诺基亚论坛

PIM API 简介

版本 1.1；20005 年 8 月 28 日

Java™

NOKIA

目录

修订记录

2004 年 11 月 25 日	版本 1.0	初始文档
2005 年 8 月 28 日	版本 1.1	对 3.1、3.2、3.3 节做了微小改动

1 引言

本文档讲述了 PIM API [JSR-075]，其中包括 API 简介和一个范例。本文档假定读者熟悉 Java™编程，并具有移动信息设备描述 (Mobile Information Device Profile， MIDP)编程的基础，MIDP 编程基础可参见诺基亚论坛中的文档 *MIDP 1.0: Introduction to MIDlet Programming* [MIDPPROG]。PIM API 具有安全方面的限制，因此，读者还应该熟悉 MIDP 2.0 安全架构的概念；诺基亚论坛中的文档 *MIDP 2.0: Tutorial On Signed MIDlets* [SIGNMID]阐述了安全模式。

PIM API 是在 JSR-75: PDA Optional Packages for the J2ME™ Platform 中定义的，JSR-75 共包含两个 Java™ 2 Platform, Micro Edition (J2ME™)可选包，它们用于支持 PDA 之类的设备的功能。这两个可选包提供了对个人信息管理(PIM API)数据库的访问以及处理本地文件系统(FileConnection API)的能力。这两个包相互之间完全独立，因此，设备可以包含其中任意一个包，也可以同时包含两个包。

2 PIM API

2.1 简介

PIM API 是一个可选的 J2ME 包，利用它，可以访问并修改存在于 MIDP 设备中的 PIM 数据库。该包的目的是提供一个可以安全地用于多种设备上的访问 PIM 数据库的标准接口。为了避免发生潜在的死锁，与设备中需要 I/O 操作的多数 API 相同，对 PIM API 的调用也应在异于 GUI 线程的其它线程中完成。

目前，PIM API 共支持三类数据库或者列表：名片夹列表日历项列表、待办事项列表。在某个特定设备中，这三类数据库没有必要同时存在，但是，规范要求，如果在设备中实现了 PIM API，则必须至少有一类数据库可用。一个实现可以包含同种类型的多个列表，例如，移动设备可以在设备内存中保存一份名片夹列表，与此同时，还可以在设备的 SIM 卡中保存另外一份名片夹列表。

PIM API把数据库称为PIMList对象，把单个的数据条目称为PIMItem。PIM API共支持三类PIMList：ContactList、EventList、 ToDoList。类PIM提供了两种openPIMList方法用于访问特定的PIMList。方法的参数包括数据库类型和使用的访问模式(READ_ONLY、WRITE_ONLY、READ_WRITE)。类PIM是singleton的，可以通过静态方法getInstance()获得它。为了减少对安全许可的请求，建议尽量减少打开数据库时请求的访问权限。

每个数据库由唯一的名称来识别，在打开数据库时，需要使用该名称。数据库的命名因实现而异，但是，利用方法listPIMLists可以获得特定类型数据库的可用名称的列表。而且，在诺基亚设备中，这些名称与数据库的本地化名称是一致的。也就是说，根据设备使用的语言，名片夹数据库命名可能为“Contacts”、“Contactos”或“ Puhelinluettelo”。此外，高端设备还允许用户自定义名称或创建新列表。因此，建议在打开数据库时使用方法listPIMList获得名称列表，而不要对列表名称进行硬编码。

为了访问PIMItems的全部内容和选择其特定子集，PIMList包含了若干items()方法。如果底层数据库非常大，则使用这些方法访问数据库的速度将会很慢，因此要慎用这些方法。

PIMList还支持PIMItems的创建、修改和删除。注意，在创建新条目或者修改现有条目时，修改不会立刻映射到本地数据库中，只有在调用方法commit()后，才能完成这种映射。与此不同，删除条目的操作一经请求就立刻完成。为了实现上述操作，必须先以WRITE_ONLY或 READ_WRITE模式打开数据库。

PIM API共有三类PIMItems，它们分别与三类PIMList(Contact、 Event和ToDo)对应。PIMItems包含了特定条目的实际数据。数据被存储在特定字段中，其中包含电话号码、地址等数据。Contact、 Event和ToDo对象的字段是不同的，这些字段由在每个接口上定义的数字常量来标识。一个字段可以包含同一字段类型的多个条目，例如，Contact可以包含多个电话号码。

一个PIMList没有必要支持API中描述的所有字段。相反，每个实现可以定义其支持的字段，对于诺基亚设备来说，其定义的字段与底层本地数据库中的字段相对应。因此，建议在读取某个字段之前，调用方法isSupportedField()或getSupportedFields()验证特定PIMList中的该字段是否可用。注意，即使是同一设备中的数据库，其字段集也可能不同；例如，SIM卡中的名片夹列表仅包含两个字段，与此相比，设备内存中的名片夹列表还包含若干个额外的字段。在设计应用时，应该考虑到这些不同的字段集，这样，才能使应用与多种设备类型兼容。每个被支持的字段可能包含特定PIMItem的 0 个或多个条目。为了确保某个PIMItem的字段可用，并得到该字段包含的条目个数，应该使用方法countValues()。否则，将可能导致抛出IndexOutOfBounds异常。

每个条目还可以包含额外的属性，用以进一步描述该条目。例如，TEL字段通常具有表明它是固定号码、传真号码或是移动号码的属性。在使用属性之前，验证PIMList是否支持该属性非常重要；这可以通过使用方法isSupportedAttribute() 和 getSupportedAttributes()完成。

API中描述的字段和属性都是逻辑值，因此，通常还有必要为特定项目命名一个适合于人阅读的标签。PIMList中的方法getFieldLabel() 和 getAttributeLabel() 能够返回实现中使用的本地化标签。对于实现友好的用户界面来说，方法getSupportedFields()也非常有用，它能够按照设备UI建议的顺序返回支持的字段。

为了进行字段分组，API 还支持分类。诺基亚设备支持的分类与本地数据库中支持的分类相一致。

PIM API具有serializing和deserializing PIMItems的方法，这使得MIDlet能够与外部应用进行交互。Contacts的serialization格式是标准vCard [VCARD]，Event 和 ToDo条目的serialization格式是vCalendar [VCALENDAR]。支持标准的具体版本与实现方式是相关的，版本信息可以用方法PIM.supportedSerialFormats获得。

