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

Certain features outlined in this documentation require a minimum version of BlackBerry® Enterprise Server, BlackBerry® Desktop Software, BlackBerry® Device Software and/or BlackBerry Handheld Software and may require additional development or Third Party Products and Services for access to corporate applications.

Certain products mentioned in this documentation include software developed by the Apache Software Foundation (http://www.apache.org/) and/or licensed pursuant to Apache License, Version 2.0 (http://www.apache.org/licenses/). For more information, see the NOTICE.txt file included with the software. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

The terms of use of any RIM product or service are set out in a separate license or other agreement with RIM applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY RIM FOR PORTIONS OF ANY RIM PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.
Chapter 5
Data structures and memory management on mobile devices

Objectives

- Explain why memory management is important for mobile devices
- Identify techniques to minimize memory usage
- Identify data structures that minimize memory consumption
- Identify and describe the purpose of BlackBerry persistence model; MIDP record store; runtime store; File Connection API
- Describe how to back up and synchronize data

This chapter introduces data structures and memory management. This chapter explains why memory management is important when developing applications for mobile devices. The chapter also explains how to manage low memory availability and how data structure selection can minimize memory consumption. This chapter describes the following storage options: BlackBerry persistence model; MIDP record store; runtime store; and File Connection API.
Memory management

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

Low memory has a performance impact on the BlackBerry smartphone. Java garbage collections automatically reclaim memory that the operating system no longer uses. Garbage collection also combines free memory into large blocks. If the BlackBerry smartphone runs out of SRAM space, the BlackBerry Java Virtual Machine swaps memory out to the flash memory. If flash memory is also full, then the BlackBerry Java Virtual Machine cannot perform this task effectively, so garbage collection runs more frequently and for a longer period.

Techniques to minimize memory use

You need to know how to minimize memory usage when you are designing programs. The following list highlights some of the main techniques to minimize memory use:

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

Low memory manager

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

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

- The amount of available flash memory on the BlackBerry smartphone falls below a certain threshold. The flash memory threshold depends on the amount of free RAM in the system. The flash memory threshold ranges between 400 KB and 800 KB.
Data structures and memory management on mobile devices

- The number of persistent object handles that are available on the BlackBerry smartphone falls below 1000 persistent object handles.
- The number of object handles that are available on the BlackBerry smartphone falls below 1000 object handles.

Data structure selection

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

Consider the following guidelines:
- Ensure that the data structure consists of the minimum possible number of objects, especially when you use a high-level object such as a vector or a hash table. These classes provide significant functionality but are not efficient storage mechanisms. Avoid using them in the persistent store if possible.
- When possible, use primitive data types instead of objects because primitive data types reduce the number of object handles that are consumed on the BlackBerry smartphone. An array of primitive data types is an object and consumes an object handle.
- String objects are as efficient as byte arrays. A String object consumes only one object handle and is equivalent to a byte array if your application stores all of the characters as a byte. The value of each character is less than or equal to the decimal value of 255. If your application cannot store characters as a byte, you can store the characters as a String because it is equivalent to storing a character array.

Object consolidation

The amount of flash memory on the BlackBerry smartphone determines the fixed number of persistent object handles that are available in the system. The data structure selection determines the rate at which the stored records exhaust the number of persistent object handles. A persistent object consumes a persistent object handle and an object handle. A transient object consumes only an object handle.

For example, a persistent object that contains ten String fields, which represent items such as a name, a phone number, and an address, consumes 11 persistent object handles, one for the record object and one for each String. If a BlackBerry Java Application stores 3000 records, the application consumes 33,000 persistent object handles, which may exceed the number of persistent object handles available on the flash memory of a BlackBerry smartphone.

You can use the net.rim.device.api.system.ObjectGroup class to consolidate the object handles for an object into one group. If you use the example in the previous paragraph, grouping the record, the record consumes one persistent object handle instead of 11. The object handles for the String fields consolidate under the record object handle.
Chapter 5

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

When you ungroup an object it has an impact on performance. The system creates a copy of the grouped object and allocates handles to each of the objects inside that group. Therefore, ungroup objects only when necessary.

Garbage collection

An unreachable object is a dynamically allocated object that has no reachable reference to it. The garbage collector determines whether an object is reachable. The garbage collection system reclaims and reuses resources as it identifies and discards objects that a program no longer needs.

