



BlackBerry Java Development Environment

Version 4.3.0

Fundamentals Guide

BlackBerry Java Development Environment Version 4.3.0 Fundamentals Guide

Last modified: 9 August 2007

Part number: 12080720

At the time of publication, this documentation is based on the BlackBerry Java Development Environment Version 4.3.0.

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

©2007 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images, and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, BlackBerry, "Always On, Always Connected" and the "envelope in motion" symbol are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

AOL is a trademark of America Online, Inc. Bluetooth is a trademark of Bluetooth SIG. Java and Javadoc are trademarks of Sun Microsystems, Inc. Microsoft, Hotmail, MSN, Windows Mobile, ActiveX, and Windows are trademarks of Microsoft Corporation. Yahoo! is a trademark of Yahoo! Inc. Palm is a trademark of Palm Trademark Holding Company, LLC. HotSync is a trademark of Palmsource, Inc. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

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

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

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

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

Published in Canada

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

Contents

| | | |
|----------|--|-----------|
| 1 | Understanding BlackBerry and programming for BlackBerry devices | 9 |
| | BlackBerry Enterprise Solution and BlackBerry Internet Service | 9 |
| | BlackBerry Enterprise Solution | 9 |
| | BlackBerry Internet Service | 10 |
| | Java ME and Java APIs for BlackBerry | 10 |
| | Java ME | 10 |
| | Java ME and Java API extensions for a BlackBerry device application | 11 |
| | Programming in Java for BlackBerry | 12 |
| | API control and code signing | 12 |
| | Object modelling | 13 |
| | Multithreading | 13 |
| | Localization | 13 |
| | Release cycles and BlackBerry JDE versions | 13 |
| | Application and runtime models | 14 |
| | MIDlet applications | 14 |
| | CLDC applications | 14 |
| | Common application models | 15 |
| 2 | Designing BlackBerry device applications | 17 |
| | Designing applications for mobile device users | 17 |
| | Mobile device use versus computer use | 17 |
| | Designing the UI and navigation | 18 |
| | BlackBerry device user input and navigation | 18 |
| | Creating a UI that is consistent with standard BlackBerry UIs | 20 |
| | Screens and fields | 20 |
| | Support for multiple screen resolutions and sizes | 21 |
| 3 | Best practices for writing BlackBerry device applications | 23 |
| | Guidelines for writing efficient code | 23 |
| | Reducing the size of compiled code | 28 |
| | Minimizing memory use | 29 |
| | Key resources to reserve | 29 |
| | Conserving resources | 30 |

| | |
|---|-----------|
| Using objects judiciously | 30 |
| Conserving storage space | 31 |
| Best practices | 31 |
| 4 Memory and data storage | 33 |
| BlackBerry device memory | 33 |
| How the BlackBerry device manages memory | 33 |
| Garbage collection on BlackBerry devices | 34 |
| RAM garbage collection | 34 |
| Full garbage collection | 34 |
| Idle garbage collection | 34 |
| Low memory manager | 35 |
| Low memory conditions | 35 |
| Memory management and persistent objects | 35 |
| Persistent storage | 36 |
| Storage on removable media | 36 |
| File encryption | 36 |
| Implementing code security and file data encryption | 37 |
| Accessing data on the microSD media card | 37 |
| Backing up data | 38 |
| Synchronizing data | 38 |
| 5 Wireless data transport | 39 |
| Wireless gateways | 39 |
| Using the BlackBerry Enterprise Server as an intranet gateway | 39 |
| Using the wireless service provider's Internet gateway | 40 |
| Transport options | 40 |
| Alternate transport methods | 40 |
| 6 Integrating multimedia features | 41 |
| Audio support | 41 |
| Audio Recording and Voice Playback | 41 |
| Imaging support | 41 |
| Video support | 42 |
| 7 Integrating with BlackBerry applications | 43 |
| Adding custom menu items | 43 |

| | |
|--|-----------|
| Invoking native applications and phone calls | 43 |
| Accessing email and PIM data | 44 |
| Using BlackBerry Messenger with a BlackBerry application | 44 |
| Using listeners to respond to application changes | 44 |
| 8 Security | 45 |
| Data encryption | 45 |
| Data encryption in transport | 45 |
| Data encryption on the BlackBerry device | 45 |
| Authentication | 46 |
| BlackBerry device authentication and IT policy | 46 |
| Application authentication | 46 |
| Back-end authentication | 46 |
| Controlled APIs and code signing | 46 |
| Controlled APIs | 47 |
| Application control | 48 |
| Access to memory | 48 |
| IT policies | 48 |
| 9 Localizing applications | 49 |
| Storing text strings in resource files | 49 |
| Storing resources for a locale | 49 |
| 10 Testing and setting up applications | 51 |
| Why it is not necessary to obfuscate applications | 51 |
| Preverify applications | 51 |
| Testing applications using the BlackBerry JDE | 52 |
| Testing applications on BlackBerry devices | 52 |
| Setting up applications through the computer application | 52 |
| Setting up applications over the wireless network | 53 |
| Acronym list | 55 |

Understanding BlackBerry and programming for BlackBerry devices

BlackBerry Enterprise Solution and BlackBerry Internet Service
 Java ME and Java APIs for BlackBerry
 Programming in Java for BlackBerry
 Application and runtime models
 Common application models

BlackBerry Enterprise Solution and BlackBerry Internet Service

| Service | Description |
|---------------------------------|--|
| BlackBerry® Enterprise Solution | <p>The BlackBerry Enterprise Solution makes possible the wireless extension of corporate email messages and BlackBerry Java® Applications through the BlackBerry® Enterprise Server.</p> <p>The BlackBerry Enterprise Solution includes the BlackBerry® Enterprise Server, which exists behind the corporate firewall of the enterprise customer. The BlackBerry Enterprise Server provides a wireless gateway that allows applications on the BlackBerry device to connect to corporate application servers using standard HTTP and TCP/IP protocols.</p> |
| BlackBerry® Internet Service | <p>The BlackBerry Internet Service provides a wireless messaging solution for small enterprises and individual BlackBerry device users. It simplifies wireless connectivity to publicly accessible messaging and collaboration systems.</p> <p>The BlackBerry Internet Service also provides centrally hosted gateways that let BlackBerry device users access public email and other Internet-based applications without using a BlackBerry Enterprise Server.</p> |

BlackBerry device users can choose to use one of these two services, or they can use both services on the same device. Understanding the difference between these two solutions and which types of users you plan to support in your applications is important, as they might impact which modes of transport you use and how you manage data synchronization.

BlackBerry Enterprise Solution

Most organizations that choose the BlackBerry® Enterprise Solution™ for accessing corporate email messages or applications run the BlackBerry Enterprise Server as part of the BlackBerry Enterprise Solution. The BlackBerry Enterprise Server exists behind the corporate firewall to provide a wireless gateway for all BlackBerry devices in the organization to access corporate email and PIM 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 custom Java® applications

BlackBerry MDS

To let BlackBerry® Java® Applications access resources behind the corporate firewall, the BlackBerry Enterprise Server includes the BlackBerry® Mobile Data System (MDS). The BlackBerry MDS provides an HTTP and TCP/IP proxies for third-party applications, which lets the BlackBerry device 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 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, letting server-side applications behind the corporate firewall push content to applications on BlackBerry devices.

BlackBerry Internet Service

Individuals and small groups of BlackBerry® device users who do not run a BlackBerry Enterprise Server use the BlackBerry Internet Service. BlackBerry Internet Service users can forward messages from their public email account such as Microsoft® MSN® Hotmail®, AOL®, and Yahoo!®, as well as POP3 and IMAP-based email accounts to their BlackBerry devices. BlackBerry Internet Service users might also have a hosted messaging account that is specific to their BlackBerry service.

Devices that register with the BlackBerry Internet Service include support for direct HTTP and TCP/IP connectivity to the Internet from third-party BlackBerry Java® Applications.

Java ME and Java APIs for BlackBerry

The BlackBerry® Java® Development Environment is a fully integrated development and simulation environment for building custom applications for BlackBerry devices. With the BlackBerry JDE, developers can build custom client applications using the Java Micro Edition programming language and the extended Java APIs for BlackBerry. Unless otherwise stated, the Java APIs discussed in this document are relevant to all BlackBerry devices running BlackBerry Device Software Version 4.0 or later.

Java ME

Java® ME is an industry standard platform that defines common sets of Java APIs for different types of mobile and embedded devices. It uses an object-oriented programming language with the same syntax and programming model as the standard Java language that is used for developing computer and server-side applications. A Java ME application on a BlackBerry® device runs within the BlackBerry JVM, which provides all of the runtime services to the applications and performs functions such as typical memory allocations, security checks, and garbage collection.