当有必要验证特定设备中的PIM API是否可用时，应该检验系统属性microedition.pim.version。如果该包可用，则该属性的值为可用的版本号。当前版本号为1.0，否则，其值为null。

2.2 安全

很明显，访问个人数据存在安全和私密问题。许多操作都要求MIDlet获得适当的许可，这可以由用户显式地确认，或将其作为可信任的MIDlet而授予许可。在这些操作中，可能会抛出SecurityException，认识到这一点并对异常进行适当的处理是非常重要的。

MIDP 2.0 MIDlet 既可以是可信的，也可以是不可信的[SIGNMID]。在后一种情况下，设备无法确知 MIDlet 的由来和真实性，因此，在没有显式的用户许可前，不允许调用受限的 API。也就是说，如果需要打开或修改一个 PIM 数据库，则将会出现用户提示，而用户必须显式地确认该操作。

在MIDlet是可信的情况下，设备可以通过X.509 证书判断MIDlet的由来和真实性。这些MIDlet可以根据其所属的安全域自动地获得许可。如果可信的MIDlet希望获得特定PIM API的许可，则必须在Java 应用描述符(Java Application Descriptor，JAD)文件中的属性MIDlet-Permission下声明这些许可，并且要保证对MIDlet进行了适当地签名。

目前有若干种许可能够供 PIM API 利用读写权限访问每种数据库类型：

javax.microedition.pim.ContactList.read

- javax.microedition.pim.ContactList.write
- javax.microedition.pim.EventList.read
- javax.microedition.pim.EventList.write
- javax.microedition.pim.ToDoList.read
- javax.microedition.pim.ToDoList.write

在调用方法 openPIMList 和 listPIMLists 时，需要检验是否具有上述许可。注意，在以 READ 模式打开数据库的情况下，如果试图执行写操作，则会抛出 SecurityException。

MIDP 2.0 规范[MIDP 2.0]定义了与用户数据操作相关的两个功能组。这两个组分别是“Read User Data Access”和 “Write User Data Access”，它们分别包含读操作和写操作许可。功能组为用户提供了更高层面的许可，使用户能够整体地改变许可。例如，用户可以授予第三方访问所有 PIM 数据库的权限，而不需要对每一个许可单独授权。表 1 给出了不可信和可信的第三方 MIDlet 的默认的模式和允许的许可模式。

功能组	可信的第三方域		不可信的域	
	默认设置	允许的设置	默认设置	允许的设置
Read User Data Access	Oneshot	Session, Blanket, Oneshot, No	Oneshot	Oneshot, No
Write User Data Access	Oneshot	Session, Blanket, Oneshot, No	No	Oneshot, No

表 1：允许的和默认的许可模式

2.3 诺基亚实现方式说明

由于 PIM API 为不同制造商预留了多样化的空间，所以，了解诺基亚设备的某些实现细节非常重要。诺基亚实现的 API 可以因开发平台(即 Series 40 和 Series 60 开发平台)而异，但是，同一平台的不同设备中的 API 实现是相同的。

特定实现的细节主要是指从本地字段到 PIM API 字段的映射。以下各节将讨论这些细节。

2.3.1 名片夹列表数据库

在诺基亚设备中，至少有一种名片夹数据库可用，但是，通常有两个数据库，一个用于设备内存中的名片夹列表，另一个用于 SIM 卡中的名片夹。另外，有的设备也可能具有更多数据库。可以根据名称，使用方法 `openPIMList(int pimListType, int mode, String name)`选择数据库。用于识别数据库的名称要与设备使用的本地化名称一致，在某些情况下，还可以由用户来定义。因此，在应用中，不应该对数据库名称进行硬编码，而应该使用方法 `listPIMLists` 获得其名称。设备还决定了方法 `listPIMLists` 返回的数据库名称的顺序。一般情况下，可以使用的数据库有两个。第一个是“内存”数据库，在默认情况下，应该使用该数据库，第二个是 SIM 卡中的列表。打开默认数据库的方法有两种：1. 调用 `openPIMList(int pimListType, int mode);` 2. 读取 `listPIMLists` 中的第一个条目，并调用 `openPIMList(int pimListType, int mode, String name)`。

每个数据库中包含的字段和属性都因设备型号和数据库类型而异。以下各表列举了 SIM、Series 40 和 Series 60 名片夹数据库支持的字段。

本地名片夹字段	PIM API 名片夹字段类型	PIM API 名片夹字段属性
Name	Contact.FORMATTED_NAME	
Phone number	Contact.TEL	Contact.ATTR_PREFERRED

表 2：SIM 名片夹数据库支持的字段

本地名片夹字段	PIM API 名片夹字段类型	PIM API 名片夹字段属性
Name	Contact.FORMATTED_NAME	PIMItem.ATTR_NONE
General phone number	Contact.TEL	Contact.ATTR_NONE
Mobile number	Contact.TEL	Contact.ATTR_MOBILE
Home number	Contact.TEL	Contact.ATTR_HOME
Office number	Contact.TEL	Contact.ATTR_WORK
Fax number	Contact.TEL	Contact.ATTR_FAX

本地名片夹字段	**PIM API** 名片夹字段类型	**PIM API** 名片夹字段属性
E-mail	Contact.EMAIL	Contact.ATTR_HOME
Web address	Contact.URL	PIMItem.ATTR_NONE
Postal address	Contact.ADDR	Contact.ATTR_HOME
Note	Contact.NOTE	PIMItem.ATTR_NONE
Image	Contact.PHOTO	PIMItem.ATTR_NONE

