# JSR 75 - PIM

**Version 0.9, Draft**


# API GUIDE

# COPYRIGHT



### Trademarks and Service Marks

# About This Document

This document describes the PIM optional package javax.microedition.pim (JSR-75) and provides sample code snippets.

## Scope

This document is for user who has the knowledge of Java ME and wants a brief introduction of JSR 75 - PIM.

## Document History:

| Date | Version | Comment |
|---|---|---|
| 18/06/09 | 0.9 | Draft |

## References:

1. JSR 75 specification: http://jcp.org/en/jsr/detail?id=75

2. PIM Article: http://developers.sun.com/mobility/apis/pim/pim5/

## Abbreviations:

| | |
|---|---|
| Java ME | Java Micro Edition |
| JSR | Java Specification Request |
| API | Application Programming Interface |
| MIDP | Mobile Information Device Profile |
| PIM | Personal Information Management |
| MSA | Mobile Service Architecture |
| SIM | Subscriber Identity Module |

# Table of Contents



# List of Figures

# Introduction

The PIM API is an optional JSR 75 package *javax.microedition.pim.*;* that gives support to access and modify the PIM database that may exist in a MIDP device. The purpose of PIM API is to give a standardized interface to those databases, which could be used across different type of devices in secure fashion.

The PIM API currently supports the three types of database or lists:

- Contact lists
- Event lists
- To-Do lists

These three types of database may not be necessarily available in a particular device, but the specification mentions that the API that has been implemented in a device, at least one database of one type should be available. An implementation may contain more than one list of the same type on device, for example, a mobile device can have a contact list contained in the device memory and another in the device SIM card.

# Overview

The PIM API is one of the mandate APIs defined by the Mobile Service Architecture (MSA) specification. PIM API encapsulated in a single package *javax.microedition.pim.** includes the following interfaces, classes and exception.

Table 1: the javax.microedition.pim Optional Package

| Interface | Description |
|---|---|
| Contact | It represents a single Contact entry in a PIM Contact database. The supported field list for a Contact is also a subset of the fields defined by the vCard specification. It is also responsible for UnsupportedFieldException. |
| ContactList | It represents a Contact list containing Contact items. A Contact List is responsible for determining which of the fields from a Contact are retained when a Contact is persisted into the List. It is also responsible for java.lang.IllegalArgumentException. |
| Event | It represents a single Event entry in a PIM Event database. The fields are a subset of the fields in the vEvent object defined by the vCalendar 1.0 specification. It is also responsible for UnsupportedFieldException. |

| | |
|---|---|
| EventList | It represents an Event list containing Event items. An Event List is responsible for determining which of the fields from an Event are retained when an Event is persisted into the List. It is also responsible for java.lang.IllegalArgumentException. |
| PIMItem | It represents the common interfaces of an item for a PIM list.<br>A PIM item represents a collection of data for a single PIM entry. |
| PIMList | It represents the common functionality of a PIM list. PIMLists contain zero or more PIMItems (represented by the class PIMItem ). |
| ToDo | It represents a single To Do item in a PIM To Do database. The fields are a subset of the fields in VTODO defined by the vCalendar specification. It is also responsible for UnsupportedFieldException. |
| ToDoList | It represents a ToDo list containing ToDo items. A ToDo List is responsible for determining which of the fields from a ToDo are retained when a ToDo is persisted into the List. It is also responsible for java.lang.IllegalArgumentException |

| Classes | Description |
|---|---|
| PIM | This Class for accessing PIM lists on a device and performing opening the lists, converting raw data streams to and from PIM items for importing and exporting into those lists. |
| RepeatRule | It represents a description for a repeating pattern for an Event item. The fields are a subset of the capabilities of the RRULE field in VEVENT defined by the vCalendar 1.0 specification. |

| Exceptions | Description |
|---|---|
| FieldEmptyException | It represents an exception thrown when an attempt is made to access a field that does not have any data values associated with it. |
| FieldFullException | It represents an exception thrown when an attempt is made to add data to a field but the field already has all available slots for data values assigned. |
| PIMException | It represents exceptions thrown by the PIM classes. This class has a reason code optionally associated with it to provide more information about the PIM exception that occurred. |
| UnsupportedFieldException | It represents an exception thrown when a field is referenced that is not supported in the particular PIM list that an element belongs to. |

PIM data is organized into databases or lists. Multiple lists can exist for calendar events, address-book contacts, and to-do items. PIM lists contain Item(s), which are grouping of the related fields. A field consists of a label, data type, values and attributes.