Java ME includes subcategories, or profiles, which represent different classes of mobile and embedded devices. For example, the Java ME MIDP standard addresses the API and BlackBerry JVM needs of constrained mobile devices with a user interface, such as cellular phones or PDAs. Other profiles within the Java ME cover other types of embedded devices, such as TV set-top boxes and vending machine controllers. The BlackBerry device supports the Java ME MIDP standard as defined in JSR 118. If you have developed applications for other mobile devices, you might be familiar with the Java ME MIDP standard because several third-party BlackBerry JVM implementations run Java ME MIDP on Palm® OS, Windows Mobile®, and other mobile device operating systems.

MIDP provides a common API set that any BlackBerry device can support, regardless of its underlying operating system. Therefore, developers can often build one Java application using the MIDP standard APIs and run that application on many different types of devices.

For more information about the Java ME standard, visit <http://java.sun.com/javame>.

Java ME and Java API extensions for a BlackBerry device application

The BlackBerry® device and the BlackBerry JDE support the Java® ME MIDP standard, which provides a core set of Java APIs you can use to build mobile device applications. The BlackBerry device supports the following JSRs:

| JSR | Description |
|---|--|
| JSR 75 and File Connection API for Java ME | Available only on devices that run BlackBerry Device Software Version 4.2 or later. <ul style="list-style-type: none"> See "Persistent storage" on page 36 for more information. See "Implementing code security and file data encryption" on page 37 for more information. See "Accessing data on the microSD media card" on page 37 for more information. See "Imaging support" on page 41 for more information. |
| JSR 82: Java APIs for Bluetooth® | BlackBerry devices are designed to support the Serial Port Profile (SPP), Object Exchange Profile (OBEX), and Object Push Profile (OPP) profiles. <ul style="list-style-type: none"> See the <i>BlackBerry Java Development Environment Development Guide</i> for more information. |
| JSR 118: MIDP v2.0 | <ul style="list-style-type: none"> See "Java ME" on page 10 for more information. See "Application and runtime models" on page 14 for more information. See "Persistent storage" on page 36 for more information. See the <i>BlackBerry Java Development Environment Development Guide</i> for more information. |
| JSR 120: WMA v1.1 | <ul style="list-style-type: none"> See the <i>BlackBerry Java Development Environment Development Guide</i> for more information. |
| JSR 135: MMA v1.1 (subset defined within the MIDP specification) | <ul style="list-style-type: none"> See the <i>BlackBerry Java Development Environment Development Guide</i> for more information. |
| JSR 139: CLDC v1.1 | <ul style="list-style-type: none"> See "CLDC applications" on page 14 for more information. |
| JSR 172: J2ME Web Services | <ul style="list-style-type: none"> The JSR 172: J2ME Web Services specification provides access from J2ME to web services. You can create a BlackBerry device application that lets a BlackBerry device act as a web service client. |
| JSR 177 | <ul style="list-style-type: none"> See the <i>BlackBerry Java Development Environment Development Guide</i> for more information. |
| JSR 179: Location API for Java ME (supported on BlackBerry devices with GPS capabilities) | <ul style="list-style-type: none"> See the <i>BlackBerry Java Development Environment Development Guide</i> for more information. |
| JSR 185: JTWI | <ul style="list-style-type: none"> See JSR 139: CLDC v1.1, JSR 118: MIDP v2.0, JSR 120: WMA v1.1 in this guide. |

| JSR | Description |
|--|--|
| JSR 205: Wireless Messaging API 2.0 | <ul style="list-style-type: none"> Lets developers create a BlackBerry Application that sends and receives messages that contain images, sound, video, and text. |
| JSR 211: Content Handler API | <ul style="list-style-type: none"> Lets developers create a BlackBerry Application that can register as a content handler application and handle Uniform Resource Identifiers based on MIME-types or schemes. |
| JSR 238: Mobile Internationalization API | <ul style="list-style-type: none"> The Mobile Internationalization API (JSR 238) allows you to create localizable application resources outside of the application source code. At runtime, an application can access, select, and use the localizable resources that match the BlackBerry device's locale. |

Visit <http://jcp.org/en/home/index> for more information on JSRs.

BlackBerry® devices also support a large set of additional Java® APIs that are not part of the standard JSR definitions. These additional APIs provide you with more features and capabilities when you develop applications for BlackBerry devices. You do not need to use these additional APIs in your applications, but they can provide greater features and functionality over what is available in the standard MIDP API libraries. Between the MIDP standard APIs and the BlackBerry specific Java API extensions, the following types of APIs are available to you when developing your applications:

| API | Description |
|------------------------------|--|
| User Interface APIs | Use to create screens, menu items, and all components of the user interface. |
| Persistent Data Storage APIs | Use to store custom data locally within your application. BlackBerry services do not provide a relational database model. |
| Networking and I/O APIs | Use to establish network connections and read or write data to a server-side application. |
| Event Listeners | Use to respond to BlackBerry device user or system-initiated events on a BlackBerry device. |
| Application Integration APIs | Use to integrate with the existing BlackBerry email, phone, calendar, contacts, browser, camera, media player, and task list applications. |
| Additional Utilities | Additional APIs for data encryption and compression, XML parsing, Bluetooth connectivity, location-based services, etc. |

Programming in Java for BlackBerry

API control and code signing

When you develop BlackBerry® Java® Applications for BlackBerry devices, you can use only the public Java APIs that are published and documented in the BlackBerry Java Development Environment Javadocs. The BlackBerry JVM 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 methods to access BlackBerry device user data or other information on the BlackBerry device that is considered sensitive. You can use these APIs, but you must request and receive a set of code signing keys from Research In Motion (RIM). You must then digitally sign your application before you load it on a BlackBerry device. Code signing does not certify or approve an application; it lets RIM identify the author of an application that uses sensitive APIs, if the application is malicious.

To request a set of code signing keys, complete the web form at <http://www.blackberry.com/developers/downloads/jde/api.shtml>. You will receive your set of code signing keys in about 10 days.

Object modelling

Whether you use the MIDlet or the CLDC application model, you must use an object-oriented approach when designing 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 environment 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.

Localization

BlackBerry® supports international localization and resource files. To create resource files and strings for multiple languages that the BlackBerry Java® Application can access dynamically, place your application's location information in resource files. If users change the language settings on their BlackBerry devices, the BlackBerry JVM automatically switches your application to the resource files of the new language.

Release cycles and BlackBerry JDE versions

All BlackBerry® devices include a specific BlackBerry Device Software version and a pre-loaded BlackBerry JVM. To determine the BlackBerry Device Software version 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.

The BlackBerry Device Software makes available certain Java® APIs and determines the version of the BlackBerry JDE that you can use to develop applications.

RIM releases Java APIs and different versions of the BlackBerry JDE with each major BlackBerry Device Software and BlackBerry JVM release. 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 the APIs that were introduced in BlackBerry Device Software Version 4.0 and BlackBerry JVM Version 4.0. applications you create using BlackBerry JDE Version 4.0 only work on BlackBerry devices running BlackBerry Device Software Version 4.0 or later.

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.

Application and runtime models

When building an application for BlackBerry® devices, use either the MIDlet or CLDC application model.

MIDlet applications

The MIDlet application model is part of the Java® Platform MicroEdition 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 can be portable to other devices that also support the MIDP standard. | <ul style="list-style-type: none">• Applications can use only the user interface APIs that exist in the <code>javax.microedition.lcdui</code> library.• The model assumes that all application processes will terminate once the application closes.• Applications cannot be designed to start automatically in the background when the device turns on. |

CLDC applications

CLDC applications typically extend the `UiApplication` class and start with a standard `main()` method.

| Advantages | Disadvantages |
|---|---|
| <ul style="list-style-type: none">• BlackBerry® User interface APIs provide more functionality and flexibility than the standard <code>javax.microedition.lcdui</code> library.• Applications can run active background threads after they turn off.• Applications 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. |

Most of the sample applications that the BlackBerry Java® Development Environment includes 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.