表 3：Series 40 名片夹数据库支持的字段

本地名片夹字段	**PIM API** 名片夹字段类型	**PIM API** 名片夹字段属性
First name	Contact.NAME_GIVEN	PIMItem.ATTR_NONE
Last name	Contact.NAME_FAMILY	PIMItem.ATTR_NONE
Last name <space> First name	Contact.FORMATTED_NAME	
Company	Contact.ORG	PIMItem.ATTR_NONE
Job title	Contact.TITLE	PIMItem.ATTR_NONE
Address	Contact.ADDR	PIMItem.ATTR_NONE
Address (home)	Contact.ADDR	Contact.ATTR_HOME
Address (business)	Contact.ADDR	Contact.ATTR_WORK
Telephone	Contact.TEL	PIMItem.ATTR_NONE
Telephone (home)	Contact.TEL	Contact.ATTR_HOME
Telephone (business)	Contact.TEL	Contact.ATTR_WORK
Mobile	Contact.TEL	Contact.ATTR_MOBILE
Mobile (home)	Contact.TEL	Contact.ATTR_MOBILE + Contact.ATTR_HOME
Mobile (business)	Contact.TEL	Contact.ATTR_MOBILE + Contact.ATTR_WORK
Fax	Contact.TEL	Contact.ATTR_FAX
Fax (home)	Contact.TEL	Contact.ATTR_FAX + Contact.ATTR_HOME
Fax (business)	Contact.TEL	Contact.ATTR_FAX + Contact.ATTR_WORK
E-mail	Contact.EMAIL	PIMItem.ATTR_NONE
E-mail (home)	Contact.EMAIL	Contact.ATTR_HOME
E-mail (business)	Contact.EMAIL	Contact.ATTR_WORK
Pager	Contact.TEL	Contact.ATTR_PAGER
Birthday	Contact.BIRTHDAY	PIMItem.ATTR_NONE
URL	Contact.URL	PIMItem.ATTR_NONE
URL (home)	Contact.URL	Contact.ATTR_HOME

本地名片夹字段	PIM API 名片夹字段类型	PIM API 名片夹字段属性
URL (business)	Contact.URL	Contact.ATTR_WORK
Note	Contact.NOTE	PIMItem.ATTR_NONE
Photo	Contact.PHOTO	PIMItem.ATTR_NONE

表 4：Series 60 名片夹数据库支持的字段

在 Series 60 设备中，地址就是由地址的不同部分组成的字符串数组。字符串数组的索引如表 5 所示。

地址组成部分	PIM 名片夹字段索引
P.O. Box	Contact.ADDR_POBOX
Extension	Contact.ADDR_EXTRA
Street	Contact.ADDR_STREET
Postal/ZIP	Contact.ADDR_POSTALCODE
City	Contact.ADDR_LOCALITY
State/province	Contact.ADDR_REGION
Country/region	Contact.ADDR_COUNTRY

表 5：Series 60 设备中地址字符串组成部分的数组

如表 2 至表 4 所示，不同数据库类型支持的字段存在很大差异。应用必须合理地处理这些差异，例如，应用仅用于某个特定设备型号，或者使用常见的命名方法。可以用方法 `isSupportedField` 检测实际处理的数据库类型。

2.3.2 日历项列表数据库

同样地，Series 40 和 Series 60 设备支持的字段不尽相同。Series 40 和 Series 60 设备都能支持若干种日历项列表数据库，这些数据库的名称是本地化的，代表了各种日历项类别。但是，`listPIMLists` 以抽象的方式返回如下列表：

- Series 40: {“Meeting”, “Call”, “Birthday”, “Memo”, “Reminder”}
- Series 60: {“Appointment”, “Event”, “Anniversary”}

在调用 `listPIMLists` 时，要保持这些列表的顺序，这样就可以保证，尽管第一个条目在 Series 40 设备中的本地化名称是“Meeting”，但是其逻辑意义仍然相同。

不同类型的列表中，日历项数据库所支持的字段也不尽相同。表 6 列举了 Series 40 设备支持的字段以及这些字段与本地字段的映射关系。

日历项数据库	本地字段	PIM API 字段
Meeting	Subject	Event.SUMMARY
	Location	Event.LOCATION
	Start time	Event.START
	End time	Event.END
	Alarm	Event.ALARM

日历项数据库	本地字段	PIM API 字段
Call	Phone number	Event.SUMMARY
	Name	Event.NOTE
	Time	Event.START
	Alarm	Event.ALARM
Birthday	Name	Event.SUMMARY
	Year of birth	Event.NOTE
	Time	Event.START
	Alarm	Event.ALARM
Memo	Subject	Event.SUMMARY
	Start date	Event.START
	End date	Event.END
	Alarm	Event.ALARM
Reminder	About	Event.SUMMARY
	Start time	Event.START
	Alarm	Event.ALARM

表 6：Series 40 日历项数据库支持的字段

在 Series 60 设备中，日历项数据库的映射关系与 Series 40 不同，如表 7 所示。

日历项数据库	本地字段	PIM API 字段
Meeting	Subject	Event.SUMMARY
	Location	Event.LOCATION
	Start time	Event.START
	Start date	
	End time	Event.END
	End date	
	Alarm	Event.ALARM
Memo	Subject	Event.SUMMARY
	Start date	Event.START
	End date	Event.END
Anniversary	Occasion	Event.SUMMARY

表 7：Series 60 日历项数据库支持的字段

在 Series 40 和 Series 60 设备中，日历项都可以具有重复规则，并且可以用方法 `getRepeat` 获得该规则。重复规则属性与本地字段的映射关系如表 8 所示。

Series 40 重复类型	Series 60 重复类型	PIM API 重复规则属性
Never	Not repeated	`getRepeat` returns null
Every day	Daily	(FREQUENCY, DAILY)
Every week	Weekly	(FREQUENCY, WEEKLY)
Every two week(s)	Fortnightly	(FREQUENCY, WEEKLY) + (INTERVAL, 2)
Every month	Monthly	(FREQUENCY, MONTHLY)
Every year	Yearly	(FREQUENCY, YEARLY)

表 8：Series 40 和 Series 60 设备中重复类型的映射关系

2.3.3 待办事项数据库

Series 40 和 Series 60 设备中的待办事项数据库具有相同的属性集。因为它们仅支持一个待办事项数据库，所以没有查询数据库名称的必要。字段之间的映射关系如表 9 所示。

本地字段	PIM API 待办事项字段
Subject	ToDo.SUMMARY
Priority	ToDo.PRIORITY
Due date	ToDo.DUE
Completed	ToDo.COMPLETED
Completion date	ToDo.COMPLETION_DATE

