Research In Motion

A50 How to Debug and Optimize

On BlackBerry SmartPhones

Andre Fabris

09

Contents

A50 – How to Debug and Optimize

Writing applications for the BlackBerry platform is easy.

Every now and then the application will not do what we intended. I will show you how to debug the application in Eclipse, as well as how to monitor and optimize it for the best user experience.

This tutorial will cover debugging and profiling, and the available tools in Eclipse.

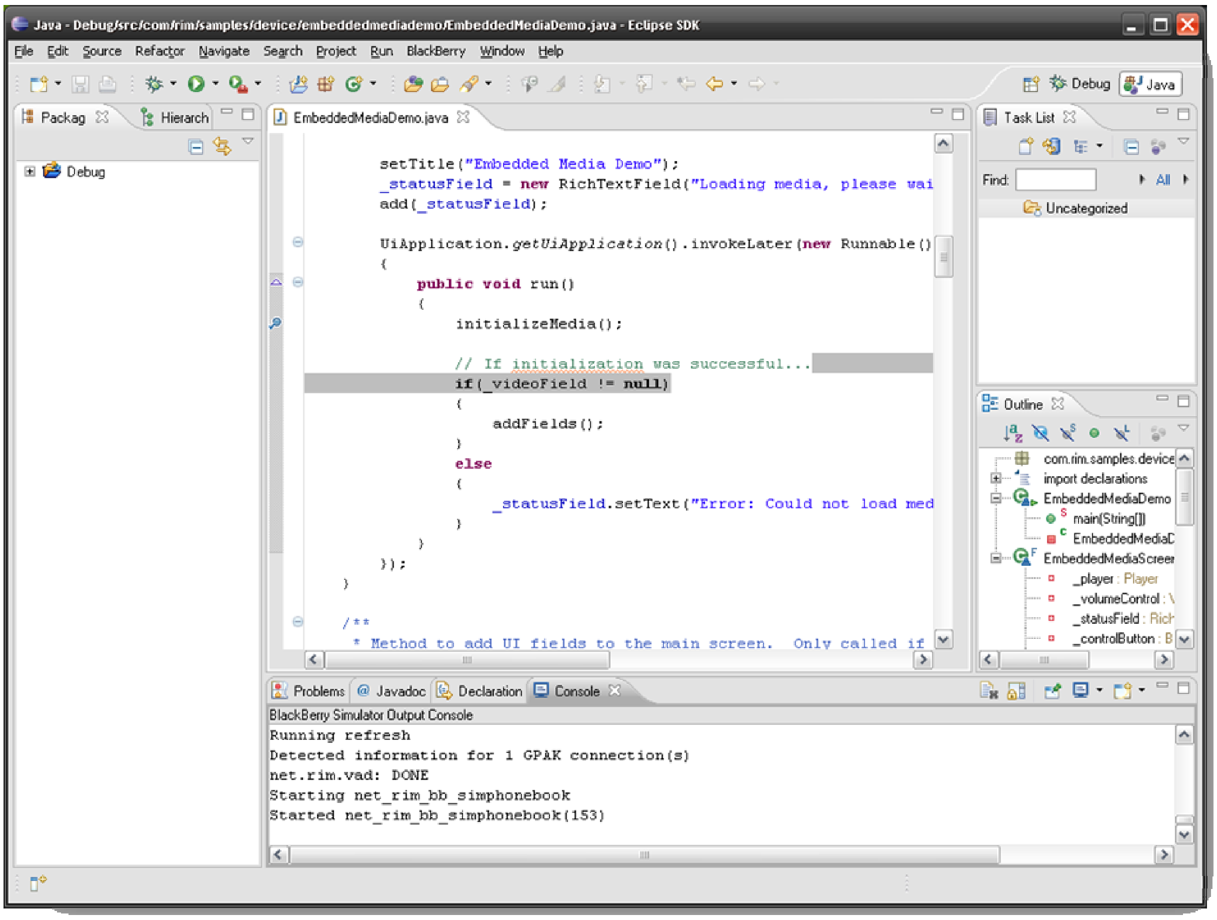
Debugging Tools

Debugging is finding errors and then troubleshooting them in your application. There are number of different ways to debug your application. You can run the application on the simulator and you can run it on the device.

Most of the time you will debug your application on the simulator as it gives you better tools to find the issues. Sometimes you will need to debug on the device as some applications do not run in the simulator (for example, making network connections through a carrier WAP gateway).

There are two ways to run your application in Eclipse: Click on Run / Run option (Ctrl – F11) or Run / Debug (F11). To debug your application you will need to use the latter one.

The Run / Debug command takes longer to launch the simulator but gives you access to a number of tools which I will describe below.



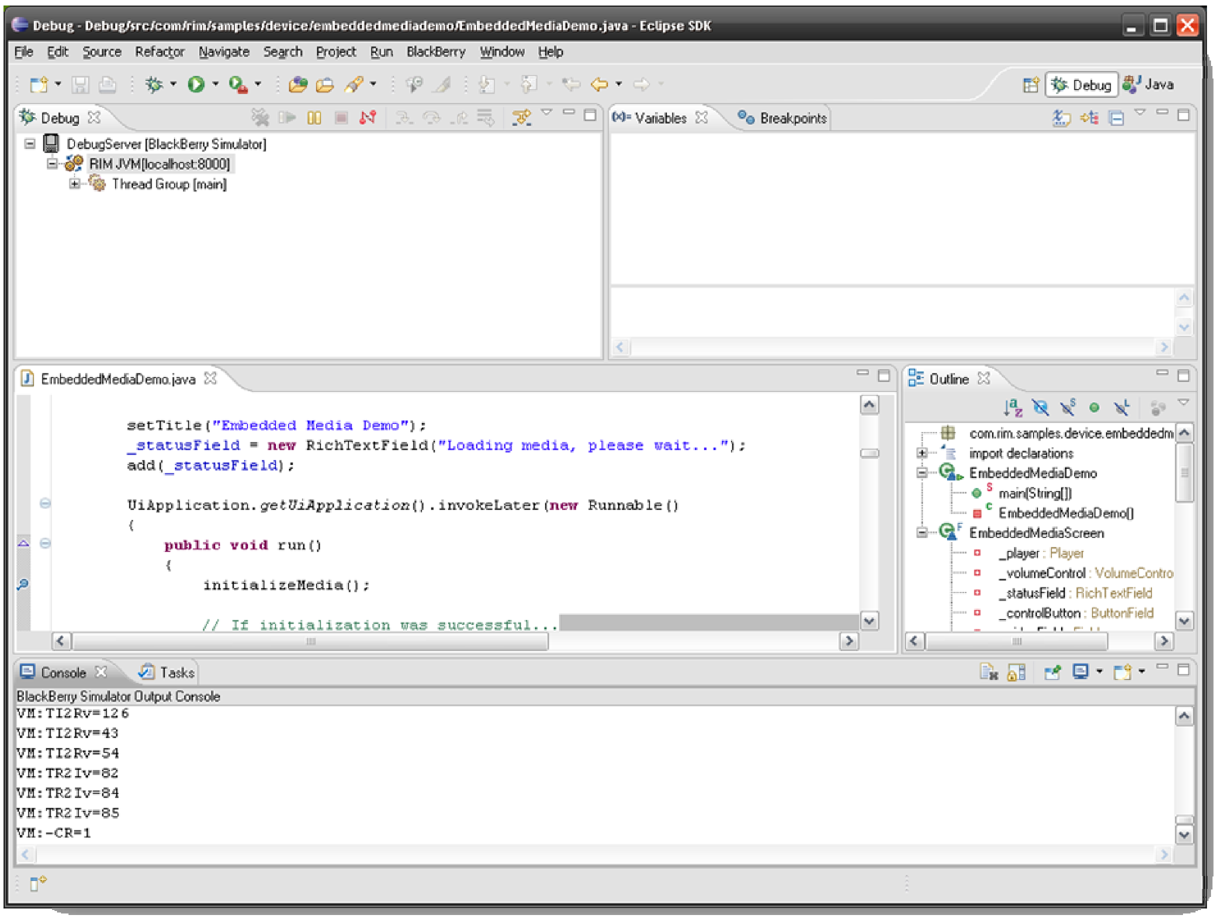
```
        setTitle("Embedded Media Demo");
        _statusField = new RichTextField("Loading media, please wai
        add(_statusField);

        UiApplication.getUiApplication().invokeLater(new Runnable()
        {
            public void run()
            {
                initializeMedia();

                // If initialization was successful...
                if(_videoField != null)
                {
                    addFields();
                }
                else
                {
                    _statusField.setText("Error: Could not load med
                }
            }
        });
    }

    /**
     * Method to add UI fields to the main screen.  Only called if
```

BlackBerry Simulator Output Console

```
Running refresh
Detected information for 1 GPAK connection(s)
net.rim.vad: DONE
Starting net_rim_bb_simphonebook
Started net_rim_bb_simphonebook(153)
```

Figure 1

Eclipse allows you to arrange your workspace in a number of different ways. On Figure 1 we can see a typical edit layout. You will notice in the top right corner that the Java button is pressed. Next to it is Debug button which will switch the view into the debug view.


