BlackBerry Java Development Environment

Version: 4.6.0

Fundamentals Guide

BlackBerry

SWDT499912-499912-1208024835-001

Contents

Understanding BlackBerry and programming for BlackBerry devices

1

BlackBerry® devices provide a Java® ME wireless environment that supports client/server applications. Application developers can create a BlackBerry® Java Application that has sophisticated UIs for data entry and searching, and that supports multithreading, internationalization, network communication, and local data storage. Applications can communicate with networks using standard TCP and HTTP connections, regardless of the underlying wireless network.

Application developers can also create a BlackBerry Java Application that integrates tightly with core BlackBerry device applications, such as the message list, organizer applications, phone, and browser, for an essentially seamless user experience.

Design principles for BlackBerry devices

Applications designed for BlackBerry® devices should provide a balance between the best possible user experience and a long battery life. When you design your BlackBerry device application, consider the differences between mobile devices and computers. Mobile devices

- have a smaller screen size that can display a limited number of characters
- have slower processor speeds
- use wireless networks that have a longer latency period than standard LANs
- have less available memory
- have shorter battery life
- display one screen at a time

Mobile device users use applications on their mobile device differently than they would use applications on a computer. On mobile devices, users expect to find information quickly. For example, a CRM system can provide a massive amount of information, but users only require a small amount of that information at one time. The BlackBerry device UI is designed so that users can perform tasks easily and access information quickly.

When you design applications for BlackBerry devices, try to be as consistent as possible with other BlackBerry device applications. Consider the following guidelines:

- Use or extend existing UI components where possible so that your application can inherit the default behavior of the component.
- Follow the standard navigation model as closely as possible so that users can make full use of the keyboard and trackball.
- Make all actions available from the menu. Verify that the actions available in the menu are relevant to users' current context.

When you design your application, also consider the following guidelines:

- Stay focused on users' immediate task. Simplify data selection and presentation to display only the information that users need at any one moment.

- Display information in a way that makes effective use of the small screen.

Before you design your application, consider using the core applications on the BlackBerry device or the BlackBerry Smartphone Simulator to learn more about the navigation model and best practices for designing your application's UI.

Release cycles and versions

All BlackBerry® devices include a specific version of the BlackBerry® Device Software and the BlackBerry® Java® Virtual Machine. To determine the version of the BlackBerry Device Software for a BlackBerry device, in the device Options, click **About**. You can upgrade the BlackBerry Device Software. For example, you can upgrade a BlackBerry device with BlackBerry Device Software version 4.0 to BlackBerry Device Software version 4.1.

With each major release of the BlackBerry Device Software and the BlackBerry JVM, Research In Motion includes the corresponding Java APIs and version of the BlackBerry® Java® Development Environment. The version of the BlackBerry Device Software determines the version of the BlackBerry® Integrated Development Environment that you can use to develop applications. For example, RIM released BlackBerry Device Software version 4.0 and BlackBerry JDE version 4.0 at the same time. BlackBerry JDE version 4.0 includes support for the APIs that were introduced in BlackBerry Device Software version 4.0 and BlackBerry JVM version 4.0. Applications that you create using BlackBerry JDE Version 4.0 only work on BlackBerry devices running BlackBerry Device Software version 4.0 or later.

You can use the following criteria to decide which version of the BlackBerry JDE to use to develop an application:

- If the application does not need to use specific BlackBerry device hardware features or newly released API extensions, use BlackBerry JDE version 4.0 to develop the application.
- If the application is designed to run only on the BlackBerry® Pearl™ 8100 smartphone, use BlackBerry JDE version 4.2 or later.

BlackBerry Java Development Environment

The BlackBerry® Java® Development Environmentis a fully integrated development and simulation environment for building a BlackBerry® Java Application for BlackBerry devices. With the BlackBerry JDE, developers can build applications using the Java® ME programming language and the extended Java APIs for BlackBerry.

The BlackBerry Java Development Environment includes the following development tools:

- BlackBerry® Integrated Development Environment
- BlackBerry Smartphone Simulator
- Java ME APIs and BlackBerry APIs
- sample applications

The BlackBerry IDE includes a full suite of editing and debugging tools that are optimized for the development of a BlackBerry Java Application. TheBlackBerry Smartphone Simulator provides a complete Windows® type environment, and is designed to simulate UIs and user interaction, network connections, email services, and wireless data synchronization.

The BlackBerry Java Development Environment Component Package includes the following development tools for development within third-party IDEs such as NetBeans™ or Eclipse™:

- RAPC: You can use this command prompt compiler to compile .java and .jar files into .cod files that you can run in the BlackBerry Smartphone Simulator or on a BlackBerry device.
- JavaLoader: You can use this tool to add or update an application on a BlackBerry device for testing, and to view information about application .cod files.
- BlackBerry® Signature Tool: You can use this tool to send code signature requests to the BlackBerry® Signing Authority Tool.
- Preverify Tool: You can use this tool to partially verify your classes before you load your application onto a BlackBerry device.
- JDWP: You can use this tool to debug applications using third-party integrated development environments.

Java ME and Java APIs for BlackBerry

Java® ME is an industry standard platform that defines common sets of Java APIs for different types of wireless and embedded devices. A Java ME application on a BlackBerry® device runs in the BlackBerry® Java® Virtual Machine, which provides all of the runtime services to the applications and performs functions such as typical memory allocations, security checks, and garbage collection.

The Java ME MIDP standard addresses the API and BlackBerry JVM needs of a constrained wireless device with a user interface. The BlackBerry device supports the Java ME MIDP standard as defined in JSR 118. The Java MEMIDP standard provides a core set of Java APIs that any BlackBerry device can support, regardless of its underlying operating system. Developers can often build one Java application using the MIDP standard APIs and run that application on many different types of devices.

Support for standard Java APIs

The BlackBerry® device and the BlackBerry® Java® Development Environment support the Java® ME MIDP standard, which provides a core set of Java APIs that you can use to develop wireless device applications. The BlackBerry device and the BlackBerry® Java® Development Environment also support the following JSRs:

- JSR 30: Connected Limited Device Configuration Version 1.0

  (supported on devices with BlackBerry® Device Software version 4.0 or earlier)
- JSR 37: Mobile Information Device Profile Version 1.0

  (supported on devices with BlackBerry Device Software Version 4.0 or earlier)
- JSR 75: Portable Optional Packages for the J2ME Platform (PDAP) support for the PIM APIs only and the File Connection API for Java ME (supported on devices with BlackBerry Device Software version 4.2 or later)
- JSR 82: Java APIs for Bluetooth®
- JSR 118: Mobile Information Device Profile Version 2.0
- JSR 120: Wireless Messaging API (WMA) Version 1.1
- JSR 135: Mobile Media APIs (MM API) Version 1.1

- JSR 139: Connected Limited Device Configuration Version 1.1
- JSR 172: J2ME Web Services
- JSR 177: Security and Trust Services API for J2ME (SATSA)
- JSR 179: Location API for Java ME
- JSR 185: Java Technology for the Wireless Industry (JTWI)
- JSR 205: Wireless Messaging API 2.0
- JSR 211: Content Handler API
- JSR 226: Scalable 2D Vector Graphics API for Java ME
- JSR 238: Mobile Internationalization API

Support for Java API extensions

BlackBerry® devices support the following Java® APIs that are not part of the standard JSR definitions and that can provide greater features and functionality over what is available in the standard MIDP API libraries.

