

Create BlackBerry applications with open source tools, Part 3: Building a social-networking application

Skill Level: Intermediate

[Frank Ableson](#)
Author

17 Mar 2009

The most intriguing and powerful phenomenon in the digital age is the impact of social-networking applications on the consumer and business markets. Marketing professionals are changing how they interact with prospective clients, peer groups can easily form, and, as demonstrated in the fall of 2008, social networking can be part of a successful political campaign. Combined with the popularity and power of the BlackBerry platform, you have the makings of a dynamic combination. Part 3 of this "[Create BlackBerry applications with open source tools](#)" series explores how the BlackBerry is a great device for writing and deploying social-networking applications.

Section 1. Before you start

This series explores open source and Java™ technology for BlackBerry application development in the context of a mobile data-collection application. [Part 1](#) provides an introduction to BlackBerry development with a quick introduction to the platform, a tour of the BlackBerry development tools, and construction of a complete data-collection application. [Part 2](#) explores the RSS data-distribution format by creating a BlackBerry RSS reader.

This tutorial is for Java developers interested in BlackBerry application development in the context of social-networking applications. The example demonstrates a rudimentary social-networking application that leverages the freely available BlackBerry development tools. Mobile development experience is helpful, but not required. Java programming skills are required for BlackBerry applications, but are

not an explicit requirement for this tutorial. Likewise, familiarity with social-networking concepts is helpful, but not required.

About this tutorial

Why write a social-networking application for BlackBerry? Social networking is all the rage, and people want to take this experience with them wherever they go. Many social-networking users carry BlackBerry devices. The BlackBerry platform is rich and ripe for social networking. Its capability as an Internet-accessing device is solid, its reputation as a messaging platform is legendary, and one important feature that most BlackBerry devices boast is a full keyboard. Having a camera is handy, but the power lies in a keyboard where users can rapidly type messages to friends or business associates. The BlackBerry environment also presents an open programming infrastructure where you can "catch" or hook many events of interest. These capabilities open many possibilities for feature-rich social-networking applications.

This tutorial is not singularly blazing new trails; there are already commercially available social-networking applications for the BlackBerry. This tutorial's sample application demonstrates how to build a useful open source application centered around social networking. Learn to add a custom menu into an application and implement a "Share this Event" feature that highlights interacting with the contacts database and the calendar events on the BlackBerry. Contacts and events are part of the larger, more general set of data known as *Personal Information Management (PIM)*. Once you understand how to interact with the PIM data on a device, the social-networking opportunities are limited only by the imagination.

This tutorial provides a brief introduction to PIM data, then dives into the requirements for a BlackBerry social-networking application. [Download](#) the complete source code for the example BlackBerry application.

System requirements

This tutorial demonstrates how to use BlackBerry development tools to construct an open source social-networking application for the BlackBerry. You will need the [BlackBerry Java Development Environment \(JDE\)](#) or equivalent to construct the application.

Sample code highlights

In this tutorial, a social-networking application named IBMCalendar is constructed for the BlackBerry. As you go through, try to think beyond the basics; this data may be used in the social-networking applications you use today. [Download](#) the source code. Snippets include:

ContactList

To work with a list of contacts from the PIM database.

IBMCalendar

The `Application` class, and contains the entry point of the application.

IBMCalendar constructor

Method that demonstrates how to add a menu item to the built-in BlackBerry applications.

CalendarMenus

Class containing the Menu UI element, which is added into the calendar/date-book application on the BlackBerry.

CalendarMenus.run()

Method invoked when a user selects the custom menu added to the DateBook application.

CalendarMenus.toString()

Method that sounds rather simple (and it is), but is very important. This is where the textual name, or label, of the menu is controlled.

CalendarMenus.handleCalendarEvent()

Method invoked when a date-book entry or event is manipulated by the custom menu added to the date-book application.

BlackBerryContactList

Class employed to provide a contact lookup for finding a user's e-mail address. This class demonstrates interacting with the PIM database from a high level.

Event and Contact

Classes used to extract data from the BlackBerry PIM database.

SendEmail

Class that interacts with the BlackBerry's messaging infrastructure to send an e-mail programmatically.

Section 2. PIM basics

Before jumping into the installation of the BlackBerry JDE and building the application, let's cover some PIM basics.

PIM data elements

The most common and universal data elements on a mobile device, including the BlackBerry platform, include:

Addresses or contacts

Such as your cousin's work phone number

Calendar/date-book events

Such as a lunch meeting with a prospective client next Thursday

To-do items

For example, bring home milk and eggs from the grocery store

These data elements have been with us forever, or at least since the Palm Pilot revolutionized the Personal Digital Assistant (PDA) market a few years ago. Collectively, the data elements are known as PIM items.