The two types of garbage collection are RAM and full.

RAM garbage collection

A RAM garbage collection operation removes unreferenced objects from RAM. The BlackBerry Java Virtual Machine initiates RAM garbage collection only when the BlackBerry Java Virtual Machine cannot allocate an object because of a lack of space in RAM. The RAM garbage collection operation typically takes 500 to 600 milliseconds to execute. The garbage collection operation removes freshly allocated variables that are no longer referenced in RAM.

Full garbage collection

The full garbage collection operation executes for 1 second on average and takes less than 2 seconds to complete. The full garbage collection operation performs the following actions:
  - performs a RAM garbage collection operation
  - marks objects in flash memory that are no longer referenced or persistent
  - releases non-persistent object handles in RAM and flash memory

The system initiates a full garbage collection in the following situations:
  - The BlackBerry Java Virtual Machine cannot allocate an object because of a lack of available space in RAM.
  - A process is about to exceed its currently allocated heap size.
  - The BlackBerry Java Virtual Machine cannot allocate a new object because the object handles are not available.
  - The BlackBerry smartphone is idle.
Idle garbage collection

The system attempts to perform the following garbage collection operations when the BlackBerry smartphone idles. This operation lets the system improve its performance without affecting the BlackBerry smartphone user experience.

- Full garbage collection operation occurs when the BlackBerry smartphone idles for a relatively small amount of time, if required.
- Thorough garbage collection operation occurs when the BlackBerry smartphone idles for a significant period of time and a garbage collection operation is required.

Garbage collection does not occur every time that the BlackBerry smartphone is idle. It occurs only when the system calculates that a garbage collection operation is beneficial for optimal system performance and maximized battery performance.
1. Which of the following statements about full garbage collection is false?
   A. Full garbage collection releases persistent object handles in RAM and flash memory.
   B. Full garbage collection performs a RAM garbage collection operation.
   C. Full garbage collection releases non-persistent object handles in RAM and flash memory.

2. Which of the following is a good tip for minimizing memory usage
   A. Use objects rather than primitive data types.
   B. Use primitive data types rather than objects.
   C. Avoid setting object references to null.
   D. Filter and sort all data on the BlackBerry smartphone instead of on the server.

3. Object groups can be updated. True or False?

4. A persistent object containing five string fields consumes how many object handles?
   A. Five
   B. Six
   C. Seven
   D. None

5. The Java Virtual Machine initiates garbage collection under which conditions? Choose all that apply.
   A. Flash memory is available.
   B. The Java Virtual Machine cannot allocate an object.
   C. The smartphone is idle and battery life can be maximized.
   D. A process falls below its minimum heap size.
Answers

1. A
2. B
3. False
4. B
5. B and C
Data storage

Applications can save data in the BlackBerry smartphone’s non-volatile flash memory or microSD expandable memory card. This data persists across BlackBerry smartphone resets. With either the BlackBerry Persistent Store APIs or the MIDP RMS APIs, you can store data persistently to flash memory. Persistent storage permits a BlackBerry smartphone to maintain data in non-volatile memory even when you remove the battery from the BlackBerry smartphone.

You can store data on the BlackBerry smartphone in one of the following ways:

- BlackBerry persistence model
- MIDP record store
- Runtime store
- FileConnection API
- SQLite®

Access to memory

The BlackBerry Java Virtual Machine is designed to inhibit applications from causing problems accidentally or maliciously in other applications or on the BlackBerry smartphone. BlackBerry smartphone applications can write to the BlackBerry smartphone memory that the BlackBerry Java Virtual Machine uses. BlackBerry smartphone applications can 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.

File systems and paths

Every BlackBerry smartphone includes memory where you can store data. BlackBerry smartphones include the following types of memory: internal flash, an internal SD card, and microSD memory card. Different models support different memory configurations and associated file systems. To read and write data to and from each type of file system, you must know the corresponding path to use.

- Internal flash: Internal flash is the internal file system on smartphones that do not support the built-in file system type. The size of this file system is from 5 to 128 MB. To read and write to this file system use the path file://store/home/user.

- Built-in: An internal SD card controls transient data objects and runtime processes. The internal SD card of this file system is currently supported only on BlackBerry® Bold™ Series and BlackBerry® Storm™ Series smartphones. The size of this file system on these smartphones is 860 MB, but the size might change in future on smartphones that support this type of file system. To read and write to this file system use the path file:///system.

