BlackBerry Java Application
Core
Version: 5.0
Development Guide
## Contents

1 Managing applications
   - Application manager .................................................................................................................. 5
     - Retrieve information about a BlackBerry Java Application .................................................. 5
     - Communicate with another BlackBerry Java Application ....................................................... 5
     - Determine the services that are available to a BlackBerry Java Application ......................... 6
   - Application control ...................................................................................................................... 6
     - Request access to resources ........................................................................................................ 6
   - Code modules ............................................................................................................................... 7
     - Retrieve module information ....................................................................................................... 7
   - Access control messages ............................................................................................................... 7
     - Displaying a message for an operation that requires user permission .................................. 7
     - Display application control message to a user ........................................................................... 7

2 Storing data
   - Data management ......................................................................................................................... 9
   - Access to memory ........................................................................................................................ 9
   - File systems and paths .................................................................................................................. 9
   - File Connections APIs ............................................................................................................... 9
   - Storage on removable media ....................................................................................................... 10
     - Accessing data on the microSD media card ............................................................................... 10
     - Using the microSD media card with more than one BlackBerry device .................................. 10
     - File encryption on microSD cards ............................................................................................. 10
     - IT policies and the microSD media card .................................................................................... 10
     - Encryption of data on a microSD media card ......................................................................... 10
   - Code sample: Reading sections of a binary file .......................................................................... 11
   - Persistent data storage ............................................................................................................... 12
     - Create a persistent data store .................................................................................................... 12
     - Store persistent data .................................................................................................................. 13
     - Retrieve persistent data ............................................................................................................. 13
     - Remove persistent data ............................................................................................................. 14
   - MIDP record storage ................................................................................................................... 14
     - Create an MIDP record store ...................................................................................................... 14
     - Add a record to a record store .................................................................................................. 14
     - Retrieve a record from a record store ....................................................................................... 14
     - Retrieve all records from a record store ................................................................................... 15
   - Collections ................................................................................................................................. 15
     - Retrieve a collection from persistent storage ............................................................................ 15
Create a collection listener to notify the system when a collection changes................................. 16
Remove a collection listener that notifies the system when a collection changes.............................. 17
Notify the system when a collection changes......................................................................................... 17
Runtime storage....................................................................................................................................... 18
Retrieve the runtime store.......................................................................................................................... 18
Add an object in the runtime store............................................................................................................ 18
Replace an object in the runtime store....................................................................................................... 18
Retrieve a registered runtime object.......................................................................................................... 19
Retrieve an unregistered runtime object.................................................................................................... 19

3 Creating connections............................................................................................................................... 20
Network connections and transport types............................................................................................... 20
Using the BlackBerry Enterprise Server as an intranet gateway............................................................. 20
Using the wireless service provider's Internet gateway............................................................................ 21
Retrieve the wireless network name.......................................................................................................... 21
Connections................................................................................................................................................ 21
Use HTTP authentication............................................................................................................................ 21
Use an HTTPS connection........................................................................................................................... 23
Use a socket connection............................................................................................................................... 23
Use a datagram connection.......................................................................................................................... 25
Use a USB or serial port connection............................................................................................................ 26
Use a Bluetooth serial port connection..................................................................................................... 27
Wi-Fi connections...................................................................................................................................... 28
Wireless access families............................................................................................................................. 28
Retrieve the wireless access families that a BlackBerry device supports.................................................... 29
Determine if a BlackBerry device supports multiple wireless access families........................................ 29
Determine the wireless access family transceivers that are turned on.................................................... 29
Turn on the transceiver for a wireless access family.................................................................................. 29
Turn off the transceiver for a wireless access family.................................................................................. 29
Check if the Wi-Fi transceiver is turned on................................................................................................. 29
Check if the Wi-Fi transceiver is connected to a wireless access point...................................................... 29
Retrieve the status of the wireless access point or the active Wi-Fi profile.............................................. 30
Retrieve the wireless network name.......................................................................................................... 30
Open a Wi-Fi socket connection.................................................................................................................. 30
Open a Wi-Fi HTTP connection................................................................................................................... 31
Open a Wi-Fi HTTPS connection.................................................................................................................. 31
Enhanced Network API.............................................................................................................................. 31
Code sample: Displaying available transport types.................................................................................. 32
Code sample: Creating a connection over HTTP by using the first available transport............................ 33
### Controlling access to APIs and application data

1. Check if a code signature is required
2. Java APIs with controlled access
3. Register to use controlled APIs
4. Restrictions on code signatures
5. Request a code signature
6. Register a signature key using a proxy server
7. Sign an application using a proxy server
8. View the signature status for an application
9. Using keys to protect APIs and data
10. Protect APIs using code signing keys
11. Protect runtime store data using code signing keys
12. Protect persistent data using code signing keys

### Testing a BlackBerry device application

1. Testing applications on a BlackBerry Smartphone Simulator
2. Testing applications on a BlackBerry device
3. Testing applications using the compiled .cod files
   - Install and remove a .cod file for testing
   - Save a .cod file from a device to a computer
   - Retrieve information about a .cod file

### Packaging and distributing a BlackBerry Java Application

1. Preverify a BlackBerry device application
2. Application distribution over the wireless network
3. Wireless pull (user-initiated)
4. Wireless push (server-initiated)
5. Distributing BlackBerry Java Applications over the wireless network
   - Extract sibling .cod files
   - Modifying information for a MIDlet suite
   - Properties of BlackBerry device application .jad files
   - Correct the .cod file sizes listed in a .jad file
   - Create .jad files that reference multiple .cod files
6. Distributing BlackBerry device applications with the BlackBerry Desktop Software
   - Elements in BlackBerry device application .alx file
7. Application distribution through a computer connection
   - Distribute an application from a computer
   - Distribute an application from a web page
Managing applications

Application manager

The BlackBerry® Java® Virtual Machine on the BlackBerry® device includes an application manager that is the central dispatcher of operating system events for other BlackBerry device applications.

The `net.rim.device.api.system.ApplicationManager` class lets applications interact with the application manager to perform the following actions:

- interact with processes, such as retrieving the IDs for foreground applications
- post global events to the system
- run an application immediately or at a specific time

Retrieve information about a BlackBerry Java Application

1. Import the required classes and interfaces.

   ```java
   import net.rim.device.api.system.ApplicationManager;
   import net.rim.device.api.system.ApplicationDescriptor;
   import java.lang.String;
   import net.rim.device.api.system.*;
   ```

2. To retrieve information about the processes that are running, invoke `ApplicationManager.getVisibleApplications()`.

   ```java
   ApplicationManager manager = ApplicationManager.getApplicationManager();
   ApplicationDescriptor descriptors[] = manager.getVisibleApplications();
   ```

3. To retrieve descriptions of the objects for the BlackBerry® device applications that are running, invoke `ApplicationDescriptor.getName()`.

   ```java
   String appname1 = descriptors[0].getName();
   ```

4. To retrieve a description of the current application, invoke `ApplicationDescriptor.currentApplicationDescriptor()`.

   ```java
   ApplicationDescriptor descriptor = ApplicationDescriptor.currentApplicationDescriptor();
   ```

Communicate with another BlackBerry Java Application

1. Import the `net.rim.device.api.system.ApplicationManager` class.

2. To post a system-level event to another BlackBerry® device application, invoke one of the `ApplicationManager.postGlobalEvent()` methods.
Determine the services that are available to a BlackBerry Java Application

The service book consists of service records. Each service record defines a service on a BlackBerry® device. Service records define the communication protocol (WAP or IPPP), the network gateway, and the configuration information such as the browser settings.

1. Import the net.rim.device.api.servicebook class.
2. To let your BlackBerry device application interact with the BlackBerry® Infrastructure, use the service book API (net.rim.device.api.servicebook).

Application control

The BlackBerry® Application Control IT policy rules provide administrators with the ability to establish the capabilities of an application when it runs on a specific BlackBerry device. For example, administrators can use the BlackBerry Application Control IT policy rule to make sure that a game that exists on the BlackBerry device cannot access the phone API. The BlackBerry Application Control IT policy rule works only when the BlackBerry device is connected to a BlackBerry® Enterprise Server. This IT policy does not apply to BlackBerry devices that use the BlackBerry® Internet Service only.

If the administrator or a user denies the application access to one of the protected areas, the associated method throws a ControlledAccessException. For class-level checks, the method throws a NoClassDefFoundError. Depending on which APIs that you use, your application might need to handle both types of errors.

Request access to resources

1. Import the required classes and interfaces.

   ```java
   import net.rim.device.api.applicationcontrol.ApplicationPermissions;
   import net.rim.device.api.applicationcontrol.ApplicationPermissionsManager;
   ```

2. Create an instance of the ApplicationPermissions class.

   ```java
   ApplicationPermissions permissions = new ApplicationPermissions();
   ```

3. Specify the build request to ask for event injection privileges.

   ```java
   permissions.addPermission( ApplicationPermissions.PERMISSION_EVENT_INJECTOR );
   ```

4. Determine the access control settings that the BlackBerry® device user specifies.

   ```java
   if(ApplicationPermissionsManager.getInstance().invokePermissionsRequest (permissions))
   {
       System.out.println("The user allowed requested permissions.");
   } else
   {
       System.out.println("The user denied requested permissions.");
   }
   ```
Code modules

Retrieve module information

1. Import the net.rim.device.api.system.CodeModuleManager class.
2. To retrieve a handle for a module, invoke getModuleHandle(), and provide the name of the code module as a parameter.
   ```java
   int handle = CodeModuleManager.getModuleHandle("test_module");
   ```
3. To retrieve specific information about a module, invoke the methods of the CodeModuleManager class, and provide the module handle as a parameter to these methods.
   ```java
   String name = CodeModuleManager.getModuleName(handle);
   String vendor = CodeModuleManager.getModuleVendor(handle);
   String description = CodeModuleManager.getModuleDescription(handle);
   int version = CodeModuleManager.getModuleVersion(handle);
   int size = CodeModuleManager.getModuleCodeSize(handle);
   int timestamp = CodeModuleManager.getModuleTimestamp(handle);
   ```
4. To retrieve an array of handles for existing modules on a BlackBerry® device, invoke getModuleHandles().
   ```java
   int[] handles = CodeModuleManager.getModuleHandles();
   String name = CodeModuleManager.getModuleName(handles[0]);
   ```

Access control messages

Displaying a message for an operation that requires user permission

You can use the components of the net.rim.device.api.applicationcontrol package to let a BlackBerry® device application display custom messages to a BlackBerry device user when the application attempts an operation that the user must permit. The application displays information about the type of permission that the user must provide. For example, you can use PERMISSION_PHONE for an operation that requires access to the phone functionality of the device.

You can use the applicationcontrol package to include a custom message with the default message that an application displays in response to an application control.

Display an application control message to a user

A BlackBerry® device application can include more than one registered ReasonProvider. The application displays messages from ReasonProviders in the order that each ReasonProvider registers with the application. For example, if an application registers a ReasonProvider A before ReasonProvider B, the application displays the message from ReasonProvider A, followed by the message from ReasonProvider B.

1. Import the net.rim.device.api.applicationcontrol.ReasonProvider interface.
2. In your implementation of the `ReasonProvider.getMessage(int permissionID)` method, return a `String` value that contains the message to display to the user.
Storing data

Data management

You can store data to persistent memory on the BlackBerry® device. The BlackBerry Persistent Store APIs and the MIDP RMS APIs provide support for JSR 37 and JSR 118. A BlackBerry device that runs BlackBerry® Device Software version 4.2 or later also provides a traditional file system and support for saving content directly to the file system using JSR 75 APIs. With either the BlackBerry Persistent Store APIs or the MIDP RMS APIs, you can store data persistently to flash memory. The data persists even if you remove the battery from the BlackBerry device.

A BlackBerry device that runs BlackBerry® Device Software version 5.0 or later also supports the use of SQLite®. You must use the JSR 75 APIs or SQLite to store data on a media card.

Access to memory

The BlackBerry® Java® environment is designed to inhibit applications from causing problems accidentally or maliciously in other applications or on the BlackBerry device. BlackBerry device applications can write only to the BlackBerry device memory that the BlackBerry Java Virtual Machine uses; they cannot access the virtual memory or the persistent storage of other applications (unless they are specifically granted access to do so). Custom applications can only access persistent storage or user data, or communicate with other applications, through specific APIs. Research In Motion must digitally sign applications that use certain BlackBerry APIs, to provide an audit trail of applications that use sensitive APIs.

File systems and paths

The file systems on a BlackBerry® device have corresponding file paths that you use to access them.

