screen.

We use synchronization to make sure we have sole access to the data:

```
private MenuItem saveItem = new MenuItem("Save", 110, 10) {
public void run() {
      StoreInfo info = new StoreInfo();
      info.setElement(StoreInfo.NAME, namefield.getText());
      info.setElement(StoreInfo.ADDRESS, addressfield.getText());
      info.setElement(StoreInfo.PHONE, phonefield.getText());
      info.setElement(StoreInfo.SPECIALTY, specialtyfield.getText());
      _data.addElement(info);
      synchronized (store) {
            store.setContents(_data);
            store.commit();
      }
      Dialog.inform("Success!");
      namefield.setText(null);
      addressfield.setText(null);
      phonefield.setText("");
      specialtyfield.setText("");
      }
};
```

Get Menu Item

To retrieve the data we use the `store.getContents` method and cast the object into our "_data" vector. The last object in this vector is the one we are retrieving and to display the data we assign it to the corresponding fields.

```
private MenuItem getItem = new MenuItem("Get", 110, 11) {
public void run() {
synchronized (store) {
      _data = (Vector) store.getContents();
      if (!_data.isEmpty()) {
            StoreInfo info = (StoreInfo) _data.lastElement();
      namefield.setText(info.getElement(StoreInfo.NAME));
      addressfield.setText(info.getElement(StoreInfo.ADDRESS));
      phonefield.setText(info.getElement(StoreInfo.PHONE));
      specialtyfield.setText(info.getElement(StoreInfo.SPECIALTY));
      }
}
}
};
```

Again, we use the synchronized() call to access the data.

StoreInfo class

There are no secrets in our StoreInfo class.

It has one vector, which is initialized in the constructor and has 4 empty Strings. Get and set methods, just get and set the elements in their places. We do not want elements to change places, as it will result in corrupted data. That is why we use constant values NAME, ADDRESS, PHONE and SPECIALITY to make sure the data is always saved and read at and from the same location.

```
private static final class StoreInfo implements Persistable {

	private Vector _elements;
	public static final int NAME = 0;
	public static final int ADDRESS = 1;
	public static final int PHONE = 2;
	public static final int SPECIALTY = 3;

	public StoreInfo() {

	_elements = new Vector(4);

		for (int i = 0; i < _elements.capacity(); ++i) {
			_elements.addElement(new String(""));
		}
	}

	public String getElement(int id) {
		return (String) _elements.elementAt(id);
	}
	public void setElement(int id, String value) {
		_elements.setElementAt(value, id);
	}
}
```

The persistent magic

And here is the Persistent Store itself:

```
static {
      store = PersistentStore.getPersistentObject (0xdec6a67096f833cL);
            synchronized (store) {
                  if (store.getContents() == null) {
                        store.setContents(new Vector());
                        store.commit();
                  }
            }
            _data = new Vector();
            _data = (Vector) store.getContents();
}
```

As you can see all the magic happens in one line. The rest of the code just checks to see if there is existing data, and if not, creates empty data. Next, the code retrieves this data.

The key is that we use the `PersistentStore` class and its `getPersistentObject(long key)` method. Each persistent object should have a unique key that is used to identify it when we want to perform save or retrieve actions. In our case it is `0xdec6a67096f833cL`.

I also owe you an explanation for the 3 methods we used in the code above: `getContents`, `commit` and `setContents`.

You can also find them in our API reference.

In the API reference, click on “All Classes”, find the PersistentObject class and you will see all the methods you can use on this object (Figure 3).