表 9：Series 40 和 Series 60 设备中支持的待办事项字段

待办事项条目还可以具有优先级顺序； PIM API 支持十种优先级，而目前的诺基亚设备仅支持四种优先级。它们之间的映射关系如表 10 所示。

PIM API 优先级	本地优先级
0	Not used
1, 2, 3	High
4, 5, 6	Medium
7, 8, 9	Low

表 10：待办事项优先级的映射关系

3 日历项共享 MIDlet

本文档给出的范例应用可以与其它用户共享某个特定日历项。该应用提供构建日历项、把日历项序列化成 vCalendar，并把日历项内容作为消息发送给其它设备(可以从名片夹列表中获得该设备的号码)的方法。此外，它还可以把日历项添加到本地日历项数据库中。该应用利用 PIM API 中提供的序列化功能，通过 Wireless Messaging API [WMA]，以短消息服务 (Short Message Service ， SMS)的方式发送消息。该范例非常简单；一个实际的应用应该具有更多功能，如冲突解决和确认等。还应注意，在通常情况下，当捕获到异常时，只是简单地把异常显示给用户，而没有给出一条用户界面友好的消息。

MIDlet 启动后使用一个屏幕来定义并发送日历项。在该屏幕中，用户可以设置目的号码、日历项主题、开始日期/时间和持续时间(以分钟为单位)。此外，还可以通过利用菜单条目“Find Contacts”，在名片夹列表中查找目的号码。这将显示名片夹姓名和电话号码的列表。一旦消息被发送，该日历项就被插入到本地数据库中。

MIDlet 通过在指定 SMS 端口(6553)上监听发送来的消息来建立自己。当接收到消息时，MIDlet 将弹出一个新屏幕，然后对消息进行解码，并把日历项显示在屏幕上。此时，接收者可以接受把新数据插入到日历中的操作。

MIDlet 还在 `PushRegistry` 中进行自我注册，并在指定的 SMS 端口中等待发送来的消息。利用 JAD 文件中的 MIDlet-Push 条目，MIDlet 可以完成静态注册。

为了避免 UI 线程的死锁，应该在一个独立的线程中完成所有与读、写 PIM 数据库相关的操作。

图 1 至图 3 给出了该应用软件的屏幕截屏。

图 1：诺基亚 6630 设备中的会议请求发送屏幕

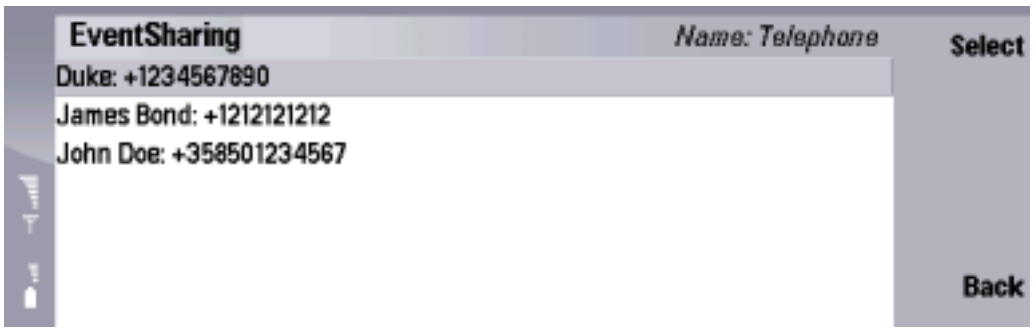

图 2：诺基亚 9500 通讯器中的名片夹查找屏幕

图 3：诺基亚 6630 设备中的会议请求接收屏幕

图 4 给出了日历项共享 MIDlet 应用的一个简化类图。

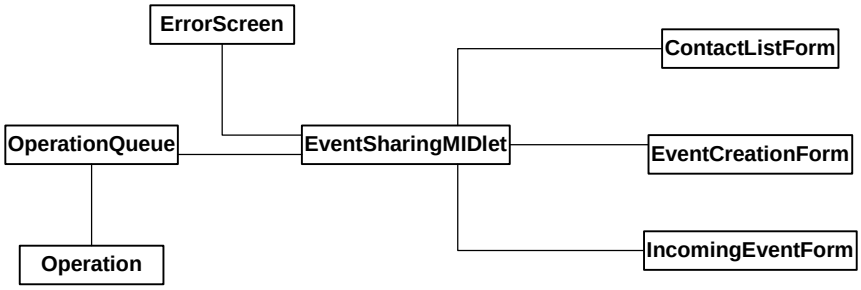

图 4：日历项共享 MIDlet 的类图

3.1 EventSharingMIDlet

这是 MIDlet 的主类；它负责 MIDlet 的初始化，并处理不同屏幕之间的切换。该类还负责完成所有 SMS 处理，其中包括创建 `MessageConnection`、接收消息和发送新消息。在启动 MIDlet 时，该类还检查 MIDlet 是否是由于发送来的消息而被 `PushRegistry` 激活的。