Debug - Debug/src/com/rim/samples/device/embeddedmediademo/EmbeddedMediaDemo.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run BlackBerry Window Help
Debug
Java
Debug
DebugServer [BlackBerry Simulator]
RIM JVM[localhost:8000]
Thread Group [main]
Variables
Breakpoints
EmbeddedMediaDemo.java
Outline
com.rim.samples.device.embeddedm
import declarations
EmbeddedMediaDemo
main(String[])
EmbeddedMediaDemo()
EmbeddedMediaScreen
_player : Player
_volumeControl : VolumeContro
_statusField : RichTextField
_controlButton : ButtonField
Console
Tasks

```
        setTitle("Embedded Media Demo");
        _statusField = new RichTextField("Loading media, please wait...");
        add(_statusField);

        UiApplication.getUiApplication().invokeLater(new Runnable()
        {
            public void run()
            {
                initializeMedia();

                // If initialization was successful...
```

BlackBerry Simulator Output Console

```
VM:TI2Rv=126
VM:TI2Rv=43
VM:TI2Rv=54
VM:TR2Iv=82
VM:TR2Iv=84
VM:TR2Iv=85
VM:-CR=1
```

Figure 2

If these buttons are not available you can find it in Window menu and then Show View / Debug. In Figure 2 we can see a typical debug layout. The Debug tab (top left) shows us the running threads.

If the simulator is not running most of the windows here will be empty.

Just below the debug layout is our source code.

Here we can add breakpoints. We can add them before we run the simulator or while we are running it. The simulator will stop execution of our application when it hits the breakpoint.

Below the source code window is the Console window. Here, you will see the output from the Virtual Machine as well as the messages from your own application.

In the top right corner we can see the list and values of our variables, as well as a list of breakpoints.

I will show you how to use these windows in more detail.

Console Window

Apart from giving you a lot of useful messages from the JVM, we can add our own custom messages.

If we use for example `System.out.println(“message”);` or

`System.err.println(x);` the text message or value of x will be displayed in console window.

It is good practice to use try-catch blocks to catch exceptions, and then print as much information as you need:

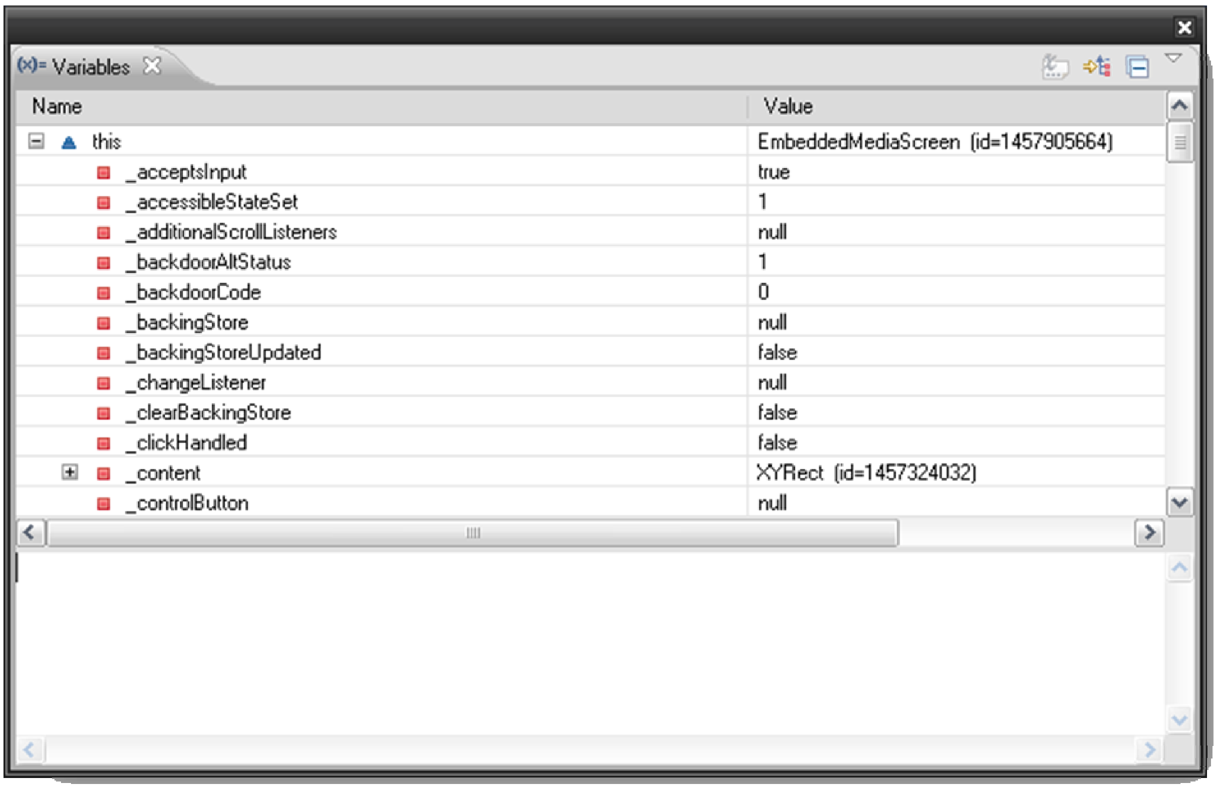
```
    try {
            ...
        }
        catch (Exception e) {
            System.out.println("Error 123");
            System.out.println(e.toString());
            e.printStackTrace();
        }
```

This will print your custom error number, exception name and a Stack if available, which you might find very useful when debugging.

It is also good practice to add this sort of message to the event log which I will cover later in this tutorial.

Variables & Debug Window

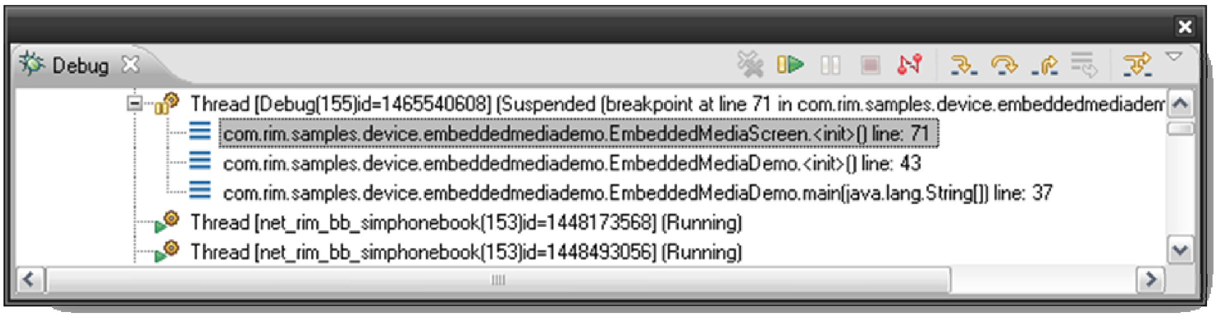
Once the application hits the breakpoint and stops we can have a look at the values of the local variables.



Name	Value
this	EmbeddedMediaScreen (id=1457905664)
_acceptsInput	true
_accessibleStateSet	1
_additionalScrollListeners	null
_backdoorAltStatus	1
_backdoorCode	0
_backingStore	null
_backingStoreUpdated	false
_changeListener	null
_clearBackingStore	false
_clickHandled	false
_content	XYRect (id=1457324032)
_controlButton	null

Figure 3

The Variables window will show the values of all local variables at the breakpoint (Figure 3).



```
Thread [Debug(155)id=1465540608] (Suspended (breakpoint at line 71 in com.rim.samples.device.embeddedmediadem
    com.rim.samples.device.embeddedmediademo.EmbeddedMediaScreen.<init>() line: 71
    com.rim.samples.device.embeddedmediademo.EmbeddedMediaDemo.<init>() line: 43
    com.rim.samples.device.embeddedmediademo.EmbeddedMediaDemo.main(java.lang.String[]) line: 37
Thread [net_rim_bb_simphonebook(153)id=1448173568] (Running)
Thread [net_rim_bb_simphonebook(153)id=1448493056] (Running)
```

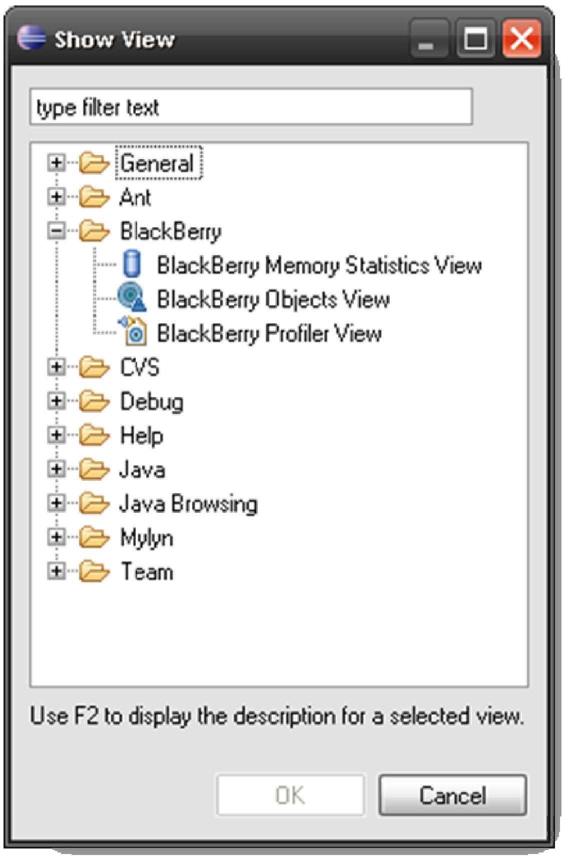
Figure 4

On the top right part of Debug window there are control icons (Figure 4). They allow you to resume your application (F8), step Into (F5), step over (F6) your code. This can be very useful as you can see step by step what is going on in your application.