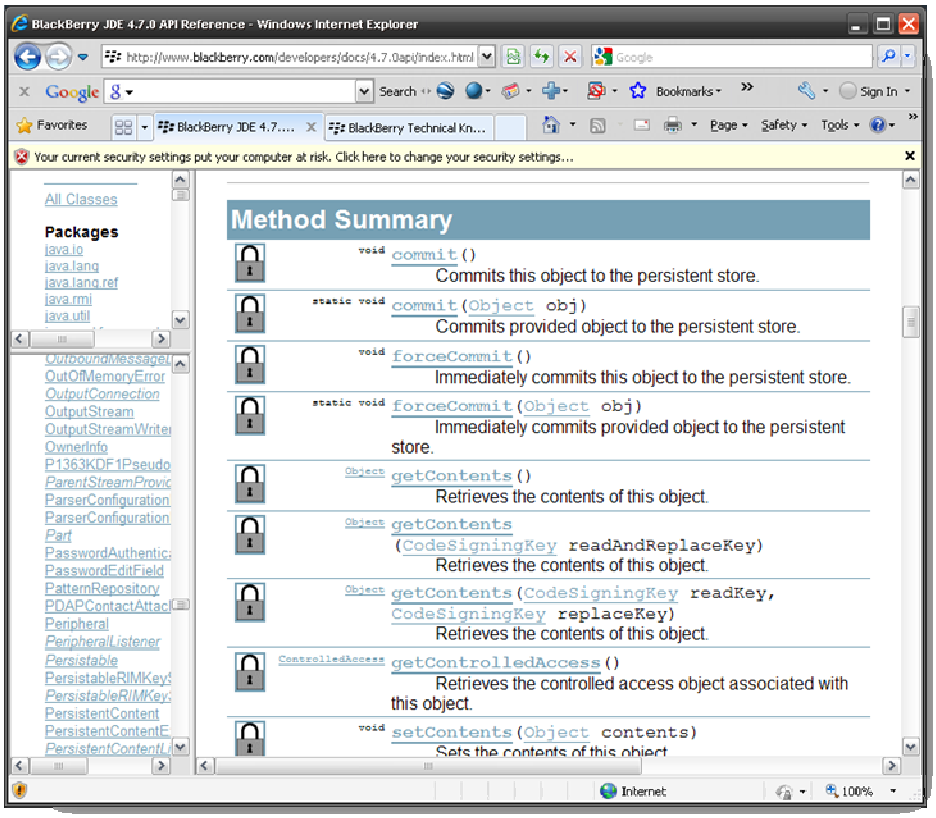


Figure 3

`void commit()` - Commits this object to the persistent store.

`Object getContents()` - Retrieves the contents of this object.

`void setContents(Object contents)` - Sets the contents of this object.

You will also notice padlocks next to all these methods. It means only one thing – to use this method your application has to be signed.

Code signing

Before you try running this code on a device, you should know that accessing persistent storage is a protected operation and therefore your code must be signed.

You can run the code on the simulator without the need to sign the code.

If you do not know how to sign your code please check out the A60 tutorial.

Note that Eclipse displays a light bulb with an exclamation mark next to all the code lines requiring signatures. If you place your mouse above the icon it will tell you that usage is discouraged (Figure 4).