<table>
<thead>
<tr>
<th>File system mounting point</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal flash</td>
<td>file:///store/</td>
</tr>
<tr>
<td>media card</td>
<td>file:///SDCard</td>
</tr>
</tbody>
</table>

File Connections APIs

The File Connection APIs provide a traditional file system, and support for saving data directly to the file system on the BlackBerry® device or to a microSD card. You can view data in the file system and move the data to a computer by using Windows®.
Storage on removable media

Accessing data on the microSD media card

The `javax.microedition.io.file` package supports the JSR 75 File Connection APIs and is used in applications to access the file system for the microSD media card. You can also implement the `FileConnection` interface to access BlackBerry® device ring tones and camera images.

<table>
<thead>
<tr>
<th>Class or interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionClosedException</td>
<td>This exception is thrown when an application invokes a method on a closed file connection.</td>
</tr>
<tr>
<td>FileConnection</td>
<td>An application can use this API to access files or directories.</td>
</tr>
<tr>
<td>FileSystemListener</td>
<td>An application can use this API to receive status notifications when the application adds or removes a file system root.</td>
</tr>
<tr>
<td>FileSystemRegistry</td>
<td>An application can use this API as a central registry for file system listeners that listen for the addition or removal of file systems.</td>
</tr>
<tr>
<td>IllegalModeException</td>
<td>This exception is thrown when a method requires a specific security mode (for example READ or WRITE) and the open connection is not in that mode.</td>
</tr>
</tbody>
</table>

Using the microSD media card with more than one BlackBerry device

If the BlackBerry® device user moves the microSD media card to a BlackBerry device that does not use a device password or uses a password that does not successfully decrypt the microSD media card master key, the BlackBerry device prompts the BlackBerry device user to enter the microSD media card password. If the BlackBerry device has a password, the BlackBerry device user can use the prompt to change the microSD media card password to the BlackBerry device password.

File encryption on microSD cards

IT policies and the microSD media card

You can apply the IT policy Encrypt data written to the microSD media card to any new or modified files that you store on the microSD media card. Only the files that you store on the microSD media card after an administrator sets the IT policy are encrypted. Except for media files, all content is encrypted.

Encryption of data on a microSD media card

When a BlackBerry® Java® Application accesses a file on the microSD memory card, file decryption occurs and the file moves to main memory for an application to read. For a BlackBerry Java Application to access a file that is password protected, the BlackBerry device must not be locked. Encrypted files have a .rem extension and cannot be decrypted on platforms other than BlackBerry.
If the NVRAM is removed and the microSD media card is locked with a BlackBerry device key, the data on the microSD media card is no longer accessible. To remove data that is not accessible, start the BlackBerry device and remove all encrypted media files.

The BlackBerry device uses a master key stored on the microSD media card to encrypt BlackBerry device media files. The master key prevents the BlackBerry device from having to decrypt or re-encrypt all media files when you disable encryption or change the password.

**Code sample: Reading sections of a binary file**

This code sample demonstrates how to read sections of a binary file by reading header information from a .gif file. The application reads the width and height of the image from the header. To run the code sample you must place a .gif file in the root folder of a media card in a BlackBerry® device.

```java
import net.rim.device.api.ui.*;
import net.rim.device.api.io.*;
import javax.microedition.io.file.*;
import javax.microedition.io.*;
import java.io.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public class RandomFileAccess extends UiApplication {
    public static void main(String[] args) {
        RandomFileAccess app = new RandomFileAccess();
        app.enterEventDispatcher();
    }
    public RandomFileAccess() {
        pushScreen(new HomeScreen());
    }
}

class HomeScreen extends MainScreen {
    public HomeScreen() {
        setTitle("Random File Access Sample");
        try {
            FileConnection fc = (FileConnection)Connector.open("file:///SDCard/test.gif");
            boolean bFileExists = fc.exists();
            if (!bFileExists) {
                Dialog.alert("Cannot find specified GIF file.");
                System.exit(0);
            }
            DataInputStream in = fc.openDataInputStream();
            // Read sections of the image from the header...
        } catch (Exception e) {
            // Handle exceptions...
        }
    }
}
```
Persistent data storage

Create a persistent data store

Each PersistentObject has a unique long key.

1. Import the following classes:
   - `net.rim.device.api.system.PersistentObject`
   - `net.rim.device.api.system.PersistentStore`
   - `java.lang.String`
   - `net.rim.device.api.ui.component.Dialog`

2. To create a unique long key, in the BlackBerry® Integrated Development Environment, type a string value.

   com.rim.samples.docs.userinfo

3. Right-click the string and click **Convert 'com.rim.samples.docs.userinfo' to long**.

4. Include a comment in your code to indicate the string that you used to generate the unique long key.

5. To create a persistent data store, create a single static `PersistentObject` and invoke

   `PersistentStore.getPersistentObject`, using the unique long key as a parameter.
static PersistentObject store;
static {
    store = PersistentStore.getPersistentObject( 0xa1a569278238dad2L );
}

Store persistent data

1. Import the following classes:
   • net.rim.device.api.system.PersistentStore
   • net.rim.device.api.system.PersistentObject

2. Invoke setContents() on a PersistentObject. This method replaces existing content with the new content.

3. To save the new content to the persistent store, invoke commit().

   ```java
   String[] userinfo = {username, password};
   synchronized(store) {
       store.setContents(userinfo);
       store.commit();
   }
   ```

4. To use a batch transaction to commit objects to the persistent store, invoke PersistentStore.getSynchObject(). This method retrieves the persistent store monitor that locks the object.
   a. Synchronize on the object.
   b. Invoke commit() as necessary. If any commit in the batch fails, the entire batch transaction fails.

5. To commit a monitor object separately from a batch transaction, invoke forceCommit() while synchronizing the monitor object.

Retrieve persistent data

1. Import the following classes:
   • net.rim.device.api.system.PersistentObject
   • net.rim.device.api.ui.component.Dialog

2. Invoke getContents() on a PersistentObject.

3. To convert to your desired format, perform an explicit cast on the object that PersistentObject.getContents() returns.

   ```java
   synchronized(store) {
       String[] currentinfo = (String[])store.getContents();
       if(currentinfo == null) {
           Dialog.alert(_resources.getString(APP_ERROR));
       } else {
           currentusernamefield.setText(currentinfo[0]);
           currentpasswordfield.setText(currentinfo[1]);
       }
   }
   ```
Remove persistent data

If you delete the .cod file that defines a PersistentStore, then all persistent objects that the .cod file created are deleted.

1. Import the following classes:
   • net.rim.device.api.system.PersistentStore
   • net.rim.device.api.system.PersistentObject

2. To remove all persistent data from a BlackBerry® device application, invoke PersistentStore.destroyPersistentObject(), providing as a parameter a unique key for the PersistentObject.

3. To remove individual data, treat the data as normal objects, and remove references to it. A garbage collection operation removes the data.

MIDP record storage

Create an MIDP record store

1. Import the javax.microedition.rms.RecordStore class.

2. Invoke openRecordStore(), and specify true to indicate that the method should create the record store if the record store does not exist.

   RecordStore store = RecordStore.openRecordStore("Contacts", true);

Add a record to a record store

1. Import the javax.microedition.rms.RecordStore class.

2. Invoke addRecord().

   int id = store.addRecord(_data.getBytes(), 0, _data.length());

Retrieve a record from a record store

1. Import the following classes:
   • javax.microedition.rms.RecordStore
   • java.lang.String

2. Invoke getRecord(int, byte[], int). Pass the following parameters:
   • a record ID
   • a byte array
   • an offset
Retrieve all records from a record store

1. Import the `javax.microedition.rms.RecordStore` class.
2. Import the following interfaces:
   - `javax.microedition.rms.RecordEnumeration`
   - `javax.microedition.rms.RecordFilter`
   - `javax.microedition.rms.RecordComparator`
3. Invoke `openRecordStore()`.
4. Invoke `enumerateRecords()`. Pass the following parameters:
   - `filter`: specifies a `RecordFilter` object to retrieve a subset of record store records (if null, the method returns all records)
   - `comparator`: specifies a `RecordComparator` object to determine the order in which the method returns the records (if null, the method returns the records in any order)
   - `keepUpdated`: determines if the method keeps the enumeration current with the changes to the record store

```java
RecordStore store = RecordStore.openRecordStore("Contacts", false);
RecordEnumeration e = store.enumerateRecords(null, null, false);
```

Collections

Retrieve a collection from persistent storage

1. Import the following classes:
   - `net.rim.device.api.system.PersistentStore`
   - `java.util.Vector`
2. Import the `net.rim.device.api.synchronization.SyncCollection` interface.
3. To provide the BlackBerry® device application with access to the newest `SyncCollection` data from the `PersistentStore`, invoke the `PersistentStore.getPersistentObject()` method using the ID of the `SyncCollection`.

```java
private PersistentObject _persist;
private Vector _contacts;
private static final long PERSISTENT_KEY = 0x266babf899b20b56L;
_persist = PersistentStore.getPersistentObject( PERSISTENT_KEY );
```

4. Store the returned data in a vector object.
   ```java
   _contacts = (Vector)_persist.getContents();
   ```
5. Create a method to provide the BlackBerry device application with the newest `SyncCollection` data before a wireless data backup session begins.
public void beginTransaction()
{
    _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);
    _contacts = (Vector)_persist.getContents();
}

6. Create code to manage the case where the SyncCollection you retrieve from the PersistentStore is empty.

if( _contacts == null )
{
    _contacts = new Vector();
    _persist.setContents( _contacts );
    _persist.commit();
}

---

Create a collection listener to notify the system when a collection changes

The system invokes CollectionEventSource.addCollectionListener() to create a CollectionListener for each SyncCollection the BlackBerry® device application makes available for wireless backup.

1. Import the net.rim.device.api.util.ListenerUtilities class.

2. Import the following interfaces:
   - java.util.Vector
   - net.rim.device.api.collection.CollectionEventSource
   - net.rim.device.api.collection.CollectionListener
   - net.rim.device.api.synchronization.SyncCollection

3. Create a private vector object to store the collection of SyncCollection listeners for the BlackBerry device application.

   private Vector _listeners;
   _listeners = new CloneableVector();

4. Implement the CollectionEventSource.addCollectionListener() method, making sure the method adds a CollectionListener to the Vector that contains listeners. In the following code sample, we implement CollectionEventSource.addCollectionListener() to invoke ListenerUtilities.fastAddListener() to add a listener to the Vector that contains listeners.

   public void addCollectionListener(Object listener)
   {
       _listeners = ListenerUtilities.fastAddListener( _listeners, listener );
   }
Remove a collection listener that notifies the system when a collection changes

When a CollectionListener is no longer required, the system invokes CollectionEventSource.removeCollectionListener.

1. Import the following classes:
   • net.rim.device.api.util.ListenerUtilities
   • java.util.Vector

2. Import the following interfaces:
   • net.rim.device.api.collection.CollectionEventSource
   • net.rim.device.api.collection.CollectionListener

3. Implement the following interfaces:
   • net.rim.device.api.collection.CollectionEventSource
   • net.rim.device.api.collection.CollectionListener

4. Implement the CollectionEventSource.removeCollectionListener() method, using the ListenerUtilities.removeListener() method to remove a CollectionListener from the Vector that contains SyncCollection listeners for the BlackBerry® device application. In the following code sample, we implement CollectionEventSource.removeCollectionListener() to invoke ListenerUtilities.removeListener() to remove a listener from the Vector that contains listeners

   ```java
   public void removeCollectionListener(Object listener)
   {_listeners = ListenerUtilities.removeListener( _listeners, listener );}
   ```

Notify the system when a collection changes

1. Import the net.rim.device.api.collection.CollectionListener interface.

2. To notify the system when an element is added to a SyncCollection, invoke CollectionListener.elementAdded().

   ```java
   for( int i=0; i<_listeners.size(); i++ )
   {
   CollectionListener cl = (CollectionListener)_listeners.elementAt( i );
   cl.elementAdded( this, object );
   } return true;
   }
   ```

3. To notify the system when an element in a SyncCollection is replaced, invoke CollectionListener.elementUpdated().

4. Invoke CollectionListener.elementRemoved().
Runtime storage

BlackBerry® devices use a runtime store as a central location in which BlackBerry Java® Applications can share runtime objects. By default, only BlackBerry Java Applications that Research In Motion digitally signs can access data in the runtime store. Contact RIM for information about how to control access to your data.

The runtime store is not persistent. When you restart the BlackBerry device, the data in the runtime store clears.

Retrieve the runtime store

1. Import the net.rim.device.api.system.RuntimeStore class.
2. Invoke RuntimeStore.getRuntimeStore().

   ```java
   RuntimeStore store = RuntimeStore.getRuntimeStore();
   ```

Add an object in the runtime store

1. Import the following classes:
   • net.rim.device.api.system.RuntimeStore
   • java.lang.String
   • java.lang.IllegalArgumentException
2. Invoke RuntimeStore.put(long, String) and provide as parameters a unique long ID and the runtime object to store.
3. Create a try - catch block to manage the IllegalArgumentException that put() throws if a runtime object with the same ID exists.