API	Description
User Interface APIs	You can use these APIs to create screens, menu items, and all the components of the user interface.
Persistent Data Storage APIs	You can use these APIs to store custom data locally within your application.
Networking and I/O APIs	You can use these APIs to establish network connections and read or write data to a server-side application.
Event Listeners	You can use the Event Listeners to respond to BlackBerry device user or system-initiated events on a BlackBerry device.
Application Integration APIs	You can use these APIs to integrate with the existing BlackBerry email, phone, calendar, contacts, browser, camera, media player, and task list applications.
Additional Utilities	You can use these additional APIs for data encryption and compression, XML parsing, Bluetooth® connectivity, location-based services, and so on.

BlackBerry solutions

BlackBerry® device users might use either the BlackBerry® Enterprise Server or the BlackBerry® Internet Service, or they can use both on the same device. Understanding the differences between theBlackBerry Enterprise Server and the BlackBerry Internet Service, and which types of users you plan to support, is important, as it might impact which modes of transport you use and how you manage data synchronization.

BlackBerry Enterprise Solution

The BlackBerry® Enterprise Server is part of the BlackBerry® Enterprise Solution. The BlackBerry Enterprise Server exists behind the corporate firewall and provides a wireless gateway for BlackBerry device users in an organization to access corporate email and organizer data. The BlackBerry Enterprise Server also provides the following key features:

- data encryption and compression
- BlackBerry device management and monitoring utilities
- simplified application provisioning
- authenticated gateway for intranet access from a BlackBerry® Java Application

BlackBerry Internet Service

BlackBerry® device users who are not associated with a BlackBerry® Enterprise Server can use the BlackBerry® Internet Service. The BlackBerry Internet Service is an email and Internet service for BlackBerry devices that is designed to provide users with automatic delivery of email messages, wireless access to email attachments, and access to Internet content.

The BlackBerry Internet Service includes support for direct HTTP and TCP/IP connectivity to the Internet from a third-party BlackBerry® Java Application.

BlackBerry MDS

To allow a BlackBerry® Java Application access to resources behind the corporate firewall, the BlackBerry® Enterprise Server includes the BlackBerry® Mobile Data System. The BlackBerry MDS provides HTTP and TCP/IP proxies for a BlackBerry Java Application, which allow the BlackBerry device to communicate with application and web servers behind the corporate firewall without additional VPN software. Applications that send data using the BlackBerry Enterprise Server as a gateway can capitalize on the simplified enterprise connectivity, data encryption and compression, and wireless network-independence that the BlackBerry® Enterprise Solution offers. BlackBerry MDS also provides an open interface, allowing server-side applications behind the corporate firewall to push content to applications on BlackBerry devices.

BlackBerry Java Application design

2

Standalone applications

You can use the BlackBerry® APIs to build standalone applications, such as games and static reference guides that can run as offline applications. You can add the required resource data to an application before you compile it. BlackBerry device users can install the application over the wireless network or with the BlackBerry® Desktop Software. After an application is installed on the BlackBerry device, it does not need to connect to the wireless network or to a computer.

Applications with desktop synchronization

You can use the BlackBerry® APIs to build applications with desktop synchronization capabilities, such as reference guides and organizer applications. The user connects the BlackBerry device to a computer to manage and synchronize data that is located on the computer.

Research In Motion® does not provide HotSync® conduits or any other direct database synchronization module. You must build the synchronization code, and the BlackBerry device user must initiate the data synchronization process manually. After the application is installed on the BlackBerry device, the BlackBerry device user must synchronize information manually by connecting their BlackBerry device to the computer with a serial connection, a USB connection, or a Bluetooth® connection.

Applications with wireless access, wireless synchronization, or wireless alerting

You can use the BlackBerry® APIs to build applications that push content proactively over the wireless network to BlackBerry devices in environments that use the BlackBerry® Enterprise Server. A BlackBerry® Java Application for BlackBerry devices uses a wireless connection to the Internet or the corporate intranet to provide BlackBerry device users with access to remote data and applications. The BlackBerry® Java® Development Environment provides APIs you can use in applications to establish network connections to servers on the Internet or the corporate intranet.

MIDlet applications

The MIDlet application model is part of the MIDP specification. The main class of a MIDlet always extends the MIDlet class and it must use methods for `startApp()`, `pauseApp()`, and `destroyApp()`.

Advantages	Disadvantages
• Applications are portable to other devices that also support the MIDP specification.	• Applications can use only the user interface APIs that exist in the `javax.microedition.lcdui` library. • The model assumes that all application processes terminate when the application closes. • Applications cannot start automatically in the background when the device turns on.

CLDC applications

The CLDC application model is a specification of a framework for Java® ME. A CLDC application extends the `UiApplication` class and starts with a standard `main()` method.

Most of the sample applications that the BlackBerry® Java® Development Environmentincludes use the CLDC application model. All of the core BlackBerry applications (including message list, contacts list, calendar, and the browser) are built as CLDC applications.

Advantages	Disadvantages
• BlackBerry User Interface APIs provide more functionality and flexibility than the standard `javax.microedition.lcdui` library. • Applications can run active background threads after they have closed. • Applications can start automatically in the background when the device turns on. • Applications can use IPC APIs to exchange information with other applications. • Developers can create reusable library modules that CLDC applications can import.	• Applications are not portable to other devices.

API control and code signing

When you develop a BlackBerry® Java Application for BlackBerry devices, you can use only the public Java APIs that are published and documented in the Javadoc™ documents in the BlackBerry® Java® Development Environment. The BlackBerry® Java® Virtual Machine on the BlackBerry device is designed to protect the underlying data and operating system, so applications cannot call undocumented or unsupported APIs or access data that is not explicitly exposed through the APIs. If you try to use Java APIs that are not publicly exposed, your application receives an error message at runtime.

Public APIs are either open or signed. Signed APIs expose the methods to access BlackBerry device user data or other information on the BlackBerry device that is considered sensitive. You can use signed APIs, but you must request and receive a set of code signing keys from Research In Motion. You must then digitally sign your application before you install it on a BlackBerry device. Code signing does not certify or approve an application; it allowsRIM to identify the author of an application that uses sensitive APIs, if the application is malicious.