Figure 1 shows the PIM Organization data model.

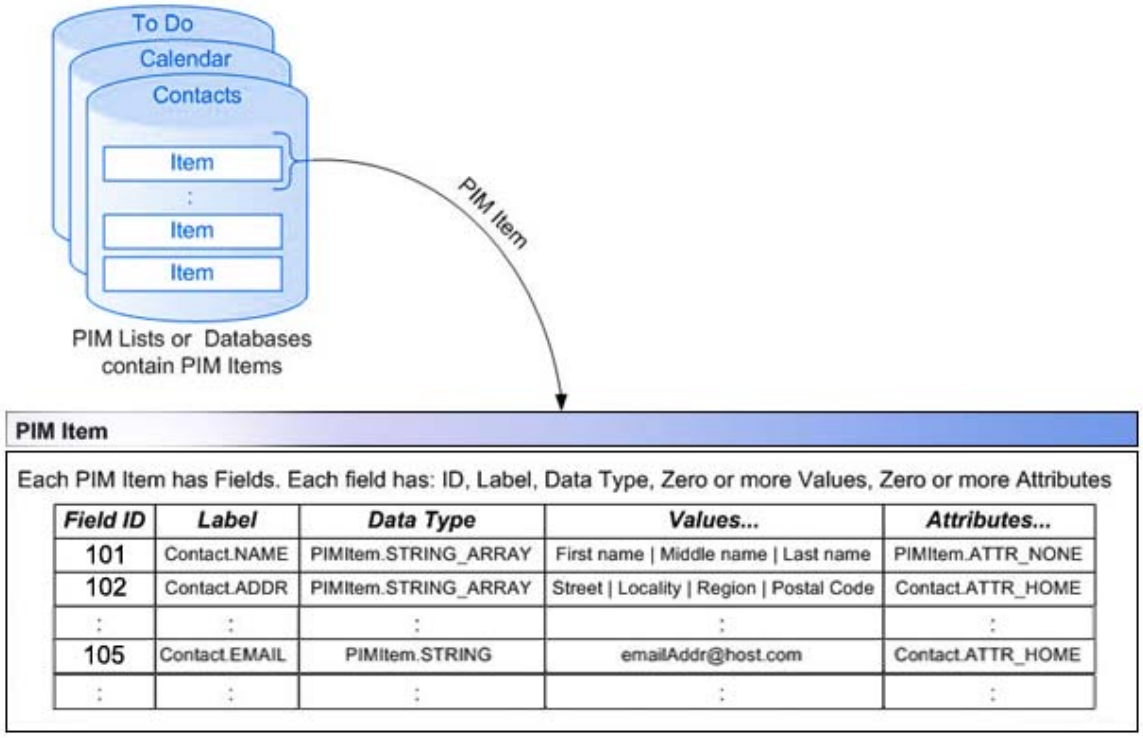


**Figure 1: PIM data organization**

In this data model, PIM lists or databases are represented by interface *javax.microedition.pim.PIMList* and its sub-interfaces. PIM Items are represented by interface *javax.microedition.pim.PIMItem* and its sub-interfaces. Fields are represented by the corresponding Java data type such as String, Date, int, boolean, arrays of bytes and Strings.

## The PIM class

The abstract PIM class provides access to the PIM implementation. The PIM class exposes a static method to retrieve the PIM instance itself and abstract methods to manage the PIM lists/databases and to import and export PIM data:

*getInstance()* - static method to retrieve the PIM instance that allows to manage the local PIM.

*listPimList(int pimListType)* - abstract method to get the names of existing PIM databases. *openPimList(int pimListType, int* mode) and *openPimList(int pimListType, int* mode, String name) - abstract method to open the default database, or open a specific database by name.

*fromSerialFormat(...)* and *toSerialFormat(...)* - abstract methods to import and export PIM data respectively.

*supportedSerialFormats(...)* - abstract method to get the supported PIM data exchange formats such as vCard version 2.1 and vCalendar version 1.0.