   ```java
   RuntimeStore store = RuntimeStore.getRuntimeStore();
   String msg = "Some shared text";
   long ID = 0x60ac754bc0867248L;
   try {
       store.put( ID, msg );
   } catch(IllegalArgumentException e) {
   }
   ```

Replace an object in the runtime store

1. Import the following classes:
   • net.rim.device.api.system.RuntimeStore
   • java.lang.String
   • net.rim.device.api.system.ControlledAccessException
2. Invoke replace().
3. Create a try - catch block to manage the ControlledAccessException that replace() throws if the runtime object with the specified ID does not exist.
RuntimeStore store = RuntimeStore.getRuntimeStore();
String newmsg = "Some new text";
try {
    Object obj = store.replace( 0x60ac754bc0867248L, newmsg);
} catch(ControlledAccessException e) {
} not exist

Retrieve a registered runtime object

1. Import the following classes:
   • net.rim.device.api.system.RuntimeStore
   • net.rim.device.api.system.ControlledAccessException
2. Invoke RuntimeStore.get() and provide as a parameter the runtime object ID.
3. Create a try-catch block to manage the ControlledAccessException that get() throws if the BlackBerry® Java® Application does not have read access to the specified runtime object.

Retrieve an unregistered runtime object

1. Import the following classes:
   • net.rim.device.api.system.RuntimeStore
   • net.rim.device.api.system.ControlledAccessException
   • java.lang.RuntimeException
2. Invoke RuntimeStore.waitFor() to wait for registration of a runtime object to complete.
3. Create code to handle exceptions.
Creating connections

Network connections and transport types

A BlackBerry® device can use various radio communication technologies such as Wi-Fi® technology, CDMA or GPRS to open a wireless connection. The wireless connection is transferred to a wired network and connected to an intranet or the Internet through a proxy or gateway. A BlackBerry device can work with different types of gateways and each gateway provides a unique set of configurable features. You can specify the type of wireless connection and gateway to use.

BlackBerry devices that run BlackBerry® Device Software 5.0 or later include a Network API that is designed to simplify how you open network connections and check the availability and the wireless coverage area of transport types. You can open HTTP, HTTPS, and socket network connections by specifying a URL and an optional list of preferred transport types. If you specify a list of transport types, the Network API checks the availability and the wireless coverage area of each transport type, in the specified order, and tries to open a connection. This process continues until a network connection is opened or the end of the list is reached. If you do not specify a transport type, all available transport types are tried.

Most of the functionality of the Network API is implemented in the ConnectionFactory and TransportInfo classes. You can use the methods that are provided in the TransportInfo class to check for available transport types and associated wireless coverage areas. You can use the ConnectionFactory class to request a network connection. All the Network API classes are provided in the net.rim.device.api.io.transport and net.rim.device.api.io.transport.options packages.

Using the BlackBerry Enterprise Server as an intranet gateway

Organizations host the BlackBerry® Enterprise Server behind their firewall to allow BlackBerry device users to access the organization's intranet. The BlackBerry® Mobile Data System component of the BlackBerry Enterprise Server includes the BlackBerry® MDS Services, which provides an HTTP and TCP/IP proxy service to let third-party Java® applications use it as a highly secure gateway for managing HTTP and TCP/IP connections to the intranet. When you use the BlackBerry Enterprise Server as an intranet gateway, all data traffic between your application and the BlackBerry Enterprise Server is encrypted automatically using AES encryption or Triple DES encryption. Because the BlackBerry Enterprise Server is located behind the organization's firewall and provides inherent data encryption, applications can communicate with application servers and web servers that reside on the organization's intranet.

If your application connects to the Internet rather than an organization's intranet, you might be able to use the BlackBerry Enterprise Server that the device is associated with as a gateway. In this case, network requests travel behind an organization's firewall to the BlackBerry Enterprise Server, which makes the network request to the Internet through the firewall. Organizations can set an IT policy to enforce the use of the BlackBerry Enterprise Server as the gateway for all wireless network traffic, including traffic destined for the Internet.
Using the wireless service provider's Internet gateway

Java® applications for BlackBerry® devices can connect to the Internet using the Internet gateway that the wireless service provider provides. Most wireless service providers provide their own Internet gateway that offers direct TCP/IP connectivity to the Internet. Some operators also provide a WAP gateway that lets HTTP connections occur over the WAP protocol. Java applications for BlackBerry devices can use either of these gateways to establish connections to the Internet. If you write your application for BlackBerry device users who are on a specific wireless network, this approach can often yield good results. However, if you write your application for BlackBerry device users on a variety of wireless networks, testing your application against the different Internet gateways and achieving a consistent and reliable experience can be challenging. In these scenarios, you may find it useful to use the BlackBerry® Internet Service, and use the wireless service provider’s Internet gateway as a default connection type if the BlackBerry Internet Service is not available.

Retrieve the wireless network name

1. Import the following classes:
   - net.rim.device.api.system.RadioInfo
   - java.lang.String
   - net.rim.device.api.ui.Field
2. Invoke RadioInfo.getCurrentNetworkName(). The BlackBerry® device must be connected to a wireless network for this method to work.

```java
String networkName = RadioInfo.getCurrentNetworkName();
System.out.println("Network Name: " + networkName);
```

Connections

Use HTTP authentication

1. Import the following classes:
   - net.rim.device.api.system.CoverageInfo
   - javax.microedition.io.Connector
   - net.rim.device.api.ui.UiApplication
   - net.rim.device.api.ui.component.Dialog
   - java.lang.String
2. Import the following interfaces:
   - javax.microedition.io.HttpConnection
   - net.rim.device.api.system.CoverageStatusListener
   - javax.microedition.io.StreamConnection
3. Use the CoverageInfo class and CoverageStatusListener interface of the net.rim.device.api.system package to verify that the BlackBerry device is in wireless network coverage area.
4. Invoke `Connector.open()`, using the HTTP location of the protected resource.

5. Cast and store the returned object as a `StreamConnection`.

```java
StreamConnection s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt");
```

6. Cast and store the `StreamConnection` object as an `HttpConnection` object.

```java
HttpConnection httpConn = (HttpConnection)s;
```


```java
int status = httpConn.getResponseCode();
```

8. Create code to manage an unauthorized HTTP connection attempt.

```java
int status = httpConn.getResponseCode();
switch (status)
    case (HttpConnection.HTTP_UNAUTHORIZED);
```

9. Create a `run()` method and within it implement a dialog object to ask the BlackBerry device user for login information.

```java
UiApplication.getUiApplication().invokeAndWait(new Runnable())
{
    public void run()
    {
        dialogResponse = Dialog.ask;
        (Dialog.D_YES_NO,"Unauthorized Access:\n Do you wish to log in?");
    }
}
```

10. To process the login information, create code to manage a Yes dialog response.
    a. Retrieve the login information and close the current connection.

```java
if (dialogResponse == Dialog.YES)
    String login = "username:password";
    //Close the connection.
    s.close();
```

b. Encode the login information.

```java
byte[] encoded = Base64OutputStream.encode(login.getBytes(),
0, login.length(), false, false);
```

11. Invoke `HTTPConnection.setRequestProperty()` using the encoded login information to access the protected resource.

```java
s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt ");
httpConn = (HttpConnection)s;
httpConn.setRequestProperty("Authorization", "Basic " + new String(encoded));
```
Use an HTTPS connection

If your BlackBerry device is associated with a BlackBerry® Enterprise Server and uses an HTTPS proxy server that requires authentication, you will not be able to use end-to-end TLS.

1. Import the following classes:
   • net.rim.device.api.system.CoverageInfo
   • javax.microedition.io.Connector
2. Import the following interfaces:
   • net.rim.device.api.system.CoverageStatusListener
   • javax.microedition.io.HttpsConnection
3. Use the CoverageInfo class and CoverageStatusListener interface of the net.rim.device.api.system package to make sure that the BlackBerry device is in a wireless network coverage area.
4. Invoke Connector.open(), specifying HTTPS as the protocol and cast the returned object as an HttpsConnection object to open an HTTP connection.
   ```java
   HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/");
   ```
5. To specify the connection mode, add one of the following parameters to the connection string that passes to Connector.open()
   • Specify that an end-to-end HTTPS connection must be used from the BlackBerry device to the target server: EndToEndRequired
   • Specify that an end-to-end HTTPS connection should be used from the BlackBerry device to the target server. If the BlackBerry device does not support end-to-end TLS, and the BlackBerry device user permits proxy TLS connections, then a proxy connection is used: EndToEndDesired.
   ```java
   HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/;EndToEndDesired");
   ```

Use a socket connection

Although you can implement HTTP over a socket connection, you should use an HTTP connection for the following reasons:
• Socket connections do not support BlackBerry® Mobile Data System features, such as push.
• BlackBerry® device applications that use socket connections typically require significantly more bandwidth than BlackBerry device applications that use HTTP connections.
1. Import the following classes:
   • net.rim.device.api.system.CoverageInfo
   • javax.microedition.io.Connector
   • java.lang.String
   • java.io.OutputStreamWriter
   • java.io.InputStreamReader
2. Import the following interfaces:
   - `net.rim.device.api.system.CoverageStatusListener`
   - `javax.microedition.io.StreamConnection`

3. Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless network coverage area.

4. Invoke `Connector.open()`, specifying `socket` as the protocol and appending the `deviceside=false` parameter to the end of the URL.
   - To open a socket connection using BlackBerry MDS Services, append `deviceside=false` to the end of the URL. BlackBerry device applications must input their local machine IP explicitly because `localhost` is not supported.
     ```java
     private static String URL = "socket://local_machine_IP:4444;deviceside=false";
     StreamConnection conn = null;
     conn = (StreamConnection)Connector.open(URL);
     ```
   - To open a socket connection over direct TCP, append the `deviceside=true` parameter to the end of the URL.
     ```java
     private static String URL = "socket://local_machine_IP:4444;deviceside=true";
     StreamConnection conn = null;
     conn = (StreamConnection)Connector.open(URL);
     ```
   - To open a socket connection over direct TCP, specifying APN information, append the `deviceside=true` parameter to the end of the URL and specify the APN over which the connection will be made. Specify the user name to connect to the APN, and the password if required by the APN.
     ```java
     private static String URL = "socket://local_machine_IP:4444;deviceside=true;apn=internet.com;tunnelauthusername=user165;tunnelauthpassword=user165password";
     StreamConnection conn = null;
     conn = (StreamConnection)Connector.open(URL);
     ```

5. Use `openInputStream()` and `openOutputStream()` to send and receive data.
   ```java
   OutputStreamWriter _out = new OutputStreamWriter(conn.openOutputStream());
   String data = "This is a test";
   _out.write(data, 0, length);
   InputStreamReader _in = new InputStreamReader(conn.openInputStream());
   char[] input = new char[length];
   for ( int i = 0; i < length; ++i ) {
       input[i] = (char)_in.read();
   }
   ```

6. Invoke `close()` on the input and output streams and the socket connection. Each of the `close()` methods throws an `IOException`. Make sure that the BlackBerry device application implements exception handling.
   ```java
   _in.close();
   _out.close();
   conn.close();
   ```
### Use a datagram connection

Datagrams are independent packets of data that applications send over networks. A **Datagram** object is a wrapper for the array of bytes that is the payload of the datagram. Use a datagram connection to send and receive datagrams.

To use a datagram connection, you must have your own infrastructure to connect to the wireless network, including an APN for GPRS networks. Using UDP connections requires that you work closely with service providers. Verify that your service provider supports UDP connections.

1. Import the following classes and interfaces:
   - `net.rim.device.api.system.CoverageInfo`
   - `javax.microedition.io.Connector`
   - `java.lang.String`

2. Import the following interfaces:
   - `net.rim.device.api.system.CoverageStatusListener`
   - `javax.microedition.io.DatagramConnection`
   - `javax.microedition.io.Datagram`

3. Use the `CoverageInfo` class and the `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless network coverage area.

4. Invoke `Connector.open()`, specify udp as the protocol and cast the returned object as a `DatagramConnection` object to open a datagram connection.

   ```java
   (DatagramConnection)Connector.open("udp://host:dest_port[;src_port]/apn");
   ```

   where:
   - **host** is the host address in dotted ASCII-decimal format.
   - **dest-port** is the destination port at the host address (optional for receiving messages).
   - **src-port** is the local source port (optional).
   - **apn** is the network APN in string format.

5. To receive datagrams from all ports at the specified host, omit the destination port in the connection string.

6. To open a datagram connection on a non-GPRS network, specify the source port number, including the trailing slash mark. For example, the address for a CDMA network connection would be `udp://121.0.0.0:2332;6343/`. You can send and receive datagrams on the same port.

