



# BlackBerry Java Development Environment

Version 4.6.0

BlackBerry Device Applications Integration Guide

BlackBerry Java Development Environment Version 4.6.0 BlackBerry Device Applications Integration Guide

Last modified: 14 July 2008

Part number: 15497807

At the time of publication, this documentation is based on the BlackBerry Java Development Environment Version 4.6.0.

Send us your comments on product documentation: <https://www.blackberry.com/DocsFeedback>.

©2008 Research In Motion Limited. All rights reserved. BlackBerry®, RIM®, Research In Motion®, SureType® and related trademarks, names, and logos are the property of Research In Motion Limited and are registered and/or used as trademarks in the U.S., Canada, and countries around the world.

Bluetooth is a trademark of Bluetooth SIG. Java and JavaScript are trademarks of Sun Microsystems, Inc. Microsoft, ActiveX, Internet Explorer, and Windows are trademarks of Microsoft Corporation. iCal is a trademark of Apple Computer, Inc. vCard is a trademark of the Internet Mail Consortium. Plazmic is a trademark of Plazmic Inc. Wi-Fi and 802.11, 802.11a, 802.11b, 802.11g are trademarks of the Wi-Fi Alliance. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The BlackBerry device and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit [www.rim.com/patents.shtml](http://www.rim.com/patents.shtml) for a list of RIM (as hereinafter defined) patents.

This document is provided "as is" and Research In Motion Limited and its affiliated companies ("RIM") assume no responsibility for any typographical, technical or other inaccuracies in this document. In order to protect RIM proprietary and confidential information and/or trade secrets, this document may describe some aspects of RIM technology in generalized terms. RIM reserves the right to periodically change information that is contained in this document; however, RIM makes no commitment to provide any such changes, updates, enhancements or other additions to this document to you in a timely manner or at all. RIM MAKES NO REPRESENTATIONS, WARRANTIES, CONDITIONS OR COVENANTS, EITHER EXPRESS OR IMPLIED (INCLUDING WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OF FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, MERCHANTABILITY, DURABILITY, TITLE, OR RELATED TO THE PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE REFERENCED HEREIN OR PERFORMANCE OF ANY SERVICES REFERENCED HEREIN). IN CONNECTION WITH YOUR USE OF THIS DOCUMENTATION, NEITHER RIM NOR ITS RESPECTIVE DIRECTORS, OFFICERS, EMPLOYEES OR CONSULTANTS SHALL BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER BE THEY DIRECT, ECONOMIC, COMMERCIAL, SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY OR INDIRECT DAMAGES, EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, INCLUDING WITHOUT LIMITATION, LOSS OF BUSINESS REVENUE OR EARNINGS, LOST DATA, DAMAGES CAUSED BY DELAYS, LOST PROFITS, OR A FAILURE TO REALIZE EXPECTED SAVINGS.

This document might contain references to third party sources of information, hardware or software, products or services and/or third party web sites (collectively the "Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third-Party Information. The inclusion of Third-Party Information in this document does not imply endorsement by RIM of the Third Party Information or the third party in any way. Installation and use of Third Party Information with RIM's products and services may require one or more patent, trademark or copyright licenses in order to avoid infringement of the intellectual property rights of others. Any dealings with Third Party Information, including, without limitation, compliance with applicable licenses and terms and conditions, are solely between you and the third party. You are solely responsible for determining whether such third party licenses are required and are responsible for acquiring any such licenses relating to Third Party Information. To the extent that such intellectual property licenses may be required, RIM expressly recommends that you do not install or use Third Party Information until all such applicable licenses have been acquired by you or on your behalf. Your use of Third Party Information shall be governed by and subject to you agreeing to the terms of the Third Party Information licenses. Any Third Party Information that is provided with RIM's products and services is provided "as is". RIM makes no representation, warranty or guarantee whatsoever in relation to the Third Party Information and RIM assumes no liability whatsoever in relation to the Third Party Information even if RIM has been advised of the possibility of such damages or can anticipate such damages.

Research In Motion Limited  
295 Phillip Street  
Waterloo, ON N2L 3W8  
Canada

Published in Canada

Research In Motion UK Limited  
Centrum House, 36 Station Road  
Egham, Surrey TW20 9LF  
United Kingdom



# Contents

<b>1</b>	<b>Adding custom menu items to applications</b>	<b>8</b>
	Adding menu items to BlackBerry Java Applications	8
	Create a menu item	8
	Register a menu item	9
<b>2</b>	<b>Integrating with BlackBerry applications</b>	<b>11</b>
	Invoke BlackBerry applications	11
<b>3</b>	<b>Using the message list</b>	<b>13</b>
	Create new messages	13
	Work with a message	15
	Open a message	16
	Retrieve the body of a message without an attachment	17
	Retrieve the body of a message with an attachment	17
	Code sample: Retrieve the body of a message	18
	Notify an application that an email message is about to be sent	20
	Notify an application that an MMS message is about to be sent	20
	Notify an application that an SMS message is about to be sent	20
	Send a message	21
	Reply to a message	22
	Forward a message	23
	Work with folders	23
	Working with attachments	25
	Create an attachment handler	25
	Retrieve attachments	25
	Send a message with an attachment	26
<b>4</b>	<b>Using PIM applications</b>	<b>27</b>
	Using the calendar	27
	Start the calendar from your BlackBerry Java Application	27
	Use the calendar	28
	Notify an application when a list of events changes	31
	Notify an application when the default list of events on a BlackBerry device changes	31

Retrieve multiple lists of events .....	32
Using the address book .....	32
Open the address book from your BlackBerry Java Application .....	32
Use contacts.....	32
Notify an application when a list of contacts changes.....	38
Using tasks.....	39
Start the task application from your BlackBerry Java Application .....	39
Use tasks.....	40
Notify an application when a list of tasks changes .....	43
Code samples.....	43
Code sample: Creating new recurring appointments .....	43
Code sample: Displaying a screen that lets BlackBerry device users add new contacts .....	45
Code sample: Using tasks .....	48
<b>5 Using the phone application .....</b>	<b>51</b>
Start the phone application from your BlackBerry Java Application .....	51
Use phone call functionality .....	51
Add DTMF tones to the send queue .....	52
Listen for phone events.....	53
Access and use call logs.....	53
Code sample .....	54
Code sample: Calculating the time spent on the phone by a participant .....	54
<b>6 Using the BlackBerry Browser .....</b>	<b>57</b>
Display content in the BlackBerry Browser .....	57
Display content in a BlackBerry Browser field .....	57
Code sample .....	60
Code sample: Using the BlackBerry Browser .....	60



# Adding custom menu items to applications

## Adding menu items to BlackBerry Java Applications

### Adding menu items to BlackBerry Java Applications

The Application Menu Item API, in the `net.rim.blackberry.api.menuitem` package, lets you add menu items to BlackBerry® Java® Applications. The `ApplicationMenuItemRepository` class lets you add or remove menu items from BlackBerry Java Applications.

#### Create a menu item

Task	Steps
Define a menu item.	<ul style="list-style-type: none"> <li>&gt; Extend the abstract <code>ApplicationMenuItem</code> class.</li> </ul> <pre>public class SampleMenuItem extends ApplicationMenuItem { ... }</pre>
Specify the position of the menu item in the menu.	<p>A higher number means that the menu item appears lower in the menu.</p> <ul style="list-style-type: none"> <li>&gt; Invoke <code>ApplicationMenuItem()</code></li> </ul> <pre>SampleMenuItem() {     super(20); }</pre>
Specify the menu item text.	<ul style="list-style-type: none"> <li>&gt; Implement <code>toString()</code>.</li> </ul> <pre>public String toString() {     return "Open the Contacts Demo application"; }</pre>
Specify the behaviour of the menu item.	<ul style="list-style-type: none"> <li>&gt; Implement <code>run()</code>.</li> </ul> <pre>public Object run(Object context) {     Contact c = (Contact)context; // An error occurs if this does not work.     if ( c != null ) {         new ContactsDemo().enterEventDispatcher();     } else {         throw new IllegalStateException( "Context is null, expected a Contact instance");     }     Dialog.alert("Viewing a message in the messaging view");     return null; }</pre>

## Register a menu item

Task	Steps
Retrieve the BlackBerry® MDS Java Application menu item repository.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>ApplicationMenuItemRepository.getInstance()</code>.</li> </ul> <pre>ApplicationMenuItemRepository repository = ApplicationMenuItemRepository.getInstance();</pre>
Create your BlackBerry Java Application menu item.	<ul style="list-style-type: none"> <li>&gt; Invoke the constructor.</li> </ul> <pre>ContactsDemoMenuItem contactsDemoMenuItem = new ContactsDemoMenuItem();</pre>
Add the menu item to the repository.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>ApplicationMenuItemRepository.addItem()</code>.</li> </ul> <pre>repository.addItem(ApplicationMenuItemRepository.MENUITEM_ADDRESSCARD_VIEW, contactsDemoMenuItem);</pre>
Add the menu item to BlackBerry® Maps.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>ApplicationMenuItemRepository.addItem()</code> using the <code>MENUITEM_MAPS</code> field.</li> </ul> <pre>repository.addItem(ApplicationMenuItemRepository.MENUITEM_MAPS, contactsDemoMenuItem);</pre>



# Integrating with BlackBerry applications

## Invoke BlackBerry applications

### Invoke BlackBerry applications

You can create BlackBerry® Java® Applications that invoke BlackBerry device applications such as the email, calendar, phone, maps, browser, and camera applications to perform an action or display information. When the third-party BlackBerry Java Application invokes the BlackBerry device application, the third-party BlackBerry Java Application can make the BlackBerry device application perform an action or display information.

Task	Steps
Start the message application and create a new blank message.	<ul style="list-style-type: none"> <li>&gt; Invoke the <code>invokeApplication()</code> method with the following parameters:               <ul style="list-style-type: none"> <li>• <code>Invoke.APP_TYPE_MESSAGES</code>: a constant parameter</li> <li>• <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_SMS</code> parameter</li> </ul> </li> </ul> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW_SMS));</pre>
Start the calendar.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)</code></li> </ul>
Start the phone application.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>Invoke.invokeApplication(APP_TYPE_PHONE, PhoneArguments)</code>.</li> </ul>
Start the maps application and display the default map view.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>invokeApplication()</code> using a new <code>MapsArguments</code> object.</li> </ul> <pre>Invoke.invokeApplication( Invoke.APP_TYPE_MAPS, new MapsArguments() );</pre>

See the *API Reference* for more information about using the `net.rim.blackberry.api.invoke.Invoke` class to invoke BlackBerry device applications.

#### Related topics

- See “Create new messages” on page 13 for more information.
- See “Open the address book from your BlackBerry Java Application” on page 32 for more information.
- See “Start the task application from your BlackBerry Java Application” on page 39 for more information.