Every major mobile platform supports these data elements and supports the common desktop software such as Microsoft® Outlook, Lotus Notes®, and many other productivity application suites. An entire industry of synchronization software has grown up around the seemingly simple, yet difficult, task of synchronizing PIM data among various data sources. Synchronization of data between the BlackBerry device and the desktop (or server environments) is a broad topic and is not the focus of this tutorial.

This tutorial focuses on interacting with the PIM data directly on the device for two reasons:

- Many users don't actually sync their data with a desktop application. If they sync at all, it is with a Web-based social-networking application.
- More importantly, at the core of a social-networking application is your PIM data. Social networking is about connecting, sharing, and, arguably, peering into what your friends and associates are doing — right now.

Next, you'll learn about some of the packages and classes available in the BlackBerry SDK for interacting with the PIM database.

PIM data APIs

PIM data is used primarily by three applications on the BlackBerry: the address book, the calendar application, and the tasks application. With these applications, data is entered, stored, retrieved, and manipulated. Figure 1 shows a simple event

recorded for a dinner at 5 p.m.

Figure 1. Event in the BlackBerry calendar application



This event is shown in the calendar application. The summary of the event is "Dinner with In-Laws" and the location is Taco Bell. Drilling down into the event shows more information.

Figure 2. Details of an event



You can set the start and end times of the event, specify a reminder for the event, and even mark the event as recurring, among other things. This is all good, but this tutorial is more interested in how you can interact with the data programmatically, so let's look at the `Event` data class in more detail.

When working with the BlackBerry SDK/APIs for PIM data, you need to be aware of two levels of classes:

`javax.microedition.pim`

A package containing the generic PIM data. The classes in this package are found on devices beyond the BlackBerry and represent the core `PIMItem`s.

`net.rim.blackberry.api.pdap`

A package containing BlackBerry-specific extensions to the `javax.microedition.pim` classes.

The `Event` class in the `javax.microedition.pim` package extends `PIMItem`, a more basic super class. The `PIMItem` is a generic PIM data element containing a collection of data fields. `PIMItem`s may be organized into a `PIMList`, which is a collection of `PIMItem`s. The specific fields supported by a particular platform may vary and are determined by the `PIMList` in which the `PIMItem` is stored.

Data fields within a `PIMItem` may be of a variety of data types. The common data types are:

- String
- Stringarray
- Date
- Integer
- Boolean
- Binary

The `PIMItem` class includes, as you might suspect, several `getters` and `setters` for manipulating the data. Each data field includes:

- A label to describe it, such as phone number.
- Zero or more data values, which are a zero-indexed list of values. The `get` and `set` methods have an `index` parameter to assist in managing this data.
- Attributes for the data values.
- A specific data type.

Field names are identified by integer values defined in the `Contact`, `Event`, and `ToDo` classes. Table 1 shows a sampling of field names and the data type found in the `Event` class.

Table 1. Sample Event data fields

Field Name	Data Type
Summary, Location	String
Start, End	Date
Alarm	Integer

The date value is actually stored as a long integer, which is compatible with the `java.util.Date` class. This will be used in the sample application code.

Retrieving PIM data

As mentioned, `PIMItems`, such as `Events`, `Contacts`, and `ToDo`s, are stored in `PIMLists`. There may be multiple `PIMLists` available on a device, though typically, you'll encounter a single default list of PIM data. To gain access to the lists of PIM data, you must first obtain a reference to the PIM database through the `javax.microedition.pim.PIM` class: `PIM pim = PIM.getInstance();`

The static method `getInstance()` retrieves an instance of the PIM database, which is required for subsequent operations. This class also includes methods for opening existing `PIMLists` and for serializing a `PIMItem` to an input/output stream. These methods, `toSerialFormat` and `fromSerialFormat`, are typically used for backup or synchronization purposes. They are not used in this tutorial's sample application.

The sample application employs the `openPIMList` method to obtain a `PIMList` containing `PIMItems`. There are two `openPIMList` methods: one opens the default `PIMList`, and the other takes a `String` argument for a specific, named `PIMList`. You can use the `listPIMLists` method to obtain a list of available `PIMList` names. This method returns a string array containing a `PIMList` name in each array element.