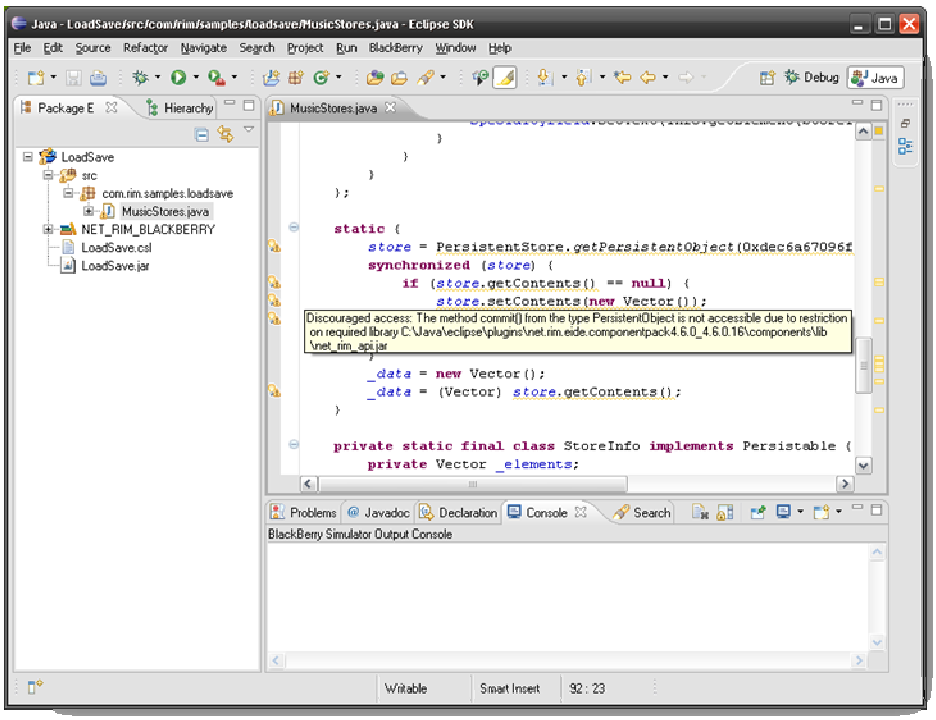


Figure 4

The whole application

```
package com.rim.samples.loadsave;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import java.util.*;

public class MusicStores extends UiApplication {

      private AutoTextEditField namefield;
      private AutoTextEditField addressfield;
      private EditField phonefield;
      private EditField specialtyfield;
      private static Vector _data;
      private static PersistentObject store;

      public static void main(String[] args) {
            MusicStores app = new MusicStores();
            app.enterEventDispatcher();
      }

      public MusicStores() {
            MainScreen mainScreen = new MainScreen();
            mainScreen.setTitle(new LabelField("Favourite Music Store"));
            namefield = new AutoTextEditField("Store Name: ", "");
            addressfield = new AutoTextEditField("Address: ", "");
            phonefield = new EditField("Phone Number: ", "",
Integer.MAX_VALUE, BasicEditField.FILTER_PHONE);
            specialtyfield = new EditField("Music Type: ", "",
Integer.MAX_VALUE, BasicEditField.FILTER_DEFAULT);
            mainScreen.add(namefield);
            mainScreen.add(addressfield);
            mainScreen.add(phonefield);
            mainScreen.add(specialtyfield);
            mainScreen.addMenuItem(saveItem);
            mainScreen.addMenuItem(getItem);
            pushScreen(mainScreen);
      }

      private MenuItem saveItem = new MenuItem("Save", 110, 10) {
            public void run() {
                  StoreInfo info = new StoreInfo();
                  info.setElement(StoreInfo.NAME, namefield.getText());
                  info.setElement(StoreInfo.ADDRESS,
addressfield.getText());
                  info.setElement(StoreInfo.PHONE, phonefield.getText());
                  info.setElement(StoreInfo.SPECIALTY,
specialtyfield.getText());
                  _data.addElement(info);
                  synchronized (store) {
                        store.setContents(_data);
                        store.commit();
```

```
                }
                Dialog.inform("Success!");
                namefield.setText(null);
                addressfield.setText(null);
                phonefield.setText("");
                specialtyfield.setText("");
            }
        };

        private MenuItem getItem = new MenuItem("Get", 110, 11) {
            public void run() {
                synchronized (store) {
                    _data = (Vector) store.getContents();
                    if (!_data.isEmpty()) {
                        StoreInfo info = (StoreInfo)
_data.lastElement();

        namefield.setText(info.getElement(StoreInfo.NAME));
        addressfield.setText(info.getElement(StoreInfo.ADDRESS));
        phonefield.setText(info.getElement(StoreInfo.PHONE));
        specialtyfield.setText(info.getElement(StoreInfo.SPECIALTY));
                    }
                }
            }
        };

        static {
            store =
PersistentStore.getPersistentObject(0xdec6a67096f833cL);
            synchronized (store) {
                if (store.getContents() == null) {
                    store.setContents(new Vector());
                    store.commit();
                }
            }
            _data = new Vector();
            _data = (Vector) store.getContents();
        }

        private static final class StoreInfo implements Persistable {
            private Vector _elements;
            public static final int NAME = 0;
            public static final int ADDRESS = 1;
            public static final int PHONE = 2;
            public static final int SPECIALTY = 3;

            public StoreInfo() {
                _elements = new Vector(4);
                for (int i = 0; i < _elements.capacity(); ++i) {
                    _elements.addElement(new String(""));
                }
            }

            public String getElement(int id) {
                return (String) _elements.elementAt(id);
            }

            public void setElement(int id, String value) {
                _elements.setElementAt(value, id);
            }
        }
```

```
}
```

Other Variations

There are a number of things you might want to try with this sample application:

- Try to change it so it can go through all the saved records.
- Try to save and load different types of objects i.e. arrays, images etc.

Links

BlackBerry Developers Web Site:

http://na.blackberry.com/eng/developers/

Developer Video Library:

- Deploying and Signing Applications:
  http://www.blackberry.com/DevMediaLibrary/view.do?name=deploying
- Introduction to BlackBerry Java Development:
  http://www.blackberry.com/DevMediaLibrary/view.do?name=java

Developer Labs:

- Storing Persistent Data
  http://na.blackberry.com/eng/developers/resources/developer_labs.jsp#tab_tab_jde

Documentation:

- Documentation for the developers can be found here:
  http://na.blackberry.com/eng/support/docs/developers/?userType=21

Knowledge Base Articles:

- Signature Tools & Code Signing
- Java APIs & Samples / Persistence
  http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/customview.html?func=ll&objId=348583

Forums:

- The link to BlackBerry Development Forums:
  http://supportforums.blackberry.com/rim/?category.id=BlackBerryDevelopment