To request a set of code signing keys, visit www.blackberry.com/developers/downloads/jde/api.shtml. You will receive your set of code signing keys in about 10 days.

Object modeling

Whether you use the MIDlet or the CLDC application model, you must use an object-oriented approach when you design your application for the BlackBerry® device. In an object-oriented approach, developers use objects to contain the code that is common to a specific process or function. For example, a developer might use separate objects to control networking activity, data storage, data processing and manipulation, and user interface interaction. When you design your application, start with a good object model.

Multithreading

The BlackBerry® operating system is a multithreaded operating system, which means that many applications and processes can run actively on the BlackBerry device at the same time. For example, applications can use background threads to manage processor-intensive tasks or network communications so that they do not affect the main thread. If an application creates background threads, and a BlackBerry device user closes the application, the background threads can remain active.

Best practice: Using multithreading

Make effective use of the multithreading capabilities of the BlackBerry® operating system. In particular, always create a new thread for network connections or other lengthy operations (more than one-tenth of a second). Use background threads for listeners or other processes that run in the background when the application starts.

Best practices for writing an efficient BlackBerry Java Application

Best practice: Writing efficient code

To allow a BlackBerry® Java® Application to use resources efficiently, consider the following guidelines:

- Use local variables.
- Use shorthand for evaluating Boolean conditions.
- Make classes final.
- Use `int` instead of `long`.
- Avoid garbage collection.
- Use static variables for `Strings`.
- Avoid the `String(String)` constructor.
- Write efficient loops.
- Optimize subexpressions.
- Optimize division operations.
- Avoid `java.util.Enumeration`.
- Perform casts using `instanceof`.
- Evaluate conditions using `instanceof`.
- Avoid using `StringBuffer.append(StringBuffer)`.
- Avoid returning `null`.
- Avoid passing `null` into methods.
- Use caution when passing `null` into a constructor.
- Use `long` for unique identifiers.
- Exit applications correctly.
- Print the stack trace.

Using local variables

Use local variables whenever possible. Access to local variables is more efficient than access to class members.

Using shorthand for evaluating Boolean conditions

To evaluate a `Boolean` condition, use shorthand. The resulting compiled code is shorter.

Code sample

```
return( boolean_expression );
```

Making classes final

When you create code libraries, mark classes as `final` if you know that developers will never extend them. The presence of the `final` keyword allows the compiler to generate more efficient code.

By default, the BlackBerry® Java® Development Environment compiler marks any classes that you do not extend in an application .cod file as `final`.

Using int instead of long

In Java®, a `long` is a 64-bit integer. Because BlackBerry® devices use a 32-bit processor, operations can run two to four times faster if you use an `int` instead of a `long`.

Avoiding garbage collection

Avoid calling `System.gc()` to perform a garbage collection operation because it might take too much time on BlackBerry® devices with limited available memory. Let the BlackBerry® Java® Virtual Machine collect garbage.

Using static variables for Strings

When you define static fields (also called class fields) of type `String`, you can increase application speed by using static variables (not `final`) instead of constants (`final`). The opposite is true for primitive data types, such as `int`.

For example, you might create a `String` object as follows:

```
private static final String x ="example";
```

For this static constant (denoted by the `final` keyword), each time that you use the constant, a temporary `String` instance is created. The compiler eliminates `"x"` and replaces it with the string `"example"` in the bytecode, so that the BlackBerry® Java® Virtual Machine performs a hash table lookup each time that you reference `"x"` .

In contrast, for a static variable (no `final` keyword), the `String` is created once. The BlackBerry JVM performs the hash table lookup only when it initializes `"x"` , so access is faster.

```
private static String x = "example";
```

You can use public constants (that is, `final` fields), but you must mark variables as private.

Avoiding the String(String) constructor

In a BlackBerry® Java Application, each quoted string is an instance of the `java.lang.String` class. Create a `String` without using the `java.lang.String(String)` constructor.

Code sample

```
String str = "abc";
String str = "found " + n + " items";
```

Writing efficient loops

If your container is likely to contain more than one element, assign the size to a local variable.

If the order in which you iterate through items is not important, you can iterate backward to avoid the extra local variable on the stack and to make the comparison faster.

Code sample

```
int size = vector.size();
for( int i = 0; i < size; ++i ) {
   ...
}
for( int i = vector.size() - 1; i >= 0; --i ) {
   ...
}
```

Optimizing subexpressions

If you use the same expression twice, use a local variable.

Code sample

```
int tmp = i+1; one( tmp ); two( tmp );
```

Optimizing division operations

Division operations can be slow on BlackBerry® devices because the processor does not have a hardware divide instruction.

When your code divides a positive number by two, use shift right by one (`>> 1`). Use the shift right (>>) only when you know that you are working with a positive value.

Code sample

```
int = width >> 1;
```

Avoiding java.util.Enumeration

Avoid using `java.util.Enumeration` objects unless you want to hide data (in other words, to return an enumeration of the data instead of the data itself). Asking a vector or hash table for an `Enumeration` object is slow and creates unnecessary garbage. If another thread might modify the vector, synchronize the iteration. The Java® SE uses an `Iterator` object for similar operations, but `Iterator` objects are not available in the Java® ME.

Code sample

```
for( int i = v.size() - 1; i >=0; --i ) {
o = v.elementAt( i );
...
}
synchronized( v ) {
for( int i = v.size() - 1; i >=0; --i ) {
o = v.elementAt( i );
...
}
}
```

Performing casts using instanceof

Use `instanceof` to evaluate whether a cast succeeds.

Code sample

```
if( x instanceof String ) {
(String)x.whatever();
} else {
...
}
x
```

Evaluating conditions using instanceof

To produce smaller and faster code, if you evaluate a condition using `instanceof`, do not evaluate explicitly whether the variable is null.

Code sample

```
if( e instanceof ExampleClass ) { ... }
if( ! ( e instanceof ExampleClass ) ) { ... }
```

Avoiding StringBuffer.append (StringBuffer)

To append a `String` buffer to another, a BlackBerry® Java Application should use `net.rim.device.api.util.StringUtilities.append ( StringBuffer dst, StringBuffer src[, int offset, int length ] )`.

Code sample

```
public synchronized StringBuffer append(Object obj) {
if (obj instanceof StringBuffer) {
StringBuffer sb = (StringBuffer)obj;
net.rim.device.api.util.StringUtilities.append( this, sb, 0, sb )
return this;
}
return append(String.valueOf(obj));
}
```

Avoiding returning null

If you write a public method that returns an object, the method should return null only under the following conditions:

- Your application expects a null value to occur during normal application operation.
- The Javadoc™ `@return` parameter for the method states that null is a possible return value.

If your application does not expect a null return value, the method should throw an appropriate exception, which forces the caller of the method to deal explicitly with the problem. The caller of the method might not need to check for a null return value unless the caller of the method throws a null exception.

