

What Java™ developers need to know about MIDP on Symbian OS

Author: Martin de Jode
Version: 1.0
Date: January 2005

1. Introduction

With support for J2ME™ CLDC/MIDP becoming ubiquitous on mobile phones Java is an increasingly attractive option for many wireless development projects. By adopting the Java route developers can simplify application porting between phones by leveraging cross platform standards in the form of the Java Programming Language and the standard APIs offered by CLDC and MIDP specifications.

Any implementation of a Java Specification Request (JSR) (for example CLDC 1.0 – JSR030, MIDP 2.0 – JSR118 etc) must pass a Technology Compatibility Kit (TCK) ensuring conformance to the specification defined by that JSR. However, this doesn't prevent different implementations of the same JSR differing in subtle ways. Furthermore, most JSRs define optional features that are not mandatory requirements for an implementation. In other words (with apologies to George Orwell) all MIDP implementations may be equal, but some are more equal than others!

Therefore, even when developing for the standard J2ME CLDC/MIDP platform there are still implementation specific features knowledge of which may prove beneficial for the developer; easing their task and allowing them to make more out of the platform. In this paper we present useful information for Java developers programming MIDP on Symbian OS, in the hope that this will enable them to write better applications, more quickly and exploit to the full the power of Java on Symbian OS.

We will not be concerned with programming APIs, as this information is presented elsewhere on [Symbian Developer Network](#) and other sites (See for instance [“Getting Started with MIDP on Symbian OS”](#)). Neither will this paper deal with defects - defects get fixed, but the underlying architecture is more stable. Instead, we will concentrate on implementation details and features that may be particular to Symbian OS and are relevant to Java Developers.

We will briefly look at:

- CLDC and VM implementation
- Features of MIDP
- Errors and debugging

2. Symbian OS Essentials

One of the attractions of Java development is that it hides the developer from the complexity of dealing directly with the OS and OS-specific issues. However, in this paper it is useful to introduce a few Symbian OS paradigms that are relevant to the following discussion. First Symbian OS is a fully multitasking OS, with support for multiple processes and multiple threads within those processes. One consequence for Java MIDlets is that there is no need for a MIDlet to be automatically paused when another application of higher priority moves into the foreground. Although Symbian OS is fully multitasking it takes a different approach to achieving this compared with some other well known OSes. Symbian OS makes extensive use of a client/server framework whereby the server makes services available to multiple clients. Rather than spawning multiple threads to service multiple clients a Symbian OS server typically has just one (event handling) thread running an active scheduler which schedules requests from one or more active objects. So Symbian OS servers use an event handling framework, rather than multiple threads, to serve multiple clients. For more information on the architecture of Symbian OS see ["Symbian OS C++ for Mobile Phones"](#).

3. CLDC and the Virtual Machine

3.1. CLDC HI

Since Version 7.0s Symbian OS has supported a port of Sun's CLDC HotSpot™ Implementation Virtual Machine (CLDC HI VM). CLDC HI is a high performance virtual machine (VM), delivering nearly an order of magnitude better performance compared with the standard KVM (see ["The CLDC HotSpot Implementation Virtual Machine"](#)). The CLDC HI VM uses various techniques to improve performance including

- Dynamic Adaptive Compilation
- Lightweight Threading System
- Generational Garbage Collection
- Fast Synchronisation
- Unified resource management

By way of comparison the standard KVM employs a conventional bytecode interpreter implemented in C, which allied to the slow clock speeds (compared to desktops or servers) provides typically an "adequate but not impressive" performance¹.

The CLDC HI VM supported by Symbian OS v7.0s implements the CLDC 1.0 specification (JSR030). Subsequent Symbian OS releases feature Sun's CLDC HI 1.1 VM implementing the CLDC 1.1 specification (JSR139) which amongst other things adds in support for floating points.

¹ "The CLDC HotSpot Implementation Virtual Machine" (www.java.sun.com). In fact Symbian OS has never supported a standard KVM, pre v7.0s releases provided a KVM with integration of ARM's VTK software acceleration, including the bytecode interpreter loop implemented in ARM assembler.

3.2. Implementation Details

VM is run as a (native) thread in its own process. In addition to the VM thread, Java components need to receive call backs from native code; this is achieved via a native Java Event Server. These Java components (or more correctly their native peers) are clients of an instance of the Java Event Server. The Java Event Server runs in a native thread external to the VM process, and makes use of the Symbian OS Client/Server framework to handle multiple client requests.