In the Debug window you can also see the running threads, pause and resume the application manually, even setup and apply step filters.

BlackBerry Memory Statistics View

To see the BlackBerry Memory Statistics view, as well as the Objects and Profiler View, click on Window / Show View / Other / expand BlackBerry and set the View you want. In this case it is the Memory Statistics View (Figure 5).

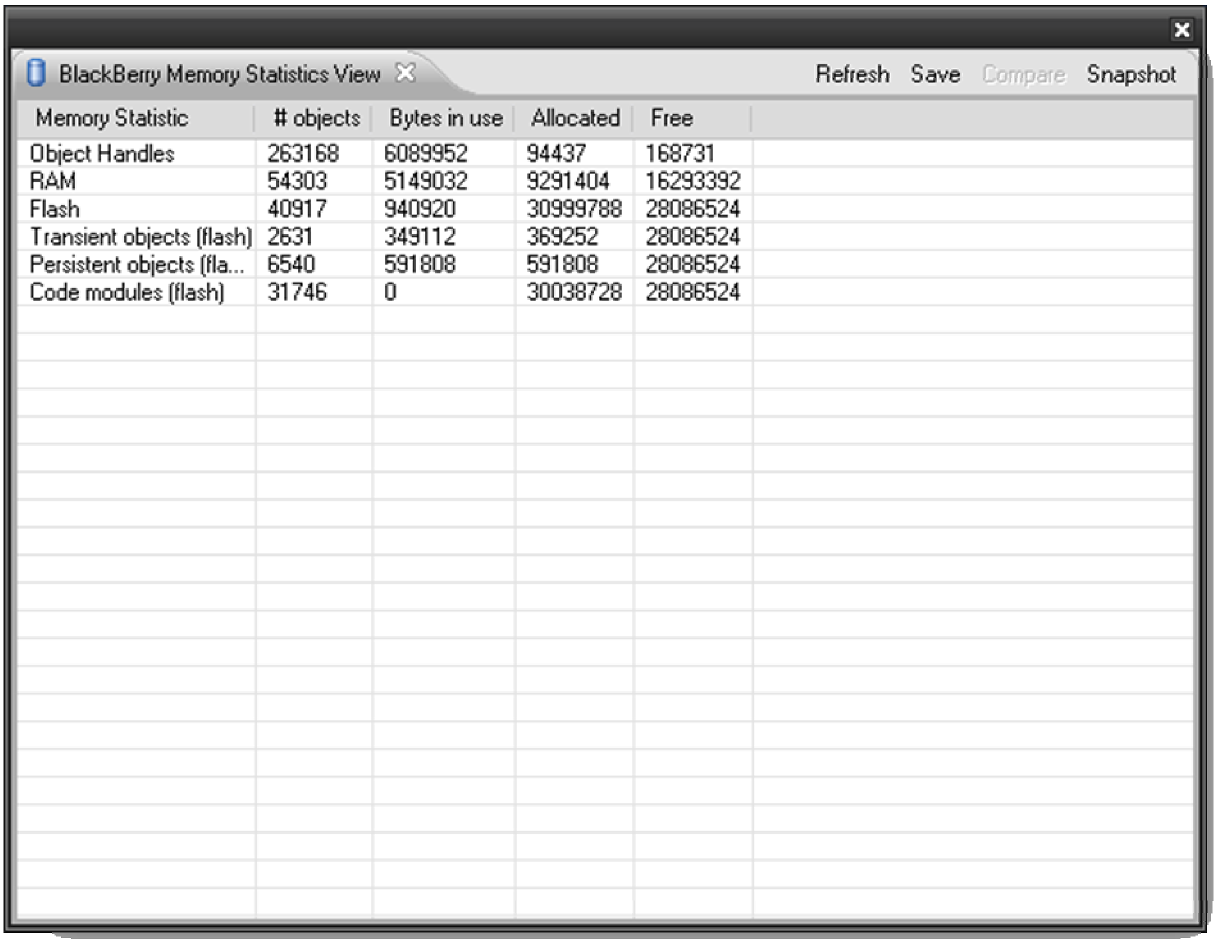


Use F2 to display the description for a selected view.

Figure 5

To use it, set up two breakpoints. When the application stops at the first breakpoint, click on the refresh button on the BlackBerry Memory Statistics View (Figure 6). Press Save and you can save the data in .csv format.

What we really want is to take a snapshot so we can compare it later.

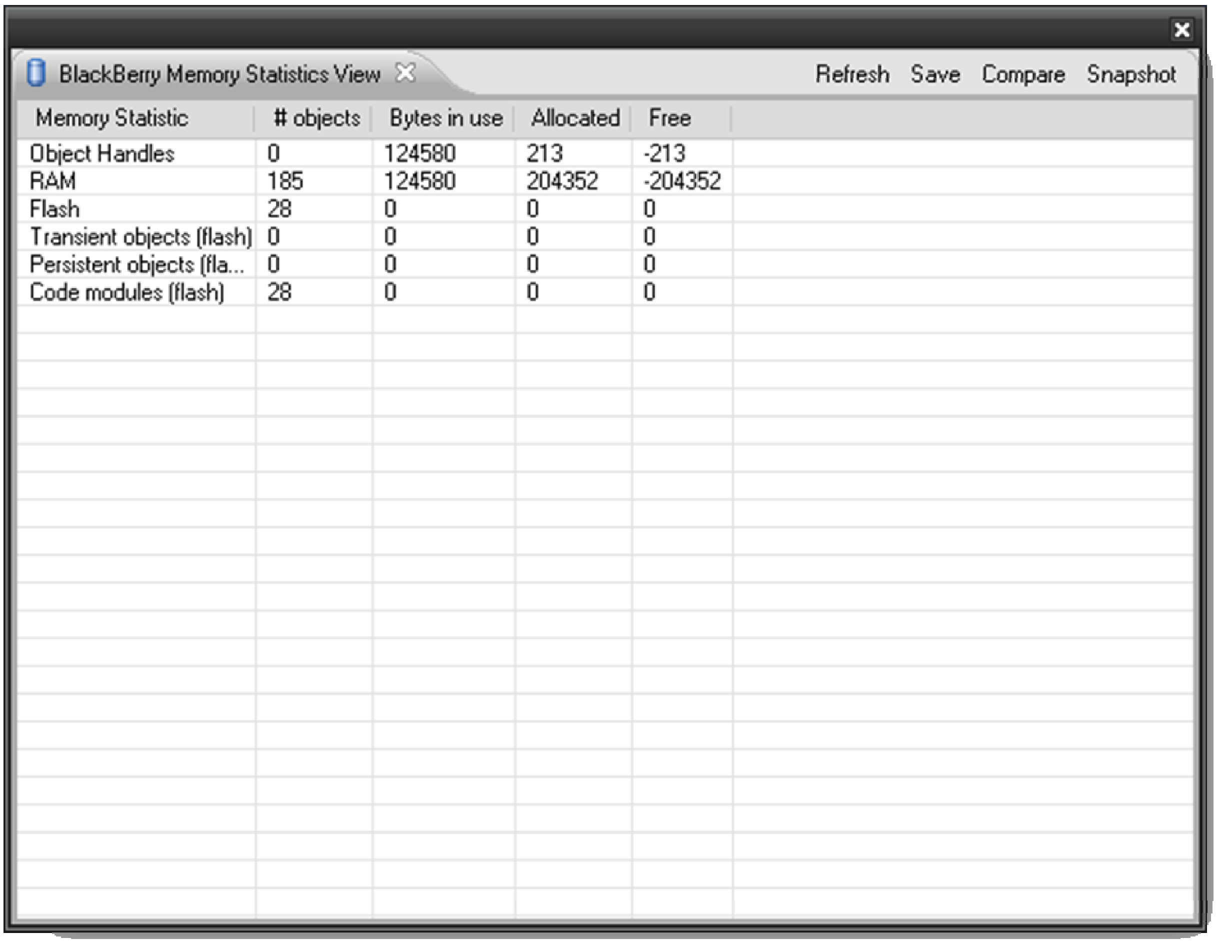


Memory Statistic	# objects	Bytes in use	Allocated	Free
Object Handles	263168	6089952	94437	168731
RAM	54303	5149032	9291404	16293392
Flash	40917	940920	30999788	28086524
Transient objects (flash)	2631	349112	369252	28086524
Persistent objects (fla...	6540	591808	591808	28086524
Code modules (flash)	31746	0	30038728	28086524

Figure 6

Go to the next breakpoint (F8) and then when the application stops, refresh the BlackBerry Memory Statistics View again. Press the compare button, and the system will show you the changes (Figure 7).