# Using the message list

Create new messages  
 Work with a message  
 Work with folders  
 Working with attachments

## Create new messages

Task	Steps
Create a new blank SMS text message.	<p>&gt; Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_SMS</code> parameter.</p> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW_SMS));</pre>
Create a new populated text message.	<p>Use the API items in the <code>javax.wireless.messaging</code> package (JSR 120).</p> <ol style="list-style-type: none"> <li>1. Create and populate a new <code>TextMessage</code> object. <pre>MessageConnection mc = (MessageConnection)Connector.open( "sms://" ); TextMessage m = (TextMessage)mc.newMessage( MessageConnection.TEXT_MESSAGE ); m.setAddress( "sms://5558888" ); m.setPayloadText( "An SMS Message for you" );</pre> </li> <li>2. Invoke <code>invokeApplication()</code> with the following parameters: <ul style="list-style-type: none"> <li>• <code>APP_TYPE_MESSAGES</code>: a constant parameter</li> <li>• <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the new <code>TextMessage</code> object.</li> </ul> <pre>Invoke.invokeApplication( Invoke.APP_TYPE_MESSAGES, new MessageArguments( m ) );</pre> </li> </ol>
Create a new SMS text message with multimedia.	<p>&gt; Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_MMS</code> parameter.</p> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW_MMS));</pre>
Create a new blank email message.	<p>&gt; Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW</code> parameter.</p> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW));</pre>

Task	Steps
Create a new populated email message.	<ol style="list-style-type: none"> <li data-bbox="485 230 1278 633">           1. Create and populate a new email message object.           <pre data-bbox="521 260 1278 460"> net.rim.blackberry.api.mail.Message m = new net.rim.blackberry.api.mail.Message(); Address a = new Address("mLi@rim.com", "Ming Li"); Address[] addresses = {a}; m.addRecipients(net.rim.blackberry.api.mail.Message.RecipientType.TO, addresses); m.setContent("A message for you..."); m.setSubject("Email for you"); </pre> </li> <li data-bbox="485 468 1278 633">           2. Invoke <code>invokeApplication()</code> with the following parameters:           <ul data-bbox="521 494 1278 581" style="list-style-type: none"> <li>• <code>APP_TYPE_MESSAGES</code>: a constant parameter</li> <li>• <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the new email <code>Message</code> object.</li> </ul> <pre data-bbox="521 581 1278 633"> Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(m)); </pre> </li> </ol>
Create a new blank PIN message.	<p data-bbox="485 638 1278 685">&gt; Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses the <code>ARG_NEW_PIN</code> parameter.</p> <pre data-bbox="485 694 1278 743"> Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( MessageArguments.ARG_NEW_PIN)); </pre>
Create a new populated PIN message.	<ol style="list-style-type: none"> <li data-bbox="485 748 1278 980">           1. Create and populate a new PIN message.           <pre data-bbox="521 781 1278 980"> net.rim.blackberry.api.mail.Message m = new net.rim.blackberry.api.mail.Message(); PINAddress pa = new PINAddress("ABCDEF99", "Mark Chapters"); Address[] addresses = {pa}; m.addRecipients( net.rim.blackberry.api.mail.Message.RecipientType.TO, addresses ); m.setContent("A message for you..."); m.setSubject("PIN message for you"); </pre> </li> <li data-bbox="485 989 1278 1121">           2. Invoke <code>invokeApplication()</code> with the following parameters:           <ul data-bbox="521 1015 1278 1067" style="list-style-type: none"> <li>• <code>APP_TYPE_MESSAGES</code>: a constant parameter</li> <li>• <code>MessageArguments</code>: a new <code>MessageArguments</code> object that uses the new PIN message.</li> </ul> <pre data-bbox="521 1076 1278 1121"> Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments(m)); </pre> </li> </ol>

## Work with a message

Task	Steps
Receive a message notification.	<ol style="list-style-type: none"> <li>1. Implement the <code>FolderListener</code> and <code>StoreListener</code> interfaces.  <pre>public class MailTest implements FolderListener, StoreListener { ... }</pre> </li> <li>2. Create code to manage a <code>ControlledAccessException</code>.</li> </ol>
Add a listener to the message store.	<ol style="list-style-type: none"> <li>1. Retrieve the <code>Store</code> object.</li> <li>2. Add a <code>StoreListener</code> instance to it.</li> <li>3. Create a try-catch block to manage a <code>NoSuchServiceException</code>.  <pre>try {     Store store = Session.waitForDefaultSession().getStore(); } catch (NoSuchServiceException e) {     System.out.println(e.toString()); } store.addStoreListener(this);</pre> </li> </ol>
Add a listener to the message store for batch updates.	<pre>&gt; Implement StoreListener.batchOperation(). void batchOperation(StoreEvent e) {     // Perform action when messages added or removed in batch operation. }</pre>
Add a listener to a folder.	<ol style="list-style-type: none"> <li>1. Retrieve the <code>Folder</code> object for which you want to receive new message notifications.  <pre>Folder[] folders = store.list(Folder.INBOX); Folder inbox = folders[0];</pre> </li> <li>2. Add the <code>FolderListener</code> instance to the folder.  <pre>inbox.addFolderListener(this);</pre> </li> <li>3. Implement <code>FolderListener.messagesAdded()</code> and <code>FolderListener.messagesRemoved()</code>.  <pre>void messagesAdded(FolderEvent e) {     // Perform processing on added messages. } void messagesRemoved(FolderEvent e) {     // Perform processing on removed messages. }</pre> </li> </ol>
Retrieve the total count of unread messages in all folders in the store.	<pre>&gt; Invoke net.rim.blackberry.api.mail.Store.getUnreadMessageCount(). int numUnread = store.getUnreadMessageCount();</pre>

Task	Steps
Get more of a message.	<p>By default, the first section of a message (typically about 2 KB) is sent to the BlackBerry® device.</p> <ol style="list-style-type: none"> <li>1. Create an instance of a subclass of the <code>BodyPart</code> abstract class.  <pre>TextBodyPart tb = new TextBodyPart(new MultiPart());</pre> </li> <li>2. To determine if more data for a body part is available on the server, invoke <code>tb.hasMore()</code>.</li> <li>3. To determine if the BlackBerry device user made a request for more data, invoke <code>tb.moreRequestSent()</code>.</li> <li>4. To obtain a <code>Transport</code> object, invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>.  <pre>Transport trans = Session.getTransport();</pre> </li> <li>5. To request more of a message, invoke <code>trans.more(BodyPart bp, boolean reqAll)</code>. The second parameter of <code>more()</code> is a Boolean value that specifies whether to retrieve only the next section of the body part (false) or all remaining sections of the body part (true).  <pre>if (( tb.hasMore() ) &amp;&amp; (! tb.moreRequestSent())) {trans.more(tb, true);} </pre> </li> </ol>

## Open a message

1. Retrieve the message store and the folder that contains the message.

```
Store store = Session.waitForDefaultSession().getStore();
Folder folder = Store.getFolder("SampleFolder");
```

2. Retrieve the message objects from the folder. Iterate through the array and retrieve information, such as the sender and subject, to display to the BlackBerry® device user.

```
Message[] msgs = folder.getMessages();
```

3. When a BlackBerry device user selects a message from the list, invoke methods on the `Message` object to retrieve the appropriate fields and body contents to display to the BlackBerry device user.

```
Message msg = msgs[0]; // Retrieve the first message.
Address[] recipients = msg.getRecipients(Message.RecipientType.TO)
Date sent = msg.getSentDate();
Address from = msg.getFrom();
String subject = msg.getSubject();
Object o = msg.getContent();
// Verify that the message is not multipart.
if ( o instanceof String ) {
String body = (String)o;} //...
```

4. Invoke `getBodyText()` on a message to retrieve the plain text contents as a `String`. If the message does not contain plain text, the method returns null.

## Retrieve the body of a message without an attachment

1. Invoke `Message.getContent()`.  

```
Object obj=message.getContent();
```
2. If the content of the message is a `Multipart` object, cast the message content as a `Multipart` object.  

```
if(obj instanceof Multipart){
    Multipart mp=(Multipart)obj;
}
```
3. If the content of the message is of type `multipart/alternative`, iterate through the `BodyPart` objects in the `Multipart` object, storing each reference to a `BodyPart` object in a `BodyPart` variable.  

```
if(mp.getContentType().equals(ContentType.MULTIPART_ALTERNATIVE){
    for(int i=0;i<mp.getCount();i++){
        BodyPart bp=mp.getBodyPart(i);
```
4. If a `BodyPart` object is of type `MimeBodyPart`, store the content type of the `BodyPart` object in a `String` variable.  

```
if(bp instanceof MimeBodyPart){
    String type=bp.getContentType();
```
5. Check if the content type of the `BodyPart` object is `text/html`.  

```
if(type.equals(ContentType.TEXT_HTML_STRING){
```
6. Check if the content type of the `BodyPart` object is `TextBodyPart`.  

```
}else if(bp instanceof TextBodyPart){
```

## Retrieve the body of a message with an attachment

1. Invoke `Message.getContent()`.  

```
Object obj=message.getContent();
```
2. Cast the message content as a `Multipart` object.  

```
Multipart mp=(Multipart)obj;
```
3. If the content of the message is of type `multipart/mixed`, iterate through `BodyPart` objects in the `Multipart` object, storing each reference to a `BodyPart` object in a `BodyPart` variable.  

```
if(mp.getContentType().equals(ContentType.MULTIPART_MIXED){
    for(int i=0;i<mp.getCount();i++){
        BodyPart bp = mp.getBodyPart(i);
```

4. If the `BodyPart` object is of type `MimeBodyPart`, check if the content type of the `Multipart` object is `MULTIPART_ALTERNATIVE`.

```
    if(bp instanceof MimeBodyPart){
        MimeBodyPart mbp=(MimeBodyPart)bp;
        Multipart mp=(Multipart)mbp.getContent();

        if(mp.getcontentType().equals(ContentType.MULTIPART_ALTERNATIVE){
```

5. If the content type of the `Multipart` object is `MULTIPART_ALTERNATIVE`, store a reference to a `BodyPart` object in a `BodyPart` variable.

```
        BodyPart xp=mp.getBodyPart(i);
```

6. If a `BodyPart` object is of type `MimeBodyPart`, store the content type of the `BodyPart` object in a `String` variable.

```
        if(xp instanceof MimeBodyPart){
            String type=xp.getcontentType();
```

7. Check if the content type of the `BodyPart` object is `text_html`.

```
        if(type.equals(ContentType.TYPE_TEXT_HTML_STRING){
            // HTML text
        }
```

8. Check if the content type of the `BodyPart` is `TextBodyPart`.

```
        }else if(xp instanceof TextBodyPart){
            // Plain text
        } //end else if
```

## Code sample: Retrieve the body of a message

```
Object obj=message.getContent();
if(obj instanceof Multipart){
    Multipart mp=(Multipart)obj;
    if(mp.getcontentType().equals(ContentType.MULTIPART_ALTERNATIVE){

        for(int i=0;i<mp.getCount();i++){

            BodyPart bp=mp.getBodyPart(i);

            if(bp instanceof MimeBodyPart){
```

```

String type=bp.getcontentType();

    if(type.equals(ContentType.TYPE_TEXT_HTML_STRING){
        // HTML text
    }
    else if( bp instanceof TextBodyPart){
        // Plain text
    } //end else if
} //end if
} // end for loop

}end if
else if(mp.getContentType().equals(ContentType.MULTIPART_MIXED){
    for(int i=0;i<mp.getCount();i++){
        BodyPart bp =mp.getBodyPart(i);
        if(bp instanceof MimeBodyPart){
            MimeBodyPart mbp=(MimeBodyPart)bp;

            Multipart mp=(Multipart)mbp.getContent();
            if(mp.getcontentType().equals(ContentType.MULTIPART_ALTERNATIVE){

                BodyPart xp=mp.getBodyPart(i);
                if(xp instanceof MimeBodyPart){
                    String type=xp.getcontentType();

if(type.equals(ContentType.TYPE_TEXT_HTML_STRING){
                    // HTML text
                }
                else if(xp instanceof TextBodyPart){
                    // Plain text
                } //end else if
            } //end if
        } //end if
    } //end if
} //end if

```

```
                }//end if
            }//end for loop
} //end else if
```

## Notify an application that an email message is about to be sent

1. Create a class that implements `net.rim.blackberry.api.mail.SendListener`.

```
public class mailSendListener implements SendListener{
```

2. Create an instance of the subclass.

```
mailSendListener mailSL = new mailSendListener();
```

3. In a try block, retrieve the `Store` object.

```
try {
    Store store = Session.waitForDefaultSession().getStore();
}
```

4. In a catch block, manage a `NoSuchServiceException`.

```
catch (NoSuchServiceException e) {
    System.out.println(e.toString());
}
```

5. Add a `SendListener` instance.

```
store.addSendListener(mailSL);
```

## Notify an application that an MMS message is about to be sent

1. Create a class that implements `net.rim.blackberry.api.mms.SendListener`.

```
public class mmsSendListener implements SendListener{
```

2. Create an instance of the subclass.

```
mmsSendListener mmsSL = new mmsSendListener();
```

3. Add a `SendListener` instance.

```
MMS.addSendListener(mmsSL);
```

## Notify an application that an SMS message is about to be sent

1. Create a class that implements `net.rim.blackberry.api.sms.SendListener`.

```
public class smsSendListener implements SendListener{
```