- Media card: This is the file system on microSD cards. The size of this file system varies with the size of the inserted microSD card. The microSD card stores media files, documents and
persistent data from third-party applications. To read and write to this file system use the path file:///SDCard.

Persistent data storage

The BlackBerry persistence model provides a flexible and efficient way to store data. String and Integer are examples of two objects that implement the persistable interface. This interface lets you save an object in the persistent store. To store custom object types, the custom type class must implement the Persistable interface.

The application that creates the data lets you share data between applications.

The BlackBerry persistence model uses the following packages:

- net.rim.device.api.system.PersistentStore
- net.rim.device.api.system.PersistentObject
- net.rim.device.api.util.Persistable

To save an object, you need to add it to the persistent store using a unique ID. You can then retrieve the object using that same ID. PersistentObject.getContents() returns a reference to the object; you can develop your code so that the changes you make to it are automatically saved when the application exits.

To be persistable, a data class needs to implement the Persistable interface. The members of a data class must be primitive data types or object types that also implement the Persistable interface. The Persistable interface defines no methods or variables. You can store objects that implement the Persistable interface.

Create a persistent data store

Each PersistentObject has a unique long key.

To create a persistent data store, complete the following steps:

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

2. To create a unique long key, in the BlackBerry JDE Plug-in for Eclipse field, type a string value that matches the package name of your application.
   com.rim.samples.docs.userinfo

3. Right-click the string and click Convert 'com.rim.samples.docs.userinfo' to long.
4. Include a comment in your code to identify the string used to generate the unique long key

5. To create a persistent data store, create a single static PersistentObject and invoke PersistentStore.getPersistentObject, using the unique long key as a parameter.

   ```java
   static PersistentObject store;

   static {
     store = PersistentStore.getPersistentObject(0xa1a569278238dad2L);
   }
   ```

**Store persistent data**

In the previous procedure, you initialized the data store. To store persistent data, complete the following steps:

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

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

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

   ```java
   String[] userinfo = {username, password};

   synchronized(store) {
     store.setContents(userinfo);
     store.commit();
   }
   ```

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

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

To retrieve persistent data, complete the following steps. In this example, you’ll retrieve two objects: the `currentusernamefield` object, and the `currentpasswordfield` object.

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

2. **Invoke `getContents()` on a persistent object.**

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

   ```java
   synchronized(store) {
     String[] userinfo = (String[])store.getContents();
     if(currentinfo == null) {
       Dialog.alert(_resources.getString("No previous user info"));
     } else {
       currentusernamefield.setText(currentinfo[0]);
       currentpasswordfield.setText(currentinfo[1]);
     }
   }
   ```

Remove persistent data

If you delete the `.cod` file that defines a Persistent Store, then all persistent objects that the `.cod` file created are deleted. To remove persistent data, complete the following steps:

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

2. **To remove all persistent data from a particular persistent store, invoke**

   ```java
   PersistentStore.destroyPersistentObject();
   ```

3. **Provide the same unique key used to create and access the store as a parameter for the persistent object.**
4. To remove individual data, treat the data as normal objects, and remove references to each object. A garbage collection operation removes the data.

**MIDP record storage**

The MIDP record store lets you port an application across multiple smartphones that are compatible with the Java® ME. The `javax.microedition.rms` package includes the MIDP record store implementation.

In MIDP, persistent data is stored as records in RecordStore objects. Each RecordStore object belongs to a single MIDlet suite. A MIDlet is an application that conforms to the MIDP standard. A MIDlet makes data available only to the MIDlet suite that created it, or to applications.

**Create an MIDP record store**

To create an MIDP record store, complete the following steps:

1. Import the `javax.microedition.rms.RecordStore` class.
2. Invoke `openRecordStore()`, and specify true to create the record store.