## The PIMList Interface and Sub-Interface

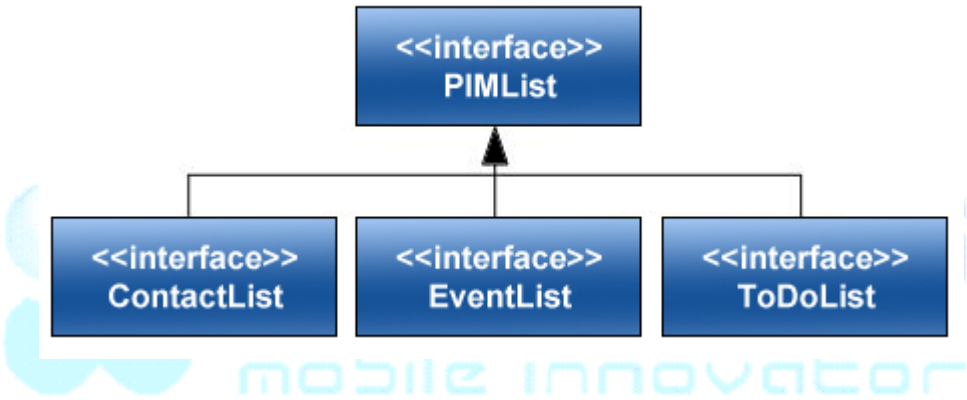


**Figure 2: PIMList Interface and Sub-Interface**

The PIM class provides access to the PIM lists to access PIM databases themselves. PIM lists are represented by the PIMList interface and its sub-interfaces as described next.

- *PIMList* represents a PIM database in general
- *Contact List* represents the contact list database
- *EventList* represents the calendar events database
- *ToDoList* represents the to-do list database

## The PIMItem Interface and Sub-Interfaces

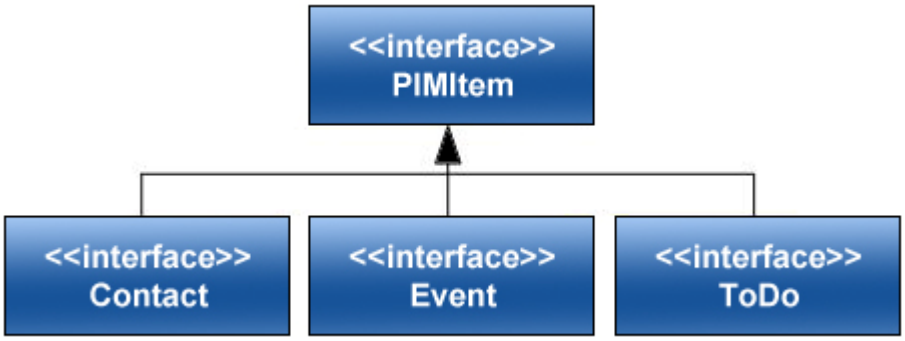


**Figure 3: PIMItem Interface and Sub-Interface**

PIM lists are collections of PIM items. PIM items as the rows or records of the database. PIM items are represented by the PIMItem super-interface and its sub-interfaces as described next:

- *PIMItem* is a generalization of PIM data, such as contact, calendar or a to-do item
- *Contact* represents a contact item in the address book database
- *Event* represents an event in the calendar database
- *ToDo* represents a to-do item in the To-Do database

Following section explains PIM data organization shown in Figure 1:

## Field

PIM items are groupings of related fields. Fields are specific to the type of PIMItem. For example, a calendar event item consists of fields such as start and end-event times, event description, location, and other. The address-book item contains fields such as first and last-names, home and work phone numbers and addresses, email addresses, and so on. Fields have an ID, a descriptive label, a data type, values, and attributes, as illustrated in figure 1.

Fields are uniquely identified within a given PIM list/item by field ID. Field IDs are constants that are defined by the specific PIM item Contact, Event and ToDo. For example, the Contact PIM item defines the fields Contact. NAME, Contact.ADDR etc.
Fields are considered standard or extended fields. Standard fields are the common fields while extended fields are PIM implementation specific fields that must be discovered. All supported fields can be discovered by calling the method getSupportedFields() for the particular PIM list. Extended fields are easy to identify because their field ID values are greater than or equal to PIMItem.EXTENDED_FIELD_MIN_VALUE.