As you can see in our sample, we used some object handles, but we used quite a lot of RAM. It is worth investigating what has caused this. In our case we loaded some bitmap images in our application.

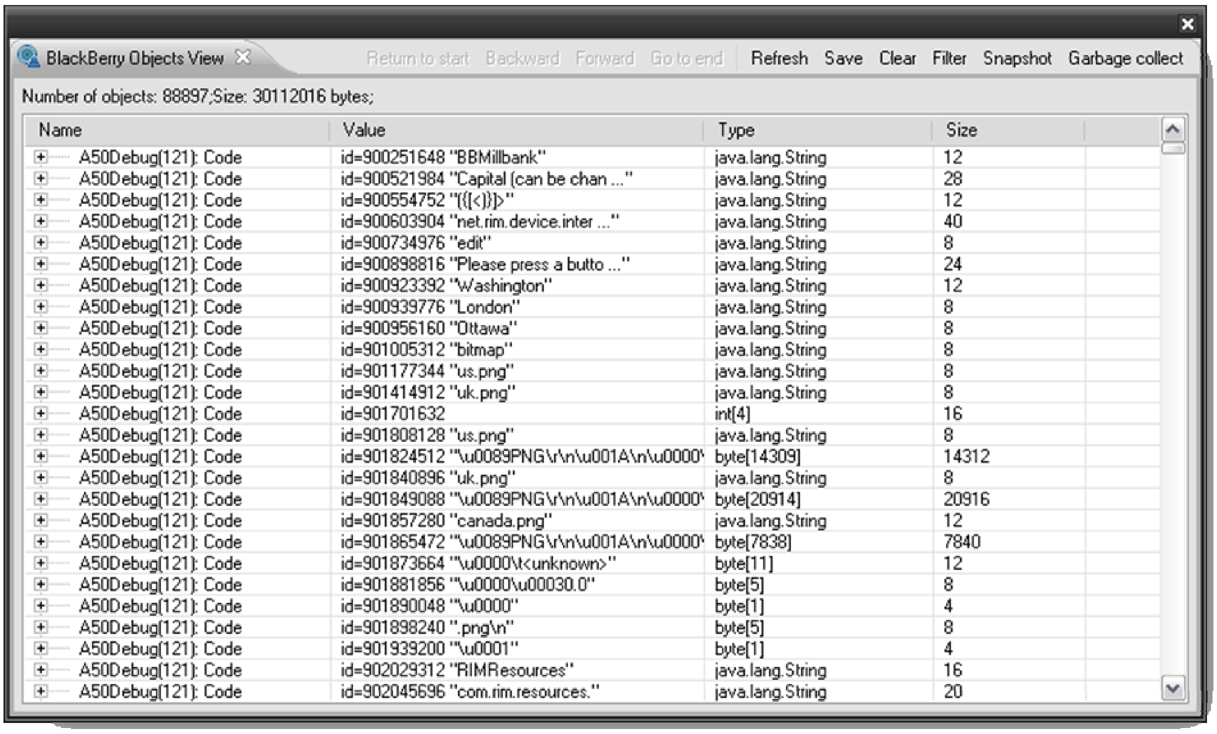


Memory Statistic	# objects	Bytes in use	Allocated	Free
Object Handles	0	124580	213	-213
RAM	185	124580	204352	-204352
Flash	28	0	0	0
Transient objects (flash)	0	0	0	0
Persistent objects (fla...	0	0	0	0
Code modules (flash)	28	0	0	0

Figure 7

BlackBerry Objects View

You will need to add breakpoints to your code. Once the application hits the breakpoint, refresh the BlackBerry Objects View (Figure 8).



Number of objects: 88897;Size: 30112016 bytes;

Name	Value	Type	Size
A50Debug(121): Code	id=900251648 "BBMillbank"	java.lang.String	12
A50Debug(121): Code	id=900521984 "Capital (can be chan ..."	java.lang.String	28
A50Debug(121): Code	id=900554752 "({[<)}]>"	java.lang.String	12
A50Debug(121): Code	id=900603904 "net.rim.device.inter ..."	java.lang.String	40
A50Debug(121): Code	id=900734976 "edit"	java.lang.String	8
A50Debug(121): Code	id=900898816 "Please press a butto ..."	java.lang.String	24
A50Debug(121): Code	id=900923392 "Washington"	java.lang.String	12
A50Debug(121): Code	id=900939776 "London"	java.lang.String	8
A50Debug(121): Code	id=900956160 "Ottawa"	java.lang.String	8
A50Debug(121): Code	id=901005312 "bitmap"	java.lang.String	8
A50Debug(121): Code	id=901177344 "us.png"	java.lang.String	8
A50Debug(121): Code	id=901414912 "uk.png"	java.lang.String	8
A50Debug(121): Code	id=901701632	int[4]	16
A50Debug(121): Code	id=901808128 "us.png"	java.lang.String	8
A50Debug(121): Code	id=901824512 "\u0089PNG\r\n\u001A\n\u0000'	byte[14309]	14312
A50Debug(121): Code	id=901840896 "uk.png"	java.lang.String	8
A50Debug(121): Code	id=901849088 "\u0089PNG\r\n\u001A\n\u0000'	byte[20914]	20916
A50Debug(121): Code	id=901857280 "canada.png"	java.lang.String	12
A50Debug(121): Code	id=901865472 "\u0089PNG\r\n\u001A\n\u0000'	byte[7838]	7840
A50Debug(121): Code	id=901873664 "\u0000\t<unknown>"	byte[11]	12
A50Debug(121): Code	id=901881856 "\u0000\u00030.0"	byte[5]	8
A50Debug(121): Code	id=901890048 "\u0000"	byte[1]	4
A50Debug(121): Code	id=901898240 ".png\n"	byte[5]	8
A50Debug(121): Code	id=901939200 "\u0001"	byte[1]	4
A50Debug(121): Code	id=902029312 "RIMResources"	java.lang.String	16
A50Debug(121): Code	id=902045696 "com.rim.resources."	java.lang.String	20

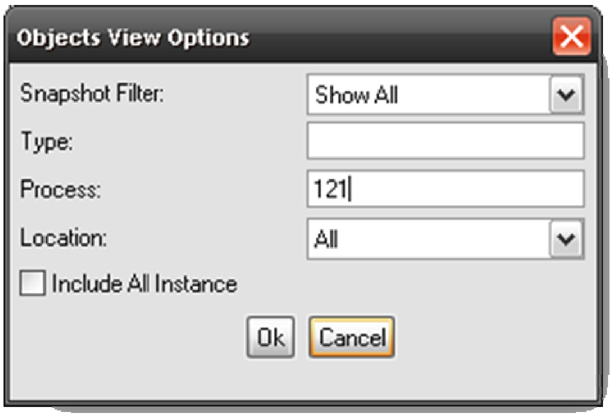
Figure 8

The object view shows you ALL objects. So you will not see only the objects from your application, you will be able to see all objects from all running applications.

The Save option will save the data in .csv format. Snapshot is to be used to compare the objects between two breakpoints. This is very useful to find memory leaks.

You can also run Garbage Collection to see whether your object references are properly de-allocated.

You can also use the Filter to limit the Object View to your process. In this case our process number is 121 – the number in the brackets. After clicking on Filter (Figure 9) enter process number and press OK. You will have to refresh the main window to see the update.