**Add a record to a record store**

To add a record to a record store, complete the following steps:

1. Import the `javax.microedition.rms.RecordStore` class.
2. Invoke `addRecord()`.

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

**Retrieve a record from a record store**

To retrieve a record from a record store, complete the following steps:

1. Import the following classes:
   - `javax.microedition.rms.RecordStore`
   - `java.lang.String`
2. Invoke `getRecord(int, byte[], int)`. Pass the following parameters:
   - record ID
   - byte array
   - offset
byte[] data = new byte[store.getRecordSize(id)];
store.getRecord(id, data, 0);
String dataString = new String(data);

Retrieve all records from a record store

To retrieve all records from a record store, complete the following steps:

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

2. **Import the following interfaces:**
   - `javax.microedition.rms.RecordEnumeration`
   - `javax.microedition.rms.RecordFilter`
   - `javax.microedition.rms.RecordComparator`

3. **Invoke `openRecordStore()`**.

4. **Invoke `enumerateRecords()`**. Pass the following parameters:
   - **filter**—specifies a `RecordFilter` object to retrieve a subset of record store records (if null, the method returns all records)
   - **comparator**—specifies a `RecordComparator` object to determine the order in which the method returns the records (if null, the method returns the records in random order)
   - **keepUpdated**—determines if the method keeps the enumeration current with the changes to the record store

The following example shows how to open a record store and enumerate records:

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

Runtime storage

BlackBerry smartphones use a **runtime store** as a central location in which BlackBerry Java Applications can share runtime objects.

The runtime store is not persistent. When you restart the BlackBerry smartphone, the data in the runtime store is cleared.
Retrieve the runtime store

To retrieve the runtime store, complete the following steps:

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

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

Add an object in the runtime store

To add an object in the runtime store, complete the following steps:

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

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

Replace an object in the runtime store

To replace an object in the runtime store, complete the following steps:

1. Import the following classes:
   - `net.rim.device.api.system.RuntimeStore`
   - `java.lang.String`
   - `net.rim.device.api.system.ControlledAccessException`
2. **Invoke** replace().

3. **Create a try-catch block to manage the** ControlledAccessException **that occurs if the runtime object with the specified ID does not exist.**

   ```java
   RuntimeStore store = RuntimeStore.getRuntimeStore();
   String newmsg = "Some new text";
   try {
       Object obj = store.replace(0x60ac754bc0867248L, newmsg);
   } catch(ControlledAccessException e) {
   }
   ```

---

**Retrieve a registered runtime object**

To retrieve a registered runtime object, complete the following steps:

1. **Import the following classes:**
   - net.rim.device.api.system.RuntimeStore
   - net.rim.device.api.system.ControlledAccessException

2. **Invoke** RuntimeStore.get(). **Provide the runtime object ID as a parameter.**

3. **Create a try-catch block to manage the** ControlledAccessException **that occurs if the BlackBerry Java Application does not have read access to the specified runtime object.**

   ```java
   RuntimeStore store = RuntimeStore.getRuntimeStore();
   try {
       Object obj = store.get(0x60ac754bc0867248L);
   } catch(ControlledAccessException e) {
   }
   ```

---

**Retrieve an unregistered runtime object**

To retrieve an unregistered runtime object, complete the following steps:

1. **Import the following classes:**
   - net.rim.device.api.system.RuntimeStore
   - net.rim.device.api.system.ControlledAccessException
2. Invoke `RuntimeStore.waitFor()` to wait for registration of a runtime object to be completed.

3. Create code to handle exceptions.

```java
RuntimeStore store = RuntimeStore.getRuntimeStore();
try {
    Object obj = store.waitFor(0x60ac754bc0867248L);
} catch (ControlledAccessException e) {
} catch (RuntimeException e) {
}
```

**SQLite**

SQLite is an open source relational database library. It is designed to make efficient use of memory resources and it includes few features. As a result, it can be a good choice for embedded and wireless applications. BlackBerry devices that run BlackBerry® Device Software version 5.0 or later have the SQLite library integrated into the operating system and virtual machine. The SQLite API enables you to develop applications that use the integrated SQLite database. It was introduced in BlackBerry Java Development Environment version 5.0.

The `net.rim.device.api.database` package includes classes that enable you to work with SQLite. You can use the following approach to work with an existing SQLite database.

- Create a SQL statement by calling `Database.createStatement()`.
- Prepare the statement to run by calling `Statement.Prepare()`. Performing this step is like compiling the statement.
- Run the statement by calling `Statement.getCursor()` if the statement might return results and `Statement.Execute()` otherwise.
- If there are results, retrieve them by iterating over the returned `Database.Cursor` row by row.