The proliferation of classes and methods including the word "lists" can be a little confusing. Take care to note data types in the BlackBerry Java API [documentation](#). The sample application simply accesses the default `PIMList`. Regardless of which `openPIMList` method is employed, there are two arguments required:

pimListType

May be `CONTACT_LIST`, `EVENT_LIST`, or `TODO_LIST`

mode

May be `READ_ONLY`, `WRITE_ONLY`, or `READ_WRITE`

Once a `PIMList` is open, the application can access the elements by employing an enumeration. To examine all of the contacts within a particular `CONTACT_LIST` or `PIMList`, you could use the code in Listing 1.

Listing 1. Accessing contacts from the PIM database

```
BlackBerryPIMList contactList = (BlackBerryPIMList)
    pim.openPIMList(PIM.CONTACT_LIST, PIM.READ_ONLY);

For (Enumeration eContacts = contactList.items(); eContacts.hasMoreElements();)
{
    Contact contact = (Contact) eContacts.nextElement();
    // do something with contact
}
```

To access a field within a specific `PIMItem`, use the `getString()` method in Listing 2.

Listing 2. getString() method to access a field within a specific PIMItem

```
String emailAddress = c.getString(Contact.EMAIL, 0);
```

The PIM database also allows categorization for easy management of PIM data. PIM categories are not discussed in this tutorial, but are straightforward to implement.

At this point, you know what PIM data looks like, where to find it, and how to access it. In the next section, you'll construct a social-networking application for the BlackBerry.

Section 3. Social networking for the BlackBerry

This section examines each of the major elements of the [sample application](#), including the relevant source-code snippets. First, a little about how the application performs and what to expect in the code.

Example social-networking application

The sample application, IBMCalendar, is a bit different from the applications in the previous parts of this series. It contains no UI screens of its own and is designed to interact directly with the BlackBerry PIM applications. That's right — there are no custom UI screens. Everything is accomplished using built-in hooks and utilities to achieve the desired functions.

A successful application must be intuitive to use or your target audience simply will not use it. Have you ever seen an application loaded with features, but you didn't actually use it because it was inconvenient to start up and access? To avoid that situation, the sample application:

- Always runs. Anytime your BlackBerry starts (not that you ever turn it off, of course) the application is started. It does not have an icon in the application ribbon.
- Installs a menu into the calendar application, making our application easy to find and always available. An effective social-networking application should be intuitive and super simple to operate.
- Presents a list of friends to connect with by using a built-in method of the `BlackBerryContactList` class. The friends are simply everyone in your contacts database.

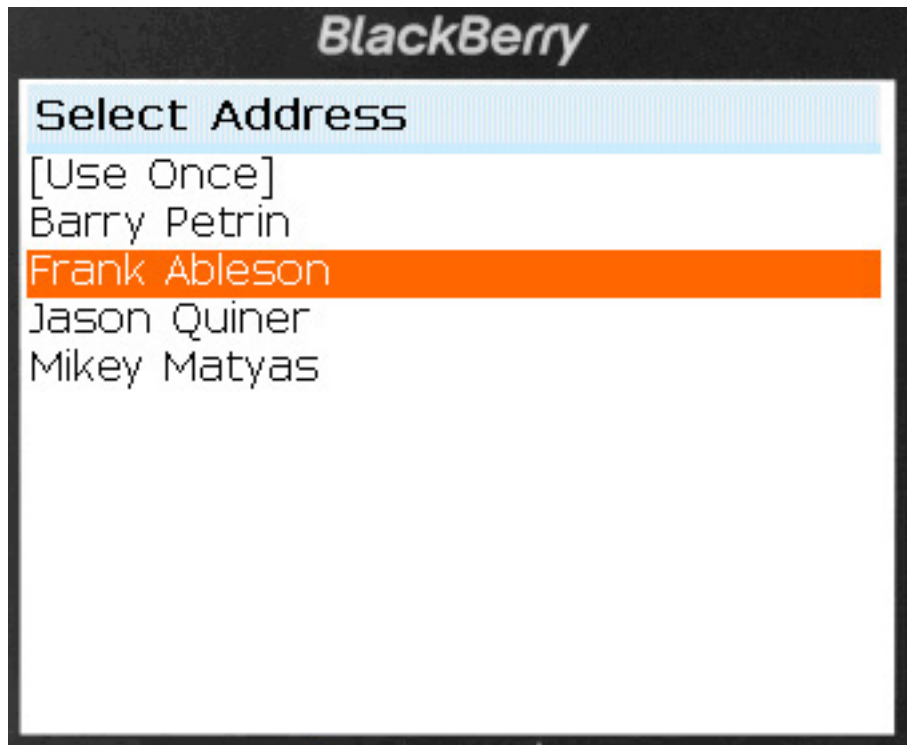
The functions of the application are simple. Whenever an event is selected in the calendar application, you can select a custom menu to **Share this Event**.