2. Create an instance of the subclass.

```
smsSendListener smsSL = new smsSendListener();
```

3. Add a `SendListener`.

```
SMS.addSendListener(smsSL);
```

## Send a message

Task	Steps
Create a message.	<ol style="list-style-type: none"> <li>1. Create a Message object.</li> <li>2. Specify a folder in which to save a copy of the sent message.</li> </ol> <pre>Store store = Session.getDefaultInstance().getStore(); Folder[] folders = store.list(Folder.SENT); Folder sentfolder = folders[0]; Message msg = new Message(sentfolder);</pre>
Specify the recipients.	<ol style="list-style-type: none"> <li>1. Create an array of Address objects.</li> <li>2. Add each address to the array.</li> <li>3. Create code to catch an AddressException, which is thrown if an address is invalid.</li> </ol> <pre>Address toList[] = new Address[1]; try { toList[0]= new Address("clyde.warren@blackberry.com", "Clyde Warren"); } catch(AddressException e) { System.out.println(e.toString()); }</pre>
Add the recipients.	<ol style="list-style-type: none"> <li>1. Invoke Message.addRecipients() and provide the type of recipient (TO, CC, or BCC) and the array of addresses to add as parameters to the method.</li> <li>2. If the message has multiple types of recipients, invoke addRecipients() once for each recipient type.</li> </ol> <pre>msg.addRecipients(Message.RecipientType.TO, toList);</pre>
Specify the name and email address of the sender.	<p>&gt; Invoke setFrom(Address).</p> <pre>Address from = new Address("clyde.warren@blackberry.com", "Clyde Warren"); msg.setFrom(from);</pre>
Add a subject line.	<p>&gt; Invoke setSubject(String).</p> <pre>msg.setSubject("Test Message");</pre>
Specify the message contents.	<p>&gt; Invoke setContent(String). Typically, the BlackBerry® Java® Application retrieves content from text that a BlackBerry device user types in a field.</p> <pre>try { msg.setContent("This is a test message."); } catch(MessagingException e) { System.out.println(e.getMessage()); }</pre>
Send the message.	<ol style="list-style-type: none"> <li>1. Invoke Session.getTransport() and store the returned object in a variable of type Transport. The Transport object represents the messaging transport protocol.</li> <li>2. Invoke trans.send(Message).</li> </ol> <pre>Transport trans = Session.getTransport(); try { trans.send(msg); } catch(MessagingException e) { System.out.println(e.getMessage()); }</pre>

## Reply to a message

Task	Steps
Reply to an existing message.	<ol style="list-style-type: none"><li>1. Invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. The <code>Transport</code> object represents the messaging transport protocol. <pre>Transport trans = Session.getTransport();</pre></li></ol> <p>&gt; Invoke <code>Message.reply( Boolean )</code> and specify <code>true</code> to reply to all message recipients or <code>false</code> to reply to only the sender.</p> <pre>Store store = Session.waitForDefaultSession().getStore(); Folder[] folders = store.list(INBOX); Folder inbox = folders[0]; Message[] messages = inbox.getMessages(); if( messages.length &gt; 0 ) {     Message msg = messages[0]; } Message reply = msg.reply(true); trans.send(reply);</pre>

## Forward a message

Task	Steps
Create a message object.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>forward()</code> on an existing <code>Message</code> object. The subject line of a forwarded message is set automatically to <code>FW:original_subject</code>.</li> </ul> <pre>Message fwdmsg = msg.forward();</pre>
Add the recipients.	<ol style="list-style-type: none"> <li>1. Create an array of addresses. <pre>Address toList[] = new Address[1];</pre> </li> <li>2. Invoke <code>addRecipients(int, Address[])</code>. <pre>toList[0]= new Address("clyde.warren@blackberry.com", "Clyde Warren"); fwdmsg.addRecipients(Message.RecipientType.TO, toList);</pre> </li> </ol>
Specify that the message content appears before the original message.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>setContent(String)</code>. <pre>try { fwdmsg.setContent("This is a forwarded message."); } catch(MessagingException e) { System.out.println(e.getMessage()); }</pre> </li> </ul>
Send the message.	<ol style="list-style-type: none"> <li>1. Invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. The <code>Transport</code> object represents the messaging transport protocol. <pre>Transport trans = Session.getTransport();</pre> </li> <li>2. Invoke <code>trans.send(Message)</code>. <pre>try { trans.send(fwdmsg); } catch(MessagingException e) { System.out.println(e.getMessage()); }</pre> </li> </ol>

## Work with folders

1. Invoke `getStore()` on the default session.

```
Store store = Session.waitForDefaultSession().getStore();
```

## 2. Complete any of the following actions:

Task	Steps
Open a folder view.	<ol style="list-style-type: none"> <li>1. Invoke <code>store.list()</code> to retrieve a list of folders.  <pre>Store store = null; store = Session.waitForDefaultSession().getStore(); Folder[] folders = store.list();</pre> </li> <li>2. Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_MESSAGES</code> constant parameter and a new <code>MessageArguments</code> object that uses a folder from the list of folders as a parameter.  <pre>Invoke.invokeApplication(Invoke.APP_TYPE_MESSAGES, new MessageArguments( folders[0]));</pre> </li> </ol>
List the folders in a mailbox store.	<pre>&gt; Invoke Store.list(). Folder[] folders = store.list();</pre>
Retrieve an array of folders by type.	<pre>&gt; Invoke list(int) and provide as a parameter the folder type. Folder[] folders = store.list(INBOX); Folder inbox = folders[0];</pre>
Retrieve an array of folders through a search.	<pre>&gt; Invoke findFolder(String). Folder[] folders = store.findFolder("Inbox");</pre>
Retrieve a folder by its name.	<ol style="list-style-type: none"> <li>1. Invoke <code>getFolder(String)</code> and provide as a parameter the absolute path to the folder.  <pre>Folder folder = store.getFolder("Mailbox - Aisha Wahl/Inbox/Projects");</pre> </li> <li>2. Create code to manage a <code>FolderNotFoundException</code> exception if the folder does not exist.</li> </ol>
Retrieve a folder by its ID.	<ol style="list-style-type: none"> <li>1. Invoke <code>getID()</code> to retrieve the folder ID.</li> <li>2. Invoke <code>getFolder()</code> with the ID as a parameter.  <pre>Folder[] folders = store.list(); long id = folders[0].getId(); Folder f2 = store.getFolder(id);</pre> </li> </ol>
File a message.	<pre>&gt; Invoke appendMessage(Message) on a Folder object. Message msg = new Message(); //... Folder folder = store.getFolder("Inbox"); folder.appendMessage(msg);</pre>

## Working with attachments

To open incoming message attachments and create outgoing attachments on the BlackBerry® device, use the mail API. A separate `BodyPart` on a `Multipart` message represents a message attachment.

### Create an attachment handler

The BlackBerry® Enterprise Server Attachment Service receives all attachments first. Third-party attachment handlers cannot override the default BlackBerry device behavior. See the *BlackBerry Enterprise Server Maintenance and Troubleshooting Guide* for more information about the BlackBerry Enterprise Server Attachment Service.

Task	Steps
Define a custom attachment handler.	> Implement the <code>AttachmentHandler</code> interface.
Register the accepted MIME types when the BlackBerry® device receives an attachment.	> Implement <code>supports(String)</code> . <pre>public boolean supports(String contentType) {     return (contentType.toLowerCase().indexOf("contenttype") != -1 ? true : false); }</pre>
Define the associated menu item string to display in the messages list when the BlackBerry device user selects an attachment.	> Implement <code>menuString()</code> . <pre>public String menuString() {     return "Custom Attachment Viewer"; }</pre>
Define attachment processing.	> Implement <code>run()</code> . When a BlackBerry device user selects a menu item from the messages list, this action invokes the <code>run()</code> method. <pre>public void run(Message m, SupportedAttachmentPart p) {     // Perform processing on data.     Screen view = new Screen();     view.setTitle(new LabelField("Attachment Viewer"));     view.add(new RichTextField(new String((byte[])p.getContent()))); }</pre>
Register an attachment.	When registering a custom attachment handler, the attachment name must be prefixed with "x-rimdevice" for the attachment to be sent and stored on the BlackBerry device. <pre>&gt; Invoke AttachmentHandlerManager.addAttachmentHandler(). AttachmentHandlerManager m = AttachmentHandlerManager.getInstance(); CustomAttachmentHandler ah = new CustomAttachmentHandler(); m.addAttachmentHandler(ah);</pre>

### Retrieve attachments

Task	Steps
Retrieve the contents of an attachment.	> Invoke <code>SupportedAttachmentPart.getContent()</code> . <pre>String s = new String((byte[])p.getContent());</pre>

Task	Steps
Retrieve information about the attachment.	<ul style="list-style-type: none"> <li>&gt; Invoke the methods of the <code>SupportedAttachmentPart</code> class. The <code>SupportedAttachmentPart</code> class represents an attachment with a corresponding viewer on the BlackBerry® device. An <code>UnsupportedAttachmentPart</code> represents an attachment that does not have a viewer on the BlackBerry device.</li> </ul>

## Send a message with an attachment

Task	Steps
Create a multipart message.	<ul style="list-style-type: none"> <li>&gt; Create a new <code>Multipart</code> object.  <pre>byte[] data = new byte[256]; // The attachment. Multipart multipart = new Multipart(); // Default type of multipart/mixed.</pre> </li> </ul>
Create each component of the attachment.	<ul style="list-style-type: none"> <li>&gt; Create a <code>SupportedAttachmentPart</code> object, designating the <code>Multipart</code> object as its parent.  <pre>SupportedAttachmentPart attach = new SupportedAttachmentPart( multipart, "application/x-example", "filename", data);</pre> </li> </ul>
Add each <code>SupportedAttachmentPart</code> object to the multipart object.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>addBodyPart(SupportedAttachmentPart)</code> on that object.  <pre>multipart.addBodyPart(attach); // Add the attachment to the multipart.</pre> </li> </ul>
Set the content of the attachment.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>setContent(Multipart)</code> on the <code>Message</code> object and provide as a parameter the <code>Multipart</code> object.  <pre>msg.setContent(multipart);</pre> </li> </ul>
Send the message.	<ol style="list-style-type: none"> <li>1. Invoke <code>Session.getTransport()</code> and store the returned object in a variable of type <code>Transport</code>. The <code>Transport</code> object represents the messaging transport protocol.  <pre>Transport trans = Session.getTransport();</pre> </li> <li>2. Invoke <code>trans.send(Message)</code>.  <pre>try { trans.send(msg); } catch(MessagingException e) { System.out.println(e.getMessage()); }</pre> </li> </ol>

# Using PIM applications

Using the calendar

Notify an application when the default list of events on a BlackBerry device changes

Using tasks

Code samples

## Using the calendar

### Start the calendar from your BlackBerry Java Application

Task	Steps
Open the calendar.	> Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)</code> .
View or change an event.	<ol style="list-style-type: none"> <li>1. Retrieve an <code>Event</code> from the list of events.  <pre>Event e = null; EventList el = (EventList)PIM.getInstance().openPIMList( PIM.EVENT_LIST, PIM.READ_WRITE ); Enumeration events = el.items(); e = (Event)events.nextElement();</pre> </li> <li>2. Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)</code> using the <code>CalendarArguments</code> object created using the <code>ARG_VIEW_DEFAULT</code> property and the retrieved <code>Event</code>.  <pre>Invoke.invokeApplication( Invoke.APP_TYPE_CALENDAR, new CalendarArguments( CalendarArguments.ARG_VIEW_DEFAULT, e ) );</pre> </li> </ol>
Manage exceptions	> Check for a <code>ControlledAccessException</code> if your BlackBerry® Java® Application invokes a BlackBerry application that you do not have permission to use or access.

Task	Steps
Open a new populated event.	<ol style="list-style-type: none"> <li>1. Create a new Event using an EventList object.  <pre>Event e = null; EventList el = (EventList)PIM.getInstance().openPIMList( PIM.EVENT_LIST, PIM.READ_WRITE ); e = el.createEvent();</pre> </li> <li>2. Add information to the Event object.  <pre>e.addString( Event.SUMMARY, 0, "Go For A Walk" ); e.addString( Event.LOCATION, 0, "The Park" ); long start = System.currentTimeMillis() + 8640000; e.addDate( Event.START, 0, start ); e.addDate( Event.END, 0, start + 72000000 );</pre> </li> <li>3. Invoke <code>Invoke.invokeApplication(APP_TYPE_CALENDAR, CalendarArguments)</code> using the <code>CalendarArguments</code> object created using the <code>ARG_NEW</code> property and the Event.  <pre>Invoke.invokeApplication( Invoke.APP_TYPE_CALENDAR, new CalendarArguments( CalendarArguments.ARG_NEW, e ) );</pre> </li> <li>4. Use an instance of the EventList class to access the calendar.</li> <li>5. Create one or more Event objects to store information for specific appointments. For each event, you can store data such as the summary, location, start and end times, and reminder notification.</li> </ol>

## Use the calendar

Task	Steps
Open an event list.	<ul style="list-style-type: none"> <li>&gt; Create an EventList object by invoking <code>openPIMList()</code>, providing as parameters the type of list to open (<code>PIM.EVENT_LIST</code>) and the mode in which to open the list: <ul style="list-style-type: none"> <li>• <code>READ_WRITE</code></li> <li>• <code>READ_ONLY</code></li> <li>• <code>WRITE_ONLY</code></li> </ul> </li> </ul> <pre>EventList eventList = null; try { eventList = (EventList)PIM.getInstance().openPIMList( PIM.EVENT_LIST, PIM.READ_WRITE); } catch (PimException e) { // Handle exception. }</pre>
Create an appointment.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>createEvent()</code> on an event list.  <pre>Event event = eventList.createEvent();</pre> </li> </ul>