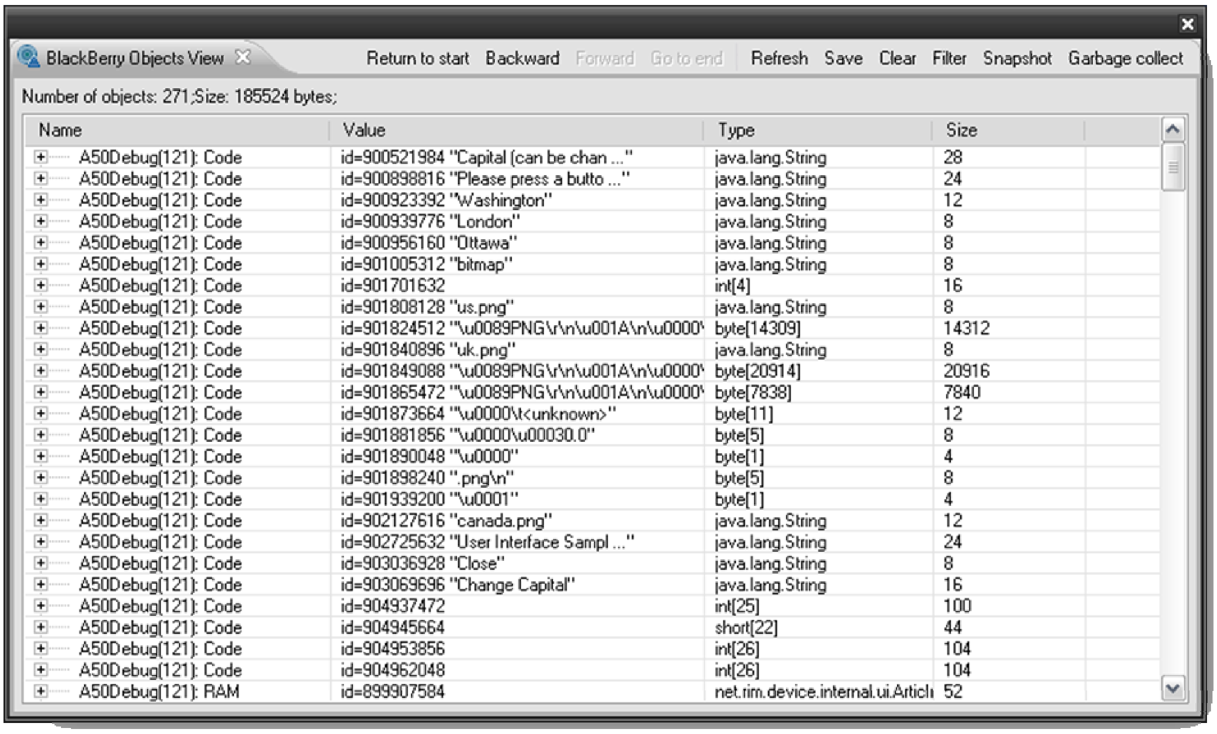


Snapshot Filter:	Show All
Type:	
Process:	121
Location:	All

☐ Include All Instance

Figure 9

As you can see now we have the list of all the objects in our application and we can monitor and analyze the memory usage (Figure 10).



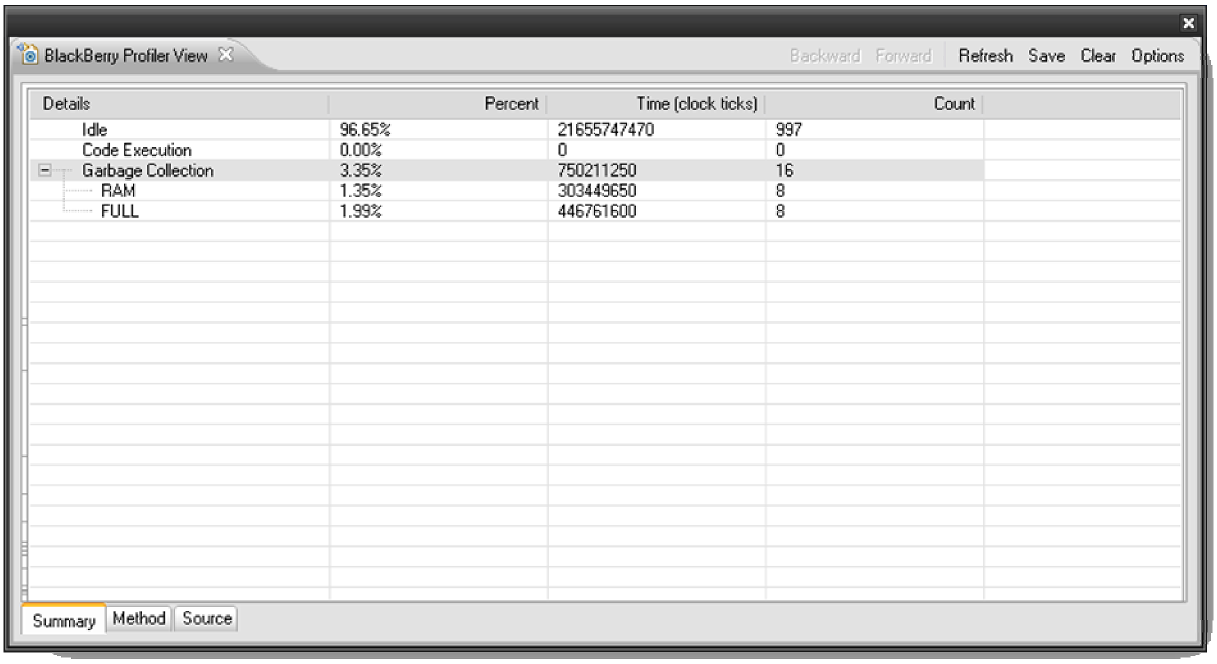
Number of objects: 271;Size: 185524 bytes;

Name	Value	Type	Size
A50Debug(121): Code	id=900521984 "Capital (can be chan ..."	java.lang.String	28
A50Debug(121): Code	id=900898816 "Please press a butto ..."	java.lang.String	24
A50Debug(121): Code	id=900923392 "Washington"	java.lang.String	12
A50Debug(121): Code	id=900939776 "London"	java.lang.String	8
A50Debug(121): Code	id=900956160 "Ottawa"	java.lang.String	8
A50Debug(121): Code	id=901005312 "bitmap"	java.lang.String	8
A50Debug(121): Code	id=901701632	int[4]	16
A50Debug(121): Code	id=901808128 "us.png"	java.lang.String	8
A50Debug(121): Code	id=901824512 "\u0089PNG\r\n\u001A\n\u0000	byte[14309]	14312
A50Debug(121): Code	id=901840896 "uk.png"	java.lang.String	8
A50Debug(121): Code	id=901849088 "\u0089PNG\r\n\u001A\n\u0000	byte[20914]	20916
A50Debug(121): Code	id=901865472 "\u0089PNG\r\n\u001A\n\u0000	byte[7838]	7840
A50Debug(121): Code	id=901873664 "\u0000\t<unknown>"	byte[11]	12
A50Debug(121): Code	id=901881856 "\u0000\u00030.0"	byte[5]	8
A50Debug(121): Code	id=901890048 "\u0000"	byte[1]	4
A50Debug(121): Code	id=901898240 ".png\n"	byte[5]	8
A50Debug(121): Code	id=901939200 "\u0001"	byte[1]	4
A50Debug(121): Code	id=902127616 "canada.png"	java.lang.String	12
A50Debug(121): Code	id=902725632 "User Interface Sampl ..."	java.lang.String	24
A50Debug(121): Code	id=903036928 "Close"	java.lang.String	8
A50Debug(121): Code	id=903069696 "Change Capital"	java.lang.String	16
A50Debug(121): Code	id=904937472	int[25]	100
A50Debug(121): Code	id=904945664	short[22]	44
A50Debug(121): Code	id=904953856	int[26]	104
A50Debug(121): Code	id=904962048	int[26]	104
A50Debug(121): RAM	id=899907584	net.rim.device.internal.ui.Articl	52