Common application models

You can build a number of types of applications that end users and enterprise customers can use on their BlackBerry® devices. Most applications for BlackBerry devices use the wireless network to provide BlackBerry device users with access to remote applications or data; however, you can also develop standalone applications or computer synchronization applications that do not rely on the wireless network. Application developers typically build applications for BlackBerry devices that fit one of the following models:

| Application model | Description |
|---|--|
| Standalone applications | Some personal productivity and entertainment applications, such as games and static reference guides, can run as offline applications. Once an application exists on the BlackBerry device, it does not need to connect to the wireless network or to the computer. You can use the BlackBerry Java® Development Environment to build standalone 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. |
| Applications with desktop synchronization | Some personal productivity applications, such as reference guides and personal document management applications, directly connect to BlackBerry device users' computers to manage and synchronize data that resides on the BlackBerry device users' computers. Once the application exists on the BlackBerry devices, BlackBerry device users must synchronize information manually by connecting their BlackBerry devices to their computers with a serial connection, a USB connection, or a Bluetooth® connection. You can use the BlackBerry JDE to build applications with desktop synchronization capabilities; however, BlackBerry 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. |
| Applications with wireless access, wireless synchronization, or wireless alerting | BlackBerry device users use their BlackBerry devices as wireless connected devices, not as disconnected PDAs. Many users no longer use their desktops to synchronize data; they use the wireless network instead. Therefore, most third party applications for BlackBerry devices use the wireless connection to the Internet or the corporate intranet for BlackBerry device users to access remote data and applications. The BlackBerry JDE provides ways for applications to establish network connections to servers on the Internet or the corporate intranet. You can also use the BlackBerry JDE to write applications that push content proactively to BlackBerry devices in environments that use the BlackBerry Enterprise Server. |

Designing BlackBerry device applications

Designing applications for mobile device users

Designing the UI and navigation

Designing the UI and navigation

Designing applications for mobile device users

Data requirements for mobile device users are often specific to the immediate task. Mobile device users expect to locate information quickly. For example, a CRM system can provide massive amounts of information. A mobile device user requires a small amount of that information at one time. When you design a mobile application, look for ways to streamline data selection and presentation so that mobile device users can efficiently retrieve the information they require. Effective, streamlined applications can encourage their acceptance and use by mobile device users.

Mobile device use versus computer use

When you design your application, remember that mobile device users and computer users often use different models and have different requirements that are dictated by the differences between the mobile environment and the computer environment.

| Feature | Mobile devices | Computers |
|-----------------|--|--|
| Display size | They display a limited number of characters. | Their browsers can display more information than mobile device browsers. |
| Processor speed | They have slower processor speeds. | They have processor speeds that can be 5 to 60 times faster than processors on mobile devices. |
| Network | Wireless networks have slower transfer rates than standard LANs. | Standard LANs have faster transfer rates than wireless networks. |
| Memory | They have lower memory availability. | They have high memory availability. |
| Battery life | They have a lower battery life. | They have a longer battery life. |

Designing the UI and navigation

BlackBerry device user input and navigation

Custom BlackBerry® Java® Applications for BlackBerry devices should follow this input and navigation model as closely as possible. Clicking the trackwheel should typically invoke a menu. Pressing the Escape key should let the BlackBerry device user go back to the previous screen or close 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.

BlackBerry devices do not include a touch screen for navigation, and BlackBerry device users do not use a stylus for input. Instead, BlackBerry devices include a keyboard along with 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.

Trackwheel versus Trackball

Most BlackBerry® devices include a trackwheel on the right-hand side of the BlackBerry device, which is the primary interface for user navigation.

BlackBerry device users typically use the trackwheel in the following ways:

- Move the cursor vertically from one field to the next by rolling the trackwheel up and down.
- Move the cursor horizontally by holding the Alt key as they roll the trackwheel.
- Select an item or show a contextual menu by clicking the trackwheel.

The BlackBerry Pearl™ 8100 smartphone is the first BlackBerry device to include a trackball instead of a trackwheel. The trackball is located on the front of the BlackBerry 8100 and is the primary interface for user navigation.

BlackBerry device users typically use the trackball in the following ways:

- Move the cursor up, down, left, or right by rolling the trackball in the appropriate direction.
- Select an item, show a default action, or show a contextual menu by clicking the trackball. For example, if the BlackBerry device user highlights an email message and clicks the trackball, the email message opens.

BlackBerry devices with a trackball also include a Menu key to the immediate left of the trackball. The Menu key is used to invoke a menu of options.

You should become familiar with both the trackwheel and trackball navigation models and verify that the BlackBerry Java® Applications you create work well under both models.

To develop an application for a BlackBerry device with a trackwheel or a trackball, use the BlackBerry JDE Version 4.0.0 to create the application.

To develop an application only for a BlackBerry device with a trackball, use the BlackBerry JDE Version 4.2.0 or later and the new navigation APIs it includes to create the application. See the BlackBerry Developer Zone at <http://www.blackberry.com/developers> for knowledge base articles about creating an application using a specific version of the BlackBerry JDE.

Accessing the trackball

Use the Trackball API to configure trackball characteristics related to sensitivity and movement.

| Trackball characteristic | Description |
|--------------------------|---|
| Trackball sensitivity | <p>Trackball sensitivity refers to the amount of trackball movement 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.</p> <p>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.</p> |
| Trackball movement | <p>You can use Filters to filter the trackball movement data that the BlackBerry device hardware sends to the software layer. Filters filter out movement “noise” or unwanted movements.</p> <p>You can also use Filters 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.</p> |

See the *API Reference* for more information about using the TrackBall API.

Simple user interaction scenario

A typical BlackBerry® device user might interact with a simple data collection form BlackBerry Java® Application in the following way:

- After opening the application, the BlackBerry device user rolls the trackwheel down until the cursor is in the first editable field.
- For a text field, the BlackBerry device user types the text with the keyboard.
- For a radio button or check box field, the BlackBerry device user presses the Space key or clicks the trackwheel to select the appropriate option.
- The BlackBerry device user rolls the trackwheel down to highlight the next editable field and completes the field, as necessary.
- Once the BlackBerry device user completes all of the fields, the BlackBerry device user clicks the trackwheel, which displays a menu with actions and options. To save or submit the data, the BlackBerry device user clicks the trackwheel to highlight the menu item and then clicks the trackwheel.
- To quit during the process, the BlackBerry device user presses the Escape key to exit the application. If the BlackBerry device user made changes, the BlackBerry device displays a dialog box that lets the BlackBerry device user save, discard, or cancel the changes.

Creating a UI that is consistent with standard BlackBerry UIs

The BlackBerry® Java® Development Environment provides the following two sets of APIs that you can use to create application UIs: standard MIDP APIs and BlackBerry APIs. The BlackBerry UI APIs are a library of prebuilt UI components that provide default layouts and behaviors that are consistent with the core BlackBerry 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.

The BlackBerry UI APIs allow you to create UIs that include tables, grids, and other specialized features. The BlackBerry Java 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.

See the *BlackBerry MIDlet Developer Guide* for more information on standard MIDP APIs.

Screens and fields

To initialize simple Screen objects, use the UI APIs. Once you create a screen, you can add fields and a menu. To display the screen to the BlackBerry® device user, push it on the UI stack. The menu object will contain associated menu items that are runnable objects that perform specific tasks when BlackBerry device users select them. For example, menu items might invoke the code that is necessary to establish a network connection, to commit a data object to memory, or to close the BlackBerry Java® Application. For more sophisticated custom applications, you can customize the UI and add new field types, as required. You can also provide custom navigation and trackwheel or trackball behavior.

Navigation and user interface considerations for BlackBerry devices

If you port your application from another mobile device operating system to the BlackBerry® device, keep the following items in mind:

| Item | Description |
|--|---|
| Moving to the trackwheel, trackball, and menu model. | <ul style="list-style-type: none"> • Trackwheel or trackball navigation: BlackBerry® device users cannot move to a point on the screen by tapping on a point on the screen. Design your field layout so that when BlackBerry device users move the trackwheel or trackball, they can move from field to field in a logical way. • Screen versus menu: Consider which action items you place on the BlackBerry device screen versus in the menu. For example you might place a "Submit" button at the bottom of the screen in a Palm® application but place it as a menu item in your BlackBerry Java® Application. • Screen length: To prevent BlackBerry device users from having to scroll too much, do not add too many fields or too much text on the screen. |

| Item | Description |
|--|---|
| Using trackball click events and context menus | <ul style="list-style-type: none"> • Your BlackBerry® Java® Application should perform one common action when the user clicks the trackball. For example, in the Message list application, from the message list view, if a user highlights a message, a trackball click opens the message. • If a BlackBerry device user clicks the trackball and more than one common action could occur given the context, your application should display a Short Menu of context sensitive tasks and actions items. For example, when reading an email, if a BlackBerry device user clicks the trackball, a Short Menu will be displayed with the following menu items: <ul style="list-style-type: none"> • File • Reply • Forward • Reply To All • Delete <p>-----</p> <ul style="list-style-type: none"> • Full Menu <p>Use the Full Menu to display less common menu items or a list of all available menu items for an application. Clicking "Full Menu" or pressing the dedicated Menu button will display the Full Menu.</p> |
| Using custom fields | <p>The BlackBerry APIs do not include more advanced fields, such as tables and dynamic trays. To add advanced fields, you can override some of the existing UI elements and customize the layout and navigation.</p> |

Before you design your application, consider using the core applications on the BlackBerry device or the BlackBerry Device Simulator to learn more about the navigation model and best practices for designing your screens. See the *BlackBerry Device Simulator User Guide* for more information about the BlackBerry Device Simulator.