Task	Steps
Add appointment information.	<pre> &gt; To verify that an item supports a field, invoke isSupportedField(int). if (event.isSupportedField(Event.SUMMARY)) { event.addString(Event.SUMMARY, Event.ATTR_NONE, "Meet with customer"); } if (event.isSupportedField(Event.LOCATION)) { event.addString(Event.LOCATION, Event.ATTR_NONE, "Conference Center"); } Date start = new Date(System.currentTimeMillis() + 8640000); if (event.isSupportedField(Event.START)) { event.addDate(Event.START, Event.ATTR_NONE, start); } if (event.isSupportedField(Event.END)) { event.addDate(Event.END, Event.ATTR_NONE, start + 72000000); } if (event.isSupportedField(Event.ALARM)) { if (event.countValues(Event.ALARM) &gt; 0) { event.removeValue(Event.ALARM,0); event.setInt(Event.ALARM, 0, Event.ATTR_NONE, 396000); } } </pre>
Create a recurring appointment.	<ol style="list-style-type: none"> <li>1. Create a RepeatRule object. The RepeatRule class defines fields for the properties and values that you can set, such as COUNT, FREQUENCY, and INTERVAL.</li> <li>2. To retrieve an array of supported fields, invoke RepeatRule.getFields().</li> <li>3. To define a recurring pattern, invoke setInt(int, int) or setDate(int, int, int, long) on a new RepeatRule object. <pre> RepeatRule recurring = new RepeatRule(); recurring.setInt(RepeatRule.FREQUENCY, RepeatRule.MONTHLY); recurring.setInt(RepeatRule.DAY_IN_MONTH, 14); </pre> </li> <li>4. To assign a recurrence pattern to an appointment, invoke setRepeat(RepeatRule) on an event. <pre> EventList eventList = (EventList)PIM.getInstance().openPIMList( PIM.EVENT_LIST, PIM.READ_WRITE); Event event = eventList.createEvent(); event.setRepeat(recurring); </pre> </li> </ol>
Change appointment information.	<ol style="list-style-type: none"> <li>1. To replace an existing value with a new one, invoke the appropriate set method, such as setString().</li> <li>2. To determine if a value is already set for the field, invoke countValues().</li> <li>3. To change an existing value, use the corresponding set method, such as setString(). <pre> if (event.countValues(Event.LOCATION) &gt; 0) { event.setString(Event.LOCATION, 0, Event.ATTR_NONE, "Board Room"); } </pre> </li> </ol>

Task	Steps
Save an appointment.	<p>To save an appointment, use the <code>importEvent()</code> method; you do not have to invoke <code>commit()</code>.</p> <ol style="list-style-type: none"> <li>Before you save the appointment, to identify appointment fields that have changed since the appointment was last saved, invoke <code>isModified()</code>.</li> <li>Invoke <code>commit()</code>.</li> </ol> <pre>if(event.isModified()) {     event.commit(); }</pre>
Retrieve appointment information.	<ol style="list-style-type: none"> <li>To retrieve an enumeration of appointments, invoke <code>PIMList.items()</code>. <pre>EventList eventList = (EventList)PIM.getInstance().openPIMList(     PIM.EVENT_LIST, PIM.READ_ONLY); Enumeration e = eventList.items();</pre> </li> <li>To retrieve an array of IDs of fields that have data for a particular task, invoke <code>PIMItem.getFields()</code>.</li> <li>To retrieve the field values, invoke <code>PIMItem.getString()</code>. <pre>while (e.hasMoreElements()) {     Event event = (Event)e.nextElement();     int[] fieldIds = event.getFields();     int id;     for(int index = 0; index &lt; fieldIds.length; ++index) {         id = fieldIds[index];         if(e.getPIMList().getFieldDataType(id) == STRING) {             for(int j=0; j &lt; event.countValues(id); ++j) {                 String value = event.getString(id, j);                 System.out.println(event.getFieldLabel(id) + "=" + value);             }         }     } }</pre> </li> </ol>
Export an appointment.	<ol style="list-style-type: none"> <li>To import or export PIM item data, use an output stream writer to export tasks from the BlackBerry® device to a supported serial format, such as iCal®.</li> <li>To retrieve a string array of supported serial formats, invoke <code>PIM.supportedSerialFormats()</code>, and then specify the list type (<code>PIM.EVENT_List</code>).</li> <li>To write an item in serial format, invoke <code>toSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE". This parameter cannot be null.</li> </ol> <pre>EventList eventList = (EventList)PIM.getInstance().openPIMList(     PIM.EVENT_LIST, PIM.READ_ONLY ); ByteArrayOutputStream bytestream = new ByteArrayOutputStream(); Enumeration e = eventList.items(); while (e.hasMoreElements()) {     Event event = (Event)e.nextElement();     PIM.getInstance().toSerialFormat(event, byteStream, "UTF8",     dataFormats[0]); }</pre>

Task	Steps
Import an appointment.	<ol style="list-style-type: none"> <li>To return an array of PIMItem objects, invoke <code>SerialFormat(java.io.InputStream is, java.lang.String enc)</code>.</li> <li>To add a new appointment, invoke <code>EventList.importEvent()</code>.  // Convert an existing appointment into a iCal and then import the iCal as a new  // appointment.  String[] dataFormats = PIM.eventSerialFormats();  // Write appointment to iCal.  ByteArrayOutputStream os = new ByteArrayOutputStream();  PIM.getInstance().toSerialFormat(event, os, "UTF8", dataFormats[0]);  // Import appointment from iCal.  ByteArrayInputStream is = new  ByteArrayInputStream(outputStream.toByteArray());  PIMItem[] pi = PIM.getInstance().fromSerialFormat(is, "UTF8");  EventList eventList = (EventList)PIM.getInstance().openPIMList(  PIM.EVENT_LIST, PIM.READ_WRITE);  Event event2 = eventList.importEvent((Event)pi[0]);</li> </ol>
Close an event list.	<ol style="list-style-type: none"> <li>Invoke <code>close()</code>.</li> <li>Create a try-catch block to manage a <code>PimException</code>.  <pre>try { eventList.close(); } catch (PimException e) { // Handle exception. }</pre></li> </ol>

See "Code sample: Creating new recurring appointments" on page 43 for more information.

## Notify an application when a list of events changes

- Create a class that implements the `PIMListListener` interface.  

```
public class myEventListListener implements PIMListListener {
```
- To register to receive notifications of changes to an event list, invoke `BlackBerryPIMList.addListener()`.  

```
BlackBerryPIMList eventList =
(BlackBerryPIMList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE);
eventList.addListener(new myEventListListener());
```

## Notify an application when the default list of events on a BlackBerry device changes

When an update occurs to the list of services on a BlackBerry device, the list of organizer data databases on a device may change. This action may result in the creation of a new default list of events.

- Create a class that implements the `PIMListListener3` interface.  

```
public class myEventListListener3 implements PIMListListener3 {
```

2. To register to receive notifications of changes to an event list, invoke `BlackBerryPIMList.addListener()`.

```
BlackBerryPIMList eventList =
    (BlackBerryPIMList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE);
eventList.addListener(new myEventListListener3());
```

3. To have an application always listen to the default PIMList, add the following to the `defaultChanged()` method.

```
BlackBerryPIMList defaultEventList =
    (BlackBerryPIMList)PIM.getInstance().openPIMList(PIM.EVENT_LIST, PIM.READ_WRITE,
    newDefault);
defaultEventList.addListener(this);
pimList.removeListener(this);
```

## Retrieve multiple lists of events

The first element in the array of lists is the default list.

- > Invoke `PIM.listPIMLists(int pimListType)`.  
`String[] allLists = PIM.listPIMLists(EVENT_LIST);`

# Using the address book

## Open the address book from your BlackBerry Java Application

Task	Steps
Open the address book.	> From a BlackBerry® Java® Application, invoke <code>Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, AddressBookArguments)</code> .
Open a contact using PIM data.	<ol style="list-style-type: none"> <li>1. Create an instance of an <code>AddressBookArguments</code> object, specifying as a parameter a <code>Contact</code> object.  <code>AddressBookArguments abArg = AddressBookArguments(String arg, Contact contact);</code></li> <li>2. Invoke <code>Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, AddressBookArguments)</code> using the <code>AddressBookArguments</code> object for the contact.  <code>Invoke.invokeApplication(APP_TYPE_ADDRESSBOOK, abArg);</code></li> </ol>
Manage exceptions.	> Check for a <code>ControlledAccessException</code> if your BlackBerry® Java® Application invokes a BlackBerry application that you do not have permission to use or access.

## Use contacts

Task	Steps
Provide access to the PIN BlackBerry® device contacts field.	> Use the <code>BlackBerryContact.PIN</code> constant.

Task	Steps
Provide access to the USER1 through USER4 BlackBerry® device contacts fields.	<ul style="list-style-type: none"> <li>&gt; Use the following constants:               <ul style="list-style-type: none"> <li>• BlackBerryContact.USER1</li> <li>• BlackBerryContact.USER2</li> <li>• BlackBerryContact.USER3</li> <li>• BlackBerryContact.USER4</li> </ul> </li> </ul>
Define labels for the USER1 through USER4 BlackBerry® device contacts fields.	<p>Changing a label affects all contacts on the BlackBerry device.</p> <ul style="list-style-type: none"> <li>&gt; Invoke BlackBerryPIMList.setFieldLabel().</li> </ul>
Open a contacts list.	<ol style="list-style-type: none"> <li>1. Create a contacts list.  <pre>ContactList contactList = null;</pre> </li> <li>2. Invoke PIM.openPIMList() and provide as parameters the type of list to open (PIM.CONTACT_LIST) and the access mode with which to open the list (READ_WRITE, READ_ONLY, or WRITE_ONLY).  <pre>try {     contactList = (ContactList)PIM.getInstance().openPIMList(         PIM.CONTACT_LIST, PIM.READ_WRITE); } catch (PimException e) {     return; }</pre> </li> </ol>
Create a contact.	<p>To add a contact to the database, you must commit it. See "Save a contact" on page 173 for more information about committing contact data.</p> <ul style="list-style-type: none"> <li>&gt; Invoke createContact() on a contact list.  <pre>Contact contact = contactList.createContact();</pre> </li> </ul>

Task	Steps
Add contact information.	<ol style="list-style-type: none"> <li>1. Invoke one of the following methods: <ul style="list-style-type: none"> <li>• addString()</li> <li>• addStringArray()</li> <li>• addDate()</li> <li>• addInt()</li> <li>• addBoolean()</li> <li>• addBinary()</li> </ul> </li> <li>2. Before you set or retrieve a field, to verify that the item supports the field, invoke <code>ContactList.isSupportedField(int)</code>.</li> <li>3. To let fields store multiple values, use field attributes. For example, the TEL field supports the ATTR_HOME, ATTR_WORK, ATTR_MOBILE, and ATTR_FAX attributes to store numbers for work, home, mobile, and fax numbers.</li> <li>4. To determine how many values a field supports, invoke <code>PIMList.maxValues(int field)</code>.</li> <li>5. To verify that a field supports a particular attribute, invoke <code>isSupportedAttribute(int, int)</code>.</li> </ol> <pre> // Create string array for name. try {ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.WRITE_ONLY);} catch (PIMException e) {} Contact contact = contactList.createContact();String[] name = new String[contactList.stringArraySize(Contact.NAME)]; // 5 name elements try {name[Contact.NAME_PREFIX] = "Mr.";name[Contact.NAME_FAMILY] = "McPherson";name[Contact.NAME_GIVEN] = "Scott";} catch (IllegalArgumentException iae) {// handle exception} // Add name. if(contactList.isSupportedField(Contact.NAME)) {contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, name); } // Create string array for address. String[] address = new String[7]; // 7 address elements try {address[Contact.ADDR_COUNTRY] = "United States";address[Contact.ADDR_LOCALITY] = "Los Angeles";address[Contact.ADDR_POSTALCODE] = "632300";address[Contact.ADDR_REGION] = "California";address[Contact.ADDR_STREET] = "323 Main Street";} catch (IllegalArgumentException iae) {// Handle exception.} // Add address.contact.addStringArray(Contact.ADDR, Contact.ATTR_NONE, address); // Add home telephone number. if (contactList.isSupportedField(Contact.TEL) &amp;&amp;contactList.isSupportedAttribute(Contact.TEL, Contact.ATTR_HOME)) {contact.addString(Contact.TEL, Contact.ATTR_HOME, "555-1234");} // Add work telephone number.if (contactList.isSupportedField(Contact.TEL)) {contact.addString(Contact.TEL, Contact.ATTR_HOME, "555-5555");} // Add work internet messaging address. if (contactList.isSupportedField(Contact.EMAIL)) {contact.addString(Contact.EMAIL, Contact.ATTR_WORK, "aisha.wahl@blackberry.com");} </pre>