7. To create a datagram, invoke `DatagramConnection.newDatagram()`.

   ```java
   Datagram outDatagram = conn.newDatagram(buf, buf.length);
   ```

8. To add data to a diagram, invoke `Datagram.setData()`.

   ```java
   byte[] buf = new byte[256];
   outDatagram.setData(buf, buf.length);
   ```

9. To send data on the datagram connection, invoke `send()` on the datagram connection.
conn.send(outDatagram);

If a BlackBerry®Java® Application attempts to send a datagram on a datagram connection and the recipient is not listening on the specified source port, an IOException is thrown. Make sure that the BlackBerry Java Application implements exception handling.

10. To receive data on the datagram connection, invoke receive() on the datagram connection. The receive() method blocks other operations until it receives a data packet. Use a timer to retransmit the request or close the connection if a reply does not arrive.

    byte[] buf = new byte[256];
    Datagram inDatagram = conn.newDatagram(buf, buf.length);
    conn.receive(inDatagram);

11. To extract data from a datagram, invoke getData(). If you know the type of data that you are receiving, convert the data to the appropriate format.

    String received = new String(inDatagram.getData());

12. Close the datagram connection, invoke close() on the input and output streams, and on the datagram connection object.

    conn.close();

Use a USB or serial port connection

Using a serial or USB connection, BlackBerry® device applications can communicate with desktop applications when they are connected to a computer using a serial or USB port. This type of connection also lets BlackBerry device applications communicate with a peripheral device that plugs into the serial or USB port.

1. Import the following classes:
   • javax.microedition.io.Connector
   • java.io.DataOutputStream
   • java.lang.String
   • java.io.DataInputStream

2. Import the javax.microedition.io.StreamConnection interface.

3. Invoke Connector.open(), and specify comm as the protocol and COM1 or USB as the port to open a USB or serial port connection.

    private StreamConnection _conn = (StreamConnection)Connector.open("comm:COM1;baudrate=9600;Bitsperchar=8;parity=none;stopbits=1");

4. To send data on the USB or serial port connection, invoke openDataOutputStream() or openOutputStream().

    DataOutputStream _dout = _conn.openDataOutputStream();

5. Use the write methods on the output stream to write data.

    private String data = "This is a test";
    _dout.writeChars(data);
6. To receive data on the USB or serial port connection, use a non-main event thread to read data from the input stream. Invoke openInputStream() or openDataInputStream().

   ```java
   DataInputStream _din = _conn.openInputStream();
   Use the read methods on the input stream to read data.
   ```

7. Use the read methods on the input stream to read data.

   ```java
   String contents = _din.readUTF();
   ```

8. To close the USB or serial port connection, invoke close() on the input and output streams, and on the port connection object. The close() method can throw IOExceptions. Make sure the BlackBerry device application implements exception handling.

   ```java
   _din.close();
   _dout.close();
   conn.close();
   ```

### Use a Bluetooth serial port connection

You can use the Bluetooth® API (`net.rim.device.api.bluetooth`) to let your BlackBerry® device application access the Bluetooth Serial Port Profile and initiate a server or client Bluetooth serial port connection to a computer or other Bluetooth enabled device.

1. Import the following classes:
   - `javax.microedition.io.Connector`
   - `net.rim.device.api.bluetooth.BluetoothSerialPort`
   - `java.io.DataOutputStream`
   - `java.io.DataInputStream`
   - `java.lang.String`
   - `java.io.IOException`

2. Import the `javax.microedition.io.StreamConnection` interface.

3. Invoke `Connector.open()`, providing the serial port information that `BluetoothSerialPort.getSerialPortInfo()` returns as a parameter to open a Bluetooth connection.

   ```java
   BluetoothSerialPortInfo[] info = BluetoothSerialPort.getSerialPortInfo();
   StreamConnection _bluetoothConnection = (StreamConnection)Connector.open( info [0].toString(), Connector.READ_WRITE );
   ```

4. To send data on the Bluetooth connection, invoke `openDataOutputStream()` or `openOutputStream()`.

   ```java
   DataOutputStream _dout = _bluetoothConnection.openDataOutputStream();
   ```

5. Use the write methods on the output stream to write data.

   ```java
   private static final int JUST_OPEN = 4;
   _dout.writeInt(JUST_OPEN);
   ```

6. To receive data on the Bluetooth connection, in a non-main event thread, invoke `openInputStream()` or `openDataInputStream()`. Use the read methods on the input stream to read the data.
DataInputStream _din = _bluetoothConnection.openDataInputStream();
String contents = _din.readUTF();

7. Invoke close() on the input and output streams, and on the Bluetooth serial port connection object to close the Bluetooth connection. The close() method can throw IOExceptions. Make sure the BlackBerry device application implements exception handling.

```java
if (_bluetoothConnection != null) {
    try {
        _bluetoothConnection.close();
    } catch (IOException ioe) {
    }
}
if (_din != null) {
    try {
        _din.close();
    } catch (IOException ioe) {
    }
}
if (_dout != null) {
    try {
        _dout.close();
    } catch (IOException ioe) {
    }
}
_bluetoothConnection = null;
_din = null;
_dout = null;
```

Wi-Fi connections

Wireless access families

Working with the BlackBerry® device transceiver involves using APIs that make reference to wireless access families.

<table>
<thead>
<tr>
<th>Wireless access family</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>includes GPRS, EDGE, UMTS®, GERAN, UTRAN, and GAN</td>
</tr>
<tr>
<td>CDMA</td>
<td>includes CDMA1x and EVDO</td>
</tr>
<tr>
<td>WLAN</td>
<td>includes 802.11™, 802.11a™, 802.11b™, 802.11g™</td>
</tr>
</tbody>
</table>

For more information about wireless access families, see the API reference for the BlackBerry® Java® Development Environment
Retrieve the wireless access families that a BlackBerry device supports

1. Import the net.rim.device.api.system.RadioInfo class.
2. Invoke RadioInfo.getSupportedWAFs().

Determine if a BlackBerry device supports multiple wireless access families

1. Import the net.rim.device.api.system.RadioInfo class.
2. Invoke RadioInfo.areWAFsSupported(int wafs).

Determine the wireless access family transceivers that are turned on

1. Import the net.rim.device.api.system.RadioInfo class.
2. Invoke RadioInfo.getActiveWAFs().

Turn on the transceiver for a wireless access family

1. Import the net.rim.device.api.system.Radio class.
2. Invoke Radio.activateWAFs(int WAFs). The WAFs parameter is a bitmask.

Turn off the transceiver for a wireless access family

1. Import the net.rim.device.api.system.Radio class.
2. Invoke Radio.deactivateWAFs(int WAFs). The WAFs parameter is a bitmask.

Check if the Wi-Fi transceiver is turned on

1. Import the net.rim.device.api.system.RadioInfo class.
2. Create an IF statement that tests the value of RadioInfo.WAF_WLAN and the value returned by RadioInfo.getActiveWAFs().
   ```java
   if (( RadioInfo.getActiveWAFs() & RadioInfo.WAF_WLAN ) != 0) { ... }
   ```

Check if the Wi-Fi transceiver is connected to a wireless access point

1. Import the net.rim.device.api.system.WLANInfo class.
2. Create an IF statement that tests the value of WLANInfo.WLAN_STATE_CONNECTED and the value returned by WLANInfo.getWLANState(). The WLANInfo.getWLANState() method checks if a BlackBerry® device has an IP address and can transfer data over a Wi-Fi® network. If the transceiver for the WLAN wireless access family is off, this method returns WLANInfo.WLAN_STATE_DISCONNECTED.
if (WLANInfo.getWLANState() == WLANInfo.WLAN_STATE_CONNECTED) {...}

Retrieve the status of the wireless access point or the active Wi-Fi profile

You can let a BlackBerry® device application retrieve status information such as the data rate of the connection, the wireless LAN standards used (802.11a™, 802.11b™, or 802.11g™), the SSID of the associated access point, or the name of the Wi-Fi profile in use. The transceiver for the WLAN wireless access family must be connected to a wireless access point.

1. Import the net.rim.device.api.system.WLANInfo class.
2. Invoke WLANInfo.getAPInfo(), storing a reference to WLANInfo.WLANAPInfo that this method returns. The WLANInfo.WLANAPInfo object contains a snapshot of the current wireless network.

WLANInfo.WLANAPInfo info = WLANInfo.getAPInfo();

If the BlackBerry device is not connected to an access point, the WLANInfo.getAPInfo() method returns null.

Retrieve the wireless network name

1. Import the following classes:
   - net.rim.device.api.system.RadioInfo
   - java.lang.String
   - net.rim.device.api.ui.Field
2. Invoke RadioInfo.getCurrentNetworkName(). The BlackBerry® device must be connected to a wireless network for this method to work.

   String networkName = RadioInfo.getCurrentNetworkName();
   System.out.println("Network Name: " + networkName);

Open a Wi-Fi socket connection

The interface=wifi parameter applies only to TCP/UDP connections. To establish a Wi-Fi® connection and use a Wi-Fi API in a BlackBerry® device application, the wireless service provider must support Wi-Fi access.

1. Import the following classes:
   - java.lang.String
   - javax.microedition.io.Connector
2. Import the javax.microedition.io.StreamConnection interface.
3. Invoke Connector.open(), specify socket as the protocol, and append the deviceside=true parameter and the interface=wifi parameter to the end of the URL string value.

   private static String URL = "socket://local_machine_IP:4444;deviceside=true;interface=wifi";
   StreamConnection conn = null;
   conn = (StreamConnection)Connector.open(URL);
Open a Wi-Fi HTTP connection

The `interface=wifi` parameter applies only to TCP/UDP connections. To establish a Wi-Fi® connection and use a Wi-Fi API in a BlackBerry® device application, the wireless service provider must support Wi-Fi access.

1. Import the following classes:
   - `java.lang.String`
   - `javax.microedition.io.Connector`

2. Import the `javax.microedition.io.HttpConnection` interface.

3. Invoke `Connector.open()`, specify `http` as the protocol, and append the `interface=wifi` parameter to the end of the URL string value.

4. Cast the returned object as an `HttpConnection` or a `StreamConnection` object.

```java
HttpConnection conn = null;
String URL = "http://www.myServer.com/myContent;deviceside=true;interface=wifi";
conn = (HttpConnection)Connector.open(URL);
```

Open a Wi-Fi HTTPS connection

The `interface=wifi` parameter applies only to TCP/UDP connections. To establish a Wi-Fi® connection and use a Wi-Fi API in a BlackBerry® device application, the wireless service provider must support Wi-Fi access.

1. Import the following classes:
   - `java.lang.String`
   - `javax.microedition.io.Connector`

2. Import the `javax.microedition.io.HttpsConnection` interface.

3. Invoke `Connector.open()`, specify `https` as the protocol, and append the `interface=wifi` parameter to the end of the URL string value.

4. Cast the returned object as an `HttpsConnection` object.

```java
HttpsConnection conn = null;
String URL = "https://host:443/;deviceside=true;interface=wifi";
conn = (HttpsConnection)Connector.open(URL);
```

Enhanced Network API

You can specify how your application opens a connection to a wireless network by using the enhanced Network API that is provided in the `net.rim.device.api.io.transport` and `net.rim.device.api.io.transport.options` packages. The Network API simplifies how your application establishes a connection to a wireless network and enables you to find the wireless transport types available on the BlackBerry® device. For example, you can use the Network API to find the availability and signal strength of a Wi-Fi® network that your application can use to open an HTTP connection.
You can use the Network API to specify a preferred list of wireless transport types. For example, if a Wi-Fi network is not available or does not have sufficient signal strength, you can have your application search for other transport types, such as a cellular network. If you do not specify a transport type, by default the BlackBerry device searches for all available transport types.

For more information about the Network API, see the *Blackberry Java Application Development Guide* and the API reference for the BlackBerry® Java® Development Environment.