### Label

The field's label is a String that describes the field. To discover a field's label call the PIMList method getFieldLabel(int field ID), passing as argument the field ID of interest.

### Data Type

Field data types are INT, BINARY, BOOLEAN, DATE, STRING_ARRAY or STRING, as defined in PIMList. To discover a field's data type, call the PIMList method PIMList.getFieldDataType(int field ID), passing as argument the field ID of interest.

### Data Value

Fields can have zero or more data values. Simple fields are represented by the corresponding Java data type, while a multi-value or compound field is represented by an array, such as String or byte arrays. And example of a multi-value field is *NAME*, which consists of first, middle and last name. To manipulate the field's value, you first need to know or discover its data type as mentioned above, and invoke the appropriate method.

### Attributes

Attributes further define field data values. For example, a Telephone (Tel) field, which has a data type of *PIMItem.STRING*, can be further qualified using predefined attributes such as the home telephone number (*ATTR_HOME*), work telephone number (*ATTR_WORK*), mobile telephone number (*ATTR_MOBILE*) or other type of telephone number. Attributes are optional, and are specified when adding data values to a field.

### Test Presence of PIM API

To test the presence of PIM API on handset, call *System.getProperty(String property)* method, passing as argument the String value *"microedition.pim.version"*. The method call will return (a non-null String) the version of the API, if the optional package is present, and null if the API is not present.

```
…
private Form form=null;
   private String pimVersion=null;

   public PIMMidlet()
   {
      form = new Form("PIM version");
      pimVersion = System.getProperty("microedition.pim.version");
      if (pimVersion!=null)
      form.append(pimVersion);
      else
      form.append("PIM package is not supported");

      Display.getDisplay(this).setCurrent(form);
   }
…
```

### Test PIM database

To test given PIM database type ContactList, EventList, ToDoList is supported or not on given handset call method *PIM.openList (int listType, mode)* to test if a specific list type is supported.

```
…
private PIM pim=null;
   private EventList eventList=null;
   public PIMMidlet() {
      pim = PIM.getInstance();
      try {
         eventList = (EventList) pim.openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE);
      }
         catch (SecurityException ex)
         {
            ex.printStackTrace();
         }
        catch (PIMException ex) {
            ex.printStackTrace ();
         } finally {
         if (eventList != null) {
           try {
              eventList.close();
           } catch (PIMException ex) {
              ex.printStackTrace();
           }
         }
      }
```

```
    }
…
```

If the specified list type is not supported, call to *PIM.openList()* throws a *PIMException*. If the MIDlet does not have permission to use the PIM API, a *java.lang.SecurityException* is thrown, which means that we cannot determine if the list type is supported, since access to the API was denied. Typically all three types of databases will be found on a handset, and that multiple lists may exist for each list type. For example, some handsets support various Event lists, for meetings, reminders, calls, memos and birthdays.

## PIM API for Java ME – Permissions

To open the PIM list in read or write mode, PIM read or write permission is required or both are to be required. Accessing the PIM resources without the proper permission will result the *java.lang.SecurityException* being thrown.

The following code snippet shows how to request *EventList* read and write permission in the JAD file.

```
MIDlet-Permissions: javax.microedition.pim.EventList.read,
                    javax.microedition.pim.EventList.write
```

### Testing for Supported Fields

Not all PIM implementations support all the fields or attributes defined in the PIM API specification. Before using a field or an attribute, you should test if it is supported, otherwise you will get an *UnsupportedFieldException* or an *IllegalArgumentException*.

To test if a particular field is supported, call method *PIMList.isSupportedField(int fieldID)*

To test if a particular *STRING_ARRAY* field is supported, call method

*PIMList.isSupportedArrayElement(int stringArrayFieldID, int arrayElement)*

To test if a particular attribute for a particular field is supported, call method

*PIMList.isSupportedAttribute(int fieldID, int attributeID)*

To test if a particular *RepeatRule* field is supported for a specific recurring frequency, call method

*EventList.getSupportedRepeatRuleFields(int frequency)*
The following code snippet shows how to test whether a field is supported or not.

```
        …
    if (eventList.isSupportedField(Event.LOCATION))
```

```
        form.append("Field is supported");
    else
        form.append("Field not supported");
    …
```

The following code snippet shows how to add new contact to the address book using explicit field access that handles optionally supported fields by use of a try catch block with *UnsupportedFieldException* and *PIMException*. In this case, the setting of the whole Contact is rejected if any of the fields are not supported in the particular list implementation.