Task	Steps
Change contact information.	<ol style="list-style-type: none"> <li>1. To change the name and address fields, invoke the appropriate set method to replace an existing value with a new value.</li> <li>2. Perform one of the following actions: <ul style="list-style-type: none"> <li>• To change the fields that support a single value, retrieve the array and then change one or more indexes in the array before adding the array back to the <code>Contact</code> object. <pre data-bbox="586 413 1243 670"> if (contact.countValues(Contact.NAME) &gt; 0) { String[] newname = contact.getStringArray(Contact.NAME, 0); } // Change the prefix to Dr. and add the suffix, Jr. newname[Contact.NAME_PREFIX] = "Dr."; newname[Contact.NAME_SUFFIX] = "Jr."; contact.setStringArray(Contact.NAME, 0, Contact.ATTR_NONE, newname); </pre> </li> <li>• To change the contacts fields that support multiple values, before adding another value, verify that the number of values does not exceed the maximum number of values. For example: <pre data-bbox="586 760 1153 852"> if (contact.countValues(Contact.EMAIL) &lt; contactList.maxValues(Contact.EMAIL)) { contact.addString(Contact.EMAIL, Contact.ATTR_NONE, "aisha.wahl@blackberry.com");} </pre> </li> </ul> </li> <li>3. Create code to manage a <code>FieldFullException</code>, which occurs if you invoke an add method, such as <code>addString()</code>, for a field that already has a value.</li> </ol>
Save a contact.	<ol style="list-style-type: none"> <li>1. To determine if any contact fields have changed since the contact was last saved, invoke <code>isModified()</code>.</li> <li>2. Invoke <code>commit()</code>. <pre data-bbox="554 1013 846 1083"> if(contact.isModified()) { contact.commit(); } </pre> </li> </ol>

Task	Steps
Retrieve contact information.	<ol style="list-style-type: none"> <li>1. Invoke <code>PIMList.items()</code>.</li> <li>2. Perform one of the following actions: <ul style="list-style-type: none"> <li>• To retrieve an array of IDs for fields that have data for a particular contact, invoke <code>PIMItem.getFields()</code> .</li> <li>• To retrieve the field values, invoke <code>PIMItem.getString()</code>.</li> </ul> </li> <li>3. When you invoke <code>PIMList.items()</code> to retrieve an enumeration of items in a contacts list, your BlackBerry® Java® Application must sort items as necessary.</li> </ol> <pre> ContactList contactList = (ContactList)PIM.getInstance().openPIMList( PIM.CONTACT_LIST, PIM.READ_WRITE); Enumeration enum = contactList.items(); while (enum.hasMoreElements()) { Contact c = (Contact)enum.nextElement(); int[] fieldIds = c.getFields(); int id; for(int index = 0; index &lt; fieldIds.length; ++index) { id = fieldIds[index]; if(c.getPIMList().getFieldDataType(id) == Contact.STRING) { for(int j=0; j &lt; c.countValues(id); ++j) { String value = c.getString(id, j); System.out.println(c.getPIMList().getFieldLabel(id) + "=" + value); } } } } </pre>
Select a contact from the address book.	<p>&gt; Invoke the <code>BlackBerryContactList.choose()</code> to return a <code>Contact</code> or <code>BlackBerryContactGroup</code> <code>PIMItem</code>.</p> <pre> BlackBerryContactList list = (BlackBerryContactList)PIM.getInstance().openPIMList(PIM.CONTACT_ LIST, PIM.READ_WRITE); PIMItem item = list.choose(); if (item instanceof Contact) { Contact contact = (Contact)item; String email = contact.getString(Contact.EMAIL, 0); System.out.println("Name is: " + email); } else if (item instanceof BlackBerryContactGroup) { ... } </pre>

Task	Steps
Export a contact.	<ol style="list-style-type: none"> <li>1. To import or export PIM item data, use an output stream writer to export tasks from the BlackBerry® device to a supported serial format, such as vCard®.</li> <li>2. To retrieve a string array of supported formats, invoke <code>PIM.supportedSerialFormats()</code> and specify the list type (<code>PIM.Contact_LIST</code>).</li> <li>3. To write an item to a supported serial format, invoke <code>toSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null.</li> </ol> <pre> ContactList contactList = (ContactList)PIM.getInstance().openPIMList( PIM.CONTACT_LIST, PIM.READ_ONLY); String[] dataFormats = PIM.getInstance().supportedSerialFormats( PIM.CONTACT_LIST); ByteArrayOutputStream byteStream = new ByteArrayOutputStream(); Enumeration e = contactList.items(); while (e.hasMoreElements()) {     Contact c = (Contact)e.nextElement();     PIM.getInstance().toSerialFormat(c, byteStream, "UTF8", dataFormats[0]); } </pre>
Import a contact.	<ol style="list-style-type: none"> <li>1. To return an array of PIM items, invoke <code>fromSerialFormat()</code>.</li> <li>2. To create a new contact using the PIM item, invoke <code>ContactList.importContact()</code>.</li> <li>3. To specify the character encoding to use when writing to the output stream, use the <code>enc</code> parameter.</li> </ol> <pre> // Import contact from vCard. ByteArrayInputStream istream = new ByteArrayInputStream(outputStream.toByteArray()); PIMItem[] pi = PIM.getInstance().fromSerialFormat(istream, "UTF8"); ContactList contactList = (ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE); Contact contact2 = contactList.importContact((Contact)pi[0]); contact2.commit(); </pre>
Delete a contact.	<p>&gt; Invoke <code>removeContact()</code> on a contact list.</p> <pre> contactList.removeContact(contact); </pre>
Close a contacts list.	<p>&gt; Invoke <code>close()</code>.</p> <pre> try { contactList.close(); } catch(PIMException e) { Dialog.alert(e.toString()); } </pre>

See "Code sample: Displaying a screen that lets BlackBerry device users add new contacts" on page 45 for more information.

## Notify an application when a list of contacts changes

1. Create a class that implements the `PIMListListener` interface.

```
public class myContactListListener implements PIMListListener {
```

2. To register to receive notifications of changes to a contact list, invoke `BlackBerryPIMList.addListener()`.

```
BlackBerryPIMList contactList =  
(BlackBerryPIMList)PIM.getIntsance().openPIMList(PIM.CONTACT_LIST, PIM.READ_WRITE);  
contactList.addListener(new myContactListListener());
```

## Using tasks

### Start the task application from your BlackBerry Java Application

Check for a `ControlledAccessException` if your BlackBerry® Java® Application invokes a BlackBerry application that you do not have permission to use or access.

Task	Steps
Open the task application.	<p>The <code>TaskArguments</code> (<code>net.rim.blackberry.api.invoke.TaskArguments</code>) cannot be updated without changes to the <code>Task</code> application.</p> <pre>&gt; Invoke invoke.invokeApplication(APP_TYPE_TASKS, TaskArguments);</pre>
View or change a task.	<ol style="list-style-type: none"> <li>1. Create an instance of a <code>ToDoList</code> and store it in an <code>Enumeration</code>. <pre>ToDoList tdl = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE); Enumeration todos = tdl.items();</pre> </li> <li>2. Create a <code>ToDo</code> object using an element from the <code>Enumeration</code>: <pre>ToDo todo = (ToDo)todos.nextElement();</pre> </li> <li>3. Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_TASKS</code> constant parameter, and a new <code>TaskArguments</code> object created using the <code>ARG_VIEW</code> parameter and the <code>ToDo</code> object. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments(TaskArguments.ARG_VIEW, todo));</pre> </li> </ol>
Create a new blank task.	<pre>&gt; Invoke invokeApplication() using the APP_TYPE_TASKS constant parameter, and a new TaskArguments object created using the ARG_NEW parameter.</pre> <pre>Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments(TaskArguments.ARG_NEW));</pre>
Create a new populated task.	<ol style="list-style-type: none"> <li>1. Create an instance of a <code>ToDoList</code>. <pre>ToDoList tdl = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE);</pre> </li> <li>2. Invoke <code>createToDo()</code> to create a new <code>ToDo</code> object and add information to the new <code>ToDo</code> object. <pre>ToDo todo = tdl.createToDo(); todo.addString(ToDo.SUMMARY, 0, "Walk the Dog");</pre> </li> <li>3. Invoke <code>invokeApplication()</code> using the <code>APP_TYPE_TASKS</code> constant parameter, and a new <code>TaskArguments</code> object created using the <code>ARG_NEW</code> parameter and the new <code>ToDo</code> object. <pre>Invoke.invokeApplication(Invoke.APP_TYPE_TASKS, new TaskArguments(TaskArguments.ARG_NEW, todo));</pre> </li> </ol>

## Use tasks

Task	Steps
Open a task list.	<p>&gt; Invoke <code>PIM.openPIMList()</code> and provide as parameters the type of list to open (<code>PIM.TODO_LIST</code>) and the access mode with which to open the list (<code>READ_WRITE</code>, <code>READ_ONLY</code>, or <code>WRITE_ONLY</code>).</p> <pre> ToDoList todoList = null; try { todoList = (ToDoList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE); } catch (PimException e) { //an error occurred return; } </pre>
Create a task.	<p>&gt; Invoke <code>createToDo()</code> on a task list.</p> <pre> ToDo task = todoList.createToDo(); </pre>
Add task information.	<ol style="list-style-type: none"> <li>Before you set or retrieve a field, verify that the item supports the field by invoking <code>isSupportedField(int)</code>.</li> <li>To retrieve the field data type, invoke <code>PIMList.getFieldDataType(int)</code>.</li> <li>To set the field data, invoke one of the following methods: <ul style="list-style-type: none"> <li><code>addString()</code></li> <li><code>addDate()</code></li> <li><code>addInt()</code></li> <li><code>addBoolean()</code></li> <li><code>addBinary()</code></li> </ul> </li> </ol> <pre> if (todoList.isSupportedField(ToDo.SUMMARY)) { task.addString(ToDo.SUMMARY, ToDo.ATTR_NONE, "Create project plan"); } if (todoList.isSupportedField(ToDo.DUE)) { Date date = new Date(); task.addDate(ToDo.DUE, ToDo.ATTR_NONE, (date + 17280000)); } if (todoList.isSupportedField(ToDo.NOTE)) { task.addString(ToDo.NOTE, ToDo.ATTR_NONE, "Required for meeting"); } if (todoList.isSupportedField(ToDo.PRIORITY)) { task.addInt(ToDo.PRIORITY, ToDo.ATTR_NONE, 2); } </pre>
Set the status of a task.	<p>&gt; Use the PIM extended field <code>ToDo.EXTENDED_FIELD_MIN_VALUE + 9</code>:</p> <ul style="list-style-type: none"> <li><code>STATUS_NOT_STARTED: 1</code></li> <li><code>STATUS_IN_PROGRESS: 2</code></li> <li><code>STATUS_COMPLETED: 3</code></li> <li><code>STATUS_WAITING: 4</code></li> </ul> <pre> task.addInt(BlackBerryToDo.STATUS, ToDo.ATTR_NONE, BlackBerryToDo.STATUS_COMPLETED); </pre>

Task	Steps
Change task information.	<ol style="list-style-type: none"> <li>1. To replace an existing value with a new value, invoke the appropriate set method, such as <code>setString()</code>.</li> <li>2. To determine if a value is already set for the field, invoke <code>countValues()</code>.</li> <li>3. To change an existing value, use the corresponding <code>set()</code> method.</li> <li>4. Create code to manage a <code>FieldFullException</code> which a method such as <code>addString()</code> throws when a value already exists.</li> </ol> <pre> if (task.countValues(ToDo.SUMMARY) &gt; 0) { task.setString(ToDo.SUMMARY, 0, ToDo.ATTR_NONE, "Review notes"); } </pre>
Save a task.	<ol style="list-style-type: none"> <li>1. Before you commit your changes, to determine whether any task fields have changed since the task was last saved, invoke <code>isModified()</code></li> <li>2. Invoke <code>commit()</code>.</li> </ol> <pre> if(task.isModified()) { task.commit(); } </pre>
Retrieve task information.	<ol style="list-style-type: none"> <li>1. To retrieve an enumeration, invoke <code>PIMList.items()</code> on the task list. <pre> ToDoList todoList = (ToDoList)PIM.getInstance().openToDoList( PIM.TODO_LIST, PIM.READ_ONLY); Enumeration enum = todoList.items(); </pre> </li> <li>2. To retrieve an array of IDs for fields that have data for a particular <code>ToDo</code> item, invoke <code>PIMItem.getFields()</code>.</li> <li>3. To retrieve the field values, invoke <code>PIMItem.getString()</code>. <pre> while (enum.hasMoreElements()) { ToDo task = (ToDo)enum.nextElement(); int[] fieldIds = task.getFields(); int id; for(int index = 0; index &lt; fieldIds.length; ++index) { id = fieldIds[index]; if(task.getPIMList().getFieldDataType(id) == STRING) { for(int j=0; j &lt; task.countValues(id); ++j) { String value = task.getString(id, j); System.out.println(task.getFieldLabel(id) + "=" + value); } } } } } </pre> </li> </ol>