Avoiding passing null into methods

Do not pass null parameters into an API method unless the API Reference states explicitly that the method supports them.

Using caution when passing null into a constructor

To avoid ambiguity when passing null into a constructor, cast null to the appropriate object.

If a class has two or more constructors, passing in a null parameter might not uniquely identify which constructor to use. As a result, the compiler reports an error.

By casting null to the appropriate object, you indicate precisely which constructor the compiler should use. This practice also provides forward compatibility if later releases of the API add new constructors.

Code sample

```
new someObject ((someObject)null );
```

Using longs for unique identifiers

Use a `long` identifier instead of a `String` identifier for unique constants, such as GUIDs, hash table keys, and state or context identifiers.

For identifiers to remain unique across a BlackBerry® Java Application, use keys that an application generates based on a hash of a `String`. In the input `String`, include enough information to make the identifier unique. For example, use a fully qualified package name such as `com.rim.samples.docs.helloworld`.

Exiting applications correctly

Before you invoke `System.exit(int status)`, perform any necessary cleanup, such as removing objects from the runtime store that applications no longer require.

Printing the stack trace

When you debug your application, to view the stack trace, catch a `Throwable` instance.

Code sample

```
catch (Throwable t) {
t.printStackTrace();
}
```

Best practice: Using objects judiciously

To allow a BlackBerry® Java Application to efficiently use memory resources, consider the following questions:

- Given the size of an application, are all of the objects necessary?
- Can your application store any objects that represent primitives, such as `Long`, `Integer`, and `Boolean`, as primitives instead of as objects?
- Are all of the persisted objects necessary?
- Do any instances of `Vector` and `Hashtable` exist? Are these instances necessary? If so, how many `Object` handles are not used in the `Vector` or `Hashtable` because the initial size is greater than the needed size?
- How many `Objects` does your application create and then throw away? In other words, how many scope-specific `Objects` does your application create?

Best practice: Reducing the size of compiled code

To reduce the size of compiled code, consider the following guidelines:

- Set appropriate access.
- Avoid creating interfaces.
- Use static inner classes.
- Avoid unnecessary field initialization.
- Import individual classes.

Setting appropriate access

When you create code libraries, you can significantly reduce the size of your compiled code by using the appropriate access modifiers for fields and methods. Declare fields as private whenever possible. In addition to being good coding practice, this allows the compiler to optimize the .cod file. When possible, use the default (`package`) access instead of public access (that is, omit the `public` and `protected` keywords).

Avoiding creating interfaces

When you create API libraries, avoid creating interfaces unless you foresee multiple implementations of the API. Interfaces produce larger, slower code.

Using static inner classes

When you use an inner class to hide one class inside another, but the inner class does not reference the outer class object, declare the inner class as static. This action prevents the creation of a reference to the outer class.

If you use an inner class for name scoping, make it static.

Code sample

```
class outer {
static class inner {
...
}
}
```

Avoiding unnecessary field initialization

Where possible, allow fields to initialize automatically as follows:

- object references are initialized to null
- `int`, `byte`, or `long` is initialized to 0
- Boolean is initialized to false

You must explicitly initialize local variables in a method.

Code sample

```
class BetterExample {
private int fieldsCount;
private Field _fieldWithFocus;
private boolean _validLayout;
private boolean _validLayout;
}
```

Importing individual classes

A BlackBerry® Java Application that uses only a small number of classes from a package should only import the individual classes.

Code sample

```
import net.rim.blackberry.api.browser.Browser;
```

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.

TheBlackBerry 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.

Best practice: Storing text strings in resource files

Instead of using text in source code, design applications to use resource files for localization (adapt to specific languages and regions).

Consider the following guidelines:

- Store the text strings for each locale in a single resource file.
- In your source code, use unique identifiers to make use of the appropriate resource files.
- Design the application to dynamically retrieve the appropriate resource file to display to the BlackBerry® device user based on the locale of the BlackBerry device user.

Multimedia support

Audio support

You can create a BlackBerry® Java Application that works with the audio formats that a BlackBerry device supports. The type of audio format that a BlackBerry device supports depends on the BlackBerry device model number.

For more information about audio support on a BlackBerry device, visit www.blackberry.com/developers.

Imaging support

On a BlackBerry® device that includes a camera, when a BlackBerry device user takes a picture, the BlackBerry device stores the picture in the file system on the BlackBerry device. A BlackBerry® Java Application can access the pictures by using the File Connection API for Java ME that is available in BlackBerry® Java® Development Environment version 4.2 or later. The BlackBerry Java Application can invoke the camera application and listen for events when images are added to the file system.

Video support

You can create a BlackBerry® Java Application that displays images and uses the graphics API classes to work with multimedia content to play a video file on BlackBerry devices that include an integrated media player.

UI and navigation design

3

BlackBerry device user input and navigation

BlackBerry® devices include a keyboard, a trackwheel or trackball, and an Escape key, for input and navigation. The Escape key provides an easy way for BlackBerry device users to go back to the previous screen or remove a menu or dialog box from the screen.

A BlackBerry® Java Application for BlackBerry devices should use the following input and navigation model as closely as possible.

- Clicking the trackwheel or trackball typically invokes a menu.
- Pressing the Escape key changes the display to the previous screen or closes the application from the main screen.

By default, the BlackBerry screen objects provide this functionality without customization; however, you must add menu items and additional UI and navigation logic.

Trackwheel versus Trackball

Trackball sensitivity

Trackball sensitivity refers to the amount of trackball movement that is required for the system to identify the movement as a navigation event, and to dispatch a navigation event to the software layer. The BlackBerry® device hardware measures physical trackball movement using units called ticks. When the number of ticks along an axis surpasses the threshold of the system or a BlackBerry® Java Application, a navigation event along that axis is dispatched to the software layer, and the system resets the tick count to zero. Tick counts are also reset to zero after a certain amount of idle time passes.

You can use the TrackBall API to set the trackball sensitivity. High trackball sensitivity equates to a smaller tick threshold, which means that small trackball movements will trigger navigation events. Conversely, low trackball sensitivity equates to a larger tick threshold, which means that larger trackball movements are required to generate navigation events.

Trackball movement

You can use the Trackball API to filter the trackball movement data that the BlackBerry® device hardware sends to the software layer. The Trackball API can filter out movement "noise" or unwanted movements.

You can also use the Trackball API to change settings such as trackball movement acceleration. Increasing the trackball movement acceleration setting can result in the software layer identifying trackball movements as moving at a faster rate than the rate detected by the BlackBerry device hardware, as long as the user continually rolls the trackball. The trackball sensitivity temporarily increases as the user rolls the trackball without pausing.

Trackwheel

BlackBerry® devices that precede the BlackBerry® Pearl™ 8100 Series do not include a trackball. Instead, they include a trackwheel on the right side of the device. The trackwheel is the primary control for user navigation.