该类包括一个 `OperationsQueue` 实例，其它类使用该实例在一个独立线程中执行与 PIM 相关的操作。

```
import java.io.*;
import java.util.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.pim.*;
import javax.wireless.messaging.*;

// Main class of the EventSharing MIDlet
public class EventSharingMIDlet
  extends MIDlet
  implements MessageListener
{
  private final static int DEFAULT_PORT = 6553;
  private final static String SMS_PREFIX = "sms://:";

  private final Image logo;
  private final EventCreationForm mainScreen;

  private int port;
```

```
private MessageConnection conn;
private Message nextMessage;
private OperationsQueue queue = new OperationsQueue();
private final static int CONTACTS_LIST_OP = 0;
private final Display display;


public EventSharingMIDlet()
{
  // init basic parameters
  logo = makeImage("/logo.png");
  try
  {
    port = Integer.parseInt(getAppProperty("port"));
  }
  catch (Exception e)
  {
    // in case the property is missing or with a wrong format
    port = DEFAULT_PORT;
  }
  ErrorScreen.init(logo, Display.getDisplay(this));
  mainScreen = new EventCreationForm(this, port);
  display = Display.getDisplay(this);
}


void sendMessage(String number, byte[] text)
{
  showMessage("Going to send the mgs " + text + " " + conn);
  if (conn != null)
  {
    try
    {
      // create a new message
      BinaryMessage requestSMS = (BinaryMessage) conn.newMessage(
        MessageConnection.BINARY_MESSAGE);
      String address = new StringBuffer("sms://")
        .append(number).append(":").append(port).toString();
      requestSMS.setAddress(address);

      requestSMS.setPayloadData(text);
      conn.send(requestSMS);
    }
    catch (IOException e)
    {
      showMessage(e.getMessage());
    }
    catch (SecurityException e)
    {
      showMessage(e.getMessage());
    }
  }
}


public void startApp()
{
  Displayable current = Display.getDisplay(this).getCurrent();
  if (current == null)
  {
    // check that the API is available
    boolean isAPIAvailable =
      (System.getProperty("microedition.pim.version") != null);
    // shows splash screen
    StringBuffer splashText =
```

```
      new StringBuffer(getAppProperty("MIDlet-Name")).
        append("\n").append(getAppProperty("MIDlet-Vendor")).
        append(isAPIAvailable?"":"\nPIM API not available");
    Alert splashScreen = new Alert(null,
      splashText.toString(),
      logo,
      AlertType.INFO);
    splashScreen.setTimeout(3000);
    if (!isAPIAvailable)
    {
      display.setCurrent(splashScreen, mainScreen);
    }
    else
    {
      display.setCurrent(splashScreen, mainScreen);
      // List connections from PushRegistry
      String smsConnections[] = PushRegistry.listConnections(true);

      // Check the connections. We'll assume only one is of interest
      for (int i = 0; i < smsConnections.length; i++)
      {
        if (smsConnections[i].startsWith(SMS_PREFIX))
        {
          try
          {
            conn =
              (MessageConnection) Connector.open(smsConnections[i]);
            BinaryMessage incomingMessage =
              (BinaryMessage) conn.receive();
            showIncomingMessage(incomingMessage);
            conn.close();
          }
          catch (IOException e)
          {
            showMessage(e.getMessage());
          }
          catch (SecurityException e)
          {
            showMessage(e.getMessage());
          }
        }
      }

      // Build the connection string
      String connection = SMS_PREFIX + port;
      try
      {
        // Initiate the connection and add listener
        conn = (MessageConnection) Connector.open(connection);
        conn.setMessageListener(this);
      }
      catch (IOException e)
      {
        showMessage(e.getMessage());
      }
      catch (SecurityException e)
      {
        showMessage(e.getMessage());
      }
    }
  }
  else
  {
    Display.getDisplay(this).setCurrent(current);
  }
```

```
}


public void pauseApp()
{
}


public void destroyApp(boolean unconditional)
{
  try
  {
    // Close the connection on exit
    if (conn != null)
    {
      conn.close();
    }
  }
  catch (IOException e)
  {
    // Ignore since we are closing anyway
  }
  queue.abort();
}


// Asynchronous callback for inbound message.
public void notifyIncomingMessage(MessageConnection conn)
{
  if (conn == this.conn && conn != null)
  {
    try
    {
      // Create a ReceiveScreen upon message removal
      BinaryMessage incomingMessage = (BinaryMessage) conn.receive();
      showIncomingMessage(incomingMessage);
    }
    catch (IOException e)
    {
      showMessage(e.getMessage());
    }
    catch (SecurityException e)
    {
      showMessage(e.getMessage());
    }
  }
}


void showIncomingMessage(BinaryMessage msg)
{
  display.setCurrent(new IncomingEventForm(this, msg));
}


void showMessage(String message)
{
  ErrorScreen.showError(message, mainScreen);
}


void showMessage(String message, Displayable mainScreen)
{
  ErrorScreen.showError(message, mainScreen);
}
```

```
  // adds an operation the queue
  void enqueueOperation(Operation operation)
  {
    queue.enqueueOperation(operation);
  }


  // shows the main screen
  void showMain()
  {
    display.setCurrent(mainScreen);
  }


  // shows the contacts list screen
  void showContactsList()
  {
    display.setCurrent(
      new ContactListForm(EventSharingMIDlet.this));
  }


  // callback from contacts list when a particular
  // telephone number has been selected
  void contactSelected(String telephoneNumber)
  {
    mainScreen.setTargetPhoneNumber(telephoneNumber);
    showMain();
  }


  // loads a given image by name
  static Image makeImage(String filename)
  {
    Image image = null;

    try
    {
      image = Image.createImage(filename);
    }
    catch (Exception e)
    {
      // use a null image instead
    }

    return image;
  }

}
```

3.2 EventCreationForm

该表单的功能是创建日历项，并把日历项发送给另一台设备。它包含若干字段，分别用于定义日历项的主题、开始时间和持续时间。在该屏幕中，需要输入目的电话号码。电话号码可以从名片夹数据库中查找。为了实现电话号码的查找，需要用“Find Contacts”命令请求显示 ContactListForm，然后，把选中的号码插入到目的号码字段中。