Figure 3. Sharing an event



When you choose to share the event, you get a list of your friends to share the event with.

Figure 4. Choosing a friend to share event with



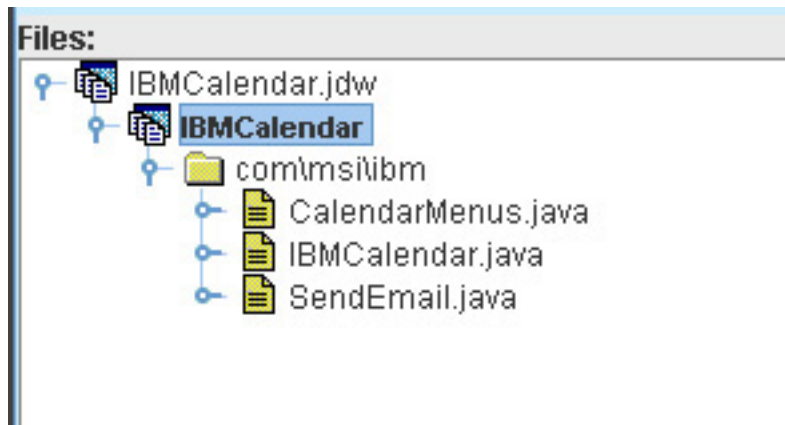
Once you've chosen a friend to share this event with, the application automatically sends an e-mail to the friend and drops the user back into the calendar application. The whole process takes no more than a moment. It's nonintrusive and simple to use.

Now that you know what the application looks like, it's time to look at the structure and code. To follow along by building the sample application, install the [BlackBerry Java Development Environment](#) if you haven't done so already.

Application structure

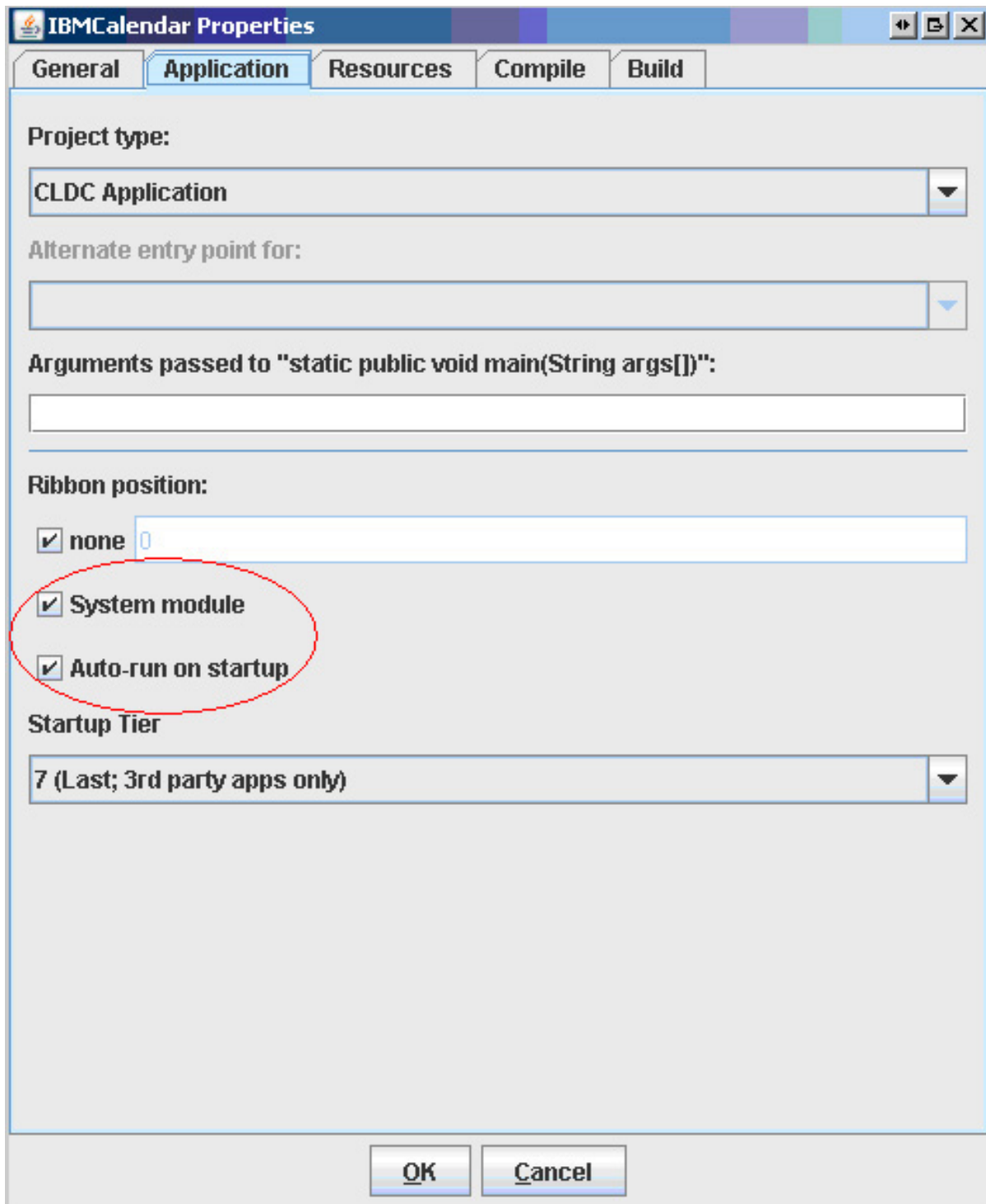
The example application will be built in parts throughout the tutorial. You can [download](#) the complete source code. Figure 5 shows the source files in use in the sample application.