**Code sample: Displaying available transport types**

```java
/*
* DisplayAvailableTransportTypesScreen.java
*
* Research In Motion Limited proprietary and confidential
* Copyright Research In Motion Limited, 2009
*/
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.io.transport.*;
import java.lang.StringBuffer;

public class DisplayAvailableTransportTypes extends UiApplication
{
    public static RichTextField _message = new RichTextField("";

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

    public DisplayAvailableTransportTypes()
    {
        pushScreen(new DisplayAvailableTransportTypesScreen());
    }
}

class DisplayAvailableTransportTypesScreen extends MainScreen
{
    private RichTextField strMessage = new RichTextField("";

    public DisplayAvailableTransportTypesScreen()
    {
        LabelField title = new LabelField("Transport Types Sample",
        LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Detected the following transport types:
    int TransportTypes[] = TransportInfo.getAvailableTransportTypes();
    StringBuffer sbTmp = new StringBuffer("";
```
for(int i=0; i<TransportTypes.length; i++)
{
    int tt = TransportTypes[i];
    switch(tt)
    {
        case TransportInfo.TRANSPORT_BIS_B:
            sbTmp.append(" BIS-B\n"); break;
        case TransportInfo.TRANSPORT_MDS:
            sbTmp.append("* MDS\n"); break;
        case TransportInfo.TRANSPORT_TCP_CELLULAR:
            sbTmp.append("* TCP Cellular\n"); break;
        case TransportInfo.TRANSPORT_TCP_WIFI:
            sbTmp.append("* Wifi\n"); break;
        case TransportInfo.TRANSPORT_WAP:
            sbTmp.append("* WAP 1.0 or 1.1\n"); break;
        case TransportInfo.TRANSPORT_WAP2:
            sbTmp.append("* WAP 2.0\n"); break;
    }
    strMessage.setText(sbTmp.toString());
    add(strMessage);
}

Code sample: Creating a connection over HTTP by using the first available transport

/*
 * HTTPFirstAvailable.java
 * Research In Motion Limited proprietary and confidential
 * Copyright Research In Motion Limited, 2010
 */
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.io.transport.*;
import javax.microedition.io.*;
import java.io.*;

public class HTTPFirstAvailable extends UiApplication
{
    public static void main(String[] args)
    {
        HTTPFirstAvailable theApp = new HTTPFirstAvailable();
        theApp.enterEventDispatcher();
    }
```java
public HTTPFirstAvailable()
{
    pushScreen(new HTTPFirstAvailableScreen());
}
}

class ConnectionThread extends Thread
{
    public void run()
    {
        ConnectionFactory connFact = new ConnectionFactory();
        ConnectionDescriptor connDesc;
        connDesc = connFact.getConnection("http://www.example.com");
        if (connDesc != null)
        {
            HttpConnection httpConn;
            httpConn = (HttpConnection)connDesc.getConnection();
            try
            {
                final int iResponseCode = httpConn.getResponseCode();
                UiApplication.getUiApplication().invokeLater(new Runnable()
                {
                    public void run()
                    {
                        Dialog.alert("Response code: " +
                        Integer.toString(iResponseCode));
                    }
                });
            } catch (IOException e)
            {
                System.err.println("Caught IOException: "
                + e.getMessage());
            }
        }
    }
}

class HTTPFirstAvailableScreen extends MainScreen
{
    public HTTPFirstAvailableScreen()
    {
        setTitle("HTTP First Sample");
        add(new RichTextField("Trying to make HTTP connection...
    ");
        ConnectionThread ct = new ConnectionThread();
        ct.start();
    }
}
Controlling access to APIs and application data

Check if a code signature is required

Research In Motion tracks the use of sensitive APIs in the BlackBerry® Java® Development Environment for security and export control reasons.

Locate the item in the API reference for the BlackBerry Java Development Environment. If the item has a lock icon or is noted as ‘signed’ your BlackBerry device application requires a signed key or signature, which RIM provides, before you can load the BlackBerry device application .cod files onto a BlackBerry device.

Java APIs with controlled access

RIM controls Runtime APIs, BlackBerry® Application APIs, and BlackBerry Cryptography APIs.

You can test BlackBerry device applications that use controlled APIs in the BlackBerry® Smartphone Simulator without code signatures; however, you must obtain code signatures from RIM before you can load the BlackBerry device applications onto BlackBerry devices.

If you use any of the following BlackBerry API packages, your BlackBerry device application requires code signatures before you can load it onto a BlackBerry device:

- net.rim.blackberry.api.browser
- net.rim.blackberry.api.invoke
- net.rim.blackberry.api.mail
- net.rim.blackberry.api.mail.event
- net.rim.blackberry.api.menuitem
- net.rim.blackberry.api.options
- net.rim.blackberry.api.pdap
- net.rim.blackberry.api.phone
- net.rim.blackberry.api.phone.phonelogs
- net.rim.device.api.browser.field
- net.rim.device.api.browser.plugin
- net.rim.device.api.crypto
- net.rim.device.api.io.http
- net.rim.device.api.notification
- net.rim.device.api.servicebook
- net.rim.device.api.synchronization
- net.rim.device.api.system
Register to use controlled APIs

2. Save the .csi file that Research In Motion sends to you in an email message. The .csi file contains a list of signatures and your registration information. If the BlackBerry® Signing Authority Tool administrator does not provide you with the .csi file or the Client PIN and you are an ISV partner, contact your ISV Technical Partnership Manager. If you are not an ISV partner, send an email message to jde@rim.com.
3. Double-click the .csi file.
4. If a dialog box appears that states that a private key cannot be found, complete steps 5 to 8 before you continue. Otherwise, proceed to step 9.
5. Click Yes to create a new key pair file.
6. In the Private Key Password field, type a password of at least eight characters, and type it again to confirm. The private key password protects your private key. If you lose this password, you must register again with RIM. If this password is stolen, contact RIM immediately.
7. Click OK.
8. Move your mouse to generate data for a new private key.
9. In the Registration PIN field, type the PIN that RIM provided.
10. In the Private Key Password field, type the private key password.
11. Click Register.
12. Click Exit.

Restrictions on code signatures

The BlackBerry® Signing Authority Tool administrator might place restrictions on your .csi file to limit your access to code signatures. To request changes to these restrictions, contact your administrator.

<table>
<thead>
<tr>
<th>.csi file restriction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of requests</td>
<td>This restriction specifies the maximum number of requests you can make using a particular .csi file. When you reach the maximum number of requests, the .csi file becomes invalid. To make new code signature requests, you must apply for a new .csi file. Although an administrator can permit an infinite number of requests, the number of requests is often specified to be a finite number for security reasons.</td>
</tr>
<tr>
<td>expiry date</td>
<td>This restriction specifies the expiry date for your .csi file. After the expiry date, you can no longer apply for code signatures with this .csi file. To make new signature requests, you must apply for a new .csi file.</td>
</tr>
</tbody>
</table>
Request a code signature

The BlackBerry® Signature Tool is included in the BlackBerry® Java® Development Environment. The BlackBerry JDE is available for download at http://www.blackberry.com/developers. The Web Signer application is installed when you install the BlackBerry® Signing Authority Tool.

For more information about the Web Signer application, see the BlackBerry Signing Authority Tool version 1.0 - Password Based Administrator Guide.

Before you begin: You must obtain a .csi file from Research In Motion.

1. In Windows® Internet Explorer®, locate the .cod file for the BlackBerry device application for which you are requesting a signature.
2. Make sure that a .csl file with the same name as the .cod file exists in the same folder as the .cod file. The BlackBerry® Integrated Development Environment compiler automatically generates the .csl file.
3. Double-click the .cod file to add it to the signature list. The signature list contains information on the .cod files that you want permission to access and are requesting signatures for.
4. Repeat steps 1 through 3 for each .cod file that you want to add to the signature list.
5. On the BlackBerry Signature Tool menu, click Request.
6. In the dialog box, type your private key password.
7. Click OK.

The BlackBerry Signature Tool uses the private key password to append the signature to the request, and it sends the signature list of .cod files to the Web Signer application for verification.

Register a signature key using a proxy server

You can register each .csi file only once.

1. At the command prompt, navigate to the BlackBerry® Signature Tool bin directory. For example:
   C:\Program Files\Research In Motion\BlackBerry JDE 4.6.0\bin
2. Type Java -jar -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80 SignatureTool.jar SigKey.csi with the following parameters:
   - SigKey: The name of each signature key (.csi) file. Use the following naming conventions for the keys: client-RRT-*.csi, client-RBB-*.csi, client-RCR-*.csi.
   - Dhttp.proxyHost: The name or IP address of the proxy server.
   - Dhttp.proxyPort: The proxy server port number if you do not specify 80 as the default port number.
3. Repeat step 2 for each .csi file that you want to register.
Sign an application using a proxy server

Your registration key and .csk file are stored together. If you lose the registration key or the .csk file, you cannot request code signatures. If you are not an ISV partner, contact your ISV Technical Partnership Manager. If you are a non ISV partner, send an email message to jde@rim.com.

1. At the command prompt, navigate to the BlackBerry® Signature Tool bin directory. For example:
   C:\Program Files\Research In Motion\BlackBerry JDE 4.6.0\bin
2. Type `Java -jar -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80 SignatureTool.jar`
3. In the File Selection window, select the .cod file(s) to sign.
4. Click Open.

View the signature status for an application

For files that are not signed, the Status column contains Failed. The Web Signer might have rejected the .cod file because the private key password was typed incorrectly.

1. Start the BlackBerry® Signature Tool.
2. Select a .cod file.
3. View the Status column.
   For files the Web Signer has signed, the Status column contains Signed.

Using keys to protect APIs and data

To create an internal key pair to use with an internal signing authority system, or an external key pair to use with an external signing authority system, you must apply protection to sensitive APIs, protect data in the runtime store, and protect data in a persistent object.

The RSAE.key is an external key, and ACMI.key is an internal key.

Protect APIs using code signing keys

1. After you receive an internal key, an external key, or both keys, in the BlackBerry® Integrated Development Environment, open the project that contains the APIs that you want to control access to.
2. In the Workspace window, right-click the project file.
3. Click Add File to Project.
4. In the Look In field, browse to C:\Program Files\Research In Motion\BlackBerry Password Based Code Signing Authority\data or the location where the .key file is saved.
5. Perform one of the following tasks:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| Use an internal key    | a. Select the internal .key file, for example, the ACMI.key file.  
|                        | b. Click Open.  
|                        | c. In the Workspace window, double click the .key file.  
|                        | d. Select the Use as default for public classes option and Use as default for non-public classes settings.  
|                        | e. Click OK.  
|                        | f. In the Workspace window, right-click the project file.  
|                        | g. Click Add File to Project.  
|                        | h. In the Look In field, browse to C:\Program Files \Research In Motion\BlackBerry Password Based Code Signing Authority\data.                                                                                                                                               |
| Use an external key    | a. Select the external .key file, for example, the RSAE.key file.  
|                        | b. Click Open.  
|                        | c. In the Packages and classes protection window, find the name of the package that contains the sensitive API items.  
|                        | d. Expand the package contents.  
|                        | e. Select each API element that requires access control.  
| 6.                     | Click Ok.  
| 7.                     | Re-compile the project.                                                                                                                                                                                                                                                      |

### Protect runtime store data using code signing keys

1. Import the following classes:
   - java.util.Hashtable
   - net.rim.device.api.system.RuntimeStore

2. Create a hash ID for the object you want to store in the runtime store.

   ```java
   long MY_DATA_ID = 0x33abf322367f9018L;  
   Hashtable myHashtable = new Hashtable();
   ```

3. Store the object in the runtime store and protect the object with the CodeSigningKey object. Only applications signed with the key can read or change the object.

   ```java
   RuntimeStore.put( MY_DATA_ID, new ControlledAccess( myHashtable, key ) );
   ```

4. Make sure that the object is protected with a particular code signing key, invoke RuntimeStore.get, providing as parameters the hash ID for the object and the CodeSigningKey object.
Protect persistent data using code signing keys

1. Import the following classes:
   - java.util.Hashtable
   - net.rim.device.api.system.PersistentObject

2. Create a hash ID for the object you want to store in a persistent object.
   ```java
   long MY_DATA_ID = 0x33abf322367f9018L;
  Hashtable myHashtable = new Hashtable();
   ```

3. Store the object in the persistent object and protect the object with the CodeSigningKey object. For example, after a BlackBerry device application runs the following line of code, only code files that are signed with the RSAE.key file can read or overwrite the object in the persistent object.
   ```java
   persistentObject.setContents( new ControlledAccess( myHashtable, key ) );
   ```

4. Make sure that the object is protected, invoke `getContents` using the CodeSigningKey object as a parameter.
   ```java
   Hashtable myHashtable = (Hashtable) persistentObject.getContents( key );
   ```
Testing a BlackBerry device application

Testing applications on a BlackBerry Smartphone Simulator

After you develop and compile your application, you can test it on the BlackBerry® device. The most common first step is to set the BlackBerry® Java® Development Environment to use a BlackBerry® Smartphone Simulator. The BlackBerry Smartphone Simulator runs the same Java code as the BlackBerry devices, so the BlackBerry Smartphone Simulator provides an accurate environment for testing how applications will function on a BlackBerry device. The BlackBerry JDE includes current versions of the BlackBerry Smartphone Simulator. To download additional versions of the BlackBerry Smartphone Simulator, visit http://www.blackberry.com/developers/index.shtml.

Testing applications on a BlackBerry device

After you test your application on the BlackBerry® Smartphone Simulator, you can install your application on a BlackBerry device. If your application uses signed APIs, you might need code signing keys. After you install the application on the BlackBerry device, you can open the application and test its functionality and performance.

For debugging purposes, you can attach your device to the BlackBerry® Integrated Development Environment and use the debugging tool to step through your application code. The BlackBerry IDE can be useful if you are trying to identify a network or Bluetooth® issue, or other issues that are difficult to simulate.

Testing applications using the compiled .cod files

When you build a project using the BlackBerry® Integrated Development Environment, the BlackBerry IDE compiles your source files into Java® bytecode, performs preverification, and creates a single .cod file and .jad file for a BlackBerry® device application.

If a BlackBerry device application contains more than 64 KB of bytecode or resource data, the BlackBerry IDE creates a .cod file that contains sibling .cod files. Only the BlackBerry® Browser supports wireless installation of a .cod file that contains sibling .cod files. To determine if a .cod file contains sibling .cod files, extract the contents of the .cod file. Any .cod files within the original .cod file are the sibling files.

To identify modules that a BlackBerry device application requires, but are not provided with it, examine the Java® application descriptor (.jad) file RIM-COD-Module-Dependencies attribute.
Install and remove a .cod file for testing

To load, remove, or save .cod files when testing a BlackBerry® device application, use the JavaLoader tool, included with the BlackBerry® Java® Development Environment. For production applications, use the BlackBerry® Desktop Software. You must load BlackBerry device applications with dependencies in the correct order. If project A is dependent on project B, load the project B .cod file before loading the project A .cod file.

Save a .cod file from a device to a computer

To load, remove, or save .cod files when testing a BlackBerry® device application, use the JavaLoader tool, included with the BlackBerry® Java® Development Environment.

1. Connect the BlackBerry device to the computer.
2. Open a command prompt, and navigate to the location of the JavaLoader.exe file.
3. Perform one of the following actions:

<table>
<thead>
<tr>
<th>Task</th>
<th>Steps</th>
</tr>
</thead>
</table>
| Save a BlackBerry device application .cod file from the BlackBerry device to your computer. | Issue a command using the following format: javaloader save .cod file  
For example: javaloader.exe save MyApplication.cod |
| Save BlackBerry device application .cod files listed in the same .jad file from the BlackBerry device to your computer. | Issue a command using the following format: javaloader save .jad file  
For example: javaloader.exe save MyApplication.jad |
| Save BlackBerry device application .cod files stored in the same CodeModuleGroup from the BlackBerry device to your computer. | Issue a command using the following format: javaloader save [-g] module  
For example: javaloader.exe save -g MyApplication |

Retrieve information about a .cod file

To load, remove, or save .cod files when testing a BlackBerry® device application, use the JavaLoader tool, included with the BlackBerry® Java® Development Environment.

1. Connect the BlackBerry® device to the computer.
2. Open a command prompt and navigate to the location of the JavaLoader.exe file.
3. Perform one of the following actions:
<table>
<thead>
<tr>
<th>Task</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve the Name, Version, Size, and Date created information for a .cod file.</td>
<td>Issue a command using the following format: <code>javaloader info .cod file</code>&lt;br&gt;For example: <code>javaloader.exe info MyApplication.cod</code></td>
</tr>
<tr>
<td>Retrieve a list of .cod files that a .cod file requires to run.</td>
<td>Issue a command using the following format: <code>javaloader info -d .cod file</code>&lt;br&gt;For example: <code>javaloader.exe info -d MyApplication.cod</code></td>
</tr>
<tr>
<td>Retrieve information on&lt;br&gt;• sibling .cod files&lt;br&gt;• size of code section&lt;br&gt;• size of data section&lt;br&gt;• size of initialized data&lt;br&gt;• number of class definitions&lt;br&gt;• list of signatures applied to a .cod file</td>
<td>Issue a command using the following format: <code>javaloader info -v .cod file</code>&lt;br&gt;For example: <code>javaloader.exe info -v MyApplication.cod</code></td>
</tr>
</tbody>
</table>
Packaging and distributing a BlackBerry Java Application

Preverify a BlackBerry device application

To reduce the amount of processing the BlackBerry® device performs when you load your BlackBerry device application, partially verify your classes. You may also use the BlackBerry® Smartphone Simulator to preverify .cod files.

At the command prompt, type:

```
preverify.exe [-d] output -classpath directory input; directory
```

Application distribution over the wireless network

You can distribute your applications over the wireless network to help provide a better experience to BlackBerry® device users and to simplify application distribution to a large group of people since you do not require a computer application. A BlackBerry device user can install your applications over the wireless network.

Wireless pull (user-initiated)

You can post compiled applications on a public or private web site. BlackBerry® device users can visit the web site to download the applications over the wireless network by using the browser on their BlackBerry devices. The browser prompts the users to install the application and then the application downloads over the wireless network and installs on the BlackBerry device.

Wireless push (server-initiated)

In the BlackBerry® Enterprise Server environment, the administrator can push applications to BlackBerry device users over the wireless network for mandatory installation. The administrator creates a new policy and specifies that the BlackBerry device requires the application. The application is pushed to users without any user interaction required. Organizations might find this approach useful when sending new applications to a large number of BlackBerry device users.
Distributing BlackBerry Java Applications over the wireless network

Extract sibling .cod files.

To ensure a BlackBerry® device user does not override the original .cod file, on the content server, extract the sibling .cod files into a different directory than the directory where the original file exists.

1. Unzip the original .cod file and extract the sibling .cod files.
2. Place each sibling .cod file on a content server.
3. In the .jad file, list the sibling .cod files separately. Use the following naming convention for sibling .cod files: name of original .cod file - sequential number.
4. Create a RIM-COD-URL-<#> parameter for each sibling .cod file, and place the name of the sibling file to the right of this parameter. # is a number that starts at 1 and increases by 1 for each sibling file. Give each sibling .cod files the same name as the original .cod file, followed by -<#>.
5. Create a RIM-COD-Size-<#> parameter for each sibling .cod file, and place the size of the sibling file to the right of this parameter. # is the same number that is appended to the name of the sibling file. Place the RIM-COD-Size-<#> parameter immediately below the RIM-COD=URL-<#> parameter.

Example: Listing sibling .cod files in a .jad file

The following example contains two sibling files named myApp-1.cod and myApp-2.cod, after the original .cod file myAPP. The developer appends the `.cod` file extension to each sibling file name. The developer creates a RIM-COD-Size-<#> parameter for each sibling file.

```
Manifest-Version: 1.0
MIDlet-Version: 1.0.0
MIDlet-1: ,
RIM-COD-Module-Dependencies: net_rim_cldc
MicroEdition-Configuration: CLDC-1.0
RIM-COD-Module-Name: MyApp
MIDlet-Name: My Application
RIM-COD-URL: myApp.cod
RIM-COD-Size: 55000
RIM-COD-URL-1: myApp-1.cod
RIM-COD-Size-1: 50000
RIM-COD-URL-2: myApp-2.cod
RIM-COD-Size-2: 25000
MicroEdition-Profile: MIDP-1.0
```

Modifying information for a MIDlet suite

You can use the Updatejad tool, part of the BlackBerry® Java® Development Environment, to process .jad files and perform the following actions:
• Correct the .cod file sizes listed in a .jad file. The .cod file sizes listed in the .jad file change after you use the BlackBerry® Signing Authority Tool to sign .cod files.
• Create .jad files that reference multiple .cod files.

Use the Updatejad tool only on .jad files created using the BlackBerry® Integrated Development Environment or the RAPC command-line tool, and signed using the BlackBerry Signing Authority Tool.

The Updatejad tool commands have the following format:

```
updatejad.exe -q -n input.jad [additional.jad]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-q</td>
<td>This option suppresses the creation of success output messages for .jad file processing operations. If an error occurs during .jad file processing, a non-zero exit code is produced.</td>
</tr>
<tr>
<td>-n</td>
<td>This option suppresses the backup of the original .jad file.</td>
</tr>
<tr>
<td><code>input.jad</code></td>
<td>This option specifies the .jad file to update.</td>
</tr>
<tr>
<td><code>additional.jad</code></td>
<td>This option specifies other attributes to add to the input.jad file.</td>
</tr>
</tbody>
</table>

For more information, see the BlackBerry Integrated Development Environment Help or the BlackBerry Signing Authority Tool version 1.0 - Password Based Administrator Guide.

**Properties of BlackBerry device application .jad files**

The BlackBerry® Integrated Development Environment lets you create a dual-purpose .jad file to support the downloading of MIDlets onto BlackBerry devices and other wireless devices. To do this, create a .jad file that contains both the RIM-COD-URL and RIM-COD-Size attributes and the MIDlet-Jar-URL and MIDlet-Jar-Size attributes. On BlackBerry devices, download the .cod files; on other wireless devices, download the .jar files.

<table>
<thead>
<tr>
<th>Required RIM attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIM-COD-Creation-Time</td>
<td>creation time of the .cod file</td>
</tr>
<tr>
<td>RIM-COD-Module-Dependencies</td>
<td>list of modules that the .cod file requires</td>
</tr>
<tr>
<td>RIM-COD-Module-Name</td>
<td>name of the module that the .cod file contains</td>
</tr>
<tr>
<td>RIM-COD-SHA1</td>
<td>SHA1 hash of the .cod file</td>
</tr>
<tr>
<td>RIM-COD-Size</td>
<td>size (in bytes) of the .cod file</td>
</tr>
<tr>
<td>RIM-COD-URL</td>
<td>URL from which the .cod file can be loaded</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional RIM attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIM-Library-Flags</td>
<td>reserved for use by Research In Motion</td>
</tr>
<tr>
<td>RIM-MIDlet-Flags</td>
<td>reserved for use by RIM</td>
</tr>
<tr>
<td>RIM-MIDlet-NameResourceBundle</td>
<td>name of the resource bundle on which the BlackBerry device application depends</td>
</tr>
</tbody>
</table>
Correct the .cod file sizes listed in a .jad file.

1. Use the BlackBerry® Integrated Development Environment to create two BlackBerry® device application files, for example, test.cod and test.jad.
2. Use the BlackBerry® Signing Authority Tool to sign the .cod file.
3. From a command-prompt, navigate to the location of the Updatejad tool.
4. Type a command to correct the .cod file sizes listed in test.jad.
   
   updatejad.exe test.jad

Create .jad files that reference multiple .cod files.

1. Use the BlackBerry® Integrated Development Environment to create two BlackBerry® device application files, for example, lib.cod and lib.jad.
2. Use the BlackBerry® Signing Authority Tool to sign the .cod file.
3. Use the BlackBerry IDE to create two other BlackBerry device application files that use the .jad file, for example, test.cod and test.jad.
4. Use the BlackBerry Signing Authority Tool to sign the new .cod file.
5. From a command-prompt, navigate to the location of the Updatejad tool.
6. Type a command to add the .cod file names from the first .jad file to the new one.

   updatejad.exe test.jad lib.jad

Distributing BlackBerry device applications with the BlackBerry Desktop Software

Elements in BlackBerry device application .alx file

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>id</td>
<td>The application element contains the elements for a single BlackBerry® device application.</td>
</tr>
<tr>
<td></td>
<td>platformVersion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>blackBerryVersion</td>
<td></td>
</tr>
</tbody>
</table>
### Element | Attributes | Description
--- | --- | ---
|  |  | The application element can also contain additional nested application elements. Nesting lets you require that, when a BlackBerry device application is loaded on the BlackBerry device, its prerequisite modules are also loaded on the BlackBerry device. The `id` attribute specifies a unique identifier for the BlackBerry device application. To provide uniqueness, use an ID that includes your company domain in reverse order. For example, `com.rim.samples.docs.helloworld`. The `platformVersion` attribute specifies the version of the operating system software on a BlackBerry device that a BlackBerry device application requires. The `blackBerryVersion` attribute specifies the version of the BlackBerry® Device Software that a BlackBerry device application requires.
|  |  | The `copyright` element provides copyright information, which appears in the application loader tool of the BlackBerry® Desktop Manager.
|  |  | The `description` element provides a brief description of the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager.
| directory | platformVersion | The `directory` element provides the location of a set of files. The `directory` element is optional. If you do not specify a directory, the files must exist in the same location as the .alx file. The `directory` element specifies the directory relative to the location of the .alx file. `directory` elements are cumulative within a BlackBerry device application.
| directory | BlackBerryVersion | For example:

```xml
<application id="com.abc.my.app">
<directory>MyCodFiles</directory>
<fileset Java="1.0">
<files>
a.cod //resolves to <.alx location>
\MyCodFiles
b.cod
</files>
</fileset>
<directory>MyCodFiles</directory>
<fileset Java="1.0">
```
<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>files</td>
<td>—</td>
<td>The <code>files</code> element provides a list of one or more BlackBerry device application .cod files, in a single directory, to load on the BlackBerry device.</td>
</tr>
<tr>
<td>fileset</td>
<td>Java, radio, langid, Colour, platformVersion, blackBerryVersion</td>
<td>The <code>fileset</code> element includes an optional <code>directory</code> element and one or more <code>files</code> elements. It specifies a set of .cod files, in a single directory, to load onto the BlackBerry device. To load files from more than one directory, include one or more <code>fileset</code> elements in the .alx file.</td>
</tr>
</tbody>
</table>

The `platformVersion` attribute specifies the version of the operating system software on a BlackBerry device that a BlackBerry device application requires.