填写完各个字段并发出“Send”命令后，MIDlet 首先对字段中的内容进行验证，然后创建一个新日历项，并把日历项导入到首个 EventList 数据库中，序列化成一条 vCalendar 消息。稍后，可以通过请求 EventSharingMIDlet 把该消息发往目的号码。以上操作是在一个独立线程中完成的，该线程的逻辑包含在内部类 SendEventMesssageOperation 中。

```
import java.io.*;
import java.util.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.pim.*;
import javax.wireless.messaging.*;

// Form used to create a new event and send it to a contact
class EventCreationForm
 extends Form
 implements CommandListener
{
  private final Command sendCommand, exitCommand, findCommand;
  private final TextField number, duration, topic;
  private final DateField dateTime;
  private final EventSharingMIDlet parent;
  private final int port;


  public EventCreationForm(EventSharingMIDlet parent, int port)
  {
    super("Create an event");
    this.parent = parent;
    this.port = port;

    // init UI
    sendCommand = new Command("Send", Command.OK, 1);
    findCommand = new Command("Find Contacts", Command.OK, 0);
    exitCommand = new Command("Exit", Command.EXIT, 1);

    number = new TextField("Dest. number",
                           "",
                           20,
                           TextField.ANY);
    dateTime = new DateField("Date",
                             DateField.DATE_TIME,
                             TimeZone.getDefault());
                             dateTime.setDate(new Date());
    duration = new TextField("Duration (min)",
                             "30",
                             24,
                             TextField.ANY);
    topic = new TextField("Subject",
                          "",
                          24,
                          TextField.ANY);

    // create the form
    append(number);
    append(dateTime);
    append(duration);
    append(topic);

    addCommand(findCommand);
    addCommand(sendCommand);
    addCommand(exitCommand);
    setCommandListener(this);
  }


  void setTargetPhoneNumber(String phoneNumber)
  {
    number.setString(phoneNumber);
```

```
}


public void commandAction(Command cmd, Displayable displayable)
{
  if (cmd == sendCommand)
  {
    // do a sanity check and then
    // send the message in a separate thread
    parent.enqueueOperation(
      new SendEventMessageOperation(number.getString(),
                                    dateTime.getDate(),
                                    duration.getString(),
                                    topic.getString()));
  }
  else if (cmd == exitCommand)
  {
    parent.notifyDestroyed();
  }
  else if (cmd == findCommand)
  {
    parent.showContactsList();
  }
}


private class SendEventMessageOperation implements Operation
{
  private final String number, duration, topic;
  private final Date date;


  SendEventMessageOperation(String number,
    Date date,
    String duration,
    String topic)
  {
    this.number = number;
    this.date = date;
    this.duration = duration;
    this.topic = topic;
  }


  public void execute()
  {
    int length = 0;
    try
    {
      length = Integer.parseInt(duration);
    }
    catch (NumberFormatException e)
    {
      parent.showMessage("Duration not valid",
        EventCreationForm.this);
      return;
    }
    if (length <= 0)
    {
      parent.showMessage("Duration needs to be positive",
        EventCreationForm.this);
    }
    else if (number == null || number.length() == 0)
    {
      parent.showMessage("Number not entered",
```

```
    EventCreationForm.this);
}
else if (topic == null || topic.length() == 0)
{
  parent.showMessage("Subject not entered",
    EventCreationForm.this);
}
else
{
  ByteArrayOutputStream out = new ByteArrayOutputStream();
  EventList eventList = null;
  try
  {
    PIM pim = PIM.getInstance();
    String listNames[] = pim.listPIMLists(PIM.EVENT_LIST);
    if (listNames.length > 0)
    {
      eventList = (EventList) pim.openPIMList(PIM.EVENT_LIST,
                                              PIM.READ_WRITE,
                                              listNames[0]);
      Event newEvent = eventList.createEvent();
      if (eventList.isSupportedField(Event.SUMMARY))
      {
        newEvent.addString(Event.SUMMARY,
          PIMItem.ATTR_NONE,
          topic);
      }
      if (eventList.isSupportedField(Event.START))
      {
        newEvent.addDate(Event.START,
          PIMItem.ATTR_NONE,
          date.getTime());
      }
      if (eventList.isSupportedField(Event.END))
      {
        newEvent.addDate(Event.END,
          PIMItem.ATTR_NONE,
          date.getTime() + 60 * 1000 * length);
      }
      // let's check that VCALENDAR/1.0 is supported
      String supportedFormats[] = PIM.getInstance()
        .supportedSerialFormats(PIM.EVENT_LIST);
      for (int i=0;i<supportedFormats.length;i++) {
        if (supportedFormats[i].equals("VCALENDAR/1.0")) {
          PIM.getInstance().toSerialFormat(newEvent,
                                           out,
                                           "UTF-8",
                                           "VCALENDAR/1.0");
          break;
        }
      }
      if (out.size() == 0)
      {
        parent.showMessage("VCALENDAR/1.0 not supported",
          EventCreationForm.this);
      }
      // let's add the event locally
      // an advanced version should wait for an ack
      eventList.importEvent(newEvent);
      newEvent.commit();
    }
    else
    {
      parent.showMessage("No Event list available",
             EventCreationForm.this);
```

```
            }
          }
          catch (PIMException e)
          {
            parent.showMessage(e.getMessage(), EventCreationForm.this);
          }
          catch (SecurityException e)
          {
            parent.showMessage(e.getMessage(), EventCreationForm.this);
          }
          catch (UnsupportedEncodingException e)
          {
            // should not happen since UTF-8 is mandatory
          }
          finally
          {
            try
            {
              if (eventList != null)
              {
                eventList.close();
              }
            }
            catch (PIMException e)
            {
              // ignore, we are closing anyway
            }
          }
          // out could be empty if there is no support
          // for VCALENDAR/1.0
          if (out.size() > 0)
          {
            parent.sendMessage(number, out.toByteArray());
          }
        }

      }
    }

  }
```