Support for multiple screen resolutions and sizes

The screen size of a BlackBerry® device depends on the BlackBerry device model number.

- 240 x 160 pixels: BlackBerry 7200 and 7500 Series of wireless handhelds
- 240 x 240 pixels: BlackBerry 7700 Series of wireless handhelds
- 240 x 260 pixels: BlackBerry 7100 Series and the BlackBerry Pearl™ 8100 smartphone
- 320 x 240 pixels: BlackBerry 8700 Series of wireless handhelds

For most BlackBerry Java® Applications, you do not need to customize your application for each screen size. If you use the standard UI fields to build your screens, then you only have to build one version of your application, and the BlackBerry device automatically renders content to fit the screen. However, if you develop games or other graphical applications, and you use the low-level graphics classes to paint directly to the screen, you might need to provide your application with the necessary logic to determine the screen size at runtime and optimize your UI methods accordingly. Either way, test your application on different BlackBerry device models and screen sizes using the BlackBerry Device Simulators.

Best practices for writing BlackBerry device applications

Guidelines for writing efficient code
 Reducing the size of compiled code
 Minimizing memory use
 Best practices

Guidelines for writing efficient code

| Guideline | Description |
|---|---|
| Use local variables. | Use local variables whenever possible. Access to local variables is more efficient than access to class members. |
| Use shorthand for evaluating Boolean conditions. | <p>Instead of evaluating a Boolean condition unnecessarily, use shorthand. The resulting compiled code is shorter.</p> <pre>// Avoid this if(boolean_expression == true) { return true; } else { return false; } // Do this return(boolean_expression);</pre> |
| Make classes final. | <p>When you create code libraries, mark classes as <code>final</code> if you know that developers will never extend them. The presence of the <code>final</code> keyword lets the compiler generate more efficient code.</p> <p>By default, the BlackBerry® Java® Development Environment compiler marks any classes that you do not extend in an application <code>.cod</code> file as <code>final</code>.</p> |
| Use <code>int</code> instead of <code>long</code> . | In Java®, a <code>long</code> is a 64-bit integer. Because BlackBerry® devices use a 32-bit processor, operations run two to four times faster if you use an <code>int</code> instead of a <code>long</code> . |
| Avoid garbage collection. | Avoid calling <code>System.gc()</code> to perform a garbage collection operation because it might take too much time on BlackBerry® devices with limited available memory. Let the BlackBerry JVM collect garbage. |

| Guideline | Description |
|--|--|
| Use static variables for Strings. | <p>When you define static fields (also called <code>class</code> fields) of type <code>String</code>, you can increase application speed by using static variables (not <code>final</code>) instead of constants (<code>final</code>). The opposite is true for primitive data types, such as <code>int</code>.</p> <p>For example, you might create a <code>String</code> object as follows:</p> <pre>private static final String x = "example";</pre> <p>For this static constant (denoted by the <code>final</code> keyword), each time that you use the constant, a temporary <code>String</code> instance is created. The compiler eliminates "x" and replaces it with the string "example" in the bytecode, so that the BlackBerry® JVM performs a hash table lookup each that time you reference "x".</p> <p>In contrast, for a static variable (no <code>final</code> keyword), the string is created once. The BlackBerry JVM performs the hash table lookup only when it initializes "x", so access is faster.</p> <p>Note: You can use public constants (that is, final fields), but you must mark variables as private.</p> |
| Avoid the <code>String(String)</code> constructor. | <p>Avoid using the <code>java.lang.String(String)</code> constructor, because it creates an unnecessary <code>String</code> object that is a copy of the string that you provided as a parameter. Because <code>String</code> objects cannot be modified after you create them, copies are not typically necessary.</p> <pre>String str = new String("abc"); // Avoid. String str = new String("found " + n + " items"); // Avoid.</pre> <p>In BlackBerry® Java® Applications, each quoted string is an instance of the <code>java.lang.String</code> class. Create a <code>String</code> without using the <code>java.lang.String(String)</code> constructor. For example,</p> <pre>String str = "abc"; // Prefer. String str = "found " + n + " items"; // Prefer.</pre> |
| Write efficient loops. | <p>Factor loop-invariant code out of a loop.</p> <pre>// Avoid. for(int i = 0; i < vector.size(); i++) { ... }</pre> <p>This setup invokes <code>vector.size()</code>, which is inefficient. If your container is likely to contain more than one element, assign the size to a local variable. The following example removes loop-invariant code:</p> <pre>// Prefer. int size = vector.size(); for(int i = 0; i < size; ++i) { ... }</pre> <p>Alternatively, if the order in which you iterate through items is not important, you can iterate backward to avoid the extra local on the stack and to make the comparison faster.</p> <pre>for(int i = vector.size() - 1; i >= 0; --i) { ... }</pre> |
| Optimize subexpressions. | <p>If you use the same expression twice, do not rely on the compiler to optimize the expression for you. Use a local variable like in the following example:</p> <pre>one(i+1); two(i+1); // Avoid. int tmp = i+1; one(tmp); two(tmp); // Prefer.</pre> |

| Guideline | Description |
|-------------------------------|--|
| Optimize division operations. | <p>Division operations can be slow on BlackBerry® devices because the processor does not have a hardware divide instruction.</p> <p>When your code divides a positive number by two, use shift right by one (>>1) instead. Use the shift right (>>) only when you know that you are working with a positive value.</p> <pre>midpoint = width / 2; // Avoid. int = width >> 1; // Prefer.</pre> |
| Avoid java.util.Enumeration. | <p>Avoid using java.util.Enumeration objects unless you want to hide data (in other words, to return an enumeration of data instead of the data itself).</p> <pre>// Avoid. for (Enumeration e = v.elements(); e.hasMoreElements();) { o = e.nextElement(); ... }</pre> <p>Asking a vector or hash table for an Enumeration object is slow and creates unnecessary garbage. Instead, iterate the elements yourself like in the following example:</p> <pre>// Prefer. for(int i = v.size() - 1; i >=0; --i) { o = v.elementAt(i); ... }</pre> <p>If another thread might modify the vector, synchronize the iteration like in the following example:</p> <pre>synchronized(v) { for(int i = v.size() - 1; i >=0; --i) { o = v.elementAt(i); ... } }</pre> <p>Note: The Java® Platform, Standard Edition uses an Iterator object for similar operations, but iterators are not available in the Java Platform, MicroEdition.</p> |

| Guideline | Description |
|---|---|
| Perform casts using <code>instanceof</code> . | <p>Use <code>instanceof</code> to evaluate whether a cast succeeds instead of catching a <code>ClassCastException</code>.</p> <pre data-bbox="485 286 856 633"> // Avoid. try { (String)x.whatever(); } catch(ClassCastException e) { ... } // Prefer. if(x instanceof String) { (String)x.whatever(); } else { ... } </pre> <p>Using <code>instanceof</code> is faster than using a <code>try/catch</code> block. Use the <code>try/catch</code> block only when a cast failure is an exceptional circumstance.</p> <p>The BlackBerry® JDE compiler and BlackBerry JVM are optimized to perform only one class check in the first block of code following a branch determined by an <code>instanceof</code> check. To use this implementation, perform the cast immediately following the branch that is determined by an <code>instanceof</code> check.</p> <p>For example, the compiler can optimize the first example but not the second.</p> <pre data-bbox="485 850 785 1111"> // Prefer. if (a instanceof type) { type instance = (type)a; x.method(instance); instance.method(x, y, z); } // Avoid. if(a instanceof type) { x.method((type)a); } </pre> |
| Evaluate conditions using <code>instanceof</code> . | <p>To produce smaller and faster code, if you evaluate a condition using <code>instanceof</code>, do not evaluate explicitly whether the variable is null. The expression <code>e instanceof type</code> evaluates to false if "e" is null.</p> <pre data-bbox="485 1215 1120 1362"> // Avoid. if(e != null && e instanceof ExampleClass) { ... } if(e == null ! (e instanceof ExampleClass)) { ... } // Prefer. if(e instanceof ExampleClass) { ... } if(! (e instanceof ExampleClass)) { ... } </pre> |

| Guideline | Description |
|--|--|
| Avoid using <code>StringBuffer.append(StringBuffer)</code> . | <p>CLDC does not include a <code>StringBuffer.append(StringBuffer)</code> method. Appending a string buffer to another in this way creates an intermediate <code>String</code> object. Instead, BlackBerry® Java® Applications should use <code>net.rim.device.api.util.StringUtilities.append(StringBuffer dst, StringBuffer src[, int offset, int length])</code>.</p> <pre> // Avoid. public synchronized StringBuffer append(Object obj) { return append(String.valueOf(obj)); } // Prefer. 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)); } </pre> |