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;
import javax.microedition.pim.PIM;
import javax.microedition.pim.ContactList;
import javax.microedition.pim.Contact;
import javax.microedition.pim.PIMException;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.pim.PIMItem;
import java.lang.Runnable;
import java.lang.Thread;
import javax.microedition.pim.UnsupportedFieldException;

public class MainMidlet extends MIDlet implements CommandListener {

  private Display display = null;
  private final Command cmd_Exit = new Command("Exit", Command.EXIT, 1);
  private final Command cmd_Commit = new Command("Commit",
  Command.SCREEN, 2);
  private Form form = null;
  private ContactList contactList = null;
  private Contact contact = null;
  private String field_Name[] = null;
  private String field_Addr[] = null;
  private PIM pim = null;
  private TextField name, familyName, country, locality, postalCode, street,
  telephone, email_id;

  public MainMidlet() {
```

```
    init();
}

public void startApp() {

    display.setCurrent(form);

    if (contactList.isSupportedField(Contact.NAME) == true) {

        field_Name = new String[contactList.stringArraySize(Contact.NAME)];
        name = new TextField("NAME:", null, 40,
                TextField.ANY);
        form.append(name);

        familyName = new TextField("FAMILY NAME:", null, 40,
                TextField.ANY);
        form.append(familyName);

    }

    if (contactList.isSupportedField(Contact.ADDR)) {
        field_Addr = new String[contactList.stringArraySize(Contact.ADDR)];

        country = new TextField("COUNTRY:", null, 40,
                TextField.ANY);
        form.append(country);

        locality = new TextField("LOCALITY:", null, 40,
                TextField.ANY);
        form.append(locality);

        postalCode = new TextField("POSTALCODE:", null,
                40, TextField.DECIMAL);
        form.append(postalCode);

        street = new TextField("STREET:", null, 40,
                TextField.ANY);
        form.append(street);

        telephone = new TextField("TELEPHONE:", null, 40, TextField.DECIMAL);
        form.append(telephone);

        email_id = new TextField("EMAIL-ID", null, 40, TextField.ANY);
        form.append(email_id);
    }
```

```
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean flag) {
    }

    public void init() {

        display = Display.getDisplay(this);
        form = new Form("Contact List Demo...");
        form.addCommand(cmd_Exit);
        form.addCommand(cmd_Commit);
        form.setCommandListener(this);
        try {
            pim = PIM.getInstance();
            contactList = (ContactList) pim.openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE,
"Contacts");
        } catch (PIMException ex) {
            ex.printStackTrace();
        }

        if (contactList != null) {
            contact = contactList.createContact();
        }
    }

    public void commandAction(Command cmd, Displayable dis) {

        if (cmd == cmd_Exit) {
            destroyApp(true);
            notifyDestroyed();
        } else {
            new Thread(new Runnable() {

                public void run() {
                    try {
                        if (name.getString() != null) {
                            System.out.println();
                            field_Name[Contact.NAME_GIVEN] = name.getString();
                        }
                        if (familyName.getString() != null) {
                            field_Name[Contact.NAME_FAMILY] = familyName.getString();
                        }
                        if (country.getString() != null) {
                            field_Addr[Contact.ADDR_COUNTRY] = country.getString();
```

```
            }
            if (locality.getString() != null) {
               field_Addr[Contact.ADDR_LOCALITY] = locality.getString();
            }
            if (postalCode.getString() != null) {
               field_Addr[Contact.ADDR_POSTALCODE] = postalCode.getString();
            }
            if (street.getString() != null) {
               field_Addr[Contact.ADDR_STREET] = street.getString();
            }
            if (field_Name != null) {
               contact.addStringArray(Contact.NAME, PIMItem.ATTR_NONE,
                                                                  field_Name);
            }
            if (field_Addr != null) {
               contact.addStringArray(Contact.ADDR, Contact.ATTR_HOME,
                                                                  field_Addr);
            }
            if (telephone.getString() != null) {
               contact.addString(Contact.TEL, Contact.ATTR_HOME,
                                                                  telephone.getString());
            }
            if (email_id.getString() != null) {
               contact.addString(Contact.EMAIL, Contact.ATTR_HOME |
                                 Contact.ATTR_PREFERRED,
                        email_id.getString());
            }
            contact.commit();
            Alert alert = new Alert("info ", "Data added to PIM", null,
                                                               AlertType.CONFIRMATION);
            alert.setTimeout(2000);
            display.setCurrent(alert, form);
         } catch (UnsupportedFieldException e) {
            e.printStackTrace();
         } catch (PIMException ex) {
            ex.printStackTrace();
         } finally {
            try {
               contactList.close();
            } catch (PIMException ex) {
               ex.printStackTrace();
            }