**Locations of SQLite database files**

Each SQLite database is stored in a single file. If you only specify the database name as the parameter value to `DatabaseFactory.create()`, the database file is created on the SD card of the device. The default location for the database file is `/SDCard/databases/<application_name>/`. The name of the application that creates the database is included in the path to avoid name collisions.
You can create database files in device memory by specifying the corresponding file system path. On the BlackBerry® Bold™ Series and the BlackBerry® Storm™ Series devices, you can create database files in the system memory by specifying the corresponding file system path.

<table>
<thead>
<tr>
<th>Type of memory</th>
<th>File system path</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD card</td>
<td>/SDCard/</td>
</tr>
<tr>
<td>device memory</td>
<td>/store/home/user</td>
</tr>
<tr>
<td>system memory</td>
<td>/system/</td>
</tr>
<tr>
<td>default</td>
<td>SDCard/databases/&lt;application_name&gt;</td>
</tr>
</tbody>
</table>

**Best practices**

The following sections describe the best practices that you can adopt when you develop applications that use the integrated SQLite database.

**Use a SQLite database browser with the BlackBerry Smartphone Simulator**

When you develop an application that uses a SQLite database, you might find it useful to work in a development environment that enables you to execute a SQL statement and look at the resulting data changes in the database. A convenient way to establish a development environment like that is to store your database on the SD card during development. As you develop your application, you can run it on the BlackBerry® Smartphone Simulator to check newly-added functionality. The BlackBerry Smartphone Simulator stores files saved on the emulated SD card in a specified directory on your development computer.

You can use a SQLite database browser to inspect and manage the SQLite databases that are stored on the file system of your desktop. You can use the BlackBerry Smartphone Simulator to run your application and access the SQLite database used by your application from your desktop file system. You can use the SQLite database browser to navigate to the SQLite database that is stored in the directory that the BlackBerry Smartphone Simulator uses to store the files saved on the emulated SD card. Using the database browser to view your database as you change it programmatically can give you real-time feedback about the changes your application is making to the database.

You might store your database on an SD card. However, BlackBerry Bold Series and BlackBerry Storm Series devices support storing SQLite databases on internal device memory. If your application is designed to store your SQLite database on internal device memory, you should implement your application so that it is easy to modify the code to change the storage location of the database.
Reduce the database size by using the vacuum command

A SQLite database is stored in a single file. Some database operations, such as dropping a table or inserting and deleting data, cause the file to become fragmented. You can use the vacuum command to defragment and reduce the size of the file.

The vacuum command copies all of the information required to recreate a database into a temporary file in memory and uses that information to create a new database file. As the command creates the new database file, it eliminates free pages, makes table data contiguous, and reorganizes the database file structure.

If you run the vacuum command on a BlackBerry device that does not have enough memory available to store the required temporary information, the command will fail with an out of memory error.

Optimizing SQLite database performance

To optimize the performance of a SQLite database on a BlackBerry device, you must consider both the database design and how your application uses the SQLite API to interact with the database.

Consider the following guidelines:

- **Store as little data as possible.**—SQLite caches frequently accessed database pages. By storing less data you can increase the probability that the SQLite library retrieves requested data more quickly from the cache rather than from the relatively slow flash memory.

- **Use temporary tables.**—Do this only if you do not need the data to be available following a reset of the BlackBerry device.

- **Prepare generic statements that use named variables.**—Execute the statements when they are required by iterating through the variable values, binding the values to the named variables in each iteration.

- **Use explicit transactions.**—Otherwise, a transaction begins before each statement is executed and ends after the statement is executed. This is inefficient. It requires the opening, reopening, writing to, and closing of the journal file for each statement.

- **Avoid subqueries.**—By default, the SQLite library stores the subquery results in a temporary file.

Data management

The BlackBerry API Set allows an application to persist data in multiple ways. The BlackBerry Persistent Store APIs and the MIDP RMS APIs (JSR 37 and JSR 118) are available on all Java® based BlackBerry smartphones. A BlackBerry smartphone that runs BlackBerry device software version 4.2 or later provides a traditional file system and support for saving content directly to the file system using JSR 75 APIs. With either the BlackBerry Persistent Store APIs or the MIDP RMS APIs, you can store data persistently to flash memory. The data persists even if you remove the battery from the BlackBerry smartphone.
File Connection APIs

The File Connection APIs provide a traditional file system, and support for saving data directly to the file system on the BlackBerry smartphone or to a microSD card. When the BlackBerry smartphone is connected to a computer, you can use Windows® Explorer to view data in the file system and move the data to the computer.