Reducing the size of compiled code

| Guideline | Description |
|----------------------------|--|
| Set appropriate access. | <p>When you create code libraries, significantly reduce the size of your compiled code by using the appropriate access modifiers for fields and methods.</p> <ul style="list-style-type: none"> • Declare fields as <code>private</code> whenever possible. In addition to being good coding practice, this lets the compiler optimize the <code>.cod</code> file. • When possible, use the default (package) access instead of <code>public</code> access (that is, omit the <code>public</code> and <code>protected</code> keywords). |
| Avoid creating interfaces. | <p>When you create API libraries, avoid creating interfaces unless you foresee multiple implementations of the API. Interfaces produce larger, slower code.</p> |
| Use static inner classes. | <p>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 <code>static</code>. This action prevents the creation of a reference to the outer class.</p> <p>For example, the following code requires a reference to the outer class object.</p> <pre>// Avoid. class outer { int i; class inner { inner() {} int example() { return i; } } }</pre> <p>In contrast, the following code only defines the scope of the inner class name.</p> <pre>// Prefer. class outer { static class inner { ... } }</pre> <p>Only use a non-static inner class when you need access to data in the outer class from within methods of the inner class. If you use an inner class for name scoping, make it <code>static</code>.</p> |

| Guideline | Description |
|---|---|
| Avoid unnecessary field initialization. | <p>Avoid unnecessarily initializing fields in classes, where fields have default values. If you do not initialize a field in a class, the field initializes automatically using the following default values:</p> <ul style="list-style-type: none"> • object references are initialized to <code>null</code> • <code>int</code>, <code>byte</code>, or <code>long</code> is initialized to <code>0</code> • <code>Boolean</code> is initialized to <code>false</code> <p>For example, the following code fragment contain no differences:</p> <pre>// Avoid. class BadExample { private int fieldsCount = 0; // Avoid. private Field _fieldWithFocus = null; // Avoid. private boolean _validLayout = false; // Avoid. } // Prefer. class BetterExample { private int fieldsCount; // Prefer. private Field _fieldWithFocus; // Prefer. private boolean _validLayout; // Prefer. }</pre> <p>Note: You must explicitly initialize local variables in a method.</p> |
| Import individual classes. | <p>BlackBerry® Java® Applications that use only a small number of classes from a package should import the individual classes rather than the entire library.</p> <pre>// Avoid. import net.rim.blackberry.api.browser.*; // Prefer. import net.rim.blackberry.api.browser.Browser;</pre> |

Minimizing memory use

Key resources to reserve

- **Flash memory:** The raw persistent storage space that is available on each 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 in the system are consumed only by persistent objects. The amount of flash memory determines the fixed number of persistent object handles in the system.

- Object handles: Each object and array of primitives in the system has an object handle associated with it. The amount of flash memory determines the fixed number of object handles in the system.



Note: A persistent object consumes a persistent object handle and an object handle. A transient object only consumes an object handle.

Conserving resources

| Guideline | Description |
|--------------------------|--|
| Data structure selection | <p>Data structure selection defines how many object handles and how much flash memory the system consumes. Improper data structure selection can consume several different key resources without improving application functionality or BlackBerry® device user experience.</p> <ul style="list-style-type: none"> • The data structure should consist of the minimum possible number of objects, especially when using high level objects like <code>Vector</code> or <code>Hashtable</code>. These classes provide significant functionality but are not efficient storage mechanisms and you should avoid using them in the persistent store if possible. • Use primitives instead of objects, when possible, because primitives reduce the number of object handles that are consumed on the BlackBerry device. Note that an array of primitives is an object and consumes an object handle. • String objects are no longer less efficient than byte arrays. A <code>String</code> object only consumes one object handle and is equivalent if your application stores all of the characters as a byte or, 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 <code>String</code> because it is equivalent to storing a <code>char</code> array. |
| Object groups. | <p>One of the most common errors 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 in the system. Depending on the data structure selection, stored records can quickly exhaust the number of persistent object handles.</p> <p>For example, a record that contains 10 <code>String</code> fields, which represent items like name, phone number, and address, consumes 11 persistent object handles, one for the record object and one for each string. If an application persists 3000 records, that consumes 33,000 persistent object handles, which exceeds the current number of persistent object handles on a 16 MB BlackBerry device.</p> <p>You can use the <code>net.rim.device.api.system.ObjectGroup</code> class to consolidate the object handles for an object into one group. Using the example in the previous paragraph, if you group the record you will use only one object handle for the record instead of 11. The object handles for the <code>String</code> fields consolidate under the record object handle.</p> <p>However, when you group an object, it is read-only. You must ungroup the object before you modify it. After you complete the changes, regroup the object using the normal group method. If you attempt to modify a grouped object without first ungrouping it, an <code>ObjectGroupReadOnlyException</code> is thrown.</p> <p>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.</p> |

Using objects judiciously

The Objects window provides information on the status of objects in the system. When you review the information in the Objects window, ask the following questions:

- Given the size of an application's the objects, 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 thrown away? In other words, how many scope-specific Objects does your application create?

Conserving storage space

Storage space on BlackBerry® devices is limited. Design your application to minimize the amount of flash memory that your application requires to store persistent data.

A typical BlackBerry device shares any storage space that is not required for standard BlackBerry applications between all applications to store user data, including calendar appointments, contact list items, and messages list items.

If the BlackBerry device operates with low memory, it might perform the following actions to free flash memory space:

- remove old messages from the BlackBerry device
- remove calendar appointments that are more than one week old from the BlackBerry device (if wireless calendar synchronization is turned on)

If the BlackBerry device removes messages or calendar appointments because of low flash memory, the data is still kept in the computer messaging application. See “Memory management and persistent objects” on page 35 for more information on memory management.

Best practices

| Guideline | Description |
|----------------------|--|
| Use 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. |
| Minimize memory use. | To minimize runtime memory, use the following guidelines: <ul style="list-style-type: none"> • 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, filter or sort data before sending it to the BlackBerry® device. |

| Guideline | Description |
|---|--|
| Avoid returning null. | <p>If you write a public method that returns an object, the method should return <code>null</code> only under the following conditions:</p> <ul style="list-style-type: none"> Your application expects a <code>null</code> value to occur during normal application operation. The Javadoc® <code>@return</code> parameter for the method states that <code>null</code> is a possible return value. <p>If your application does not expect a <code>null</code> 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 <code>null</code> return value unless the caller of the method throws a <code>null</code> exception.</p> |
| Avoid passing null into methods. | <p>Do not pass null parameters into an API method unless the <i>API Reference</i> states explicitly that the method supports them.</p> |
| Use caution when passing null into a constructor. | <p>To avoid ambiguity when passing null into a constructor, cast null to the appropriate object.</p> <pre>new someObject ((someObject)null);</pre> <p>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. Not all supported constructors appear in the <i>API Reference</i>, because some constructors are intended for internal use only.</p> <p>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.</p> |
| Use longs for unique identifiers. | <p>Use a <code>long</code> identifier instead of a <code>String</code> identifier for unique constants, such as GUIDs, hash table keys, and state or context identifiers.</p> <p>For identifiers to remain unique across third-party applications, use keys that an application generates based on a hash of a <code>String</code>. In the input string, include enough information to make the identifier unique. For example, use a fully qualified package name such as <code>com.rim.samples.docs.helloworld</code>.</p> <p>See the <i>IDE Online Help</i> for more information.</p> |
| Exit applications correctly. | <p>Before you invoke <code>System.exit(int status)</code>, perform any necessary cleanup, such as removing objects from the runtime store that applications no longer require.</p> |
| Print the stack trace. | <p>The BlackBerry® JVM is optimized to eliminate the stack trace if it locates code that catches the exception using <code>catch (Exception e)</code>. It does not eliminate the stack trace if your application catches the <code>Throwable</code> exception.</p> <p>For example, the following code does not print a stack trace:</p> <pre>catch (IOException e) { e.printStackTrace() }</pre> <p>To print a stack trace, write code like in the following example:</p> <pre>catch (Throwable t) { t.printStackTrace(); }</pre> <p>When you debug your application, to view the stack trace, catch a <code>Throwable</code> instance.</p> |

Memory and data storage

BlackBerry device memory
Garbage collection on BlackBerry devices
Low memory manager
Persistent storage
Storage on removable media
Accessing data on the microSD media card
Backing up data
Synchronizing data

BlackBerry device memory

BlackBerry® devices include the following types of on-board memory:

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

Four levels of memory capabilities exist on current BlackBerry devices.