3.3 IncomingEventForm

在接收到消息时，该表单被激活。这种情况下，MIDlet 将假设该消息使用了 PIM 实现可以接受的格式，并对消息进行解码，把其内容显示在屏幕上。MIDlet 解析接收到的消息的地址，并从中提取出电话号码，然后，搜索名片夹数据库，试图根据电话号码确定发送者的名字。

```
import java.io.*;
import java.util.*;
import javax.microedition.lcdui.*;
import javax.microedition.pim.*;
import javax.wireless.messaging.*;

// Forms displayed when a message is received
class IncomingEventForm
  extends Form
  implements CommandListener
{
  private Command acceptCommand, exitCommand, backCommand;
  private TextField number, duration, topic;
  private DateField dateTime;
  private final EventSharingMIDlet parent;
  private Event event;
```

```
IncomingEventForm(EventSharingMIDlet parent, BinaryMessage msg)
{
  super("Event Received");
  this.parent = parent;

  backCommand = new Command("Back", Command.BACK, 2);
  exitCommand = new Command("Exit", Command.EXIT, 2);

  addCommand(backCommand);
  addCommand(exitCommand);

  setCommandListener(this);
  parent.enqueueOperation(new LoadEvent(msg));
}


public void commandAction(Command cmd, Displayable displayable)
{
  if (cmd == acceptCommand)
  {
    // insert the event in a different thread
    parent.enqueueOperation(new InsertEvent());
  }
  else if (cmd == exitCommand)
  {
    parent.notifyDestroyed();
  }
  else if (cmd == backCommand)
  {
    parent.showMain();
  }
}


// finds a name based on the number in all contact lists
private String findName(String number)
{
  String[] allLists = PIM.getInstance()
        .listPIMLists(PIM.CONTACT_LIST);
  if (allLists.length > 0)
  {
    String results[] = new String[allLists.length];
    for (int i = 0; i < allLists.length; i++)
    {
      results[i] = findNameInList(number, allLists[i]);
    }
    // if there is more than one, only the first is returned
    for (int i = 0; i < allLists.length; i++)
    {
      if (results[i] != null)
      {
        return results[i];
      }
    }
  }
  return null;
}


// finds a name based on the number in a particular contact list
// if for some reason it cannot be found, return null
private String findNameInList(String number, String list)
{
  ContactList contactList = null;
  try
  {
    contactList = (ContactList) PIM.getInstance()
          .openPIMList(PIM.CONTACT_LIST, PIM.READ_ONLY, list);
    if (contactList.isSupportedField(Contact.TEL)
```

```
            && contactList.isSupportedField(Contact.FORMATTED_NAME))
        {
          Contact pattern = contactList.createContact();
          pattern.addString(Contact.TEL, PIMItem.ATTR_NONE, number);
          Enumeration matching = contactList.items(pattern);
          if (matching.hasMoreElements())
          {
            // will only return the first match
            Contact ci = (Contact) matching.nextElement();
            // FORMATTED_NAME is mandatory
            return ci.getString(Contact.FORMATTED_NAME, 0);
          }
        }
      }
      catch (PIMException e)
      {
        //just ignore and return null;
      }
      catch (SecurityException e)
      {
        //just ignore and return null;
      }
      finally
      {
        if (contactList != null)
        {
          try
          {
            contactList.close();
          }
          catch (PIMException e)
          {
            // ignore, nothing can be done here
          }
        }
      }
      return null;
    }


    private class InsertEvent implements Operation
    {
      public void execute()
      {
        EventList eventList = null;
        try
        {
          PIM pim = PIM.getInstance();
          String listNames[] = pim.listPIMLists(PIM.EVENT_LIST);
          if (listNames.length>0)
          {
            eventList = (EventList) pim.openPIMList(PIM.EVENT_LIST,
                                                    PIM.READ_WRITE,
                                                    listNames[0]);
            // Check that the fields are supported
            if (eventList.isSupportedField(Event.SUMMARY) &&
              eventList.isSupportedField(Event.START) &&
              eventList.isSupportedField(Event.END))
            {
              // event cannot be null at this stage
              eventList.importEvent(event).commit();
              parent.showMessage("Event inserted in the local database");
            }
          }
          else
          {
            parent.showMessage("No Event list found");
          }
        }
```

```
      catch (PIMException e)
      {
        parent.showMessage(e.getMessage(), IncomingEventForm.this);
      }
      catch (SecurityException e)
      {
        parent.showMessage(e.getMessage(), IncomingEventForm.this);
      }
      finally
      {
        if (eventList != null)
        {
          try
          {
            eventList.close();
          }
          catch (PIMException e)
          {
            // nothing to do here, just ignore
          }
        }
      }
    }
  }

  // This operation reads an event from a TextMessage
  private class LoadEvent implements Operation
  {
    private final BinaryMessage msg;


    LoadEvent(BinaryMessage msg)
    {
      this.msg = msg;
    }


    public void execute()
    {
      PIMItem items[] = null;
      try
      {
        items = PIM.getInstance().fromSerialFormat(
              new ByteArrayInputStream(msg.getPayloadData()),
              "UTF-8");
      }
      catch (PIMException e)
      {
        parent.showMessage(e.getMessage(), IncomingEventForm.this);
      }
      catch (UnsupportedEncodingException e)
      {
        // should not happen since UTF-8 is mandatory
      }
      if (items != null && items.length>0)
      {
        // let's assume that only one event
        // was contained in the message
        event = (Event) items[0];
        // Sanity check
        int summaryCount = event.countValues(Event.SUMMARY);
        int startCount = event.countValues(Event.START);
        int endCount = event.countValues(Event.END);
        if (summaryCount>0 && startCount>0 && endCount>0)
        {
          topic = new TextField("topic",
                                event.getString(Event.SUMMARY, 0),
                                24,
                                TextField.ANY);
```

```
                String source = msg.getAddress();
                // assume the message starts with sms:// and has a port number
                String phoneNumber = source.substring(6,
                      source.lastIndexOf(':'));
                String sourceName = findName(phoneNumber);

                if (sourceName != null)
                {
                  number = new TextField("From",
                                         sourceName,
                                         20,
                                         TextField.ANY |
                                         TextField.UNEDITABLE);
                }
                else
                {
                  number = new TextField("From",
                                         phoneNumber,
                                         20,
                                         TextField.PHONENUMBER |
                                         TextField.UNEDITABLE);
                }
                long startDate = event.getDate(Event.START, 0);
                long length = event.getDate(Event.END, 0) - startDate;
                // get the length in minutes
                length /= 60000;
                dateTime = new DateField("on",
                                         DateField.DATE_TIME,
                                         TimeZone.getDefault());
                dateTime.setDate(new Date(startDate));

                duration = new TextField("duration (min)",
                                         "" + length,
                                         24,
                                         TextField.ANY);

                append(number);
                append(topic);
                append(dateTime);
                append(duration);

                acceptCommand = new Command("Accept", Command.OK, 1);
                addCommand(acceptCommand);
              }
              else
              {
                append(new StringItem("", "Incoming event was incomplete"));
              }
            }
            else
            {
              event = null;
              acceptCommand = null;
              append(new StringItem("", "Error during message decoding"));
            }
          }
        }

    }
```

3.4 ContactsListForm

该表单显示具有电话号码的名片的列表，这样，用户就可以选中某个电话号码。没有电话号码的名片将不显示。如果名片具有多个电话号码，则一个号码需要对应一个条目。号码的排列要遵照移动号码优先显示的准则。

