



Memory Best Practices for the BlackBerry Java Development Environment

Contents

Introduction.....	1
Key resources to conserve	1
BlackBerry handheld resources	1
Conserving resources	1
Reduce the number of objects.....	1
Data structure selection	2
Object groups.....	2
Common exceptions and errors.....	2
JVM 531.....	3
OutOfMemoryError	3
Troubleshooting tips.....	3
Live Handheld	3
Simulator.....	3
Frequently Asked Questions.....	4
Java Low Memory Manager.....	4
Garbage Collection.....	4

Introduction

This document provides a set of best practices for memory management in applications for the BlackBerry® Wireless Handheld. These best practices include tools, tips and other ideas for managing critical handheld resources and minimizing memory usage.

Key resources to conserve

The following resources must be managed by applications from a memory perspective:

- **Flash memory:** This term refers to the raw persistent storage space that is available on each handheld. Each handheld has a fixed amount of flash memory, typically in the range of 8 MB to 32 MB.
- **Persistent object handles:** This term refers to the handle that is assigned to each persistent object in the system. These handles are only consumed by persistent objects. The fixed number of persistent object handles in the system is determined by the amount of flash memory. See BlackBerry handheld resources on page 1 for more information.
- **Object handles:** This term refers to the handle that is assigned to each object and array of primitives in the system. By default, each instance of a `java.lang.Object` class has an object handle associated with it. The fixed number of object handles in the system is determined by the amount of flash memory. See BlackBerry handheld resources on page 1 for more information.

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

BlackBerry handheld resources

Flash memory	Persistent object handles	Object handles
8 MB	12,000	24,000
16 MB	27,000	56,000
32 MB	65,000	132,000

By effectively managing these resources, applications can provide a full set of features without adversely affecting the user experience.

Conserving resources

Reduce the number of objects

By reducing the number of objects that are used in an application, developers can directly affect each of the three key resources.

The BlackBerry Java Development Environment (JDE) provides the Objects Window tool to assist developers with the task of reducing the number of objects.

The Objects Window displays all the objects that currently exist in the system.

To reduce the number of objects your application creates, complete the following steps:

1. Place two breakpoints in the code surrounding an area of high object creation.
2. Run the application to the first breakpoint.
3. Open the Objects Window and click **Snapshot**.

4. Run the application to the second breakpoint.
5. Open the Objects Window.
6. Click **Compare to Snapshot**.

The Objects Window displays the objects that were created between the two breakpoints. Using this data, a developer can determine which objects were necessary and which objects could be removed or combined. See the *IDE Online Help* for more information.

Data structure selection

For applications that store data using the BlackBerry persistent store mechanism, data structure selection is important. The data structure defines how many object handles and how much flash memory is consumed. Improper data structure selection can consume several different key resources without improving application functionality or user experience.

The data structure should consist of the minimum possible number of objects, especially when using high level objects like `Vectors` and `Hashtables`. These classes provide significant functionality but are not efficient storage mechanisms and should be avoided in the persistent store if possible.

Primitives should be used instead of objects when possible because they reduce the number of object handles that are consumed on the handheld. 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 `String` object only consumes one object handle and is equivalent if all of the characters can be stored as a byte or, in other words, the value of each character is less than or equal to `0xFF`. If characters cannot be stored as a byte, storing a `String` is equivalent to storing a `char` array.

Object groups

One of the most common errors encountered by application developers is an exhaustion of persistent object handles. As mentioned previously, the fixed number of persistent object handles in the system is determined by the amount of flash memory on the handheld. Depending on the data structure selection, the number of persistent object handles can be quickly exhausted by stored records.

Consider a record that contains 10 `String` fields that represents such items as name, phone number, and address. This record consumes 11 persistent object handles—one for the record object and one for each string. If 3000 records are persisted that will consume 33,000 persistent object handles, which exceeds the current number of persistent object handles on a 16 MB handheld.

In version 4.0 of the BlackBerry Handheld Software, Research In Motion (RIM) has exposed an API that provides the capability of object grouping. Using the `net.rim.device.api.system.ObjectGroup` class, developers can consolidate the object handles for an object into one group. Using the example in the previous paragraph, if you group the record you will only use one object handle for the record instead of 11. The object handles for the `String` fields would be consolidated under the record object handle.

However, when an object is grouped, it is read-only. You must ungroup the object before making any changes. After the changes are complete, regroup the object using the normal group method. If you attempt to modify a grouped object without first ungrouping it, an `ObjectGroupReadOnlyException` is thrown.

Note: A performance penalty occurs when an object is ungrouped. 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.

Common exceptions and errors

This section describes some common errors and exceptions that occur when a handheld is suffering from low memory conditions.

JVM 531

The JVM Error 531 indicates that the system has run out of at least one critical resource, such as flash memory, persistent object handles, or object handles. Usually, this error occurs when the number of object handles or the flash memory is exhausted.

The best way to determine which resource has been exhausted is to use the `MemoryStats` class during the execution of the `LowMemoryListener`. When the `LowMemoryListener` is invoked, your application can retrieve the memory statistics and log them. Using this information you can determine which critical resource is exhausted.