- 2 MB SRAM and 16 MB flash (for example, BlackBerry 7230 Wireless Handheld™)
- 4 MB SRAM and 32 MB flash (for example, BlackBerry 7290 Wireless Handheld™)
- 16 MB SRAM and 64 MB flash (for example, BlackBerry 8700 Series of wireless handhelds™)
- 16MB SRAM and 64MB flash + microSD expandible memory (for example, the BlackBerry Pearl™ 8100 smartphone)

How the BlackBerry device manages memory

The BlackBerry® JVM manages memory usage on the BlackBerry device. The BlackBerry JVM allocates memory, performs garbage collection, and automatically swaps data between SRAM and flash. It must also share available memory between the BlackBerry device applications and the third-party applications. The memory capabilities represent the total amount of available memory, which will be much larger than the available working memory once all of the applications and associated application data exist on the BlackBerry device.

Garbage collection on BlackBerry devices

RAM garbage collection

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

Full garbage collection

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

- The BlackBerry® JVM cannot allocate an object because of a lack of 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.

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 steps:

1. It performs a RAM garbage collection operation.
2. It marks unreferenced and unpersisted objects that currently exist in flash memory.
3. It releases any non-persistent object handles in both RAM and flash memory.

Idle garbage collection

The system attempts to perform the following garbage collection operations when the BlackBerry® device idles. This lets the system improve its performance without impacting the BlackBerry device user experience.

- A Full garbage collection operation occurs when the BlackBerry device idles for a relatively small amount of time, if required based on heuristics.
- A thorough garbage collection operation occurs when the BlackBerry device idles for a significant period of time and a garbage collection operation is required based on heuristics.

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.

Low memory manager

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 existing memory to provide more available memory on the BlackBerry device. All applications, including third-party applications, should work with the low memory manager to free as much memory as possible when the BlackBerry device is low on memory resources.

Low memory conditions

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

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

Memory management and persistent objects

To free persistent objects, applications invoke `LowMemoryManager.markAsRecoverable()` inside the implementations of `freeStaleObject()`.

Before invoking this method, applications should remove all references to the object and remove the object from its data structures. The application then calls `LowMemoryManager.markAsRecoverable()`, passing in the object to free. This command indicates to the underlying BlackBerry® JVM that the BlackBerry JVM can remove the object as part of the low memory management operation.

You must invoke `markAsRecoverable()` because the low memory manager tries to recover a specific amount of memory. The low memory manager distributes this amount across the registered applications. If your application frees memory resources, invoking `markAsRecoverable()` notifies the low memory manager so that it can count the freed memory against its target amount.

Implementations of `freeStaleObject()` should perform `commit()` operations when they remove data from their collections. Do not skip this step for efficiency, because the low memory manager automatically uses a transactional commit operation so that all commits are deferred until the low memory manager operation completes. This lets you use the same commit logic for low memory manager handlers and normal code.

Persistent storage

The BlackBerry® device provides three sets of APIs for storing data to persistent memory on the BlackBerry device. BlackBerry persistent store and 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 more traditional file system and support for saving content directly to the file system via JSR 75 APIs. With either option, you can store data persistently to flash memory. That data persists even if you remove the battery from the BlackBerry device.

| API | Description |
|-----------------------------|---|
| BlackBerry persistent store | The BlackBerry Persistent Store APIs 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. BlackBerry APIs do not provide a relational database model. You must create an effective object model and manage relationships between objects, as necessary, using indices and hash tables. |
| MIDP RMS | The MIDP 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. With RMS, each record is a byte array, so you must first serialize your data into this format before storing it locally. RMS does not provide any inherent indexing or relationships between records. The single RMS data store limit is a maximum of 64 KB; however an application can create multiple RMS stores to persist larger amounts of data. The RMS APIs are part of the standard MIDP specification, so all devices that support MIDP support the RMS APIs. |
| JSR 75 File Connection APIs | The JSR 75 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 Microsoft® Windows®. |

Storage on removable media

The microSD card is a small removable flash memory card. When the microSD memory card connects to a BlackBerry device, the BlackBerry device automatically creates a file system on the microSD memory card. If the microSD memory card is not formatted, you must use the BlackBerry device Options application to format the card.

File encryption

IT policies and the microSD memory card

The IT policy Encrypt data written to the microSD media card applies to any new or modified files you store on the microSD media card. Only 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.

File encryption on the microSD memory card

When a BlackBerry® device application accesses a file on the microSD memory card, the file is automatically decrypted and moved to main memory for the application to read.

For an application to access a password protected file, 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. This 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 you move the microSD media card to a BlackBerry® device with no device password set or 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 set the microSD media card password to the BlackBerry device password.

Implementing code security and file data encryption

Use the `net.rim.device.api.io.file.ExtendedFileConnection` interface, part of the JSR 75 standard, to access the file system that the BlackBerry® device creates on the microSD media card. Use the `net.rim.device.api.io.file.ExtendedFileConnection` interface to implement code signing protection and file encryption for data in the microSD media card file system.

Set automatic encryption resolve mode

Implement the `setAutoEncryptionResolveMode(boolean mode)` method.

Set a code-signing key on an instance of a file connection

Implement `setControlledAccess(CodeSigningKey csk)`.

See the *API Reference* for more information on the `ExtendedFileConnection` interface.

Accessing data on the microSD media card

The `javax.microedition.io.file` package supports the JSR 75 `FileConnection` APIs and lets applications access the microSD media card file system.

| FileConnection class or interface | Description |
|--|---|
| <code>ConnectionClosedException</code> | thrown when an application invokes a method on a closed file connection |
| <code>FileConnection</code> | an application uses this to access files or directories located on the microSD media card or file systems on a BlackBerry® device |
| <code>FileSystemListener</code> | an application uses this to receive status notifications when adding or removing a file system root |
| <code>FileSystemRegistry</code> | a central registry for file system listeners that listen for the addition or removal of file systems on a BlackBerry device |

| FileConnection class or interface | Description |
|-----------------------------------|--|
| IllegalModeException | thrown when a method requires a specific security mode (for example READ or WRITE) and the open connection is not in that mode |

You may also implement the `FileConnection` interface to access BlackBerry device ring tones and camera images.

See <http://developers.sun.com/> or the *API Reference* for more information about the `javax.microedition.io.file` package.

Backing up data

The BlackBerry® Desktop Software provides a Backup and Restore tool that lets BlackBerry device users save BlackBerry device data to a file on their computer and use this file to restore data to the BlackBerry device.

When an application uses the synchronization API, the BlackBerry Desktop Software backs up and restores the application database with the other BlackBerry device databases. You can also use the synchronization API to create data archives or to populate application databases when the BlackBerry device first connects to the BlackBerry device user's computer.

Synchronizing data

The BlackBerry® Java® Development Environment does not provide any tools or applications for synchronizing data to remote data sources, so you must build the synchronization logic into your 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. XML APIs let you generate and parse XML-formatted data to send and receive over the wireless network. However, your client - and server-side applications must read and write the data properly and acknowledge the successful transmission.

Some applications might connect to a computer-based application and send the data over the USB connection using the BlackBerry Desktop Synchronization APIs and the BlackBerry Desktop Manager. In this case, you must build a computer application for Microsoft® Windows® that can read the data from the client through the BlackBerry Desktop Manager Plug-Ins adapter. The BlackBerry device user must manually execute the synchronization by running the BlackBerry Desktop Manager Plug-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 using the native USB protocols if you prefer.

Wireless data transport

Wireless gateways
Transport options
Alternate transport methods

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 intranet gateway

Enterprise customers host the BlackBerry® Enterprise Server behind their corporate firewall to enable access from BlackBerry devices to the corporate 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 secure gateway for managing HTTP and TCP/IP connections to the intranet. When you use the BlackBerry Enterprise Server as an intranet gateway, all traffic between your application and the BlackBerry Enterprise Server is automatically encrypted using AES or triple DES encryption. Because the BlackBerry Enterprise Server resides behind the corporate firewall and provides inherent data encryption, applications can communicate with application servers and web servers that reside on the corporate intranet.

If your application connects to the Internet rather than to the corporate intranet, you might be able to use the BlackBerry Enterprise Server that belongs to the customer as a gateway as well. In this case, network requests travel behind the corporate firewall to the BlackBerry Enterprise Server, which makes the network request to the Internet through the corporate firewall. However, enterprise customers can set an IT policy to enforce 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, and you are targeting non-enterprise customers, 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

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.