Figure 5. Project file in BlackBerry JDE



Because the application does not have a custom UI, there is no class dedicated to a screen (as in previous parts of this series). To start automatically without a UI, select **System Module** and **Auto-Startup** in the project's properties dialog.

Figure 6. Project settings



In the `IBMCalendar.java` file, look at the code in Listing 3. As in any Java application, the application requires an entry point, `main`, in `IBMCalendar.java`.

Listing 3. Main method of the `IBMCalendar.java` file

```
// IBMCalendar.java
// MSI Services, Inc.
// Frank Ableson
// 973.448.0070
// fableson@msiservices.com
// code free to use for any purpose, commercial or otherwise
package com.msi.ibm;
import net.rim.device.api.ui.*;
import net.rim.blackberry.api.menuitem.*;

// our application
class IBMCalendar extends UiApplication //implements GlobalEventListener
{
    // application entry point
    public static void main(String[] args)
    {
        System.out.println("main");
        // create an instance of our app
        IBMCalendar theApp = new IBMCalendar();
        // "run" the app
        theApp.enterEventDispatcher();
    }
    // app constructor
    public IBMCalendar()
    {
        // create Menu and add it to the Calendar Application
        CalendarMenus cm = new CalendarMenus(0,"Share this Event");

        ApplicationMenuItemRepository.getInstance().addMenuItem
        (ApplicationMenuItemRepository.MENUITEM_CALENDAR,cm);
    }
}
```

The main method creates a new instance of the `IBMCalendar` class, which is an extension of the `UiApplication` class. `UiApplication` is found in the `net.rim.device.api.ui` package. The `UiApplication` class is a base class for all BlackBerry applications that have a UI.

The constructor of the `IBMCalendar` class creates an instance of the `CalendarMenus` class, which is defined and implemented in `CalendarMenus.java`. Once the instance of `CalendarMenus` is created, it is added to the calendar application with a method in the `ApplicationMenuItemRepository` class.

The BlackBerry platform offers the opportunity to add menus to built-in applications. It doesn't require nasty hacks or tricky window enumerations, as is required on other platforms. This is a basic feature of the BlackBerry SDK in the `net.rim.blackberry.api.menuitem` package.

To add a menu to an application, you must create a class that extends the `ApplicationMenuItem` class. This class typically has three methods, though ours has four to make the code a little easier to follow. The basic steps for implementing an `ApplicationMenuItem`:

1. Provide a constructor that may optionally allow the application to set up

specific data used by the `ApplicationMenuItem`. For example, this might look like passing in a custom string value you want your custom menu to display.

2. The `toString()` method returns the name you want displayed. This may be hard-coded or dynamically generated.
3. The `run` method is invoked whenever your menu is selected. It takes a single parameter of `Object` type. You can examine this `Object` to determine if you want to respond to this menu selection.

The `CalendarMenus` instance is created in Listing 4.

Listing 4: CalendarMenus instance

```
CalendarMenus cm = new CalendarMenus(0,"Share this Event");
```

The menu text for this instance, "Share this Event," can be whatever string value you want, though, obviously, you must make sure it's compact and intuitive.

Implementation of these methods in Listing 5 shows the code from `CalendarMenus.java`. We will review the major elements of this file one by one.