The `blackBerryVersion` attribute specifies the version of the BlackBerry Device Software that a BlackBerry device application requires.

The `files` element resolves to the .alx location and contains the following directory:

```
c.cod //resolves to .alx location
\MyCodFiles\MyCodFiles
d.cod
```

The `Java` attribute specifies the minimum version of the BlackBerry® Java® Virtual Machine with which the .cod files are compatible. The `Java` attribute is required.

The `radio` attribute lets you load different BlackBerry device applications or modules on the BlackBerry device depending on the network type of the BlackBerry device. Possible values include Mobitex, DataTAC, GPRS, CDMA, and IDEN. The `radio` attribute is optional.

The `langid` attribute lets you load different BlackBerry device applications or modules depending on the language support that BlackBerry device users add to the BlackBerry device. The value is a Win32 langid code; for example: 0x0009 (English), 0x0007 (German), 0x000a (Spanish), and 0x000c (French). The `langid` attribute is optional.

The `colour` attribute lets you load different BlackBerry device applications or modules for color or monochrome displays. The value is a Boolean; `true` means color display and `false` means monochrome.
<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The <code>platformVersion</code> attribute specifies the version of the operating system software on a BlackBerry device that a BlackBerry device application requires.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The <code>blackberryVersion</code> attribute specifies the version of the BlackBerry Device Software that a BlackBerry device application requires.</td>
</tr>
<tr>
<td>hidden</td>
<td>—</td>
<td>The <code>hidden</code> element hides a package so that it does not appear to BlackBerry device users in the Application Loader. To hide a package, add the following line: <code>&lt;hidden&gt;true&lt;/hidden&gt;</code>. Use this element in conjunction with the <code>required</code> element to load the BlackBerry device application by default, or set the <code>requires</code> tag to load this package if another BlackBerry device application exists. Only corporate system administrators should use the <code>hidden</code> tag. This tag is not intended for use by third-party software vendors. The BlackBerry Desktop Software version 3.6 or later supports this element.</td>
</tr>
<tr>
<td>language</td>
<td>langid</td>
<td>The <code>language</code> element lets you override the text that appears in the Application Loader when the Application Loader runs in the language that the <code>langid</code> attribute specifies. To support multiple languages, specify multiple <code>language</code> elements. To specify the name, description, version, vendor, and copyright elements for each language, nest these elements in the <code>language</code> element. If you do not nest a element, the text appears in the default language. The <code>langid</code> attribute specifies the Win32 <code>langid</code> code for the language to which this information applies. For example, some Win32 <code>langid</code> codes are: 0x0009 (English), 0x0007 (German), 0x000a (Spanish), and 0x000c (French).</td>
</tr>
<tr>
<td>library</td>
<td>id</td>
<td>You can use the <code>library</code> element instead of the <code>application</code> element. It contains the elements for a single library module. You cannot nest modules. By default, a library module does not appear in the application loader tool of the BlackBerry Desktop Manager.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Typically, use the library element as the target of a <code>&lt;requires&gt;</code> element, so that when a particular BlackBerry device application loads onto the BlackBerry device, a required library also loads onto the BlackBerry device. BlackBerry Desktop Software version 3.6 or later supports this element.</td>
</tr>
<tr>
<td>loader</td>
<td>version</td>
<td>The <code>loader</code> element contains one or more application element. The <code>version</code> attribute specifies the version of the application loader tool of the BlackBerry Desktop Manager.</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>The <code>name</code> element provides a descriptive name for the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager.</td>
</tr>
<tr>
<td>required</td>
<td></td>
<td>The <code>required</code> element lets you force an application to load. The application loader tool of the BlackBerry Desktop Manager selects the BlackBerry device application for installation, and the BlackBerry device user cannot change this selection. Add the following line: <code>&lt;required&gt;true&lt;/required&gt;</code>. Only corporate system administrators should use the <code>required</code> tag. This tag should not be used by third-party software vendors. BlackBerry Desktop Software version 3.5 or later supports this element.</td>
</tr>
<tr>
<td>requires</td>
<td>id</td>
<td>The <code>requires</code> element is an optional element that specifies the <code>id</code> of a package on which this BlackBerry device application depends. This element can appear more than once, if the BlackBerry device application depends on more than one BlackBerry device application. When a BlackBerry device application loads onto the BlackBerry device, all packages that the <code>&lt;requires&gt;</code> tag specifies also load onto the BlackBerry device. BlackBerry Desktop Software version 3.6 or later supports this element.</td>
</tr>
<tr>
<td>Element</td>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>vendor</td>
<td>—</td>
<td>The <strong>vendor</strong> element provides the name of the company that created the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager.</td>
</tr>
<tr>
<td>version</td>
<td>—</td>
<td>The <strong>version</strong> element provides the version number of the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager. This version number is for information purposes only.</td>
</tr>
</tbody>
</table>

**Application distribution through a computer connection**

**Distribute an application from a computer**

You can use the application loader tool in the BlackBerry® Desktop Manager to install applications on the BlackBerry device. The application loader tool can provide users with a simple way to download applications from their computers to their BlackBerry devices.

**Distribute an application from a web page**

You can use the BlackBerry® Application Web Loader to post your compiled application on a web site. Users can use Windows® Internet Explorer® on their computers to visit the web page and install the application on their BlackBerry devices. When BlackBerry device users visit the web page, the BlackBerry Application Web Loader prompts them to connect their devices to the USB port. They can then install the application using an ActiveX® control. The BlackBerry Application Web Loader can provide BlackBerry device users with a simple way to install applications from their computers without running the BlackBerry® Desktop Manager.

**Distribute an application for testing**

The BlackBerry® Java® Development Environment includes a command line tool called the JavaLoader tool that is located in the BIN folder in the BlackBerry JDE folder. You can use the JavaLoader tool to quickly install and remove compiled application files on the BlackBerry device directly over the USB port. You do not require any descriptor files or web pages. The JavaLoader tool can be useful when you install and remove your application frequently during testing and development; however, the JavaLoader tool is not designed for use by BlackBerry device users.
Distributing an application from a computer

Create an application loader file

You use an application loader file to distribute a BlackBerry® device application using the Application Loader tool of the BlackBerry® Desktop Manager.

1. Create an .alx file for each BlackBerry device application, and then distribute the .alx file with the .cod files to BlackBerry device users. See the Application Loader Online Help for more information about .alx files.

2. In the BlackBerry® Integrated Development Environment, select a project.

3. On the Project menu, click Generate .alx File.

Install a BlackBerry device application on a specific device

1. Open a text editor.

2. Locate the .alx file for the BlackBerry® device application.

3. In the .alx file, make sure the series attribute in the fileset opening tag refers to the BlackBerry device you want to install the BlackBerry device application on.

   `<fileset series="8700" Java="1.0">`

   For more information about the series attribute, see the Platform.alx file located in the simulator directory of your BlackBerry® Java® Development Environment installation directory:

   Program Files\Research In Motion\BlackBerry JDE 4.6.0\simulator.

4. Make sure the files tag contains a reference to the .cod file for your BlackBerry device application.

   `<files>
   My_application.cod
   </files>`

5. Update the application, description, and other tags to reflect the purpose of the .alx file.

   `<application id="Push only to 8700">
   ...
   <description>This will push the COD only to 8700s</description>`

Code sample: Loading a BlackBerry device application on a specific BlackBerry device

```
<loader version="1.0">
  <application id="Push only to 8700">
    <name>Alien</name>
    <description>This will push the COD only to 8700s</description>
    <version>2006.02.14.1838</version>
    <vendor>RIM</vendor>
    <copyright>Copyright (c) 2001-2005</copyright>
    <fileset series="8700" Java="1.0">
```

Specifying supported versions of the BlackBerry Device Software

BlackBerry® device applications that use APIs that are only available on particular versions of the BlackBerry® Device Software should specify the supported BlackBerry device versions using the _blackberryVersion attribute.

BlackBerry device applications for use with a BlackBerry device running on a specific platform version should specify supported platform versions using the _platformVersion attribute. The _platformVersion attribute can be used within the directory tag, the application tag, or the fileset tag.

You can use the following rules to specify a version range for the BlackBerry Device Software or the platform version:

- Square brackets [] indicate inclusive (closed) range matching.
- Round brackets () indicate exclusive (open) range matching.
- Missing lower ranges imply 0.
- Missing upper ranges imply infinity.

For example, [4.0,) indicates any version between 4.0 and infinity.

**Code sample: Preventing modules from loading on versions of the BlackBerry Device Software earlier than version 4.0.**

```xml
<application id="application_id" _blackberryVersion="[4.0,)">
...
</application>
```

**Code sample: Providing alternative modules for different versions of the BlackBerry Device Software.**

```xml
<application id="application_id">
  ...
  <fileset _blackberryVersion="(4.0)"
    ... modules for BlackBerry device software versions earlier than 4.0
  </fileset>
  <fileset _blackberryVersion="[4.0,)">
    ... modules for BlackBerry device software versions 4.0 and later
  </fileset>
</application>
```

**Code sample: Preventing modules from loading on versions of the platform earlier than version 2.4.0.66.**
Specifying a location for your application on a BlackBerry device

You can add your BlackBerry® device application to one of the application folders that is available with the theme on the device, to a new application folder that you create, or to the Home screen on the device. For example, if you develop a game, you can add your game to the Games folder on the BlackBerry device. If your application has multiple entry points, you can specify an application folder for each entry point. If you do not specify a location, the BlackBerry device adds your application to the default application folder (for example, the Downloads folder). You can specify the location of applications that are installed on a BlackBerry device that is running BlackBerry® Device Software 5.0 or later.

The following application folders are available with the Precision theme on a device that is running BlackBerry Device Software 5.0:

- Applications
- Downloads
- Games
- Instant Messaging
- Mail
- Media
- Music
- Setup

If you create a new application folder, the folder name must use valid characters. For example, you cannot use characters such as the less than sign (<) or greater than sign (>). For a list of characters that are not valid, see the documentation for the FilenameTextFilter class in the API reference for the BlackBerry® Java® Development Environment.
A user can change the location of your application on the device. If a user changes the location of your application, the device always uses the new location. For example, if a user installs, removes, and subsequently reinstalls your application, the device adds your application to the location that was previously specified by the user.

**Specify a location for your application on a BlackBerry device**

You can specify a location to add your application to on a BlackBerry device by adding the `Content-Folder` attribute to the .jad file for your application. If you specify an application folder and the folder cannot be created, your application is added to the default folder (for example, the Downloads folder).

**Before you begin:** Verify that the BlackBerry® device is running BlackBerry® Device Software 5.0 or later.

1. In Eclipse®, open and package your BlackBerry device application project.
2. In the Package Explorer view, open the .jad file in the `<project_folder>/deliverables/Standard/<x.x.x>` folder. If you do not see the folder or .jad file, verify that you packaged your project.
3. In a text editor, add a new line to the end of the .jad file and type one of the following:
   - To add your application to the Home screen on a device, type `Content-Folder: /`.
   - To add your application to an application folder, type `Content-Folder: <folder_name>`.
   - To specify locations for multiple entry points, for each entry point, type `Content-Folder-x: <folder_name>`. For example, type `Content-Folder-1: Games` for the first entry point and `Content-Folder-2: Applications` for the second entry point.
4. Save the .jad file.
5. In Eclipse, in the Package Explorer view, drag the .jad file from the `<project_folder>/deliverables/Standard/<x.x.x>` folder to the root folder of your application project.
6. Package your project.
7. Test and distribute your application using the new package files.

**Example: Adding an application to the Games folder by changing the .jad file**

```
Manifest-Version: 1.0
MIDlet-Version: 1.0.0
MIDlet-Jar-Size: 2912
MicroEdition-Configuration: CLDC-1.1
MIDlet-Jar-URL: MyGame.jar
RIM-COD-Module-Dependencies: net_rim_cldc
RIM-MIDlet-Flags-1: 0
RIM-COD-Module-Name: MyGame
MIDlet-Name: MyGame
RIM-COD-Size: 1504
RIM-COD-Creation-Time: 1272372497
MIDlet-1: My Game, img/mygame.png,
RIM-COD-URL: MyGame.cod
RIM-COD-SHA1: 82 2f 22 b6 e6 34 ef c3 2b 0e a4 96 22 08 c0 60 39 4d db aa
MicroEdition-Profile: MIDP-2.0
MIDlet-Vendor: Research In Motion Ltd.
Content-Folder: Games
```
Localizing BlackBerry device applications

Multilanguage support

The BlackBerry® Integrated Development Environment includes a resource mechanism for creating string resources. The Localization API is part of the net.rim.device.api.i18n package. MIDP applications do not support localization.