Figure 10

BlackBerry Profiler View

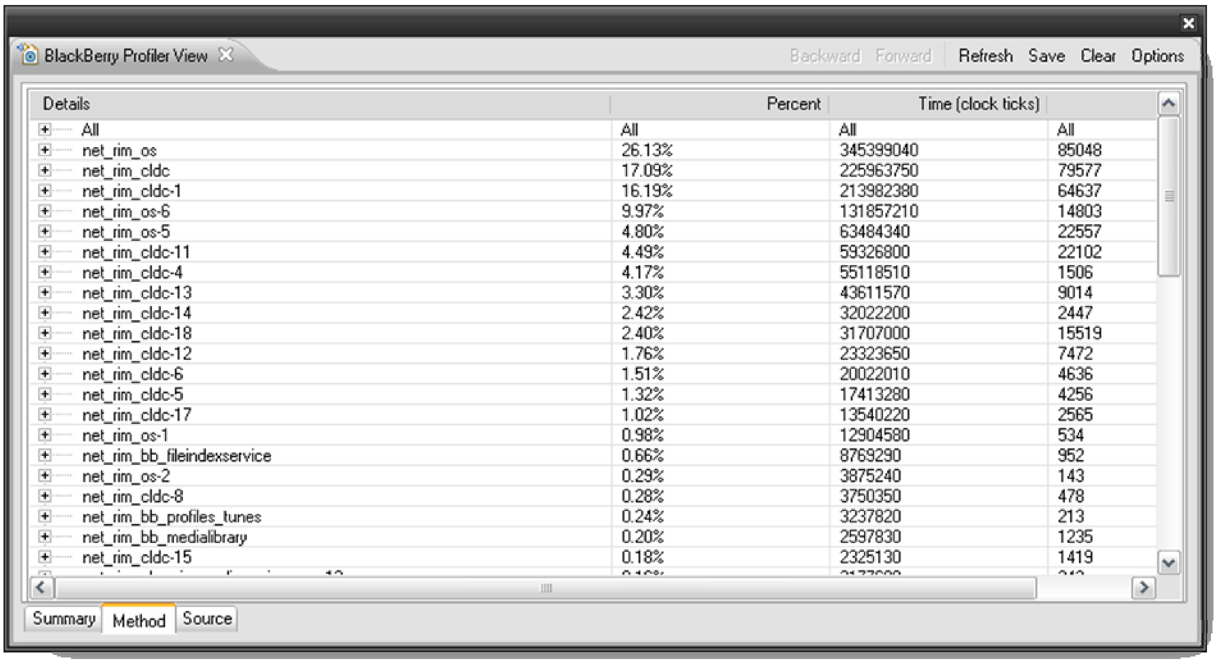
The Profiler is used to display information about where your application spends its time (Figure 11). The summary view displays percentage spent in Idle, Code Execution and Garbage Collection stages.



Details	Percent	Time (clock ticks)	Count
Idle	96.65%	21655747470	997
Code Execution	0.00%	0	0
Garbage Collection	3.35%	750211250	16
RAM	1.35%	303449650	8
FULL	1.99%	446761600	8

Figure 11

The Method view (Figure 12) will give you a detailed view of all running packages. Under options you can choose what to profile. Default settings are in ticks (time) but you can also view size and number of objects, etc.



Details	Percent	Time (clock ticks)	
All	All	All	All
net_rim_os	26.13%	345399040	85048
net_rim_cldc	17.09%	225963750	79577
net_rim_cldc-1	16.19%	213982380	64637
net_rim_os-6	9.97%	131857210	14803
net_rim_os-5	4.80%	63484340	22557
net_rim_cldc-11	4.49%	59326800	22102
net_rim_cldc-4	4.17%	55118510	1506
net_rim_cldc-13	3.30%	43611570	9014
net_rim_cldc-14	2.42%	32022200	2447
net_rim_cldc-18	2.40%	31707000	15519
net_rim_cldc-12	1.76%	23323650	7472
net_rim_cldc-6	1.51%	20022010	4636
net_rim_cldc-5	1.32%	17413280	4256
net_rim_cldc-17	1.02%	13540220	2565
net_rim_os-1	0.98%	12904580	534
net_rim_bb_fileindexservice	0.66%	8769290	952
net_rim_os-2	0.29%	3875240	143
net_rim_cldc-8	0.28%	3750350	478
net_rim_bb_profiles_tunes	0.24%	3237820	213
net_rim_bb_medialibrary	0.20%	2597830	1235
net_rim_cldc-15	0.18%	2325130	1419

Figure 12

One of the most common mistakes developers make is creating too many String objects.

```
String test = "a";
test = "b";
test += "c";
System.out.println(test.concat("d"));
```

Most of you know that the above code does have one variable test but we did not create only one String object.

Try to run the profiler to see how many String objects this code generates. You might be surprised.

Garbage Collection

BlackBerry is a purely Java based platform and it implements a system to periodically free its memory and resources called Garbage Collection (GC).

The idea is quite simple. All the objects which do not have a reference remain in memory until a GC runs and frees that memory for us.

So if we create a String object String test = “some text”; it will occupy some space in the memory. If we do not need that object, we can say test = null; however, the memory will still be occupied with “some text”. Only when a GC runs will that space in memory be freed.

There is a way to call GC to run manually, and it might seem like a good idea. However, on BlackBerry, it is highly recommended not to call GC manually, except in special cases.

When GC on BlackBerry runs nothing else runs. Nothing. You can see an hourglass on your screen. That hourglass only appears when GC is running.

GC type	Time
RAM only	0.5 sec
Flash + RAM (Full)	1 sec
Persistent	10 sec
Emergency	20+ sec
Thorough	25 – 30 sec

Table 1

In Table 1 we can see different GC types and the approximate time it takes for them to run on the device. As you can see, if the system runs any other GC than RAM or Full, it will have a significant impact on the user experience.

Even the very quick GC types will affect some applications that require fast responses like arcade games for example.

The BlackBerry heuristics monitor, checks for available free memory and runs GC as needed, and most of the time it runs quickly and not very often.

We need to keep in mind that other applications which run in the background can affect our application and our application can affect others.

Therefore it is important to reduce memory usage and GC usage.

Memory Leaks

Memory Leaks are created when we maintain a reference to the object which is not needed. Even if we design code which deletes these references we might still have a reference in the system that still prevents that object from being deleted.

Memory leaks can happen anywhere in the code but we can usually find them in our

- Data Structure,
- Local Variables,
- Runtime Store and
- Listeners

They are not easy to detect, but we can look for symptoms: Hourglass appears very often as the device is trying to do garbage collection. Emails will start being deleted. When the device is running out of memory it will try to notify all applications that use the low memory manager and ask them to free some space.

The email client will then delete some messages to try to create space.

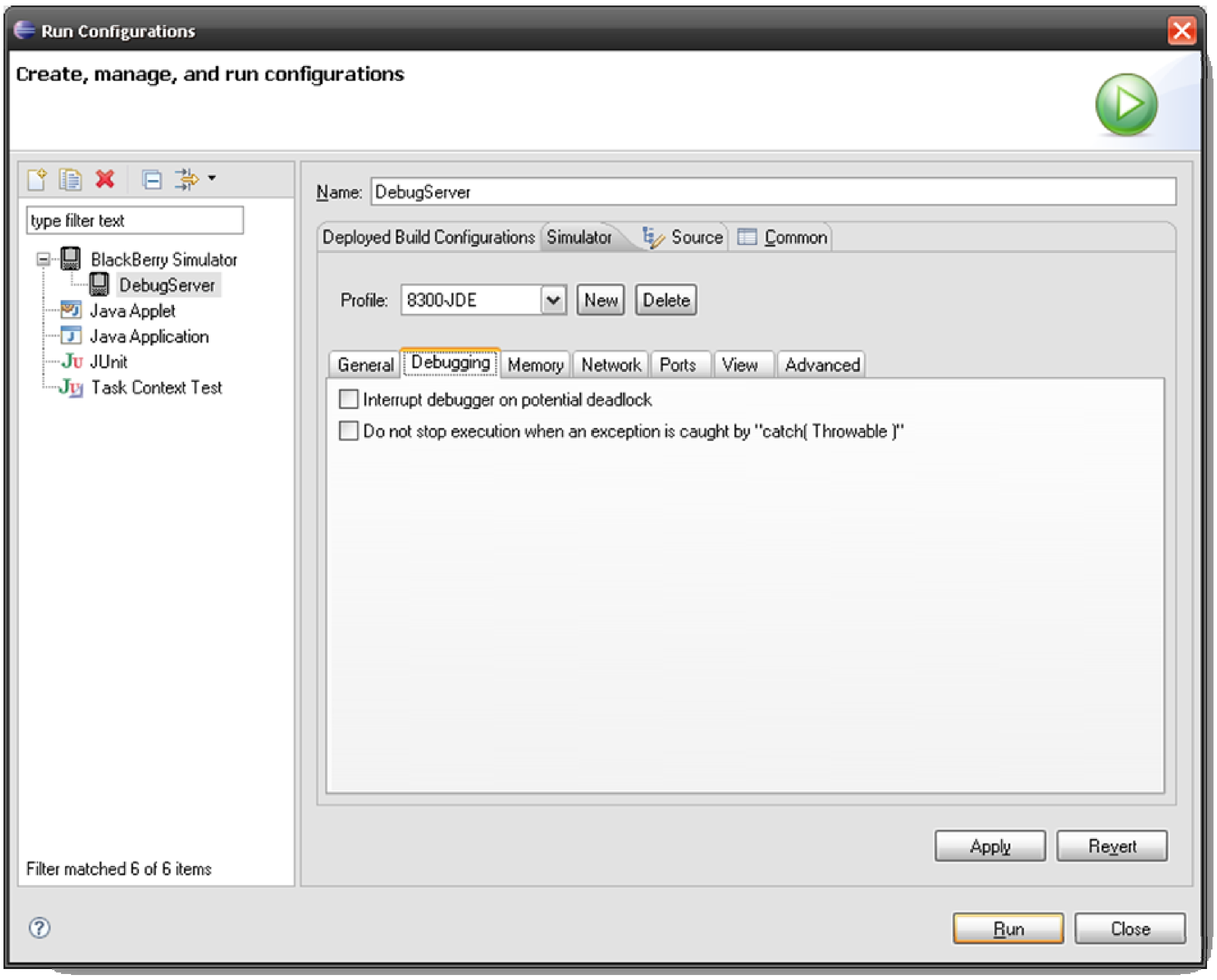
We can also see the number of free object handles if you go to Options / Status on your device. You will see the File Free number. If the number is low it may indicate a memory leak.