Listing 5. CalendarMenus.java

```
//CalendarMenus.java
// MSI Services, Inc.
// Frank Ableson
// 973.448.0070
// fableson@msiservices.com
// code free to use for any purpose, commercial or otherwise
package com.msi.ibm;
import net.rim.blackberry.api.mail.*;
import net.rim.blackberry.api.mail.event.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.blackberry.api.menuitem.*;
import java.util.Enumeration;

import javax.microedition.pim.Event;
import net.rim.blackberry.api.pdap.BlackBerryEvent;

import javax.microedition.pim.*;
import java.util.Enumeration;
import net.rim.blackberry.api.pdap.*;

class CalendarMenus extends ApplicationMenuItem
{
    String myname;
    CalendarMenus(int order,String name)
    {
        super(order);
        myname = name;
    }
}
```

```

//Run is called when the menuItem is invoked
public Object run(Object context)
{
    //context object should be an Event
    if (context instanceof Event)
    {
        try
        {
            Event evt = (Event) context;

            System.out.println("Handling our Calendar Option");
            java.lang.String s;
            handleCalendarEvent(evt);
        }
        catch (Exception ex)
        {
            System.out.println(ex);
            ex.printStackTrace();
        }
    }
    return context;
}

//toString should return the string we want to
//use as the test for the menuItem
public String toString(){
    return myname;
}

public void handleCalendarEvent(Event e)
{
    System.out.println("handling Calendar Event [" + e.toString() + "]");

    try
    {
        net.rim.blackberry.api.pdap.BlackBerryContactList contactList =
        (BlackBerryContactList) PIM.getInstance().openPIMList(PIM.CONTACT_LIST,PIM.READ_ONLY);
        Contact c = contactList.choose(null,BlackBerryContactList.
        AddressTypes.EMAIL,false);
        if (c == null)
        {
            System.out.println("no contact chosen, bail");
        }
        else
        {
            String emailAddress = c.getString(Contact.EMAIL,0);
            System.out.println(emailAddress);

            String eventData = "You're Invited!\n";
            eventData += "What: " + e.getString(Event.SUMMARY,0) + "\n";
            eventData += "Where: " + e.getString(Event.LOCATION,0) + "\n";
            eventData += "When: " + new java.util.Date(e.getDate(Event.START,0)).
            toString() + " until " + new java.util.Date(e.getDate(Event.END,0)).toString() + "\n";

            System.out.println(eventData);

            SendEmail se = new SendEmail(emailAddress,eventData);
            se.start();
        }
    }
    catch (Exception ee)
    {

```

```
        System.err.println(ee.getMessage());
        ee.printStackTrace();
    }
}
```

There are a handful of method calls to `System.out.println`. This is used for debugging purposes, as it's helpful to see a trace of the methods being invoked. This technique is even more helpful in an application that does not have a UI. To see this data in the BlackBerry JDE, view the Output window, which you can access under the **View** menu or by the key-combination of **Alt+2** in the JDE.

Starting with the constructor, notice that a parameter is passed in the order you desire the menu to appear. The example uses zero, which places the menu all the way to the top, as seen in [Figure 3](#). This value is passed to the super class. The other parameter used in this constructor is a string value that contains a name for the menu. This value is stored in the member variable `myname`. At this point, the constructor has done its job and this instance of the `CalendarMenus` class is ready.

Back in [Listing 2](#), this class is instantiated and employed in the constructor of the `IBMCalendar` class. Once the class is instantiated, it must be added to the application menu of choice, which is accomplished through the `ApplicationMenuItemRepository` class. The code gets an instance of this class from the static `getInstance` method. The `addItem` method takes two arguments:

- The location where this menu should be added
- An instance of the `CalendarMenu`

The `ApplicationMenuItemRepository` offers several possible locations where the menu may be installed, which equates to virtually all of the built-in applications. The RIM Java API Library documentation has a complete reference of the available menu locations.

When the menu is selected, the `run` method of the `CalendarMenus` class is invoked. The lone argument to this method is of type `Object`. This is interrogated at run time and, if it is found to be of type `Event`, the code processes it. In this case, the event is processed by passing it to the method `handleCalendarEvent`. The `handleCalendarEvent` method performs three main steps:

1. Asks the user to choose a contact. To do this, the PIM database is accessed and the default contacts list is opened. Once the `PIMList` is opened, the `choose` method is invoked. This method initiates a "search user interface" where the user can select a contact. The second argument, `BlackBerryContactList.AddressTypes.EMAIL`,

instructs the choose method to show contacts with e-mail addresses. If no contact is chosen, the application simply takes no further action.