         }
      }
   }).start();
```

```
    }
  }
}
```

## Discovering supported serial data formats

The PIM API provides a number of methods to import and export PIM data using standard data exchange formats. To discover the supported data exchange formats, call method *supportedSerialFormats(PIMListType)* passing argument as the type of PIM list to import or export. The following code snippet shows how to discover the data exchange formats for PIM events lists:

```
…

  private Display display=null;
  private PIM pim = null;
  private String supportedFormats[] = null;
  private static final Command cmd_Exit = new Command("Exit",Command.EXIT,1);
  private Form   form;


  public PIM2_Midlet()  {

        display = Display.getDisplay(this);
        form = new Form("PIM supported Format");
        pim = PIM.getInstance();
        if(pim != null)
        {
          //EVENT_LIST //CONTACT_LIST//
          supportedFormats = pim.supportedSerialFormats(PIM.CONTACT_LIST);
        }
        if(supportedFormats.length > 0)
        {
          for(int i=0;i<supportedFormats.length;i++)
          {
            form.append("\n"+supportedFormats[i]);
          }

        }
        form.addCommand(cmd_Exit);
        form.setCommandListener(this);
        display.setCurrent(form);

  }
…
```

To discover the other types of PIM list data exchange formats, just replace the PIM list type *PIM.CONTACT_LIST* above, for *PIM.EVENT_LIST* or *PIM.TODO_LIST* as appropriate.

The returned serial format names follow the proper common naming convention that is suitable for input to the *toSerialFormat* and *fromSerialFormat* import/export methods. Supported data exchange formats include the vCard and vCalendar formats.

Following code snippet demonstrates how to export calendar events to a file in a specified format.

```
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Command;
import java.lang.Runnable;
import java.lang.Thread;
import javax.microedition.lcdui.List;
import java.util.Vector;
import javax.microedition.pim.PIM;
import javax.microedition.pim.EventList;
import javax.microedition.pim.Event;
import javax.microedition.pim.PIMException;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import java.util.Enumeration;
import javax.microedition.pim.PIMItem;
import javax.microedition.io.file.FileConnection;
import javax.microedition.io.Connector;
import java.io.OutputStream;
import java.util.Date;
import javax.microedition.lcdui.DateField;
import javax.microedition.pim.PIMList;

public class PIMExportMidlet extends MIDlet implements CommandListener{

   private Display display=null;
   private static final Command cmd_Exit = new Command("Exit",Command.EXIT,1);
   private static final Command cmd_Export = new
                                    Command("ExportEvent",Command.SCREEN,2);
   private List event_implicitList=null;
```

```
private Vector eventVector=null;
private PIM pim=null;
private EventList eventList=null;
private final String FILE_PATH="file://localhost/root1/";
private final String FILE="exportEvent.txt";
private final String exportEncoding = "UTF-8";
private DateField startDateField;
private DateField endDateField;

public PIMExportMidlet()
{
   if(System.getProperty("microedition.pim.version")!=null)
       init();
   else
       exitMidlet();
}

public void startApp() {

   new Thread(new Runnable()
   {
      public void run()
      {
         try {
            pim = PIM.getInstance();
            eventList = (EventList)
                  pim.openPIMList(PIM.EVENT_LIST,PIM.READ_WRITE);
            if(eventList==null)
            {
               alertBox("Event List is not supported");
            }
            addEvent();
            if(eventList.isSupportedField(Event.SUMMARY))
            {
                Enumeration e = eventList.items();
                while(e.hasMoreElements())
                {
                    Event event =(Event)e.nextElement();
                    eventVector.addElement(event);
                    String field_String =
                            event.getString(Event.SUMMARY,PIMItem.ATTR_NONE);
                    event_implicitList.append(field_String,null);
                }
                eventList.close();
                display.setCurrent(event_implicitList);
            }
            else
```