If you are finding it difficult to pinpoint the memory leak in your application, try to make it worse. The more data the application uses, the more it will leak and it should be easier to detect.

Deadlocks

Deadlocks happen if you have two or more threads waiting for each other, and therefore the application gets blocked forever. The JVM will detect that the application is not responsive and will terminate it after some time.

On the BlackBerry it is quite easy to detect and prevent deadlocks.



Name: DebugServer

Profile: 8300-JDE

- [ ] Interrupt debugger on potential deadlock
- [ ] Do not stop execution when an exception is caught by "catch(Throwable)"

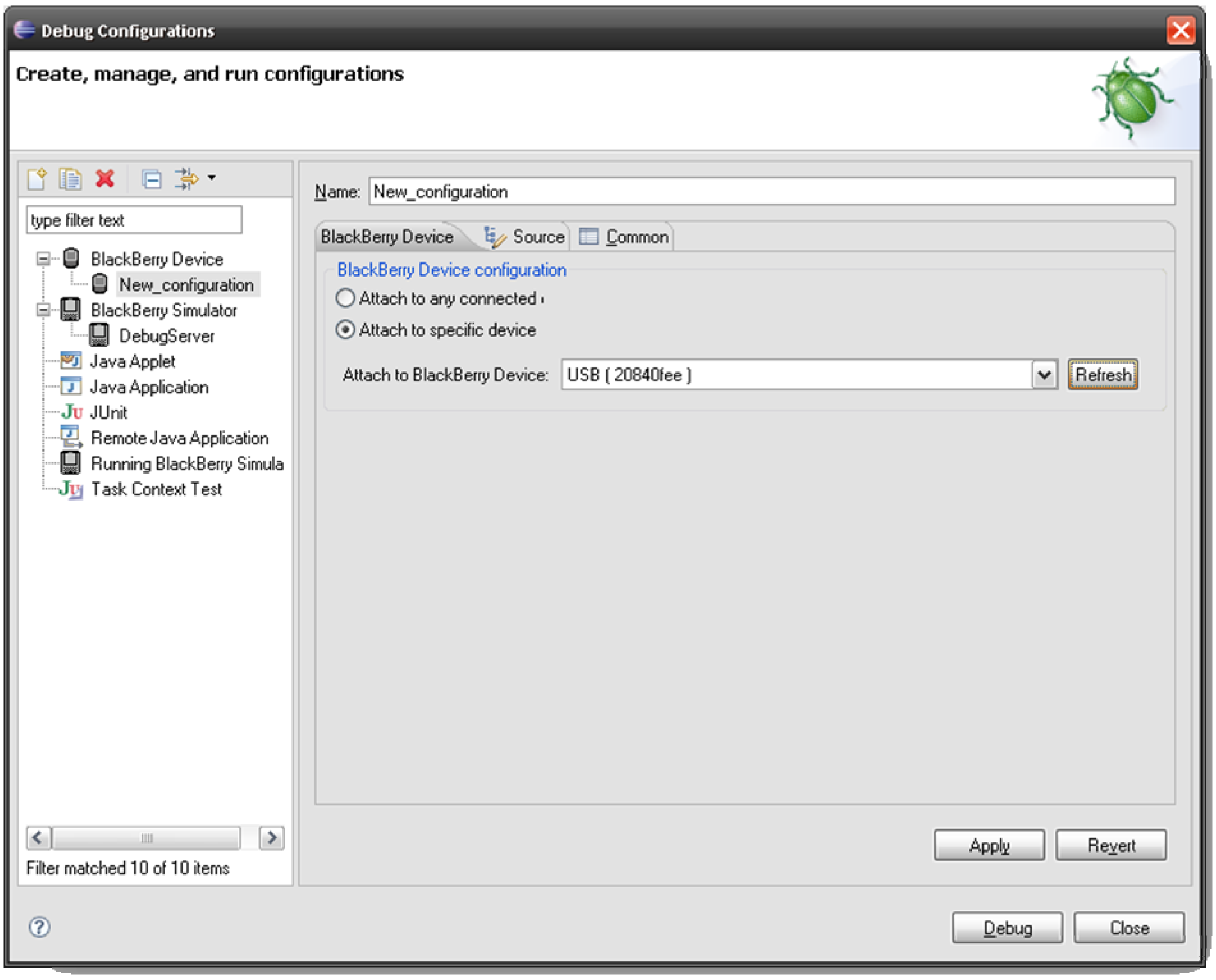
Figure 13

Click on Run in the Menu and select Debug configurations. Under the Debugging tab (Figure 13) you can check the box named ‘Interrupt debugger on potential deadlock’.

If the simulator detects a deadlock or a potential deadlock it will stop the execution of the application and will print the details in the Console window.

Debugging on the Device

It is also possible to debug your application on the device. Simulators emulate the physical device quite well but some functions need to be tested on the real device under real world conditions.


Debug Configurations

Create, manage, and run configurations

type filter text

- BlackBerry Device
  - New_configuration
- BlackBerry Simulator
  - DebugServer
- Java Applet
- Java Application
- JUnit
- Remote Java Application
- Running BlackBerry Simula
- Task Context Test

Filter matched 10 of 10 items

Name: New_configuration

BlackBerry Device | Source | Common

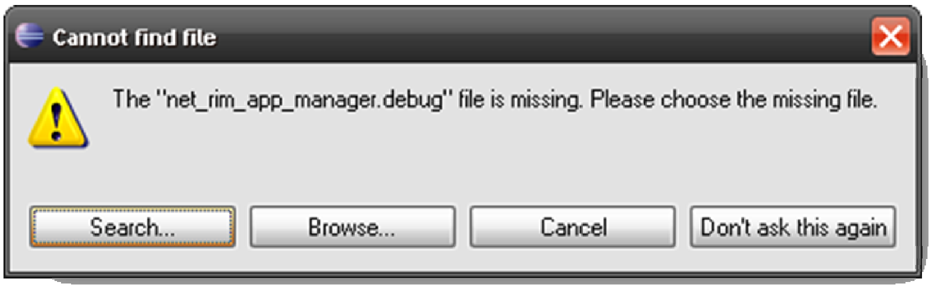
BlackBerry Device configuration

( ) Attach to any connected
(•) Attach to specific device

Attach to BlackBerry Device: USB (20840fee)

Figure 14

Under Run / Debug Configurations you can click on BlackBerry Device and add new configuration. There you can choose to attach the debugger to the specific device. After selecting the device and clicking on the Debug window you might be asked for the device password assuming you have one set.



The "net_rim_app_manager.debug" file is missing. Please choose the missing file.

Figure 15

You will probably get the message that one or more .debug files are missing (Figure 15).

The reason this might happen is that the software version on your device does not match the software version of your simulator. You can download the right simulator on our Developer Download web site:

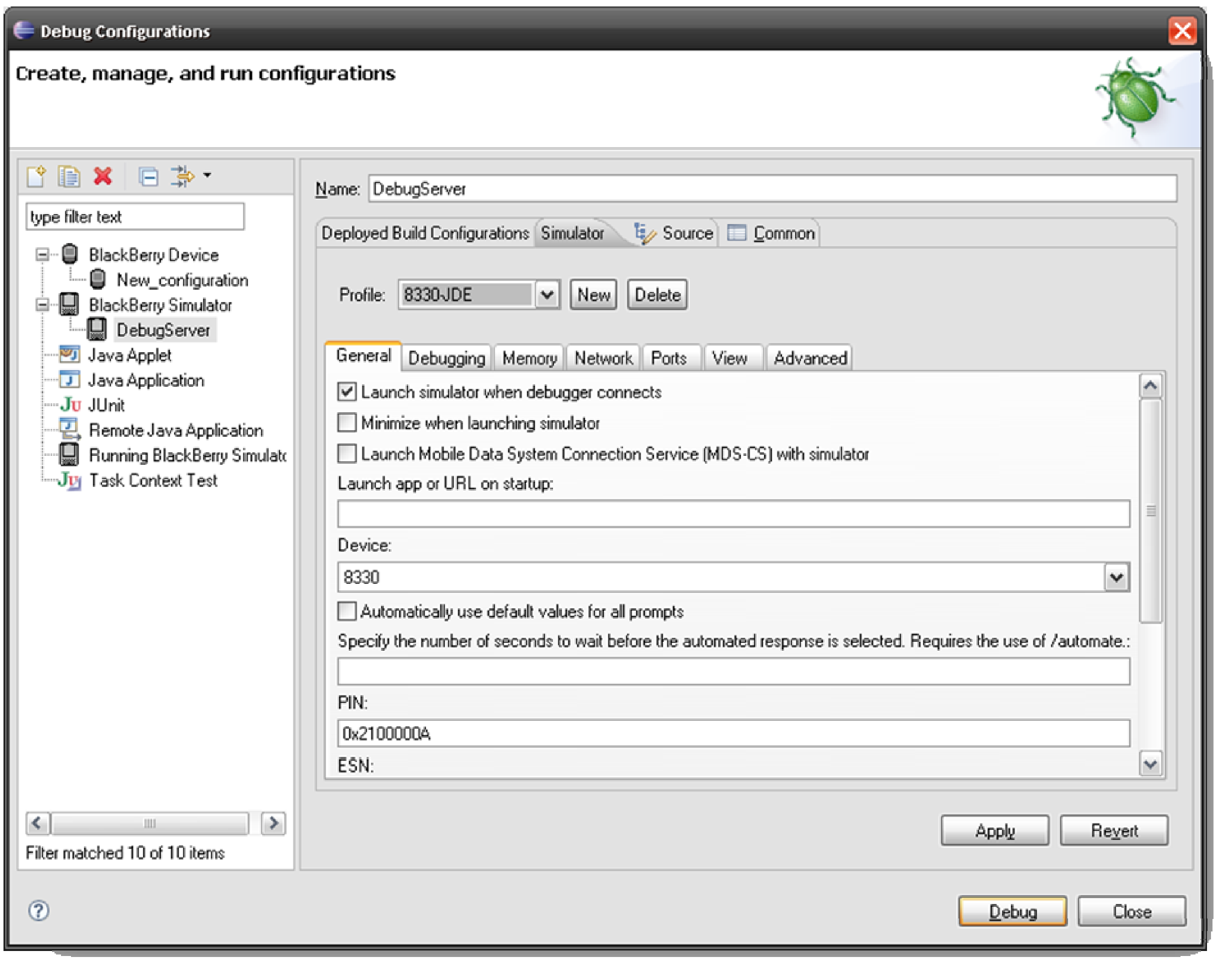
> https://www.blackberry.com/Downloads/browseSoftware.do;jsessionid=CJsgqMfoukmF9f3rzxLShA**

If the simulator is not available yet for your version of device software then you can upgrade or downgrade the software on your device to a version that matches one of the simulator versions.

You can also ignore the message, just click on 'Don't ask this again' button. Some of the debugging features will not work, for example getting stack traces, but most of the features described above will still work.

Setting up the Simulator

If you want to set up the simulator to work under specific conditions, or simulate network /file connectivity (MDS-CS, SD card, etc.) you need to change the Simulator preferences (Figure 16).


Debug Configurations

Create, manage, and run configurations

type filter text

- BlackBerry Device
  - New_configuration
- BlackBerry Simulator
  - DebugServer
- Java Applet
- Java Application
- JUnit
- Remote Java Application
- Running BlackBerry Simulatc
- Task Context Test

Filter matched 10 of 10 items

Name: DebugServer

Deployed Build Configurations | Simulator | Source | Common

Profile: 8330-JDE | New | Delete

General | Debugging | Memory | Network | Ports | View | Advanced

- [✓] Launch simulator when debugger connects
- [ ] Minimize when launching simulator
- [ ] Launch Mobile Data System Connection Service (MDS-CS) with simulator

Launch app or URL on startup:

Device: 8330

- [ ] Automatically use default values for all prompts

Specify the number of seconds to wait before the automated response is selected. Requires the use of /automate.:

PIN: 0x2100000A

ESN:

Apply | Revert

Debug | Close

Figure 16

Here is a brief overview of some of the options:

Name	Description
Launch MDS-CS	To be used when needed to simulate MDS_CS Internet connection
Device	Which device to use in the simulator
PIN	Change device's PIN number
Automatically use default values for all prompts	Skips through prompts which usually appear when you first switch on the device
System and Keyboard Locale	If you want to test i18n
Interrupt on potential	Helps detecting deadlocks

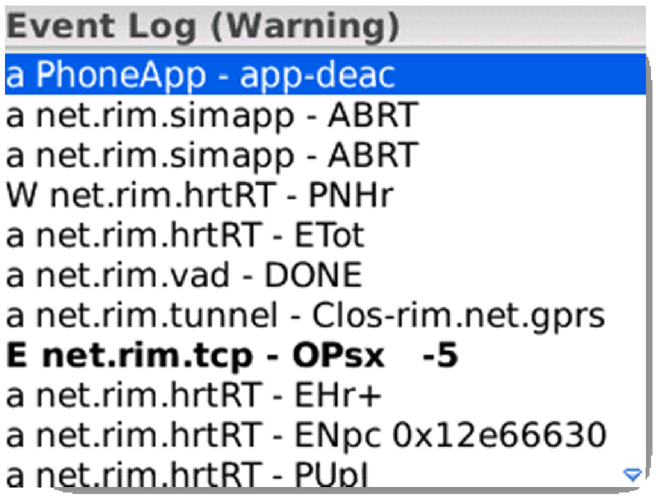
deadlock	
Reset file system and NVRAM	Clears file system and NVRAM
SD Card options	Various options to simulate whether SD card inserted, ejected etc.
Network registration	Skip network registration emulation
Phone, IMEI and other numbers	Specify emulator phone and other numbers
Ports	USB cable connected – Simulates connection to the PC
Disable backlight shutoff	Quite a useful function
Display the LCD only	Hides the keyboard part of the simulator

You can save these configurations and use them when needed.

Event Log

The Event log is the log of system events and we can write to it using the available APIs. It is a useful tool to debug your application. You can access it on the device or copy it to your PC using the application called javaloader.

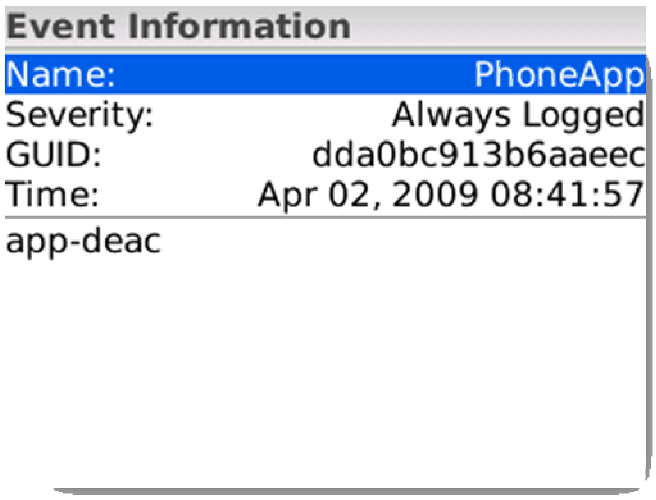
To access the event log on the device (and on the simulator), from the home screen, hold down the Alt key and while holding type lglg. You will then see on your display a list of all logged events (Figure 17).



```
Event Log (Warning)
a PhoneApp - app-deac
a net.rim.simapp - ABRT
a net.rim.simapp - ABRT
W net.rim.hrtRT - PNHr
a net.rim.hrtRT - ETot
a net.rim.vad - DONE
a net.rim.tunnel - Clos-rim.net.gprs
E net.rim.tcp - OPsx   -5
a net.rim.hrtRT - EHr+
a net.rim.hrtRT - ENpc 0x12e66630
a net.rim.hrtRT - PUpl
```

Figure 17

If you press enter you can see more details about every item in the log (Figure 18).


Event Information

```
Name:                 PhoneApp
Severity:        Always Logged
GUID:            dda0bc913b6aaeec
Time:    Apr 02, 2009 08:41:57
app-deac
```

Figure 18

To download the event log from the device to the PC type in the command prompt:

```
javaloader –u eventlog log.txt
```

The API which allows your application to write to the event log is:

`net.rim.device.api.system.EventLogger`

Examples:

```
// Register application for event logging.
 EventLogger.register(0x9c805919833654d6L, SampleApp);

 // Set minimum logging level.
 EventLogger.setMinimumLEvel(EventLogger.INFORMATION);

 // Log a numeric event.
 EventLogger.logEvent(0x9c805919833654d6L, 12, EventLogger.INFORMATION);

 // Log a String
 EventLogger.logEvent( GUID, yourString.getBytes(), level );
```

For more information on how to use EventLogger please look at the BlackBerry Java API Reference.

Links

BlackBerry Developers Web Site:

http://na.blackberry.com/eng/developers/

Developer Video Library:

http://na.blackberry.com/eng/developers/resources/videolibrary.jsp

Documentation:

http://na.blackberry.com/eng/support/docs/developers/?userType=21

Knowledge Base Articles:

http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/customview.html?func=ll&objId=348583

Forums:

http://supportforums.blackberry.com/rim/?category.id=BlackBerryDevelopment