Users can

- roll the trackwheel to move the cursor vertically
- roll the trackwheel while pressing the Alt key to move the cursor horizontally
- click the trackwheel to select objects or open the menu

Guidelines

- Familiarize yourself with both the trackwheel and trackball navigation models, and verify that your application works well for both.

Creating a UI that is consistent with standard BlackBerry UIs

You can use standard MIDP APIs and BlackBerry® UI APIs to create BlackBerry® Java Application UIs.

The BlackBerry UI APIs are a library of UI components that are designed to provide default layouts and behaviors that are consistent with the core BlackBerry device applications.

- Screen components provide a standard screen layout, a default menu, and a standard behavior when the BlackBerry device user presses the Escape key or clicks the trackwheel or trackball.
- Field components provide standard UI elements for date selection, options, check boxes, lists, text fields and labels, and progress bar controls.
- Layout managers provide an application with the ability to arrange components on a BlackBerry device screen in standard ways, such as horizontally, vertically, or in a left-to-right flow.

You can use the BlackBerry UI APIs to create UIs that include tables, grids, and other specialized features. The BlackBerry Java Development Environment uses a standard Java event model to receive and respond to specific types of events. Applications can receive and respond to BlackBerry device user events, such as when the BlackBerry device user clicks the trackwheel, clicks the trackball, or types on the keyboard, and to system events, such as global alerts, real-time clock changes, and USB port connections.

Memory management

4

Managing memory

The BlackBerry® Java® Virtual Machine manages memory usage on the BlackBerry device. The BlackBerry JVM allocates memory, performs garbage collection, and automatically swaps data between SRAM and flash memory. The BlackBerry JVM must also share available memory between the BlackBerry device applications and the BlackBerry® Java Application. The memory capabilities represent the total amount of available memory, which is larger than the available working memory when all of the applications and associated application data exist on the BlackBerry device.

BlackBerry device memory

BlackBerry® devices include the following types of memory:

Memory	Description
flash	The BlackBerry operating system and all application modules are stored persistently in flash memory. When a BlackBerry device user turns on the BlackBerry device, the core operating system and the BlackBerry® Java Application modules use approximately 10 MB to 15 MB of flash memory, depending on the version. Flash memory can store the BlackBerry device user's email messages, organizer data, and other personal information, as well as the data that a BlackBerry Java Application stores in memory.
SRAM	SRAM controls the transient data objects and runtime processes.
microSD expandible memory card	The microSD card stores media files, documents, and persistent data from a BlackBerry Java Application.

Key resources to reserve

- Flash memory: The persistent storage space that is available on the BlackBerry® device is a fixed amount of flash memory, typically in the range of 8 MB to 64 MB.
- Persistent object handles: The handles that are assigned to each persistent object are consumed only by persistent objects. The amount of flash memory on the BlackBerry device determines the fixed number of persistent object handles in the system.
- Object handles: Each object and array of primitives has an object handle associated with it. The amount of flash memory on the BlackBerry device determines the fixed number of object handles in the system.

Best practice: Minimizing memory use

To minimize runtime memory, consider the following guidelines:

- Use primitive types (such as `int` or `Boolean`) instead of objects (such as `String` or `Integer`).
- Do not depend entirely on the garbage collector.
- Avoid creating many objects quickly.
- Set object references to null when you are finished using them.
- Reuse objects as much as possible.
- Move heavy processing to the server. For example, you can filter or sort data before sending it to the BlackBerry® device.

Managing low memory availability

The low memory manager handles memory resources on the BlackBerry® device when the available memory resources fall below a certain threshold. The low memory manager attempts to free used memory to provide more available memory on the BlackBerry device. All applications, including BlackBerry® Java Applications, should work with the low memory manager to free as much memory as possible when the BlackBerry device is low on memory resources.

Identifying low memory availability on a BlackBerry device

The following conditions can cause the low memory manager to attempt to free memory resources:

- The amount of available flash memory on the BlackBerry® device falls below a certain threshold. The flash memory threshold depends on the amount of free RAM in the system. The flash memory threshold ranges between 400 KB and 800 KB.
- The number of persistent object handles that are available on the BlackBerry device falls below 1000 persistent object handles.
- The number of object handles that are available on the BlackBerry device falls below 1000 object handles.

Conserving resources

Best practice: Using efficient data structure selection

Data structure selection defines how many object handles and how much flash memory a BlackBerry® Java Application consumes. Improper data structure selection can consume key resources without improving the BlackBerry Java Application functionality or the BlackBerry device user experience.

Consider the following guidelines:

- The data structure should consist of the minimum possible number of objects, especially when you use high-level objects like a `Vector` or a `Hashtable`. These classes provide significant functionality but are not efficient storage mechanisms and you should avoid using them in the persistent store if possible.
- When possible, use primitives instead of objects, because primitives reduce the number of object handles that are consumed on the BlackBerry device. An array of primitives is an object and consumes an object handle.
- `String` objects are as efficient as byte arrays. A `String` object consumes only one object handle and is equivalent if your application stores all of the characters as a byte. In other words, the value of each character is less than or equal to the decimal value of 255. If your application cannot store characters as a byte, you can store the characters as a `String` because it is equivalent to storing a `char` array.

Best practice: Consolidating objects into object groups

One of the most common errors that application developers encounter is an exhaustion of persistent object handles. The amount of flash memory on the BlackBerry® device determines the fixed number of persistent object handles that are available in the system. Depending on the data structure selection, stored records can quickly exhaust the number of persistent object handles. A persistent object consumes a persistent object handle and an object handle. A transient object consumes only an object handle.

For example, a record that contains ten `String` fields, which represent items like a name, a phone number, and an address, consumes 11 persistent object handles, one for the record object and one for each `String`. If a BlackBerry® Java Application persists 3000 records, the application consumes 33,000 persistent object handles, which exceeds the number of persistent object handles available on a BlackBerry device with 16 MB of flash memory.

You can use the `net.rim.device.api.system.ObjectGroup` class to consolidate the object handles for an object into one group. Using the example in the previous paragraph, if you group the record, the record consumes one persistent object handle instead of 11. The object handles for the `String` fields consolidate under the record object handle.

When you consolidate object handles into one group, the object handle is read-only. You must ungroup the object before you can change it. After you complete the changes, group the object again. If you attempt to change a grouped object without first ungrouping it, an `ObjectGroupReadOnlyException` is thrown.

Ungrouping an object has a performance impact. The system creates a copy of the grouped object and allocates handles to each of the objects inside that group. Therefore, objects should only be ungrouped when necessary.