See the *BlackBerry Java Development Environment Development Guide* for more information on establishing HTTP and TCP/IP connections from a Java application for BlackBerry devices. In the Technical Knowledge Center on the BlackBerry Developer Zone, see the whitepaper *Managing Wireless Data Transport in the BlackBerry Solution v4.0 Part 1: Understanding TCP and HTTP transport options for Java applications for BlackBerry* for more information on managing wireless connectivity and how to effectively use each of the gateways.

Transport options

While most applications for BlackBerry® devices use HTTP or TCP/IP protocols, you can also use other transport protocols to send and receive data. See the *BlackBerry Java Development Environment Development Guide* for more information on network connections.

Alternate transport methods

| Transport Method | Description |
|----------------------------|---|
| Using email | The BlackBerry® APIs let an application send email messages and listen for inbound email messages. An 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. You can use these APIs to let your application use 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. |
| Using SMS | The BlackBerry APIs let an application send SMS messages and listen for inbound SMS messages. You can use these APIs to use SMS as a transport mechanism for sending and receiving data. SMS can be an effective way to implement peer-to-peer application for a variety of mobile phones. |
| Using PIN-to-PIN messaging | The BlackBerry APIs also let an application programmatically send and receive BlackBerry PIN messages. BlackBerry PIN messages are similar to SMS messages because they can be sent from one mobile device directly to another. However, PIN messaging uses the data channel rather than the voice channel and lets you 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. PIN-to-PIN messaging can be an effective way to implement peer-to-peer applications targeting BlackBerry users only. |

See the *BlackBerry Java Development Environment Development Guide* for more information on using these alternate transport modes.

Integrating multimedia features

Audio support
Imaging support
Video support

Audio support

The type of audio format that a BlackBerry® device supports depends on the BlackBerry device model number.

| BlackBerry device | AMR | Beep | MIDI | MP3 | ADPCM2 | WAV |
|--|-----|------|---------|-----|--------|-----|
| BlackBerry 5800 Series, BlackBerry 6200 Series, BlackBerry 6700 Series, BlackBerry 7200 Series, BlackBerry 7500 Series, BlackBerry 7700 Series | – | Yes | Partial | – | – | – |
| BlackBerry 7100t™, BlackBerry 7100g™, BlackBerry 7100r™ | – | Yes | Yes | – | Yes | – |
| BlackBerry 7130e™ | – | Yes | Yes | Yes | – | – |
| BlackBerry 7100i™ | – | – | Yes | – | – | – |
| BlackBerry 7130c™, BlackBerry 7130g™, BlackBerry 8100 Series, BlackBerry 8700 Series | Yes | Yes | Yes | Yes | – | Yes |

You can create BlackBerry device applications that work with the audio formats that a BlackBerry device supports by using the `Multi-Media` API. See the BlackBerry Developer Zone at <http://www.blackberry.com/developers> for more information about knowledge base articles that outline how to play audio on a BlackBerry device.

Audio Recording and Voice Playback

Only the BlackBerry 7520™ and the BlackBerry 7100i™ BlackBerry devices operating on the iDEN™ wireless networks in North America, and the BlackBerry Pearl™ 8100 smartphone, support audio recording.

- The BlackBerry 7520 and BlackBerry 7100i devices support the Motorola VoiceNotes™ capability for audio.
- The BlackBerry 8100 supports audio recording through the Java® multimedia API.

Imaging support

The BlackBerry® Pearl™ 8100 smartphone includes an integrated 1.3 megapixel camera. When a BlackBerry device user takes pictures with the camera, the BlackBerry device stores the pictures in the file system on the BlackBerry device. A BlackBerry device application can access the pictures by using the JSR 75 file connection API that is available with BlackBerry JDE Version 4.2 or later. The BlackBerry device application can invoke the camera application and listen for events when images are added to the file system.

Video support

Most BlackBerry® devices do not support the playback of video content by BlackBerry device applications. You can create BlackBerry device applications that display images and use the graphics API classes to work with rich-media content.

The BlackBerry Pearl™ 8100 smartphone includes an integrated media player that BlackBerry device applications can invoke to play a local video file.

Integrating with BlackBerry applications

Adding custom menu items
Invoking native applications and phone calls
Accessing email and PIM data
Using BlackBerry Messenger with a BlackBerry application
Using listeners to respond to application changes

Adding custom menu items

Third-party applications can register custom menu items to display in the existing BlackBerry® menus for the email, PIM, and phone applications. When a BlackBerry device user selects the custom menu item, the third-party application starts with a reference to the object that the BlackBerry device user selects. For example, an application could add a menu item called "Show Location of Sender" to the existing email application. When BlackBerry device users select the menu item, the application starts with a reference to the email object that is currently highlighted or that the user opens. The application could then use 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 come to the foreground and display a map.

Invoking native applications and phone calls

Third-party applications can invoke BlackBerry® device applications such as the email, PIM, phone, browser, and camera applications. When the third-party application invokes the BlackBerry device application, the third-party application can make the BlackBerry device application perform an action or display information. The following examples demonstrate how third-party applications can invoke BlackBerry device applications:

- Third-party applications invoke the calendar to display a specific date or calendar entry.
- Third-party applications invoke the address book to display a specific contact.
- Third-party applications invoke the browser to fetch a specific URL.
- Third-party applications invoke the phone to dial a specific number.

To start BlackBerry device applications, from a third-party application, invoke `Invoke.invokeApplication(int appType, ApplicationArguments args)` using the parameters that correspond to the intended BlackBerry device application.

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

Accessing email and PIM data

Third-party applications can also use the Java™ APIs to access the details of email messages, contacts, calendar events, tasks, and phone logs that the BlackBerry® device stores. Third-party applications can read the information, update the information, and create new entries.

Using BlackBerry Messenger with a BlackBerry application

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

To create an application that integrates with the BlackBerry Messenger application, use the classes in the `net.rim.blackberry.api.blackberrymessenger` package. See the *API Reference* for more information about using the `BlackBerryMessenger` class.

Using listeners to respond to application changes

Third-party applications can register change listeners on the email and PIM data stores as well as in the phone application. You can use the email and PIM listeners to notify an application when new entries arrive or when the BlackBerry® device user makes changes such as additions, removals, or updates, to the existing data. Use phone listeners to listen for phone call actions, such as the initiation of new calls or calls ending. These APIs let applications take immediate action when the BlackBerry device user performs a local event.

Security

- Data encryption
- Authentication
- Controlled APIs and code signing
- Application control
- Access to memory
- IT policies

Data encryption

Data encryption in transport

Enterprise applications often require that all of the data that the application sends or receives is encrypted during transport at all points outside of the corporate firewall. With BlackBerry®, this can be seamless to implement. If you use the BlackBerry Enterprise Server as the wireless gateway for your application, the BlackBerry Enterprise Server automatically encrypts data using AES or TripleDES encryption at all points in the connection between the BlackBerry device and the BlackBerry Enterprise Server behind the corporate 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

In an enterprise environment, system administrators can also set an IT policy to dictate that all BlackBerry® device user data stored in BlackBerry core applications is encrypted locally in flash memory. Third-party applications can use additional APIs to register their data with this encryption service so that the encryption service encrypts the data with the same security key before committing it to flash memory.

Authentication

BlackBerry device authentication and IT policy

BlackBerry® users can setup their BlackBerry devices with passwords for protection. When the password is active, BlackBerry device users must provide the device password periodically to access the data and applications. Using device passwords is a good first step to limiting access to your custom applications on the BlackBerry device.

In an enterprise environment, the BlackBerry Enterprise Server provides IT policies that let the system administrator enforce that BlackBerry devices in the organization are password-protected. This IT policy helps keep confidential corporate information private. A system administrator can also issue Wireless IT Commands 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. Because login screens can negatively impact the BlackBerry device user experience, if the BlackBerry device users set passwords to protect their BlackBerry devices, your application might not require a login screen.

Back-end authentication

If your application connects to an application on a server or to the Internet or intranet, you might want to enforce additional authentication features when BlackBerry® device users log into the server. Most applications that require user authentication rely on HTTP Basic authentication, which uses a simple username and password combination. You can use HTTP Basic 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 in the BlackBerry® Java® Development Environment for security and export control reasons. In the 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 load the application .cod files onto the BlackBerry device, you must sign your application using a key, or signature, from RIM.



Note: Other functionality, such as the ability to execute on startup, might require that you sign your applications.