The `javax.microedition.io.file` package supports the JSR 75 File Connection API. Applications use this file package to access the file system of the microSD media card. You can also implement the File Connection interface to access any file type on the file system, except for sample content, such as ring tones and background images. This sample content is protected.

The File Connection APIs are defined in the `javax.microedition.io.file` package. It includes two interfaces and three classes.

- **ConnectionClosedException**—This exception occurs when an application invokes a method on a closed file connection.
- **FileConnection**—Applications use this API to access files or directories.
- **FileSystemListener**—Applications use this API to receive status notifications when the application adds or removes a file system root.
- **FileSystemRegistry**—Applications use this API as a central registry for file system listeners that listen for the addition or removal of file systems.
- **IllegalModeException**—This exception occurs when a method requires a specific security mode (for example, read or write) and the open connection is not in that mode.

Persistent store APIs

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

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

MIDP record management system APIs

The MIDP RMS APIs provide a simple record management system that lets you create a data store object and retain a series of records within that object. Each record is a byte array, so you must first serialize your data into a byte array format before storing it locally. The RMS APIs do not provide inherent indexing or relationships between records. The size limit for a single RMS data store varies, based on the BlackBerry Device Software version. For example, for devices with version 4.1 or earlier
software, the maximum of 64 KB, while version 4.6 and later support 512 KB. An application can create multiple RMS data stores to persist larger amounts of data. The RMS APIs are part of the standard MIDP specification, so all smartphones that support MIDP also support the RMS APIs.

**Backing up and synchronizing data**

You can use the following tools to back up, restore, and synchronize data:

- **BlackBerry® Desktop Manager**
- Synchronization Server SDK
- BlackBerry Enterprise Server

**BlackBerry Desktop Manager**

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

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

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

A BlackBerry Java Application connects to a computer-based application to send the data over a USB connection using the BlackBerry Desktop Synchronization APIs and the BlackBerry Desktop Manager. In this case, you must build an application for Windows that can read the data from the client through an add-in task for the BlackBerry Desktop Manager. You can also use the native USB protocols to write data to the computer application.

**BlackBerry Enterprise Server**

The implementation of the backup, restore, and synchronization functions in the BlackBerry Enterprise Server is the same as the implementation in the BlackBerry Desktop Manager, with the exception that the BlackBerry Enterprise Server backup is wireless.

The automatic wireless backup process on a BlackBerry Enterprise Server is designed to back up data from the BlackBerry device to the BlackBerry Enterprise Server. By default, wireless backup is active on the BlackBerry Enterprise Server. When the automatic wireless backup process runs on the BlackBerry
Enterprise Server, the process saves BlackBerry Java Application data with the user account settings and the other BlackBerry device data that backs up.

**Synchronization Server SDK**

The Synchronization Server SDK provides third-party developers the ability to develop custom data source connectors for the wireless synchronization between a BlackBerry smartphone and data repository. The connectors ensure that when data records are updated on the server, the corresponding changes are pushed out to the local information stored on the BlackBerry smartphone. The Synchronization Server SDK includes a dynamic link library that provides a connector for backup and restore functions.
Quiz

1. Which storage option do you use if you are creating an application that is portable across multiple smartphones? Choose all that apply.
   A. BlackBerry persistence model
   B. MIDP record store
   C. File Connection API
   D. Runtime store
   E. SQLite

2. Which of the following statements about the BlackBerry persistence model is false?
   A. A data class needs to implement the Persistable interface.
   B. The members of a data class must implement the Persistable interface.
   C. The Persistable interface defines methods.

3. Which object is not included in the `javax.microedition.io.file` package?
   A. FileConnection
   B. ConnectionClosedException
   C. FileSystemListener
   D. FileConnectionOpen
   E. IllegalModeException

4. The runtime store is a central location that persists over BlackBerry smartphone resets. True or False?

5. Which package contains the classes that enable you to work with SQLite?

6. Each SQLite database is stored in a single file. True or false?
Answers

1. B and C
2. C
3. D
4. False
5. net.rim.device.api.database
6. True
Chapter 5

Summary

The BlackBerry JVM manages memory usage on the BlackBerry smartphone. The BlackBerry JVM allocates memory, performs garbage collection, and automatically swaps data between SRAM and flash memory.