```
                {
                    alertBox("Field SUMMARY not supported");
                }
            } catch (PIMException ex) {
                ex.printStackTrace();
            }
        }
    }
    ).start();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
}
private void init()
{
    display = Display.getDisplay(this);
    eventVector = new Vector();
    event_implicitList = new List("Event List...",List.IMPLICIT);
    event_implicitList.addCommand(cmd_Exit);
    event_implicitList.addCommand(cmd_Export);
    event_implicitList.setCommandListener(this);
}
private void alertBox(String data)
{
    Alert alert = new Alert("Information",data,null,
    AlertType.INFO);
    alert.setTimeout(Alert.FOREVER);
    alert.addCommand(cmd_Exit);
    alert.setCommandListener(this);
    display.setCurrent(alert);
}
private void addEvent()
{

        Event event = eventList.createEvent();
        if(eventList.isSupportedField(Event.SUMMARY) == true) {
            String summary = "Export Event";
            event.addString(Event.SUMMARY, PIMItem.ATTR_NONE, summary);
        }
        if(eventList.isSupportedField(Event.START) == true) {
            startDateField = new DateField("Start date",DateField.DATE_TIME);
            startDateField.setDate(new Date());
            long startDate = startDateField.getDate().getTime();
            event.addDate(Event.START, PIMItem.ATTR_NONE, startDate);
```

```
        }
        if(eventList.isSupportedField(Event.END) == true) {
          endDateField = new DateField("End date", DateField.DATE_TIME);
          endDateField.setDate(new Date());
          long endDate = endDateField.getDate().getTime();
          event.addDate(Event.END, PIMItem.ATTR_NONE, endDate);
        }
      try {
        event.commit();
      } catch (PIMException ex) {
        ex.printStackTrace();
      }

  }
  private void exportPIMEvent()
  {

    if(System.getProperty("microedition.io.file.FileConnection.version") == null)
    {
      alertBox("File Connection is not supported");
    }
    final String supported_format[] =
        pim.supportedSerialFormats(PIM.EVENT_LIST);
    if(supported_format.length>0)
    {
      int eventIndex = event_implicitList.getSelectedIndex();
      final Event event = (Event)eventVector.elementAt(eventIndex);
      new Thread(new Runnable(){
      public void run()
      {
        FileConnection fileConn=null;
        OutputStream outPut=null;
          try {
              fileConn = (FileConnection)
        Connector.open(FILE_PATH+FILE,Connector.READ_WRITE);
            if(!fileConn.exists())
              fileConn.create();
              outPut = fileConn.openOutputStream();
            pim.toSerialFormat(event, outPut, exportEncoding,
        supported_format[0]);
          } catch (PIMException ex) {
          ex.printStackTrace();
          } catch (UnsupportedEncodingException ex) {
          ex.printStackTrace();
          }catch (IOException ex) {
            ex.printStackTrace();
          }
```

```
            finally{
                try {

                    outPut.close();
                    fileConn.close();
                    alertBox("Data successfully exported to file...");
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }
        }
        }).start();

    }
  }
  private void exitMidlet()
  {
    destroyApp(true);
    notifyDestroyed();
  }
  public void commandAction(Command cmd,Displayable dis)
  {
    if(cmd == cmd_Exit)
      exitMidlet();
    else
    {
      exportPIMEvent();
    }

  }
}
```

Following code snippet demonstrates how to import Contact:

```
import java.io.IOException;
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Alert;

import javax.microedition.pim.PIM;
import javax.microedition.pim.PIMItem;
import javax.microedition.pim.ContactList;
import javax.microedition.pim.Contact;
```