While most controlled APIs are covered by the RIM API signature, certain 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 cover use of Certicom classes. For more information on registering and using these classes, please refer to the BlackBerry Developer Zone at <http://www.blackberry.com/developers/index.shtml>.

To test and debug your code before you receive the code signatures, use the BlackBerry Device Simulator. You must sign application before loading it onto BlackBerry devices. You never send your actual code to RIM. The BlackBerry Signature Tool sends a SHA1 hash of your code file so that the signing authority system can generate the necessary signature. See the *BlackBerry Signing Authority Tool Version 1.0 - Password Based Administrator Guide* for more information about registering and obtaining code signatures.

Controlled APIs

The following three categories of RIM controlled APIs exist: Runtime APIs, BlackBerry Application APIs, and BlackBerry Cryptography APIs. See the *API Reference* for more information on all RIM controlled APIs.

You can run applications that use controlled APIs in the BlackBerry Device Simulator without code signatures; however, you must obtain code signatures from RIM before you can load these applications onto BlackBerry devices.

If you use any of the following BlackBerry API packages, your 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`

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, system administrators can use the BlackBerry Application Control IT policy to make sure that a game that exists on the BlackBerry device cannot access the phone API.



Note: The BlackBerry Application Control IT policy works only when the BlackBerry device and a BlackBerry Enterprise Server are connected. 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.

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. Java applications can write only to BlackBerry device memory that the BlackBerry JVM uses; they cannot access the virtual memory or the persistent storage of other applications. 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.

IT policies

The BlackBerry® IT policy API (`net.rim.device.api.itpolicy`) lets applications access the IT policy information on BlackBerry devices. Applications can retrieve custom IT policy settings to change the application's behavior or functionality.



Note: System administrators use application control to limit the presence and functioning of third-party applications on BlackBerry devices. See the *BlackBerry Enterprise Server Administration Guide* for more information on application control.

Each IT policy item consists of a descriptive key and a value. The value can be a string, integer, or Boolean value. For example, the `AllowPhone` policy can have a value of `true` or `false`.

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

See the *BlackBerry Enterprise Server Policy Reference Guide* for more information on IT policies.

Localizing applications

Storing text strings in resource files
Storing resources for a locale

Storing text strings in resource files

Design applications so that they are easy to localize (adapt to specific languages and regions) without coding changes. Instead of including textual elements in your source code, store text strings in separate resource files. In your source code, use unique identifiers to make use of the appropriate resource.

Storing text strings in separate resource files has the following two benefits:

- Text translation is more efficient because you can store the text strings for a given locale in a single file that exists outside your source code.
- Applications can dynamically retrieve the appropriate text to display to the BlackBerry® device user based on the BlackBerry device user locale.

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.

Storing resources for a locale

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.

Testing and setting up applications

Why it is not necessary to obfuscate applications
Preverify applications
Testing applications using the BlackBerry JDE
Testing applications on BlackBerry devices
Setting up applications through the computer application
Setting up applications over the wireless network

Why it is not necessary to obfuscate applications

The compiler for the BlackBerry® Java® Development Environment is set to minimize the size of the application. The .cod file that results 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

As such, RIM does not believe it is necessary for you to provide obfuscation for your applications in addition to the existing obfuscation that, by default, the BlackBerry JDE provides for all applications. 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. As such, you must include a command-line procedure to obfuscate .cod files for use on BlackBerry devices.

Preverify applications

To partially verify your classes before you load your application on a BlackBerry® device, use the Preverify tool. When you preverify your classes, you reduce the amount of processing that the BlackBerry device must perform when you load your application.

See the *BlackBerry Java Development Environment Development Guide* for more information on using the Preverify tool.

You can also use the BlackBerry Device Simulator to preverify .cod files. See the *BlackBerry Device Simulator User Guide* for more information about using the BlackBerry Device Simulator.

Testing applications using the BlackBerry JDE

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 Device Simulator for testing. The BlackBerry Device Simulators run the same Java code as the live BlackBerry devices, so the BlackBerry Device Simulators provide an accurate environment for testing how applications will function on a live BlackBerry device. Each version of the BlackBerry JDE comes with the BlackBerry Device Simulators that were available when that version of the BlackBerry JDE was made public. You can download additional BlackBerry Device Simulators from the BlackBerry Developer Zone at <http://www.blackberry.com/developers/index.shtml>.

Testing applications on BlackBerry devices

After testing your application on the BlackBerry® Device Simulator, you can load your application on a live BlackBerry device. If your application uses signed APIs, you may need code signing keys for this task. Once you load the application on the BlackBerry device, you can open the application and test its functionality and performance. For debugging purposes, you can also attach your device to the BlackBerry Integrated Development Environment debugger and use the debugger 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 more difficult to simulate.

Setting up applications through the computer application

Loading an application through the computer is often the easiest approach during testing and development, and it might be the preferred approach if your application is larger than several hundred KB. You can setup an application on a computer through the USB connection in the following ways:

| Setup option | Description |
|--|---|
| BlackBerry® Desktop Manager Application Loader | The BlackBerry Desktop Manager includes a component called the Application Loader, which you use to install third-party applications as well as updated system applications for the BlackBerry device. The BlackBerry Desktop Manager Application Loader approach provides a simple way for endusers to download an application from their computers to their BlackBerry devices. |
| BlackBerry Application Web Loader | With the BlackBerry Application Web Loader, you can post your compiled application on a central web site and users can setup the application by using Internet Explorer on their computers to the visit URL. When BlackBerry device users visit the web page, The BlackBerry Application Web Loader asks them to connect their devices to the USB port. They can then install the application using an ActiveX® control. The BlackBerry Application Web Loader provides BlackBerry device users with a simple approach for setting up applications from their computers without running the BlackBerry Desktop Manager. |
| Javaloader Command Line Tool | The BlackBerry Java® Development Environment includes a command line tool called Javaloader.exe. The executable file exists in the BIN directory under the BlackBerry JDE directory. Use the Javaloader file to quickly install and remove compiled application files on the BlackBerry device directly over the USB port and does not require any descriptor files or web pages. Javaloader can be useful when you are installing and removing your application frequently during testing and development; however, the Javaloader tool is not designed for use by BlackBerry device users. |

Setting up applications over the wireless network

Using wireless application provisioning can provide a better experience to BlackBerry® device users and simplify sending the applications to a large group of people since you do not require a computer application. A BlackBerry device user can install an application over the wireless network in the following ways:

| Setup option | Description |
|----------------------------------|---|
| Wireless pull (user-initiated) | You can post a compiled applications on a public or private web site, and BlackBerry® device users can download the applications over the wireless network by using the web browser on their BlackBerry devices to visit the URL. When BlackBerry device users visit the URL, the browser prompts them to install the application. If the BlackBerry device users accept, the application downloads over the wireless connection and installs immediately. |
| Wireless push (server-initiated) | In an enterprise environment, the BlackBerry Enterprise Server administrator can push applications to BlackBerry device users over the wireless network and enforce that the application installs. The administrator creates a new policy and indicates that the BlackBerry device requires the application. Once the policy is setup on the BlackBerry Enterprise Server, the application is pushed to users without any user interaction required. Enterprises might find this approach useful when sending new wireless applications to a large number of BlackBerry device users. |

Acronym list

A

AES

Advanced Encryption Standard

API

application programming interface

C

CLDC

Connected Limited Device Configuration

G

GPS

Global Positioning System

H

HTTP

Hypertext Transfer Protocol

HTTPS

Hypertext Transfer Protocol over Secure Sockets Layer

I

I/O

input/output

IMAP

Internet Message Access Protocol

IPC

interprocess communication

J

JSR

Java Specification Request

JTWI

Java Technology for the Wireless Industry

JVM

Java Virtual Machine

L

LAN

local area network

M

MIDP

Mobile Information Device Profile

MMA

Mobile Media APIs

N

NVRAM

Nonvolatile random access memory

O

OPP

Object Push Profile

OBEX

Object Exchange Profile

P

PDA

personal digital assistant

PDAP

personal digital assistant profile

PIM

personal information management

PIN

personal identification number

POP3

Post Office Protocol 3

R

RAM

random access memory

RMS

root-mean-square

S

SHA1

Secure Hash Algorithm, Version 1

SMS

Short Message Service

SPP

Serial Port Profile

SRAM

static random access memory

SSL

Secure Sockets Layer

T

TCP

Transmission Control Protocol

TCP/IP

Transmission Control Protocol/Internet Protocol

TLS

Transport Layer Security

Triple DES

Triple Data Encryption Standard

U

UI

user interface

USB

Universal Serial Bus

V

VPN

virtual private network

W

WAP

Wireless Application Protocol

WMA

Wireless Messaging API

X

XML

Extensible Markup Language