The low memory manager handles memory resources on the BlackBerry smartphone. After the available memory resources fall below a certain threshold, the low memory manager frees used memory to provide more available memory on the BlackBerry smartphone.

You need to be aware of how to minimize memory usage when you design programs. Low memory has an impact on the BlackBerry smartphone performance; therefore, the Java garbage collection capabilities automatically reclaims file space that the operating system no longer uses. It reclaims and reuses resources as it identifies and discards objects that a program no longer needs. The two types of garbage collection include RAM and Full.

- A RAM garbage collection operation removes unreferenced objects from RAM.
- The full garbage collection operation performs a RAM garbage collection operation, marks objects in flash memory that are no longer referenced or no longer persisted, and then it releases non-persistent object handles in RAM and flash memory.

The system attempts to perform the garbage collection operations when the BlackBerry smartphone is idle. This lets the system improve its performance without impacting the BlackBerry smartphone user's experience.

Data structure selection also contributes to memory usage on the BlackBerry smartphones. It defines how many object handles and how much flash memory a BlackBerry Java Application consumes. The amount of flash memory on the BlackBerry smartphone determines the fixed number of persistent object handles that are available in the system. The data structure selection determines the rate at which the stored records exhaust the number of persistent object handles.

Applications can save data in the BlackBerry smartphone’s non-volatile flash memory or microSD card. This data persists across BlackBerry smartphone resets.

Note:
The BlackBerry® Java® Development Environment is designed to inhibit applications from causing problems accidentally or maliciously in other applications or on the BlackBerry smartphone. BlackBerry smartphone applications can write only to the BlackBerry smartphone memory that the BlackBerry Java Virtual Machine uses.
Every BlackBerry smartphone includes memory where you can store data. BlackBerry smartphones include the following types of on-board memory: internal flash, SRAM, and microSD memory card. Different models support different memory configurations and associated file systems. You can store data on the BlackBerry smartphone in one of the following ways:

- **BlackBerry persistence model**: The BlackBerry persistence model provides a more flexible and efficient way to store data. This model allows you to save an object in the persistent store. Default persistable objects include String and Integer. To store custom object types, the custom type's class must implement the Persistable interface.

- **MIDP**: The MIDP record store allows an application to be portable across multiple smartphones that are compatible with the Java® ME. The `javax.microedition.rms` package includes the MIDP record store implementation.

- **Runtime store**: The runtime store is not persistent. When you restart the BlackBerry smartphone, the data in the runtime store clears. BlackBerry smartphones use a runtime store as a central location in which BlackBerry Java Applications can share runtime objects. By default, only the BlackBerry Java Applications that RIM digitally signs can access data in the runtime store.

The BlackBerry Persistent Store APIs and the MIDP RMS APIs (JSR 37 and JSR 118) are available on all Java® based BlackBerry smartphones.

The File Connection APIs provide a traditional file system, and support for saving data directly to the file system on the BlackBerry smartphone or to a microSD card. You can view data in the file system and move the data to a computer by using Windows Explorer. File connection API Applications use this file package to access the file system for the microSD media card. You can also implement the FileConnection interface to access BlackBerry smartphone ring tones and camera images.

The BlackBerry® Persistent Store APIs are designed to provide a flexible and robust data storage interface. With the BlackBerry Persistent Store APIs, you can save entire Java® objects to memory without having to serialize the data first.

The MDIP RMS APIs provide a simple record management system that allows you to create a data store object and persist a series of records within that object. Each record is a byte array, so you must first serialize your data into a byte array format before storing it locally.

The BlackBerry® Desktop Manager provides a backup and restore tool that a BlackBerry smartphone user can use to save BlackBerry smartphone data to a file on a computer and to restore data to the BlackBerry smartphone.
1. What can you do to reduce the frequency of garbage collection?

2. Using a customer data example that consists of a name, a street address, and a city, describe how to consolidate objects into groups. What are the benefits?

3. Compare and contrast the BlackBerry persistence model, MIDP record store, runtime store, and File Connection API. When do you use each storage option?

4. What must you do to implement synchronization for your application on the BlackBerry smartphone and on the computer?

5. List and describe the five ways that you can optimize the performance of a SQLite database on a BlackBerry smartphone.