2. Once a contact is chosen, the event data is extracted and formatted into a simple "What, Where, When" human-readable format. The "date" values are used as arguments to anonymous instances of `java.util.Date`. The e-mail address is extracted from the `Contact` instance.
3. Equipped with an e-mail address and a formatted body of text, the next step is to send an e-mail.

Sending e-mail

The final step for the social-networking application is the sending of a simple e-mail to a friend to share an Event of interest. To do this, the sample application implements a class named `SendEmail`, found in `SendEmail.java`.

Listing 6. Sending the e-mail

```
// SendEmail.java
// MSI Services, Inc.
// Frank Ableson
// 973.448.0070
// fableson@msiservices.com
// code free to use for any purpose, commercial or otherwise

package com.msi.ibm;

import net.rim.blackberry.api.mail.event.*;
import net.rim.blackberry.api.mail.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.component.Dialog;

/**
 *
 */
class SendEmail extends Thread
{
    boolean bCanSend = false;
    Store msgStore;
    Folder[] folderList;
    Folder outFolder;
    Message msg;
    Transport emailTransport;
    String _emailTo,_emailBody;

    SendEmail(String emailTo,String emailBody)
    {
        try
        {
            Session s = Session.getDefaultInstance();
            if(s == null)
            {
                String errMsg = "Unable to send email message.\n";
            }
        }
    }
}
```

```

        Dialog.alert(errMsg);
        bCanSend = false;
    }
    else
    {
        bCanSend = true;

        _emailTo = emailTo;
        _emailBody = emailBody;

        emailTransport = Session.waitForDefaultSession().getTransport();
        msgStore = Session.waitForDefaultSession().getStore();
        folderList = msgStore.list(Folder.SENT);
        outFolder = folderList[0];
        msg = new Message(outFolder);
    }

    }
    catch(NoSuchServiceException nse)
    {
        nse.toString();
    }
}

public void run()
{
    System.out.println("SendEmail :: running");
    if(bCanSend == true)
    {
        try
        {
            Address [] addresses = new Address[1];
            addresses[0] = new Address(_emailTo, _emailTo);
            msg.addRecipients(Message.RecipientType.TO, addresses);
            msg.setSubject("IBM Calendar Share!");
            msg.setContent(_emailBody);
            emailTransport.send(msg);
        }
        catch(Exception e)
        {
            System.out.println("Exception caught trying to send email: " +
                e.toString());
        }
    }
}
}
}
}

```

SendEmail extends `java.util.Thread`. You want the task of sending an e-mail to take place independent of any UI thread. The class is instantiated by a constructor that takes two arguments — the e-mail recipient and the formatted message. The constructor attempts to do some initialization work by:

- Connecting to the communications subsystem of the BlackBerry device.
- Getting a reference to the outbox.
- Creating a new e-mail message, of class type `Message`, within the outbox.

When the class is started, the message is addressed and the body of the e-mail is

set up and sent.

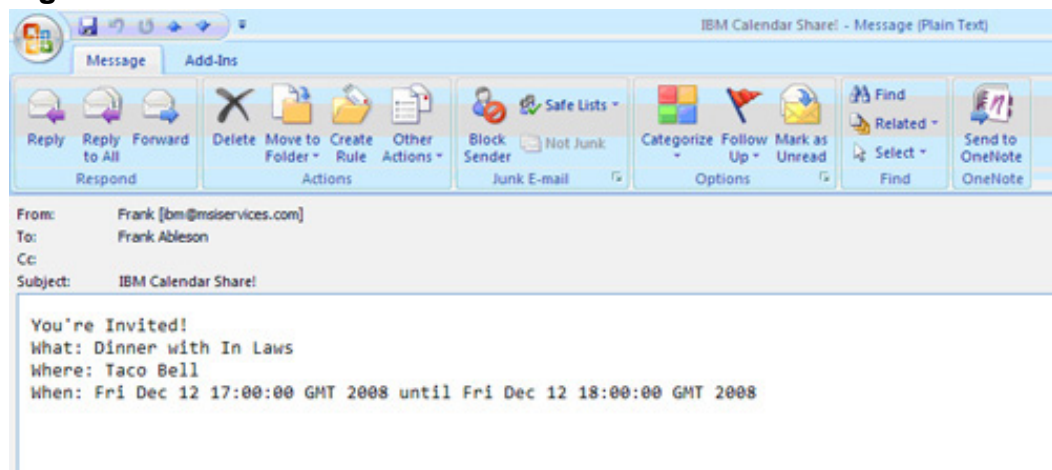
Running the application

If you are new to BlackBerry development and need help building an application in the JDE, see [Resources](#).