Task	Steps
Export a task.	<ol style="list-style-type: none"> <li>To import or export PIM item data, use an output stream writer to export tasks from the BlackBerry® device to a supported serial format.</li> <li>To retrieve a string array of supported serial formats, invoke <code>PIM.supportedSerialFormats()</code>, and then specify the list type (<code>PIM.TODO_List</code>).</li> <li>To write an item to a serial format, invoke <code>toSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null.</li> </ol> <pre> ToDoList todoList = (ToDoList)PIM.getInstance().openPIMList( PIM.TODO_LIST, PIM.READ_ONLY); ByteArrayOutputStream byteStream = new ByteArrayOutputStream(); String[] dataFormats = PIM.getInstance().supportedSerialFormats(PIM.TODO_LIST); Enumeration e = todoList.items(); while (e.hasMoreElements()) { ToDo task = (ToDo)e.nextElement(); PIM.getInstance().toSerialFormat(task, byteStream, "UTF8", dataFormats[0]); } </pre>
Import a task.	<ol style="list-style-type: none"> <li>To return an array of <code>PIMItem</code> objects, invoke <code>fromSerialFormat()</code>. The <code>enc</code> parameter specifies the character encoding to use when writing to the output stream. Supported character encodings include "UTF8," "ISO-8859-1," and "UTF-16BE." This parameter cannot be null.</li> <li>To create a new task using the PIM items, invoke <code>ToDoList.importToDo()</code>. The <code>importToDo()</code> method saves the task; you do not have to invoke <code>commit()</code>.</li> </ol> <pre> String[] dataFormats = PIM.todoSerialFormats(); // Write task to serial format. ByteArrayOutputStream os = new ByteArrayOutputStream(); PIM.getInstance().toSerialFormat(task, os, "UTF8", dataFormats[0]); // Import task from serial format. ByteArrayInputStream is = new ByteArrayInputStream(outputStream.toByteArray()); PIMItem[] pi = PIM.getInstance().fromSerialFormat(is, "UTF8"); ToDoList todoList = (ToDoList)PIM.getInstance().openPIMList( PIM.TODO_LIST, PIM.READ_WRITE); ToDo task2 = todoList.importToDo((ToDo)pi[0]); </pre>
Delete a task.	<pre> &gt; Invoke <code>removeToDo()</code> on a task list. todoList.removeToDo(task); </pre>
Close a task list.	<ol style="list-style-type: none"> <li>Invoke <code>todoList.close()</code>.</li> <li>Create code that manages exceptions.</li> </ol> <pre> try { todoList.close(); } catch (PimException e) { // Handle exception. } </pre>

See "Code sample: Using tasks" on page 48 for more information.

## Notify an application when a list of tasks changes

1. Create a class that implements the `PIMListListener` interface.

```
class myTaskListListener implements PIMListListener {
```

2. To register to receive notifications of changes to a task list, invoke `BlackBerryPIMList.addListener()`.

```
BlackBerryPIMList taskList =
(BlackBerryPIMList)PIM.getInstance().openPIMList(PIM.TODO_LIST, PIM.READ_WRITE);
taskList.addListener(new myTaskListListener());
```

## Code samples

### Code sample: Creating new recurring appointments

To let the BlackBerry® device user invite attendees to the meeting, combine this sample with `ContactsDemo.java`.

See "Code sample: Displaying a screen that lets BlackBerry device users add new contacts" on page 45 for more information.

---

#### Example: `EventDemo.java`

```
/**
 * EventDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.eventdemo;
import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public final class EventDemo extends UiApplication
{
    private EventScreen _eventScreen;

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

    private EventDemo() {
        _eventScreen = new EventScreen();
        pushScreen(_eventScreen);
    }
}
```

```

public final static class EventScreen extends MainScreen
{
    private EditField _subject, _location;
    private SaveMenuItem _saveMenuItem;
    private DateField _startTime, _endTime;
    private ObjectChoiceField _repeat;
    private Event event;

    private class SaveMenuItem extends MenuItem {
        public SaveMenuItem() {
            super(null, 0, 100000, 5);
        }

        public String toString() {
            return "Save";
        }

        public void run() {
            onSave();
        }
    }

    public EventScreen() {
        _saveMenuItem = new SaveMenuItem();
        setTitle(new LabelField("Event Demo", LabelField.ELLIPSIS |
            LabelField.USE_ALL_WIDTH) );
        _subject = new EditField("Subject: ", "");
        add(_subject);
        _location = new EditField("Location: ", "");
        add(_location);
        _startTime = new DateField("Start: ", System.currentTimeMillis() +
            3600000, DateField.DATE_TIME);
        _endTime = new DateField("End: ", System.currentTimeMillis() +
            7200000, DateField.DATE_TIME);
        add(new SeparatorField());
        add(_startTime);
        add(_endTime);
        add(new SeparatorField());
        String[] choices = {"None", "Daily", "Weekly", "Monthly", "Yearly"};
        _repeat = new ObjectChoiceField("Recurrence: ", choices, 0);
        add(_repeat);
    }

    protected boolean onSave() {
        try {
            EventList eventList = (EventList)PIM.getInstance().
                openPIMList(PIM.EVENT_LIST, PIM.WRITE_ONLY);
            event = eventList.createEvent();
            event.addString(Event.SUMMARY, PIMItem.ATTR_NONE,
                _subject.getText());
            event.addString(Event.LOCATION, PIMItem.ATTR_NONE,
                _location.getText());
            event.addDate(Event.END, PIMItem.ATTR_NONE, _endTime.getDate());
            event.addDate(Event.START, PIMItem.ATTR_NONE,
                _startTime.getDate());
            if(_repeat.getSelectedIndex() != 0) {
                event.setRepeat(setRule());
            }
        }
    }
}

```

```

    }
    // Save the appointment to the Calendar.
    event.commit();
    //reset fields on screen
    _subject.setText("");
    _location.setText("");
    _endTime.setDate(null);
    _startTime.setDate(null);
    _repeat.setSelectedIndex(0);
    return true;
} catch (PIMException e) {
    System.err.println(e);
}
}
return false;
}

private RepeatRule setRule() {
    RepeatRule rule = new RepeatRule();
    int index = _repeat.getSelectedIndex();
    if (index == 0) {
        rule.setInt(RepeatRule.FREQUENCY, RepeatRule.DAILY);
    }
    if (index == 1) {
        rule.setInt(RepeatRule.FREQUENCY, RepeatRule.WEEKLY);
    }
    if (index == 2) {
        rule.setInt(RepeatRule.FREQUENCY, RepeatRule.MONTHLY);
    }
    if (index == 3) {
        rule.setInt(RepeatRule.FREQUENCY, RepeatRule.YEARLY);
    }
    return rule;
}

protected void makeMenu(Menu menu, int instance) {
    menu.add(_saveMenuItem);
    menu.addSeparator();
    super.makeMenu(menu, instance);
}
}
}
}

```

---

## Code sample: Displaying a screen that lets BlackBerry device users add new contacts

The following sample demonstrates how to display a screen that lets BlackBerry® device users add new contacts to the address book.

---

### Example: ContactsDemo.java

```

/**
 * ContactsDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.

```

## BlackBerry Device Applications Integration Guide

```
*/  
  
package com.rim.samples.docs.contactsdemo;  
  
import java.io.*;  
import java.util.*;  
import javax.microedition.pim.*;  
import net.rim.device.api.ui.*;  
import net.rim.device.api.ui.component.*;  
import net.rim.device.api.ui.container.*;  
import net.rim.device.api.i18n.*;  
import net.rim.device.api.system.*;  
import net.rim.device.api.util.*;  
import net.rim.blackberry.api.pdap.*;  
  
public final class ContactsDemo extends UiApplication  
{  
    private ContactScreen _contactScreen;  
  
    public static void main(String[] args) {  
        new ContactsDemo().enterEventDispatcher();  
    }  
  
    public ContactsDemo() {  
        _contactScreen = new ContactScreen();  
        pushScreen(_contactScreen);  
    }  
  
    // Inner class. Creates a Screen to add a contact.  
    public static final class ContactScreen extends MainScreen  
    {  
        private EditField _first, _last, _email, _phone, _pin;  
        private SaveMenuItem _saveMenuItem;  
        private class SaveMenuItem extends MenuItem {  
            private SaveMenuItem() {  
                super(null, 0, 100000, 5);  
            }  
            public String toString() {  
                return "Save";  
            }  
            public void run() {  
                onSave();  
            }  
        }  
  
        public ContactScreen() {  
            _saveMenuItem = new SaveMenuItem();  
            setTitle(new LabelField("Contacts Demo", LabelField.ELLIPSIS |  
LabelField.USE_ALL_WIDTH));  
            _first = new EditField("First Name: ", "");  
            add(_first);  
            _last = new EditField("Last Name: ", "");  
            add(_last);  
            _email = new EditField("Email Address: ", "",  
BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_EMAIL);  
            add(_email);  
            _phone = new EditField("Work Phone: ", "",  
BasicEditField.DEFAULT_MAXCHARS, BasicEditField.FILTER_PHONE);
```

```

        add(_phone);
        _pin = new EditField("PIN:", "", 8, BasicEditField.FILTER_HEXADEDECIMAL);
        add(_pin);
    }

    protected boolean onSave() {
        String firstName = _first.getText();
        String lastName = _last.getText();
        String email = _email.getText();
        String phone = _phone.getText();
        String pin = _pin.getText();
        // Verify that a first or last name and email has been entered.
        if ((firstName.equals("") && lastName.equals("")) || email.equals("")) {
            Dialog.inform("You must enter a name and an email address!");
            return false;
        } else {
            try {
                ContactList contactList =
(ContactList)PIM.getInstance().openPIMList(PIM.CONTACT_LIST, PIM.WRITE_ONLY);
                Contact contact = contactList.createContact();
                String[] name = new String[contactList.stringArraySize(Contact.NAME)];
                // Add values to PIM item.
                if (!firstName.equals("")) {
                    name[Contact.NAME_GIVEN] = firstName;
                }
                if (!lastName.equals("")) {
                    name[Contact.NAME_FAMILY] = lastName;
                }
                contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, name);
                contact.addString(Contact.EMAIL, Contact.ATTR_HOME, email);
                contact.addString(Contact.TEL, Contact.ATTR_WORK, phone);
                if (contactList.isSupportedField(BlackBerryContact.PIN)) {
                    contact.addString(BlackBerryContact.PIN, Contact.ATTR_NONE, pin);
                }
                // Save data to address book.
                contact.commit();
                // Reset UI fields.
                _first.setText("");
                _last.setText("");
                _email.setText("");
                _phone.setText("");
                _pin.setText("");
                return true;
            } catch (PIMException e) {
                return false;
            }
        }
    }

    protected void makeMenu(Menu menu, int instance) {
        menu.add(_saveMenuItem);
        super.makeMenu(menu, instance);
    }
}

```

## Code sample: Using tasks

---

### Example: TaskDemo.java

```
/**
 * TaskDemo.java
 * Copyright (C) 2002-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.taskdemo;
import java.io.*;
import java.util.*;
import javax.microedition.pim.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;

public final class TaskDemo extends UiApplication
{
    private TaskScreen _taskScreen;

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

    private TaskDemo() {
        _taskScreen = new TaskScreen();
        pushScreen(_taskScreen);
    }

    public final static class TaskScreen extends MainScreen
    {
        // Members.
        private EditField _summary, _note;
        private DateField _due;
        private ObjectChoiceField _priority, _status;
        private SaveMenuItem _saveMenuItem;

        private class SaveMenuItem extends MenuItem
        {
            private SaveMenuItem() {
                super(null, 0, 100000, 5);
            }

            public String toString() {
                return "Save";
            }

            public void run() {
                onSave();
            }
        }

        public TaskScreen() {