The BlackBerry Integrated Development Environment stores resources for a locale in a ResourceBundle object. A ResourceBundleFamily object contains a collection of ResourceBundles, which groups the resources for an application. The application can switch languages, depending on the locale of the BlackBerry device user, without requiring new resource bundles.

You can use the BlackBerry Integrated Development Environment to compile each resource bundle into a separately compiled .cod file. You can load the appropriate .cod files onto BlackBerry devices with the other .cod files for the application.

Resources are organized in a hierarchy based on inheritance. If a string is not defined in a locale, a string from the next closest locale is used.

Files required for localization

<table>
<thead>
<tr>
<th>File required for localization</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource header file</td>
<td>This file defines descriptive keys for each localized string. When the BlackBerry® Integrated Development Environment builds a project, it creates a resource interface with Resource appended to the .rrh file name. For example, if you createAppName.rrh, the interface is named AppNameResource.</td>
<td>AppName.rrh</td>
</tr>
<tr>
<td>Resource content file (root locale)</td>
<td>This file maps resource keys to string values for the root (global) locale. It has the same name as the resource header file.</td>
<td>AppName.rrc</td>
</tr>
<tr>
<td>Resource content file (specific locales)</td>
<td>This file maps resource keys to string values for specific locales (language and country). Files have the same name as the resource header file, followed by an underscore (<em>) and the language code, and then, optionally, an underscore (</em>) and country code. Save resource content files in the folder where the .java file is located. For example, in the folder that contains CountryInfo.java, save CountryInfo.rrc (root locale), CountryInfo_en.rrc (English), and CountryInfo_fr.rrc (French).</td>
<td>AppName_en.rrc, AppName_en_GB.rrc, AppName_fr.rrc</td>
</tr>
<tr>
<td>Initialization file</td>
<td>This file initializes the resource bundle mechanism. You require this file only when you compile resources as a separate project.</td>
<td>init.java</td>
</tr>
</tbody>
</table>
Manage localization files for a suite of BlackBerry device applications

If you create a suite of BlackBerry® device applications, organize resources into separate projects for each locale. The BlackBerry® Integrated Development Environment provides a built-in initialization mechanism. You only need to create an empty initialization class with an empty `main()`. If you support a large number of locales, create a single library project for all resource header (.rrh) files and set the project type to Library. For each resource locale in the library, define a dependency between the projects.

1. Open the BlackBerry® Integrated Development Environment.
2. Create a project for each resource bundle (locale), including the root locale.
3. Give the projects for each resource locale the same name as the project for the root locale, followed by a double underscore (__), the language code, and, optionally, an underscore (_) followed by the country code. For example, if the root locale project is named `com_company_app`, the projects for each locale would be named `com_company_app__en`, `com_company_app__en_GB`, and `com_company_app__fr`.
4. Right-click the project, and then click Properties.
5. On the Build tab, in the Output file name field, type a name for the compiled file, without a file name extension.
   ```java
   package com.rim.samples.device.resource;
   import net.rim.device.api.i18n.*;
   public class init {
       public static void main (String[] args) { }
   }
   ```
7. Create one resource header file for each BlackBerry device application.
8. Copy the resource header (.rrh) files to the project for each BlackBerry device application.
9. Copy the resource header files to each resource locale project.
10. Create one resource content file for each BlackBerry device application.
11. Create one resource content file for each supported locale.
12. In each resource locale project, right-click each .rrh file, and then click Properties.
13. Select Dependency only. Do not build.
14. Add the resource content (.rrc) files to the projects for the appropriate locales.
Custom user authentication

To log in to a BlackBerry device, a user must generate a security token. By default, BlackBerry device users type a password to authenticate their identity and to generate a corresponding security token. BlackBerry® devices that run BlackBerry® Device Software 5.0 or later also support custom user authentication modules.

You must create a user authentication module to use additional hardware such as a smart card or biometric device for authentication purposes. The security framework on the BlackBerry device uses your user authentication module to interact with the hardware.

For more information about custom user authentication and the net.rim.device.api.userauthenticator package, see the API reference for the BlackBerry® Java® Development Environment.
Glossary

3GPP
Third Generation Partnership Project

AES
Advanced Encryption Standard

API
application programming interface

APN
access point name

ASCII
American National Standards Institute

.alx file
A .alx file is the application descriptor that provides information about a BlackBerry Java® Application and the location of the application’s .cod files to a BlackBerry device.

BlackBerry MDS
BlackBerry® Mobile Data System

CDMA
Code Division Multiple Access

COM port
communications port

EDGE
Enhanced Data Rates for Global Evolution

EVDO
Evolution Data Optimized

GAN
generic access network

GERAN
GSM-EDGE Radio Access Network

GPRS
General Packet Radio Service
GSM
   Global System for Mobile Communications®

HTTP
   Hypertext Transfer Protocol

HTTPS
   Hypertext Transfer Protocol over Secure Sockets Layer

IPPP
   Internet Protocol Proxy Protocol

JSR
   Java® Specification Request

MIDP
   Mobile Information Device Profile

PAP
   Push Access Protocol

PIN
   personal identification number

RAPC
   RIM Application Program Compiler

service books
   Service books determine which services are available on BlackBerry devices or BlackBerry enabled devices.

SSID
   service set identifier

TCP
   Transmission Control Protocol

TLS
   Transport Layer Security

Triple DES
   Triple Data Encryption Standard

UDP
   User Datagram Protocol
UMTS
    Universal Mobile Telecommunications System

UTRAN
    UMTS Terrestrial Radio Access Network

WAP
    Wireless Application Protocol

WLAN
    wireless local area network
Provide feedback

To provide feedback on this deliverable, visit www.blackberry.com/docsfeedback.
Related resources

For more information on developing applications for BlackBerry® devices, see the following resources.

- BlackBerry Java Application Fundamentals Guide
- BlackBerry Java Application UI and Navigation Development Guide
- BlackBerry Java Application Integration Development Guide
- BlackBerry Java Application Multimedia Development Guide
- BlackBerry Java Application Accessibility Development Guide
- BlackBerry Java Application Transitioning to Touch Screen Development Technical Note
- BlackBerry Browser Development Guide
- BlackBerry Developer Zone at www.blackberry.com/developers
- API reference for the BlackBerry® Java® Development Environment
# Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>27 April 2010</td>
<td>Changed the Specify a location for your application on a BlackBerry device topic.</td>
</tr>
<tr>
<td>6 April 2010</td>
<td>Added the following topics:</td>
</tr>
<tr>
<td></td>
<td>• Custom user authentication</td>
</tr>
<tr>
<td></td>
<td>Updated the following topics:</td>
</tr>
<tr>
<td></td>
<td>• Network connections and transport types</td>
</tr>
<tr>
<td></td>
<td>• File systems and paths</td>
</tr>
<tr>
<td></td>
<td>Added the following code samples:</td>
</tr>
<tr>
<td></td>
<td>• Code sample: Reading sections of a binary file</td>
</tr>
<tr>
<td>8 October 2009</td>
<td>Added the following topics:</td>
</tr>
<tr>
<td></td>
<td>• Specifying a location for your application on a BlackBerry device</td>
</tr>
<tr>
<td></td>
<td>• Specify a location for your application on a BlackBerry device</td>
</tr>
<tr>
<td>20 August 2009</td>
<td>Added topics and sample code showing how to use features of the Network API. Added file system and path reference information.</td>
</tr>
<tr>
<td>14 August 2009</td>
<td>Changed the System requirements topic.</td>
</tr>
</tbody>
</table>
©2010 Research In Motion Limited. All rights reserved. BlackBerry®, RIM®, Research In Motion®, and related trademarks, names, and logos are the property of Research In Motion Limited and are registered and/or used in the U.S. and countries around the world.

802.11a, 802.11b, and 802.11g are trademarks of the Institute of Electrical and Electronics Engineers, Inc. Bluetooth is a trademark of Bluetooth SIG. Eclipse is a trademark of Eclipse Foundation, Inc. Global System for Mobile Communications is a trademark of the GSM MOU Association. iDEN is a trademark of Motorola, Inc. Microsoft, ActiveX, Internet Explorer, and Windows are trademarks of Microsoft Corporation. Java is a trademark of Oracle America, Inc. UMTS is a trademark of European Telecommunications Standard Institute. Wi-Fi is a trademark of the Wi-Fi Alliance. All other trademarks are the property of their respective owners.

This documentation including all documentation incorporated by reference herein such as documentation provided or made available at www.blackberry.com/go/docs is provided or made accessible "AS IS" and "AS AVAILABLE" and without condition, endorsement, guarantee, representation, or warranty of any kind by Research In Motion Limited and its affiliated companies ("RIM") and RIM assumes no responsibility for any typographical, technical, or other inaccuracies, errors, or omissions in this documentation. In order to protect RIM proprietary and confidential information and/or trade secrets, this documentation may describe some aspects of RIM technology in generalized terms. RIM reserves the right to periodically change information that is contained in this documentation; however, RIM makes no commitment to provide any such changes, updates, enhancements, or other additions to this documentation to you in a timely manner or at all.

This documentation might contain references to third-party sources of information, hardware or software, products or services including components and content such as content protected by copyright and/or third-party web sites (collectively the "Third Party Products and Services"). RIM does not control, and is not responsible for, any Third Party Products and Services including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third Party Products and Services. The inclusion of a reference to Third Party Products and Services in this documentation does not imply endorsement by RIM of the Third Party Products and Services or the third party in any way.

EXCEPT TO THE EXTENT SPECIFICALLY PROHIBITED BY APPLICABLE LAW IN YOUR JURISDICTION, ALL CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS, OR WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS OR WARRANTIES OF DURABILITY, FITNESS FOR A PARTICULAR PURPOSE OR USE, MERCHANTABILITY, MERCHANTABLE QUALITY, NON-INFRINGEMENT, SATISFACTORY QUALITY, OR TITLE, OR ARISING FROM A STATUTE OR CUSTOM OR A COURSE OF DEALING OR USAGE OF TRADE, OR RELATED TO THE DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN, ARE HEREBY EXCLUDED. YOU MAY ALSO HAVE OTHER RIGHTS THAT VARY BY STATE OR PROVINCE. SOME JURISDICTIONS MAY NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES AND CONDITIONS. TO THE EXTENT PERMITTED BY LAW, ANY IMPLIED WARRANTIES OR CONDITIONS RELATING TO THE DOCUMENTATION TO THE EXTENT THEY CANNOT BE EXCLUDED AS SET OUT ABOVE, BUT CAN BE LIMITED, ARE HEREBY LIMITED TO NINETY (90) DAYS FROM THE DATE YOU FIRST ACQUIRED THE DOCUMENTATION OR THE ITEM THAT IS THE SUBJECT OF THE CLAIM.
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, IN NO EVENT SHALL RIM BE LIABLE FOR ANY TYPE OF DAMAGES RELATED TO THIS DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN INCLUDING WITHOUT LIMITATION ANY OF THE FOLLOWING DAMAGES: DIRECT, CONSEQUENTIAL, EXEMPLARY, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR AGGRAVATED DAMAGES, DAMAGES FOR LOSS OF PROFITS OR REVENUES, FAILURE TO REALIZE ANY EXPECTED SAVINGS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF BUSINESS OPPORTUNITY, OR CORRUPTION OR LOSS OF DATA, FAILURES TO TRANSMIT OR RECEIVE ANY DATA, PROBLEMS ASSOCIATED WITH ANY APPLICATIONS USED IN CONJUNCTION WITH RIM PRODUCTS OR SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF RIM PRODUCTS OR SERVICES OR ANY PORTION THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR SERVICES, COST OF CAPITAL, OR OTHER SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES WERE FORESEEN OR UNFORESEEN, AND EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, RIM SHALL HAVE NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO RIM AND ITS AFFILIATED COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE PROVIDERS), AUTHORIZED RIM DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND THEIR RESPECTIVE DIRECTORS, EMPLOYEES, AND INDEPENDENT CONTRACTORS.

IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF RIM OR ANY AFFILIATES OF RIM HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with RIM's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for acquiring them. You should not install or use Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and Services that are provided with RIM's products and services are provided as a convenience to you and are provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations, or warranties of any kind by RIM and RIM assumes no liability whatsoever, in relation thereto. Your use of Third Party Products and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other agreements applicable thereto with third parties, except to the extent expressly covered by a license or other agreement with RIM.
Certain features outlined in this documentation require a minimum version of BlackBerry® Enterprise Server, BlackBerry® Desktop Software, and/or BlackBerry® Device Software.

The terms of use of any RIM product or service are set out in a separate license or other agreement with RIM applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY RIM FOR PORTIONS OF ANY RIM PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.

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

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

Published in Canada