```
import javax.microedition.pim.PIMException;

import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;

import java.util.Enumeration;
import java.io.InputStream;

public class PIMImportMidlet extends MIDlet implements CommandListener  {

   private Display display;
   private List importContactList;
   private final  Command cmd_Exit= new Command("Exit",Command.EXIT,1) ;
   private final Command cmd_Import= new Command("Import contact",
                                                        Command.SCREEN,2);
    private final String importEncoding = "UTF-8";
   private final String FILE_PATH = "file://localhost/root1/";
   private final String FILE = "5.vcf";
   private FileConnection fileConn=null;
   private InputStream inPutStream=null;

   public PIMImportMidlet() {
      if(System.getProperty("microedition.pim.version")==null) {
         exitMIDlet();
      }
      init();
   }

   private void init() {

      display = Display.getDisplay(this);
      importContactList = new List("contacts", List.IMPLICIT);
      importContactList.addCommand(cmd_Import);
      importContactList.addCommand(cmd_Exit);
      importContactList.setCommandListener(this);
      addContactsToListCtrl();
   }

   private void addContactsToListCtrl() {
      try {

         ContactList contactList = (ContactList)PIM.getInstance().openPIMList(
               PIM.CONTACT_LIST, PIM.READ_WRITE);

         if(contactList.isSupportedField(Contact.NAME) == false) {
            showMsg("Info", "Contact.Name not supported");
         }
```

```
        if(contactList.isSupportedField(Contact.ADDR) == false)  {

            showMsg("Info", "Contact.ADDR not supported");
        }
        importContactList.deleteAll();
        Enumeration contacts = contactList.items();
        while(contacts.hasMoreElements() == true) {
            Contact contact = (Contact)contacts.nextElement();
            String[] name = contact.getStringArray(Contact.NAME,
                  PIMItem.ATTR_NONE);
            String
             address[]=contact.getStringArray(Contact.ADDR,PIMItem.ATTR_NONE);
            String family = name[Contact.NAME_FAMILY];
            String nameGiven = name[Contact.NAME_GIVEN];
            String country = address[Contact.ADDR_COUNTRY];
            String locality = address[Contact.ADDR_LOCALITY];
            String postalCode = address[Contact.ADDR_POSTALCODE];
            String street = address[Contact.ADDR_STREET];
            importContactList.append(family, null);
            importContactList.append(nameGiven, null);
            importContactList.append(country, null);
            importContactList.append(locality, null);
            importContactList.append(postalCode, null);
            importContactList.append(street, null);

        }
        contactList.close();

    } catch(PIMException e) {
        e.printStackTrace();
    } catch(SecurityException e) {
        e.printStackTrace();
    } catch(Exception e) {
        e.printStackTrace();
    }
}

private boolean importContact() {

    if(System.getProperty("microedition.io.file.FileConnection.version")!=null)
        return false;
    try {
    fileConn = (FileConnection)Connector.open(FILE_PATH+FILE,
          Connector.READ_WRITE);

    if(fileConn.exists() == false) {
        throw new Exception("File for importing was not found.");
```

```
    }

     inPutStream = fileConn.openInputStream();
    PIMItem[] items = PIM.getInstance().fromSerialFormat(inPutStream,importEncoding);

    if(items.length > 0) {
       ContactList contactList = (ContactList) PIM.getInstance().openPIMList(
             PIM.CONTACT_LIST, PIM.READ_WRITE);
       Contact newcontact = contactList.importContact((Contact)items[0]);
       newcontact.commit();
       contactList.close();
    }

    showMsg("Info", "Contact was imported from file.");

    } catch(Exception e) {
       showMsg("Error", e.getMessage());
       return false;
    }
    finally{
       try {
          inPutStream.close();
          fileConn.close();
       } catch (IOException ex) {
          ex.printStackTrace();
       }
    }
    return true;
}

private void showMsg(String title, String message) {
    Alert alert = new Alert(title);
    alert.setString(message);
    alert.setTimeout(Alert.FOREVER);
    display.setCurrent(alert);
}

public void startApp() {
    display.setCurrent(importContactList);
}

public void pauseApp() {

}

public void destroyApp(boolean unconditional) {
```

```
    }

    private void exitMIDlet() {
        notifyDestroyed();
    }

    public void commandAction(Command command, Displayable displayable) {

        if(command == cmd_Import) {
            if(importContact()== true) {
                new Thread(new Runnable(){
                public void run()
                {
                    addContactsToListCtrl();
                }
                }).start();

            }
        }
        else if(command == cmd_Exit) {
            exitMIDlet();
        }
    }
}
```