```

```

_saveMenuItem = new SaveMenuItem();
setTitle(new LabelField("Tasks Demo",
LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH));
_summary = new EditField("Task Summary: ", "");
add(_summary);
// In TODO.Priority, 0 to 9 is highest to lowest priority.
String[] choices = {"High", "Normal", "Low"};
_priority = new ObjectChoiceField("Priority: ", choices, 1);
add(_priority);
String[] status = { "Not Started", "In Progress", "Completed",
"Waiting on someone else", "Deferred" };
_status = new ObjectChoiceField("Status: ", status, 0);
add(_status);
_due = new DateField("Due: ", System.currentTimeMillis() + 3600000,
DateField.DATE_TIME);
add(_due);
_note = new EditField("Extra Notes: ", "");
add(_note);
}

protected boolean onSave() {
    try {
        ToDoList todoList = (ToDoList)PIM.getInstance().
openPIMList(PIM.TODO_LIST, PIM.WRITE_ONLY);
        ToDo task = todoList.createToDo();
        task.addDate(ToDo.DUE, ToDo.ATTR_NONE, _due.getDate());
        task.addString(ToDo.SUMMARY, ToDo.ATTR_NONE, _summary.getText());
        task.addString(ToDo.NOTE, ToDo.ATTR_NONE, _note.getText());
        task.addInt(ToDo.PRIORITY, ToDo.ATTR_NONE,
_priority.getSelectedIndex());
        // ToDo.EXTENDED_FIELD_MIN_VALUE + 9 represents status.
        // Add 1 to selected index so that values are correct.
        // See the RIM Implementation Notes in the API documentation for ToDo.
        task.addInt(ToDo.EXTENDED_FIELD_MIN_VALUE + 9, ToDo.ATTR_NONE,
_status.getSelectedIndex() + 1);
        // Save task to handheld tasks.
        task.commit();
        _summary.setText("");
        _note.setText("");
        _due.setDate(null);
        _priority.setSelectedIndex(1); // Reset to "Normal" priority.
        _status.setSelectedIndex(0); // Reset to "Not Started" status.
        return true;
    } catch (PIMException e) {
        return false;
    }
}

protected void makeMenu(Menu menu, int instance) {
    menu.add(_saveMenuItem);
    super.makeMenu(menu, instance);
}
}
}

```



## Using the phone application

Start the phone application from your BlackBerry Java Application  
 Use phone call functionality  
 Listen for phone events  
 Access and use call logs  
 Code sample

### Start the phone application from your BlackBerry Java Application

To open the phone application from your BlackBerry® Java® Application, invoke `Invoke.invokeApplication(APP_TYPE_PHONE, PhoneArguments)`.

The following excerpt from the `Restaurants.java` code sample on page 69 creates a menu item that invokes the phone application to call a restaurant.

```
private MenuItem phoneItem = new MenuItem(_resources.getString(MENUITEM_PHONE), 110, 12) {
public void run() {
synchronized(store) {
String phoneNumber = phonefield.getText();
if ( phoneNumber.length == 0 ) {
Dialog.alert(_resources.getString(ALERT_NO_PHONENUMBER));
} else {
PhoneArguments call = new PhoneArguments(PhoneArguments.ARG_CALL, phoneNumber);
Invoke.invokeApplication(Invoke.APP_TYPE_PHONE, call);
}
}
}
};
```

### Use phone call functionality

Task	Steps
Retrieve a phone call.	> <code>Invoke Phone.getActiveCall().</code> <code>PhoneCall call = Phone.getActiveCall();</code>
Retrieve the phone number of a BlackBerry device.	> <code>Invoke Phone.getDevicePhoneNumber(boolean format).</code> <code>String phNumber = Phone.getDevicePhoneNumber(true);</code>
Retrieve a phone call by call ID.	> <code>Invoke Phone.getCall(int).</code>

Task	Steps
Retrieve phone call information.	<pre> &gt; Use the methods of the PhoneCall class. int threshold = 120; // Alert user if outgoing calls last longer than threshold. int elapsedTime = call.getElapsedTime(); // Use getStatusString() to retrieve status as a string. int status = call.getStatus(); if ((status == PhoneCall.STATUS_CONNECTED    status == PhoneCall.STATUS_CONNECTING) &amp;&amp; call.isOutGoing() &amp;&amp; elapsedTime &gt; threshold) { // Use getCallId() to retrieve the caller ID as as an integer. String phoneNumber = call.getDisplayPhoneNumber(); Status.show("Your call to " + phoneNumber + " has lasted more than " + (String)threshold + "."); }                     </pre>

## Add DTMF tones to the send queue

Task	Steps
Add a single DTMF tone to the send queue.	> Invoke <code>sendDTMFTone()</code> .
Add multiple DTMF tones to the send queue.	> Invoke <code>sendDTMFTones()</code> .
Retrieve the send queue for the current call.	> Invoke <code>getDTMFTones()</code> .

## BlackBerry DTMF tones

BlackBerry® devices play DTMF tones as soon as no other tones are pending, overriding conversations.

DTMF tones consist of a low and a high frequency, which are played at the same time.

Key	Low Tone (Hz)	High Tone (Hz)
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477
0	941	1209
*	941	1336
#	941	1477

## Listen for phone events

Task	Steps
Listen for phone events.	> Implement the <code>PhoneListener</code> interface.
Register the phone listener.	> Invoke <code>Phone.addPhoneListener()</code> .
Remove a phone listener.	> Invoke <code>removePhoneListener()</code> .

To act on a particular event, implement one of the following methods.

Event	Method
A call is added to a conference call.	<code>callAdded(int)</code>
A BlackBerry® device user answers a call (user driven).	<code>callAnswered(int)</code>
A conference call is established.	<code>callConferenceCallEstablished(int)</code>
The network indicates a connected event (network driven).	<code>callConnected(int)</code>
A direct-connect call is connected.	<code>callDirectConnectConnected(int)</code>
A direct-connect call is disconnected.	<code>callDirectConnectDisconnected(int)</code>
A call is disconnected.	<code>callDisconnected(int)</code>
A BlackBerry device user ends the call.	<code>callEndedByUser(int)</code>
A call fails.	<code>callFailed(int, int)</code>
A call goes on hold.	<code>callHeld(int)</code>
A new call arrives.	<code>callIncoming(int)</code>
The BlackBerry device initiates an outgoing call.	<code>callInitiated(int)</code>
A call is removed from a conference call.	<code>callRemoved(int)</code>
A held call resumes.	<code>callResumed(int)</code>
A call is waiting.	<code>callWaiting(int)</code>
A conference call is ended (all members are disconnected).	<code>conferenceCallDisconnected(int)</code>

## Access and use call logs

Task	Steps
Retrieve a phone log.	The <code>PhoneLogs</code> class represents the phone call history and provides methods for opening, adding, deleting, or swapping call logs. > Invoke <code>PhoneLogs.getInstance()</code> . <code>PhoneLogs _logs = PhoneLogs.getInstance();</code>
Retrieve the number of normal or missed calls.	There are two phone log folders: <code>FOLDER_NORMAL_CALLS</code> and <code>FOLDER_MISSED_CALLS</code> . > Invoke <code>numberOfCalls(int)</code> . <code>int numberOfCalls = _logs.numberOfCalls(FOLDER_NORMAL_CALLS);</code>
Retrieve a call log.	You can instantiate two types of call logs: <code>PhoneCallLog</code> objects, which have only one participant, and <code>ConferencePhoneCallLog</code> objects, which have two or more participants. > Invoke <code>PhoneLogs.callAt(int index, long folderID)</code> . <code>PhoneCallLog phoneLog = (PhoneCallLog)_logs.callAt(0);</code>

Task	Steps
Retrieve a call participant by phone number.	<p>The PhoneCallLogID class identifies participants in a phone call log by phone number.</p> <ul style="list-style-type: none"> <li>&gt; Invoke <code>PhoneCallLog.getParticipant(int)</code> or <code>ConferencePhoneCallLog.getParticipantAt()</code>.</li> </ul> <pre>PhoneCallLogID participant = phoneCallLog.getParticipant(); PhoneCallLogID participant = ConferencePhoneCallLog.getParticipant();</pre>
Retrieve the phone number type.	<p>The PhoneCallLogID class identifies the type of phone call for a log. For example, home, mobile, work, or fax, as recorded in the address book.</p> <ul style="list-style-type: none"> <li>&gt; Invoke <code>PhoneCallLogID.getType()</code>.</li> </ul> <pre>String phoneType = PhoneCallLogID.getType();</pre>
Create a call log or conference call log.	<p>The PhoneCallLogID constructor removes dashes and other non-numeric characters from phone numbers.</p> <ol style="list-style-type: none"> <li>1. Create an instance of a <code>PhoneCallLog</code> or <code>ConferencePhoneCallLog</code> object, and provide the date, duration, participants, and notes for the call as parameters to the constructor. <pre>Date date = new Date("1000"); // date of call int duration = 60; // duration of call PhoneCallLogID caller1 = new PhoneCallLogID("555-1234"); // first participant PhoneCallLogID caller2 = new PhoneCallLogID("555-1235"); // second participant String notes = "New call."; // notes ConferencePhoneCallLog conferenceCall = new ConferencePhoneCallLog(date, duration, PhoneLogs.FOLDER_NORMAL_CALLS, caller1, caller2, notes);</pre> </li> <li>2. Update the call log: <ul style="list-style-type: none"> <li>• To update the call log, invoke <code>PhoneLogs.addCall(CallLog call)</code>.</li> <li>• To replace the call log with a new call log, invoke <code>PhoneLogs.swapCall(CallLog call, int index, long folderID)</code>.</li> </ul> <pre>_logs.swapCall(conferenceCall, 0, FOLDER_NORMAL_CALLS);</pre> </li> </ol>
Delete a call log.	<ul style="list-style-type: none"> <li>&gt; Invoke <code>PhoneLogs.deleteCall()</code>.</li> </ul> <pre>_logs.deleteCall(0);</pre>

## Code sample

### Code sample: Calculating the time spent on the phone by a participant

Example: `PhoneLogsDemo.java`

```
/**
 * PhoneLogsDemo.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.phonelogs;

import net.rim.blackberry.api.phone.phonelogs.*;
```

```

import java.lang.*;
import net.rim.device.api.system.Application;

public class PhoneLogsDemo extends Application
{
    private PhoneLogs _logs;
    private int _timeSpokenTo;

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

    private PhoneLogsDemo() {
        _logs = PhoneLogs.getInstance();
        PhoneCallLogID participant = new PhoneCallLogID("5551234");
        _timeSpokenTo = findTimeSpokenTo(participant,
            PhoneLogs.FOLDER_NORMAL_CALLS);
    }

    // Returns the number of seconds spent on the phone with a participant.
    public int findTimeSpokenTo(PhoneCallLogID participant,
        long folder) {
        int numberOfCalls = this._logs.numberOfCalls(folder);
        int timeSpokenTo = 0;
        PhoneCallLog phoneCallLog;
        ConferencePhoneCallLog conferencePhoneCallLog;
        for (int i = 0; i < numberOfCalls; i++) {
            Object o = _logs.callAt(i, folder);
            if (o instanceof PhoneCallLog) {
                phoneCallLog = (PhoneCallLog) o;
                if ( phoneCallLog.getParticipant() == participant)
                    timeSpokenTo += phoneCallLog.getDuration();
            } else {
                conferencePhoneCallLog = (ConferencePhoneCallLog) o;
                int participants = conferencePhoneCallLog.numberOfParticipants();
                for (int j = 0; j < participants; j++)
                    if (conferencePhoneCallLog.getParticipantAt(j) == participant) {
                        timeSpokenTo += conferencePhoneCallLog.getDuration();
                        j = participants;
                    }
            }
        }
        return timeSpokenTo;
    }
}

```

---



# Using the BlackBerry Browser

Display content in the BlackBerry Browser  
 Display content in a BlackBerry Browser field  
 Code sample

## Display content in the BlackBerry Browser

To display web content in the BlackBerry® Browser, use the `net.rim.blackberry.api.browser` package.

Task	Steps
Retrieve a BlackBerry® Browser session.	Retrieving the default session overrides any open sessions on the BlackBerry device. > Retrieve the default <code>BrowserSession</code> object by invoking the static method <code>Browser.getDefaultSession()</code> .
Retrieve a non-default BlackBerry® Browser session.	> Invoke <code>Browser.getSession()</code> .
Request a web page.	> Invoke <code>BrowserSession.displayPage(String url)</code> , specifying the URL that contains the web content.  The following excerpt from the <code>Restaurants.java</code> sample creates a menu item that displays a web page in the BlackBerry Browser. <pre>private MenuItem browserItem = new MenuItem(_resources.getString(MENUITEM_BROWSER), 110, 12) { public void run() { synchronized(store) {String websiteUrl = websitefield.getText(); if (websiteUrl.length == 0) { Dialog.alert(_resources.getString(ALERT_NO_WEBSITE)); } else { BrowserSession visit = Browser.getDefaultSession(); visit.displayPage(websiteUrl); } } } };</pre>

## Display content in a BlackBerry Browser field

To display web content in a BlackBerry® Browser field, use the `net.rim.blackberry.api.browser` package.