This error stops the system and a reset is required. Also, this error cannot be handled programmatically and typically occurs during a lower level system operation that requires one of the critical resources. For example, a commit operation on a `PersistentStore` can cause this error when there is not enough flash memory available on the handheld.

OutOfMemoryError

This error occurs when the VM cannot allocate any memory for a requested operation. This error typically occurs when an application asks for a `String` or other type of `Object` to be created by the system. If the VM cannot allocate the necessary space for this `Object`, it throws an `OutOfMemoryError`.

This error differs from the JVM 531 error in that an `OutOfMemoryError` is recoverable; however, your program crashes unless you explicitly catch the error.

Note: The `OutOfMemoryError` error typically occurs in your application code as opposed to during a lower level system operation.

Troubleshooting tips

Two common situations occur when the handheld experiences serious memory issues. The most common is when a handheld exhibits the JVM 531 error consistently. The second is when this same behaviour can be reproduced on the simulator. Attempting to resolve this problem on the simulator is significantly easier than using a handheld, but in the first case that is not an option.

Live Handheld

The `javaloader` utility can help diagnose errors and exceptions on the handheld. Using `javaloader`, you can retrieve the event log that contains lists of any exceptions that occurred on the handheld.

For example, run the following command to retrieve the event log from a USB handheld:

```
javaloader -u eventlog > log.txt
```

There is typically not enough information in the event log to determine the cause of memory issues. You might need to attach the handheld to the debugger, and then follow the steps outlined in *Simulator* on page 3. See the *IDE Online Help* for more information on attaching a handheld to the debugger.

Note: The profiler does not work with a connected device. The Objects window works with a connected device.

Simulator

The Objects window provides a significant amount of information on the status of objects in the system.

Review the information in the Objects window and ask the following questions:

- Consider the size of the objects attributed to the application. Are all the objects necessary?
- Can any objects representing primitives such as `Long`, `Integer`, and `Boolean` be stored as primitives instead of as objects?
- Are all of the persisted objects necessary?

- Are there any instances of `Vector` and `Hashtable`? Are these necessary? If so, how many `Object` handles are wasted inside the `Vector` or `Hashtable` because the initial size is greater than the needed size?
- How many `Objects` are being created then thrown away? In other words, how many scope-specific `Objects` are created?

Frequently Asked Questions

1. Does `LowMemoryManager.markAsRecoverable()` cascade? For example, will invoking the method on an array of `Strings`, `Integers` or other `Objects` mark the underlying objects as recoverable?

Yes. The `markAsRecoverable()` method marks all the referenced objects as recoverable.

2. Should an application invoke `LowMemoryManager.poll()` directly? If so, under what conditions?

Application should not invoke `poll()` directly. Applications can rely on the system to invoke this method when necessary. The system has more information on the current state of the handheld and is better equipped to know when to invoke `poll()`.

3. What is the optimal size for storing byte arrays?

The JVM automatically converts all byte arrays to 4-KB segments instead of storing the byte array as one contiguous array. There is typically no need for applications to break up their byte arrays.

Note: Byte arrays cannot be larger than 128 KB on the handheld.

Java Low Memory Manager

See the *Low Memory Management for the BlackBerry Java Development Environment* whitepaper for more information.

Garbage Collection

See the *Garbage Collection in the BlackBerry Java Development Environment* whitepaper for more information.

Part number: SWD_X_RIM(EN)-005.000

*Check with service provider for availability, roaming arrangements and service plans. Certain features outlined in this document require a minimum version of BlackBerry Enterprise Server software, BlackBerry Desktop Software, and/or BlackBerry handheld software. May require additional application development. Prior to subscribing to or implementing any third party products or services, it is your responsibility to ensure that the airtime service provider you are working with has agreed to support all of the features of the third party products and services. Installation and use of third party products and services 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. You are solely responsible for determining whether such third party licenses are required and are responsible for acquiring any such licenses. To the extent that such intellectual property licenses may be required, RIM expressly recommends that you do not install or use these products and services until all such applicable licenses have been acquired by you or on your behalf. Your use of third party software shall be governed by and subject to you agreeing to the terms of separate software licenses, if any, for those products or services. Any third party products or services that are provided with RIM's products and services are provided "as is". RIM makes no representation, warranty or guarantee whatsoever in relation to the third party products and services and RIM assumes no liability whatsoever in relation to the third party products and services even if RIM has been advised of the possibility of such damages or can anticipate such damages.

© 2005 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 and 'Always On, Always Connected' are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The handheld 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; D,445,428; D,433,460; D,416,256. Other patents are registered or pending in various countries around the world. Please visit www.rim.net/patents.shtml for a current listing of applicable patents.

This document is provided "as is" and Research In Motion Limited (RIM) assumes no responsibility for any typographical, technical or other inaccuracies in this document. 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 AFFILIATED COMPANIES AND THEIR 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 and/or third party web sites ("Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation the content, accuracy, copyright compliance, 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 in any way. Any dealings with third parties, including, without limitation, compliance with applicable licenses and terms and conditions, are solely between you and the third party. RIM shall not be responsible or liable for any part of such dealings.