用户可以选择一个特定的名片/号码组合，并把该信息返回给 EventCreationForm。

```
import java.util.*;
import javax.microedition.lcdui.*;
import javax.microedition.pim.*;

// This form shows a list of the contacts in the local databases
class ContactListForm
  extends List
  implements CommandListener
{
  private final Command exitCommand, selectCommand, backCommand;
  private final EventSharingMIDlet parent;
  private boolean available;
  private Vector allTelNumbers = new Vector();


  public ContactListForm(EventSharingMIDlet parent)
  {
    super("Contacts", Choice.IMPLICIT);
    this.parent = parent;

    // init UI
    selectCommand = new Command("Select", Command.OK, 0);
    backCommand = new Command("Back", Command.BACK, 1);
    exitCommand = new Command("Exit", Command.EXIT, 1);

    addCommand(backCommand);
    addCommand(exitCommand);
    setCommandListener(this);
    setFitPolicy(Choice.TEXT_WRAP_ON);

    // load the list of names in a different thread
    parent.enqueueOperation(new LoadContacts());
  }


  public void commandAction(Command cmd, Displayable displayable)
  {
    // if no names are available return
    if (!available)
    {
      parent.showMain();
      return;
    }
    else if (cmd == selectCommand)
    {
      int selected = getSelectedIndex();
      if (selected >= 0)
      {
        // will get the number from the list
        parent.contactSelected(
          (String) allTelNumbers.elementAt(selected));
      }
      else
      {
        parent.showMain();
      }
    }
    else if (cmd == backCommand)
    {
      parent.showMain();
    }
    else if (cmd == exitCommand)
    {
      parent.notifyDestroyed();
    }
  }


  // loads the names of a named contact list
```

```
private void loadNames(String name)
  throws PIMException, SecurityException
{
  ContactList contactList = null;
  try
  {
    contactList = (ContactList) PIM.getInstance()
      .openPIMList(PIM.CONTACT_LIST, PIM.READ_ONLY, name);


    // First check that the fields we are interested in are supported
    // by the PIM List
    if (contactList.isSupportedField(Contact.FORMATTED_NAME)
        && contactList.isSupportedField(Contact.TEL))
    {
      // Put an informative title while reading the contacts
      setTitle("Contacts");
      append("Reading contacts...", null);
      Enumeration items = contactList.items();
      // change the title to use the localized name.
      // getFieldLabel returns an i18n version
      delete(0);
      setTitle(contactList.getFieldLabel(Contact.FORMATTED_NAME)
               + ": "
               + contactList.getFieldLabel(Contact.TEL));
      Vector telNumbers = new Vector();
      while (items.hasMoreElements())
      {
        Contact contact = (Contact) items.nextElement();
        int telCount = contact.countValues(Contact.TEL);
        int nameCount = contact.countValues(Contact.FORMATTED_NAME);

        // we're only interested in contacts with a phone number
        // nameCount should always be > 0 since FORMATTED_NAME is
        // mandatory
        if (telCount > 0 && nameCount > 0)
        {
          // here we use the first entry in formatted named
          String contactName =
            contact.getString(Contact.FORMATTED_NAME, 0);
          // go through all the phone numbers
          for (int i = 0; i < telCount; i++)
          {
            // check if it is a cell phone and put it at the beginning
            // this doesn't necessarily work since in many cases it is
            // up to the user to indicate whether it is a mobile phone
            int telAttributes =
              contact.getAttributes(Contact.TEL, i);
            String telNumber =
              contact.getString(Contact.TEL, i);
            // check if ATTR_MOBILE is supported
            if (contactList.
              isSupportedAttribute(Contact.TEL,
                                   Contact.ATTR_MOBILE))
            {
              if ((telAttributes & Contact.ATTR_MOBILE) != 0)
              {
                telNumbers.insertElementAt(telNumber, 0);
              }
              else
              {
                telNumbers.addElement(telNumber);
              }
            }
            else
            {
              telNumbers.addElement(telNumber);
            }
            allTelNumbers.addElement(telNumber);
```

```
            }
            // Shorten names which are too long
            if (contactName.length()>20)
            {
              contactName = contactName.substring(0, 17)
                + "...";
            }
            // insert elements in the list in order
            for (int i = 0; i < telNumbers.size(); i++)
            {
              append(contactName
                + ": "
                + telNumbers.elementAt(i), null);
            }
            telNumbers.removeAllElements();
          }
        }
        available = true;
      }
      else
      {
        append("Contact list required items not supported", null);
        available = false;
      }
    }
    finally
    {
      // always close it
      if (contactList != null)
      {
        contactList.close();
      }
    }
  }


  // load the names in a separate thread
  private class LoadContacts implements Operation
  {
    public void execute()
    {
      try
      {
        // go through all the lists
        String[] allContactLists = PIM.getInstance()
          .listPIMLists(PIM.CONTACT_LIST);
        if (allContactLists.length != 0)
        {
          for (int i = 0; i < allContactLists.length; i++)
          {
            loadNames(allContactLists[i]);
          }
          addCommand(selectCommand);
        }
        else
        {
          append("No Contact lists available", null);
          available = false;
        }
      }
      catch (PIMException e)
      {
        parent.showMessage(e.getMessage(), ContactListForm.this);
        available = false;
        append("Press a key to return", null);
      }
      catch (SecurityException e)
      {
        parent.showMessage(e.getMessage(), ContactListForm.this);
```

```
                available = false;
                append("Press a key to return", null);
            }
        }
    }
}
```

3.5 支持类

MIDlet 使用的其它支持类包括：

- `ErrorScreen`：用于报告错误消息。
- `OperationsQueue` 和 `Operation`：用于创建一个操作队列，操作的执行需要在独立的线程中完成。

4 参考文献

[JSR-075] JSR-75: PDA Optional Packages for the J2ME™ Platform, Java Community Process, 2004, http://jcp.org/aboutJava/communityprocess/final/jsr075/index.html

[MIDPPROG] *MIDP 1.0: Introduction to MIDlet Programming*, Forum Nokia, 2004, http://www.forum.nokia.com | Technologies | Java | Code and Examples

[SIGNMID] *MIDP 2.0: Tutorial On Signed MIDlets*, Forum Nokia, 2004, http://www.forum.nokia.com/documents

[MIDP 2.0] *Mobile Information Device Profile 2.0*, Java Community Process, 2002, http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html

[VCARD] *vCard The Electronic Business Card Version 2.1*, versit Consortium 1996, http://www.imc.org/pdi/vcard-21.txt

[VCALENDAR] *vCalendar The Electronic Calendaring and Scheduling Exchange Format Version 1.0*, versit Consortium 1996, http://www.imc.org/pdi/vcal-10.txt

5 文档评价

为了提高文档的质量，我们诚恳邀请您填写此调查表。