You have reviewed all of the important code snippets and it's time to build and test the application. Assuming the application has been built without errors, it's time to run it in the BlackBerry simulator.

1. Make sure that the MDS simulator is running. The MDS simulator allows the BlackBerry simulator to connect to the network, including the Internet.
2. Select **F5** to start the BlackBerry simulator. The tutorial sample application starts right away in the background.
3. To test the application, make sure your device has some contact entries and an event entry. While the event is highlighted in the calendar application, select the **Share this Event** menu.
4. Choose a contact. With any luck, your friend will receive a simple and friendly e-mail inviting them to join you. Figure 7 shows the e-mail invitation.

Figure 7. A shared event



5. But wait. Does your BlackBerry already have a feature called Invite Attendee? Go ahead, try it out. The recipient receives an "ics" file, which is a standard format for a meeting invitation. Unless your mail client is equipped to process that ics file, it comes across as a bunch of noninteresting text.

The social-networking application provides a starting point for more creative applications that you will write. Instead of sending an e-mail, it could make an entry to a blog, for example. The only limitation at this point is your imagination.

Section 4. Next steps

Before wrapping up, here are a few additional notes.

Application signing

Some of the programming interfaces used in the sample application require that the application be signed in order for it to run on a real BlackBerry device. For more information on how to sign an application, please see [Part 1](#) of this series.

Error handling

Error handling is omitted from this tutorial for the sake of brevity and clarity. Of course, any production-ready code should have a healthy dose of error handling and instructions for the user in the event that something goes awry.

Multiple recipients

The sample application lets you share your event with only one friend. As a straightforward enhancement, you could add multiple recipients to the e-mail invitation by repeatedly calling the `contactList.choose()` method until a null response is returned. For each valid contact received from the choose method invocation, add the contact to a vector. Then, for each contact entry, add an individual address, which is then added to the message. Presto! Multiple invites for one event.

Section 5. Summary

In this tutorial, you explored the PIM layers of the BlackBerry platform. You learned about a handy feature that lets you add your own functions to built-in BlackBerry

applications. Using the BlackBerry extension of the `PIMList`, you performed a very easy lookup to get the e-mail address of a friend to invite to something taking place on your calendar.

Leveraging the open source technology of Java technology and the ubiquity of PIM data into a social-networking application for the most visible and revered mobile platform has great potential for bringing value to business and consumer applications alike. Who knows, maybe your social-networking application is the next Facebook or Twitter.

Downloads

Description	Name	Size	Download method
IBM Calendar source code	os-blackberry3-IBMCalendar.zip	16KB	HTTP

[Information about download methods](#)

Resources

Learn

- Check out [Part 1](#) of this series, which lays the groundwork for an open source data-collection application upon which an accessible and easy-to-use data collection service is built. And don't miss [Part 2](#), which explores the RSS data-distribution format by creating a BlackBerry RSS reader suitable for taking news wherever you and your BlackBerry go.
- [BlackBerry Desktop Software](#): Research In Motion (RIM) offers a full range of helpful user and administrator information on BlackBerry Desktop Software, including more about installing software via the Desktop Manager.
- Learn more about [Personal Information Management data](#) with Java technology specifications.
- You'll find articles about every aspect of Java programming in the developerWorks [Java technology zone](#).
- [OpenSource.org](#) presents a catalog of the most popular open source licenses.
- Learn more about the [history of RSS](#) from Harvard Law.
- To listen to interesting interviews and discussions for software developers, check out [developerWorks podcasts](#).
- Stay current with developerWorks' [Technical events and webcasts](#).
- Follow [developerWorks on Twitter](#).
- Check out upcoming conferences, trade shows, webcasts, and other [Events](#) around the world that are of interest to IBM open source developers.
- Visit the developerWorks [Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Watch and learn about IBM and open source technologies and product functions with the no-cost [developerWorks On demand demos](#).

Get products and technologies

- Get the [BlackBerry Java Development Environment](#).
- Visit [BlackBerry Code Signing Keys Order Form](#) for a developer key to access Controlled APIs within the BlackBerry environment.
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.
- Download [IBM product evaluation versions](#), and get your hands on application

development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

About the author

Frank Ableson

After his college basketball career came to an end without a multiyear contract to play for the L.A. Lakers, Frank Ableson shifted his focus to computer software design. He enjoys solving complex problems, particularly in the areas of communications and hardware interfacing. When not working, he can be found spending time with his wife Nikki and their children. You can reach Frank at frank@cfigsolutions.com.