Garbage collection on a BlackBerry device

RAM garbage collection on a BlackBerry device

The BlackBerry® Java® Virtual Machine initiates a RAM garbage collection operation only when the BlackBerry JVM cannot allocate an object because of a lack of space in RAM. The RAM garbage collection operation typically takes 500 to 600 milliseconds to execute. The garbage collection operation removes any freshly allocated variables that are no longer referenced in RAM. To make sure that the lack of a reference in RAM is a sufficient condition for removing the object, a RAM garbage collection operation can only be performed when objects have not been paged out to flash memory.

Full garbage collection on a BlackBerry device

The full garbage collection operation executes for 1 second on average and should take less than 2 seconds to complete. The full garbage collection operation performs the following actions:

- It performs a RAM garbage collection operation.
- It marks objects in flash memory that are no longer referenced or no longer persisted.
- It releases any nonpersistent object handles in RAM and flash memory.

The system might initiate a full garbage collection operation in the following situations:

- The BlackBerry® Java® Virtual Machine cannot allocate an object because of a lack of available space in RAM.
- A process is about to exceed its currently allocated heap size.
- The BlackBerry JVM cannot allocate a new object because the object handles are not available.
- The BlackBerry device is idle.

Idle garbage collection on a BlackBerry device

Garbage collection does not occur every time that the BlackBerry® device idles. It occurs only when the system considers a garbage collection operation to be beneficial for optimal system performance and maximized battery performance.

To improve performance without impacting the BlackBerry device user experience, the system attempts to perform the following garbage collection operations when the BlackBerry device idles:

- A full garbage collection operation can occur when the BlackBerry device idles for a relatively small amount of time.
- A thorough garbage collection operation can occur when the BlackBerry device idles for a significant period of time.

Data management

5

The BlackBerry® device provides APIs for storing data to persistent memory on the BlackBerry device. The BlackBerry Persistent Store APIs and the MIDP RMS APIs (support for JSR 37 and JSR 118) are available on all Java® based BlackBerry devices. A BlackBerry device that runs BlackBerry® Device Software version 4.2 or later 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.

Support for APIs to store data to persistent memory

Persistent Store APIs

The BlackBerry® Persistent Store APIs are designed to provide a flexible and robust data storage interface. With the BlackBerry Persistent Store APIs, you can save entire Java® objects to memory without having to serialize the data first. When you start the application, you can retrieve the Java object from memory and process the information. No size limit exists on a persistent store; however, the limit for an individual object within the store is 64 KB.

The BlackBerry Persistent Store APIs do not provide a relational database model. You must create an effective object model and manage the relationships between objects, as necessary, using indices and hash tables.

MIDP Record management system APIs

The RMS APIs provide a simple record management system that allows you to create a data store object and persist a series of records within that object. Each record is a byte array, so you must first serialize your data into a byte array format before storing it locally. The RMS APIs do not provide any inherent indexing or relationships between records. The size limit for a single RMS data store is a maximum of 64 KB. An application can create multiple RMS data stores to persist larger amounts of data. The RMS APIs are part of the standard MIDP specification, so all devices that can support MIDP can also support the RMS APIs.

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.

Class or interface	Description
`ConnectionClosedException`	This exception is thrown when an application invokes a method on a closed file connection.
`FileConnection`	An application can use this API to access files or directories.
`FileSystemListener`	An application can use this API to receive status notifications when the application adds or removes a file system root.
`FileSystemRegistry`	An application can use this API as a central registry for file system listeners that listen for the addition or removal of file systems.
`IllegalModeException`	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.

Backing up and synchronizing data

The BlackBerry® Desktop Manager provides a backup and restore tool that a BlackBerry device user can use to save BlackBerry device data to a file on a computer and to restore data to the BlackBerry device.

When an application uses the Synchronization API, the BlackBerry Desktop Manager can back up and restore the application database at the same time as other BlackBerry device databases. You can use the Synchronization API to create data archives or to populate application databases the first time the BlackBerry device connects to the BlackBerry device user's computer.

To synchronize data to remote data sources, you must build the synchronization logic into your BlackBerry Java® Application. Most applications send data to a server-side application using standard HTTP or TCP/IP protocols over the wireless network and the Internet or corporate intranet. You can use XML APIs to generate and parse XML-formatted data to send and receive over the wireless network. However, your client-side and server-side applications must read and write the data properly and acknowledge the successful transmission.

A BlackBerry Java Application might connect to a computer-based application to send the data over a USB connection using the BlackBerry Desktop Synchronization APIs and the BlackBerry Desktop Manager. In this case, you must build an application for Windows® that can read the data from the client through an add-in task for the BlackBerry Desktop Manager. The BlackBerry

device user must manually execute the synchronization by running the BlackBerry Desktop Manager add-in, which notifies the application on the BlackBerry device to send the data to the computer application. You can also write data to the computer application using the native USB protocols.

Wireless data transport

6

Wireless gateways

Java® applications for BlackBerry® devices can use standard HTTP, HTTPS, and TCP socket protocols to establish connections over the wireless network. When an application establishes a connection over the wireless network, it can use one of two wireless gateways to proxy the connection to the Internet or to the corporate intranet. You can design your application to rely on the default gateway that is available to the BlackBerry device user, or you can customize your code to choose a preferred gateway. Design your application to explicitly choose the preferred gateway for the connection and use the default gateway if the preferred method is not available. This might minimize the number of network connection issues that your customers face and let your application use a consistent connectivity model across all network types and wireless operators.

Using the BlackBerry Enterprise Server as an network gateway

When you use the BlackBerry® Enterprise Server as a network gateway, all traffic between your application and the BlackBerry Enterprise Server is encrypted using AES 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 are located on 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 allow the BlackBerry® Java® Application to use it as a secure gateway for managing HTTP and TCP/IP connections to the intranet.

If your application connects to the Internet, you might be able to use the BlackBerry Enterprise Server as a gateway. Network requests travel behind the organization's firewall to the BlackBerry Enterprise Server, which makes the network request to the Internet through the corporate firewall. Administrators can set an IT policy to make sure that the BlackBerry Enterprise Server is the gateway for all wireless network traffic, including traffic destined for the Internet.

If your application connects to the Internet, you can also use either the BlackBerry® Internet Service or the Internet gateway of the wireless server provider to manage connections.

Using the wireless service provider's Internet gateway

Most wireless service providers provide an Internet gateway that offers direct TCP/IP connectivity to the Internet. Some wireless service providers also provide a WAP gateway that allows HTTP connections to occur over the WAP protocol. A BlackBerry® Java® Application can use either of these gateways to connect to the Internet. If your application is for BlackBerry device users who are on a specific wireless network, using the wireless service provider's Internet gateway can often yield good results. If your application is 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. You might 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.

Alternative data transport options

Using email to transport data

You can use the BlackBerry® APIs to create aBlackBerry® Java® Application that uses email as a transport mechanism for sending and receiving data. Email can be an effective way to proactively distribute content to BlackBerry device users if the traditional push models are not available. A BlackBerry Java Application can use the BlackBerry APIs to send email messages and listen for inbound email messages. A BlackBerry Java Application can also access the details and headers of email messages that are stored locally on the BlackBerry device and register listeners for changes in the status of an email message.