Task	Steps
Access a rendering session.	1. Invoke <code>RenderingSession.getNewInstance()</code> . 2. Store the returned rendering session handle in a <code>RenderingSession</code> object. <pre>RenderingSession _renderingSession = RenderingSession.getNewInstance();</pre>

Task	Steps
Define callback functionality for a rendering session.	> Implement the <code>RenderingApplication</code> interface.
Retrieve a BlackBerry® Browser field.	<ol style="list-style-type: none"> <li>1. Invoke <code>RenderingSession.getBrowserContent(javax.microedition.io.HttpConnection, net.rim.device.api.browser.field.RenderingApplication, net.rim.device.api.browser.field.Event)</code>.</li> <li>2. Store the returned object in a <code>BrowserContent</code> object. You render web content in the <code>BrowserContent</code> object.  <pre>BrowserContent browserContent = _renderingSession.getBrowserContent(HttpConnection connection, this, Event e);</pre> </li> </ol>
Retrieve a field in which the URL content is rendered to your BlackBerry® Java® Application for display.	<p>&gt; Invoke <code>BrowserContent.getDisplayableContent()</code>, storing the returned object in a <code>Field</code> object.</p> <pre>Field field = browserContent.getDisplayableContent();</pre>
Display a BlackBerry® Browser field.	<ol style="list-style-type: none"> <li>1. To clear the current screen, invoke the <code>MainScreen.deleteAll()</code> method. <code>_mainScreen.deleteAll();</code></li> <li>2. To add field data to the BlackBerry Java® Application screen, invoke <code>MainScreen.add()</code>. <code>_mainScreen.add(field);</code></li> <li>3. Create a non-main event thread to run <code>BrowserContent.finishLoading()</code> so that the UI does not lock.</li> <li>4. To render the new BlackBerry browser content, invoke <code>BrowserContent.finishLoading()</code>. HTML files display a blank field until you invoke <code>BrowserContent.finishLoading()</code>. WML files and images might load before you invoke this method.</li> </ol>
Create a separate thread for rendering.	<p>&gt; Create a non-main thread that contains the instructions for retrieving and displaying the BlackBerry® Browser field.</p> <pre>class CreationThread extends Thread { BrowserFieldHandlerApplication _callBackApplication; BasicRenderingApplication _renderingApplication; public CreationThread(BrowserFieldHandlerApplication callBackApplication) { _callBackApplication = callBackApplication; } public void run() { _renderingApplication = new BasicRenderingApplication(_callBackApplication); BrowserField field = _renderingApplication.getBrowserField("www.blackberry.com"); _callBackApplication.displayBrowserField(field); } }</pre>
Set rendering options.	> Override <code>BrowserContent.getRenderingOptions()</code> . Your BlackBerry® Java® Application uses the default rendering options if you do not override <code>BrowserContent.getRenderingOptions()</code> .

Task	Steps
Manage events.	<p>&gt; Implement of <code>RenderingApplication.eventOccurred()</code>, specifying the actions that occur when a specific rendering event occurs.</p> <p>The following example specifies actions that occur in the event of a URL request, change in browser content, or a redirect to a different web page.</p> <pre> public Object eventOccurred(Event event) {     int eventId = event.getUID();     switch (eventId) { case Event.EVENT_URL_REQUESTED : {         URLRequestEvent urlRequestedEvent = (URLRequestEvent) event;         String absoluteUrl = urlRequestedEvent.getURL();         HttpConnection conn = null;         PrimaryResourceFetchThread thread = new         PrimaryResourceFetchThread(urlRequestedEvent.getURL(),         urlRequestedEvent.getHeaders(), urlRequestedEvent.getPostData(), event,         this);         thread.start();         break;}         case Event.EVENT_BROWSER_CONTENT_CHANGED: {             // The browser field title might have changed, so we update the title             field.             BrowserContentChangedEvent browserContentChangedEvent =             (BrowserContentChangedEvent) event;             if (browserContentChangedEvent.getSource() instanceof BrowserContent) {                 BrowserContent browserField = (BrowserContent)                 browserContentChangedEvent.getSource();                 String newTitle = browserField.getTitle();                 if (newTitle != null) {                     _mainScreen.setTitle(newTitle);}}             break;         }         case Event.EVENT_REDIRECT : {             RedirectEvent e = (RedirectEvent) event;             String referrer = e.getSourceURL();             switch (e.getType()) {                 case RedirectEvent.TYPE_JAVASCRIPT :                     break;                 case RedirectEvent.TYPE_META :                     // For MSIE and Mozilla, do not send a Referer for META Refresh.                     referrer = null;                     break;                 case Event.EVENT_SET_HEADER : // no cache support                 case Event.EVENT_SET_HTTP_COOKIE : // no cookie support                 default :                     }             return null;         }     } </pre>

## Code sample

### Code sample: Using the BlackBerry Browser

---

#### Example: BrowserFieldSampleApplication.java

```

/**
 * DefaultRenderingApplication.java
 * Copyright (C) 2004-2005 Research In Motion Limited.
 */

package com.rim.samples.docs.browser;

import java.io.IOException;
import javax.microedition.io.HttpConnection;
import net.rim.device.api.browser.field.*;
import net.rim.device.api.io.http.HttpHeaders;
import net.rim.device.api.system.Application;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.Status;
import net.rim.device.api.ui.container.MainScreen;

final public class BrowserFieldSampleApplication extends UiApplication implements
RenderingApplication
{
    private static final String REFERER = "referer";
    private RenderingSession _renderingSession;
    private MainScreen _mainScreen;
    private HttpConnection _currentConnection;

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

    private BrowserFieldSampleApplication() {
        _mainScreen = new MainScreen();
        pushScreen(_mainScreen);
        _renderingSession = RenderingSession.getNewInstance();

        PrimaryResourceFetchThread thread = new PrimaryResourceFetchThread("http://
www.google.com", null, null, null, this);
        thread.start();
    }

    public void processConnection(HttpConnection connection, Event e) {
        // cancel previous request
        if (_currentConnection != null) {
            try {
                _currentConnection.close();
            } catch (IOException e1) {
            }
        }
        _currentConnection = connection;
    }
}

```

```

BrowserContent browserContent = null;

try {
    browserContent = _renderingSession.getBrowserContent(connection, this, e);

    if (browserContent != null) {
        Field field = browserContent.getDisplayableContent();
        if (field != null) {
            synchronized (Application.getEventLock()) {
                _mainScreen.deleteAll();
                _mainScreen.add(field);
            }
        }
        browserContent.finishLoading();
    }
} catch (RenderingException re) {
} finally {
    SecondaryResourceFetchThread.doneAddingImages();
}
}

/**
 * @see
net.rim.device.api.browser.RenderingApplication#eventOccurred(net.rim.device.api.browser.Event)
 */
public Object eventOccurred(Event event) {
    int eventId = event.getUID();
    switch (eventId) {
        case Event.EVENT_URL_REQUESTED : {
            UrlRequestedEvent urlRequestedEvent = (UrlRequestedEvent) event;
            String absoluteUrl = urlRequestedEvent.getURL();

            HttpConnection conn = null;
            PrimaryResourceFetchThread thread = new
PrimaryResourceFetchThread(urlRequestedEvent.getURL(),
urlRequestedEvent.getHeaders(),
urlRequestedEvent.getPostData(),
event,
this);

            thread.start();
            break;
        } case Event.EVENT_BROWSER_CONTENT_CHANGED: {
            // browser field title might have changed update title
            BrowserContentChangedEvent browserContentChangedEvent =
(BrowserContentChangedEvent) event;
            if (browserContentChangedEvent.getSource() instanceof BrowserContent) {
                BrowserContent browserField = (BrowserContent)
browserContentChangedEvent.getSource();
                String newTitle = browserField.getTitle();
                if (newTitle != null) {
                    _mainScreen.setTitle(newTitle);
                }
            }
            break;
        } case Event.EVENT_REDIRECT : {

```

```

RedirectEvent e = (RedirectEvent) event;
String referrer = e.getSourceURL();
switch (e.getType()) {
    case RedirectEvent.TYPE_SINGLE_FRAME_REDIRECT :
        // show redirect message
        Application.getApplication().invokeAndWait(new Runnable() {
            public void run() {
                Status.show("You are being redirected to a different
page...");
            }
        });
        break;
    case RedirectEvent.TYPE_JAVASCRIPT :
        break;
    case RedirectEvent.TYPE_META :
        // MSIE and Mozilla don't send a Referer for META Refresh.
        referrer = null;
        break;
    case RedirectEvent.TYPE_300_REDIRECT :
        // MSIE, Mozilla, and Opera all send the original
        // request's Referer as the Referer for the new
        // request.
        Object eventSource = e.getSource();
        if (eventSource instanceof HttpConnection) {
            referrer =
((HttpConnection)eventSource).getRequestProperty(REFERER);
        }
        break;
}
HttpHeaders requestHeaders = new HttpHeaders();
requestHeaders.setProperty(REFERER, referrer);
PrimaryResourceFetchThread thread = new
PrimaryResourceFetchThread(e.getLocation(), requestHeaders,null, event, this);
thread.start();
break;
} case Event.EVENT_CLOSE :
    // TODO: close the application
    break;
case Event.EVENT_SET_HEADER :           // no cache support
case Event.EVENT_SET_HTTP_COOKIE :     // no cookie support
case Event.EVENT_HISTORY :             // no history support
case Event.EVENT_EXECUTING_SCRIPT :    // no progress bar is supported
case Event.EVENT_FULL_WINDOW :        // no full window support
case Event.EVENT_STOP :                // no stop loading support
default :
}

return null;
}

/**
 * @see
net.rim.device.api.browser.RenderingApplication#getAvailableHeight(net.rim.device.api.brow
ser.BrowserContent)
 */
public int getAvailableHeight(BrowserContent browserField) {
    // field has full screen
    return Graphics.getScreenHeight();
}

```

```

    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getAvailableWidth(net.rim.device.api.brows
     er.BrowserContent)
     */
    public int getAvailableWidth(BrowserContent browserField) {
        // field has full screen
        return Graphics.getScreenWidth();
    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getHistoryPosition(net.rim.device.api.brow
     ser.BrowserContent)
     */
    public int getHistoryPosition(BrowserContent browserField) {
        // no history support
        return 0;
    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getHTTPCookie(java.lang.String)
     */
    public String getHTTPCookie(String url) {
        // no cookie support
        return null;
    }

    /**
     * @see
     net.rim.device.api.browser.RenderingApplication#getResource(net.rim.device.api.browser.Req
     uestedResource,
     *     net.rim.device.api.browser.BrowserContent)
     */
    public HttpURLConnection getResource( RequestedResource resource, BrowserContent referrer)
    {

        if (resource == null) {
            return null;
        }

        // check if this is cache-only request
        if (resource.isCacheOnly()) {
            // no cache support
            return null;
        }

        String url = resource.getUrl();

        if (url == null) {
            return null;
        }

        // if referrer is null we must return the connection
        if (referrer == null) {

```

```
        HttpURLConnection connection = Utilities.makeConnection(resource.getUrl(),
resource.getRequestHeaders(), null);
        return connection;
    } else {
        // if referrer is provided we can set up the connection on a separate thread
        SecondaryResourceFetchThread.enqueue(resource, referrer);
    }
    return null;
}

/**
 * @see
net.rim.device.api.browser.RenderingApplication#invokeRunnable(java.lang.Runnable)
 */
public void invokeRunnable(Runnable runnable) {
    (new Thread(runnable)).run();
}
}

class PrimaryResourceFetchThread extends Thread
{
    private BrowserFieldSampleApplication _application;
    private Event _event;
    private byte[] _postData;
    private HttpHeaders _requestHeaders;
    private String _url;

    PrimaryResourceFetchThread(String url, HttpHeaders requestHeaders, byte[] postData,
        Event event, BrowserFieldSampleApplication application) {

        _url = url;
        _requestHeaders = requestHeaders;
        _postData = postData;
        _application = application;
        _event = event;
    }

    public void run() {
        HttpURLConnection connection = Utilities.makeConnection(_url, _requestHeaders,
        _postData);
        _application.processConnection(connection, _event);
    }
}
```

---