Using SMS to transport data

You can use the BlackBerry® APIs to create applications that use SMS as a transport mechanism for sending and receiving data. The BlackBerry APIs allow an application to send SMS messages and listen for inbound SMS messages.

Using PIN messaging to transport data

PIN messaging uses the data channel rather than the voice channel and allows you to address the destination BlackBerry® device by its unique PIN number. PIN messaging can only be used to send data from one BlackBerry device to another. The BlackBerry APIs can also allow an application to programmatically send and receive BlackBerry PIN messages. PIN messaging can be an effective way to implement PIN applications targeting BlackBerry device users only.

BlackBerry application integration

7

Adding custom menu items

A BlackBerry® Java® Application can add custom menu items to the menu on the BlackBerry device for the email, organizer, and phone applications. When a BlackBerry device user selects the custom menu item, theBlackBerry Java Application starts with a reference to the object that the BlackBerry device user selects. For example, a BlackBerry Java Application can add a menu item called Show Location of Sender to the email application. When the BlackBerry device user selects the menu item, the BlackBerry Java Application starts with a reference to the email object that is currently highlighted or that the user opens. The BlackBerry Java Application uses the email address of the sender to determine the location of the sender by retrieving the email address from the contact list, or by retrieving data from a remote server, and then comes to the foreground and displays a map.

Integrating with BlackBerry Device Software applications

A BlackBerry® device application can invoke a BlackBerry Device Software application such as the email, organizer, phone, browser, and camera applications to perform an action or display information. The following examples demonstrate how a BlackBerry device application can invoke a BlackBerry Device Software application:

- The BlackBerry device application invokes the calendar to display a specific date or calendar entry.
- The BlackBerry device application invokes the address book to display a specific contact.
- The BlackBerry device application invokes the browser to open a specific web address.
- The BlackBerry device application invokes the phone to dial a specific number.

Accessing email and organizer data

A BlackBerry® Java® Application can use the BlackBerry APIs to access the details of email messages, contacts, calendar events, tasks, and phone logs that the BlackBerry device stores. A BlackBerry Java Application can read the information, update the information, and create new entries.

Using BlackBerry Messenger with a BlackBerry Application

You can integrate a BlackBerry® Java® Application with the BlackBerry® Messenger application. This could be useful if you are creating a turn-based game application for the BlackBerry device.

To create a BlackBerry Java Application that integrates with the BlackBerry Messenger application, you can use the classes in the `net.rim.blackberry.api.blackberrymessenger` package. For more information about using the `BlackBerryMessenger` class, see the BlackBerry API Reference .

Using listeners to respond to application changes

A BlackBerry® Java® Application can register change listeners on the email and organizer data stores and in the phone application. The listeners allow the BlackBerry Java Application to take immediate action when the BlackBerry device user performs a local event. You can use the email and organizer data listeners to notify a BlackBerry Java Application when new entries arrive or when the BlackBerry device user makes changes such as additions, deletions, or updates, to the existing data. You can use phone listeners to listen for phone call actions, such as the initiation of new calls or calls ending.

Security considerations

8

Data encryption and the BlackBerry Application

Data encryption in transport

If you use the BlackBerry® Enterprise Server as the network gateway for your application, the BlackBerry Enterprise Server encrypts data using AES or TripleDES encryption at all points in the connection between the BlackBerry device and the BlackBerry Enterprise Server behind the organization's firewall. If you require data to be encrypted further between the BlackBerry Enterprise Server and the destination server, you can use the HTTPS protocol and use SSL/TLS encryption.

If your application uses the BlackBerry® Internet Service or the Internet gateway of the wireless service provider, data traffic is not encrypted. If your BlackBerry device users prefer, you can use HTTPS to encrypt the data, or you can use the Java® APIs for encryption to apply your own symmetric key or public key cryptography.

Data encryption on the BlackBerry device

Administrators can set an IT policy to make sure that all BlackBerry® device user data stored in the BlackBerry device applications is encrypted locally in flash memory. You can create aBlackBerry Java® Application that uses APIs to register the data so that the encryption service encrypts the data with the same security key before storing it in flash memory.

Access to memory

The BlackBerry® Java® Development Environment is designed to inhibit applications from causing problems accidentally or maliciously in other applications or on the BlackBerry device. BlackBerry 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). A BlackBerry® Java Application can only access persistent storage or user data, or communicate with other applications, through specific BlackBerry APIs. Research In Motion must digitally sign a BlackBerry Java Application that uses these BlackBerry APIs, to provide an audit trail of applications that use sensitive APIs.

Authentication

BlackBerry device authentication and IT policy

BlackBerry® device users can set a password for their BlackBerry devices. When the device password is active, the BlackBerry device users must provide the password to access the data and applications. Using device passwords is a good first step to limiting access to your BlackBerry Java® Application on the BlackBerry device.

Administrators can use the IT policies that are provided in the BlackBerry® Enterprise Server to make sure that BlackBerry devices in the organization are password-protected. Administrators can also use IT policies to remotely lock a BlackBerry device, change the password, or remove all of the data.

Application authentication

For applications where security features are critical, you might want to provide a login screen that requires the BlackBerry® device user to log into the application on the BlackBerry device before using it. The UI classes provide simple password fields that hide the text entry with asterisk characters. Login screens might negatively impact the BlackBerry device user experience, and if the BlackBerry device user sets a password to protect the BlackBerry device, your application might not require a login screen.

Server-side authentication

If your application connects to an application on a server or to the Internet or an intranet, you might want to include additional authentication features when the BlackBerry® device users log into the server. Most applications that require user authentication rely on HTTP Basic authentication, which uses a simple user name and password combination. You can use HTTP Basic authentication by adding the correct HTTP headers while opening the HTTP connection. You can also add more advanced authentication using certificates; however, most applications do not require it.

Controlled APIs and code signing

Research In Motion tracks the use of sensitive APIs for security and export control reasons. In the BlackBerry®API reference, RIM identifies a controlled class or method with a lock icon or a signed note. To use controlled classes or methods in your applications and before you can install the application .cod files on the BlackBerry device, you must sign your application using a key, or signature, from RIM. Other functionality, such as the ability to execute when the application starts, might require that you sign your applications.

While the RIM registration process covers the use of most controlled APIs, some cryptography classes that are related to public and private key cryptography contain technology from Certicom™. To use these classes, you must register with and obtain a license from Certicom directly. The RIM registration process does not include the use of Certicom classes.

To test and debug your code before you receive the code signatures, you can use the BlackBerry® Smartphone Simulator. You must sign the application before you install it on BlackBerry devices. You do not send your actual code to RIM. You can use the BlackBerry® Signature Tool to send a SHA-1 hash of your code file so that the signing authority system can generate the necessary signature.

For more information about registering and obtaining code signatures, see the *BlackBerry Signing Authority ToolVersion 1.0 - Password Based Administrator Guide*. For more information about registering and using classes, visit www.blackberry.com/developers/index.shtml.

BlackBerry APIs with controlled access

You can run applications that use controlled APIs in the BlackBerry® Smartphone Simulator without code signatures; however, you must obtain code signatures from Research In Motion before you can install these applications on BlackBerry devices.

You can use the following categories of RIM controlled APIs:

- Runtime APIs
- BlackBerry® Application APIs
- BlackBerry Cryptographic API

If you use any of the following BlackBerry API packages, your application requires code signatures before you can install it on 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`

For more information about RIM controlled APIs, see the BlackBerry API reference.

IT policy support

You can use the APIs in the `net.rim.device.api.itpolicy` package to access the IT policy information on the BlackBerry® device and change the behavior or functionality of a BlackBerry device application. Each IT policy item consists of a name, a description, and a value. The value can be a string, integer, or Boolean value.

BlackBerry device IT policy settings are automatically synchronized and updated over the wireless network with BlackBerry® Device Software version 3.6 or later. With earlier versions of the BlackBerry Device Software, IT policy settings are updated when the BlackBerry device user synchronizes the BlackBerry device with the computer.

For more information about IT policies, see the *BlackBerry Enterprise Server Policy Reference Guide*.

Application control

The BlackBerry® Enterprise Server application control policy rules are designed to allow or prevent the installation of specific applications on the BlackBerry device and to limit the permissions of applications on the BlackBerry device. For example, administrators can use the application control policy to make sure that a game application on the BlackBerry device cannot access the phone application.

The administrator can apply application control policies only when the BlackBerry device is associated with a BlackBerry Enterprise Server. If the administrator or a BlackBerry device user denies the application access to one of the protected areas, the application throws a `ControlledAccessException`.

File encryption on microSD cards

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 non-BlackBerry platforms.

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.

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.

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.

Test a BlackBerry Java Application

9

Obfuscating a BlackBerry Java Application

The compiler for the BlackBerry® Java® Development Environment and the BlackBerry® JDE Plug-in for Eclipse™ is set to minimize the size of the application. The compiler produces a .cod file that provides obfuscation-like services that are similar to those that obfuscation packages provide in an effort to reduce the size of the .cod file. For example, the BlackBerry Java Development Environment removes the following information from a .cod file:

- all debug information
- local variable names
- source line numbers
- private method and member names

It is not typically necessary for you to provide obfuscation for your applications in addition to the existing obfuscation that, by default, the BlackBerry JDE provides. In fact, Research In Motion does not perform any additional obfuscation of its own products.

The BlackBerry JDE does not integrate support for obfuscation through third-party tools. You must include a command-line procedure to obfuscate .cod files for use on BlackBerry devices.

Preverifying a BlackBerry Java Application

When you preverify your classes, you reduce the amount of processing that the BlackBerry® device must perform when you install your application. To partially verify your classes before you install your application on a BlackBerry device, you can use the Preverify tool, available with the BlackBerry® Java® Development Environment. You can use the BlackBerry® Smartphone Simulator to preverify the .cod files.

For more information about using the BlackBerry Smartphone Simulator, see the *BlackBerry Device Simulator User Guide*. For more information about using the Preverify tool, see the *BlackBerry Java Development Environment Development Guide*.

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 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.

BlackBerry simulators

BlackBerry Smartphone Simulator

The BlackBerry® Smartphone Simulator is designed to emulate a BlackBerry experience without using a real BlackBerry device. The BlackBerry Smartphone Simulator is an application that you install on your computer that shows an image of the BlackBerry device model of your choice. This image has the look and feel of an actual BlackBerry device. The functionality includes the same user interaction of an actual BlackBerry device (including the trackwheel or trackball and the keyboard), the same applications, and the same features, such as email messages, phone, and Internet browsing.

The BlackBerry Smartphone Simulator also serves as a platform on which applications can run. This includes the ability for the applications to make network connections, store data, and handle email messages. The BlackBerry Smartphone Simulator includes the BlackBerry device applications that are typically available on BlackBerry devices and you can install and test your own applications. You can simulate and test various connectivity and state changes using the BlackBerry Smartphone Simulator.

When you use the BlackBerry Smartphone Simulator to perform testing, you might need to simulate additional BlackBerry services. The BlackBerry® MDS Simulator and the BlackBerry® Email Simulator are available for this purpose.

To get the BlackBerry Smartphone Simulator, visit www.blackberry.com/developers and download the BlackBerry® Java® Development Environment or the BlackBerry Java Development Environment Component Package.

BlackBerry MDS Simulator

The BlackBerry® MDS Simulator is designed to simulate the BlackBerry MDS Connection Service component of the BlackBerry® Enterprise Server. When you use the BlackBerry Smartphone Simulator with the BlackBerry MDS Simulator you can test network, push HTTP, and browser applications that are designed for use with a BlackBerry Enterprise Server. To get the BlackBerry MDS Simulator, visit www.blackberry.com/developers and download the BlackBerry® Email and MDS Services Simulator Package.

BlackBerry Email Simulator

The BlackBerry® Email Simulator is designed to send and receive messages between the BlackBerry® Smartphone Simulator and either a messaging application, such as Microsoft® Outlook®, or POP3 and SMTP servers. You do not require a BlackBerry® Enterprise Server. To get the BlackBerry Email Simulator, visit www.blackberry.com/developers and download the BlackBerry® Email and MDS Services Simulator Package.

Making applications available

10

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.

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.

Glossary

11

AES
Advanced Encryption Standard

API
application programming interface

CLDC
Connected Limited Device Configuration

HTTP
Hypertext Transfer Protocol

HTTPS
Hypertext Transfer Protocol over Secure Sockets Layer

I/O
input/output

IMAP
Internet Message Access Protocol

IPC
interprocess communication

JDWP
Java® Debug Wire Protocol

JSR
Java® Specification Request

JTWI
Java® Technology for the Wireless Industry

JVM
Java® Virtual Machine

MIDP
Mobile Information Device Profile

NVRAM
Non-Volatile Random Acess Memory

PDAP
PDA Optional Packages for the J2ME Platform

PIM
personal information management

PIN
personal identification number

POP
Post Office Protocol

RAPC
RIM Application Program Compiler

RMS
Record Management System

SHA
Secure Hash Algorithm

SMS
Short Message Service

SMTP
Simple Mail Transfer Protocol

SRAM
static random access memory

SSL
Secure Sockets Layer

TCP
Transmission Control Protocol

TLS
Transport Layer Security

Triple DES
Triple Data Encryption Standard

VPN
virtual private network

WAP
Wireless Application Protocol

XML
Extensible Markup Language

Legal notice

12

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.

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