3.3. Java Threads

The Java Programming Language provides support for threads. How Java thread support is implemented is VM-specific: Java threads may be mapped to underlying OS threads or may be implemented by the VM (lightweight threads). The advantage of lightweight threads is that they provide greater platform independence since native threading models vary according to OS (for instance in terms of scheduling schemes and thread priority levels). The CLDC HI uses a lightweight threading model to support Java threads, so Java threads are not mapped directly to native Symbian OS threads.

3.4. No Limits

Unlike some MIDP implementations, the CLDC/MIDP implementation on Symbian OS imposes no restrictions on MIDlets, other than those imposed by available memory. In particular

- There are no limits on the number of RMS records or their size
- No limit is imposed on the size of the MIDlet JAR file that can be installed
- There is no limit to the number of Java threads that can be created
- There is no restriction on the number of socket connections that can be created.
- The CLDC HI VM running on Symbian OS supports a dynamic heap. In other words the size of the heap is not constrained (except by available memory) and can grow according to the demands made by an executing MIDlet suite. A Consequence of this is that the results returned by the `Runtime.getFreeMemory()` method can be confusing. The `getFreeMemory()` method only returns a snapshot of the free heap memory currently available, and becomes invalid after a subsequent allocation to the heap. The initial heap size for CLDC HI on Symbian OS is 400 kB, although this is customizable by the licensee.

4. MIDP

4.1. Implementation

Whereas the CLDC implementation running on Symbian OS is a port of Sun's CLDC HI VM, the MIDP environment is implemented by Symbian. By following this course Symbian is able to ensure optimum performance and the tight integration of MIDP with Symbian OS. One consequence is that MIDlets are treated as first class citizens, that is in most regards a MIDlet is indistinguishable from a native application. This is important consideration for a Symbian OS

phone because the user may switch between C++ applications and Java applications all the time, and it is important that a MIDP application looks, reacts, and accepts input in the same way as its native counterparts.

4.2. MIDP Lifecycle

The Mobile Information Device Profile (MIDP) specifies a lifecycle model for MIDP applications (MIDlets). Understanding and programming the MIDP lifecycle correctly is an important topic in its own right and is discussed in length in a separate White paper. However it is worth re-iterating a few important points:

- A MIDlet can be in any one of three states: Active, Paused, Destroyed.
- The Application Management System (AMS) is responsible for moving the MIDlet between states, notifying the MIDlet of the state change by calling one of the following methods of the abstract MIDlet class as appropriate.

```
startApp  
pauseApp  
destroyApp
```

However, the MIDlet may itself request a state change by invoking one of the following methods

```
notifyPaused  
resumeRequest  
notifyDestroyed
```

- On Symbian OS the behaviour of the AMS is customizable by the licensee. In particular the circumstances under which `pauseApp` is called vary between UI reference designs. On Series 60 `pauseApp` is not called when a MIDlet is backgrounded, whereas on UIQ phones the AMS should call `pauseApp` when the MIDlet moves to the background
- If the developer wishes to ensure the MIDlet moves into the Paused state in response to being backgrounded then the developer should use the `Displayable isShown` method and the `Canvas showNotify` and `hideNotify` methods to monitor the state of the display, and take appropriate action when the MIDlet is backgrounded.

For a more detailed discussion of the MIDP lifecycle on Symbian OS see the White Paper [“Programming the MIDP lifecycle on Symbian OS”](#).

4.3. LCDUI

In line with the philosophy of ensuring MIDlets are first class citizens, to all intents indistinguishable from native applications, the MIDP GUI toolkit, the LCDUI is tightly integrated with Symbian OS.

- LCDUI components are implemented using native widgets, with the javax.lcdui components mapped to the equivalent native peers, providing MIDlets with the look and feel of native applications.
- The implementation of javax.lcdui.Canvas is currently double buffered by default. So developers should not automatically provide their own implementation of double buffering. To facilitate portability of code to other platforms (which may not be double buffered) developers can (and should) use the isDoubleBuffered method of Canvas and provide the appropriate implementation.
- MIDP supports the native color depth of the underlying platform.

4.4. Networking

The MIDP 2.0 specification only mandates support for HTTP and HTTPS network protocols. The CLDC/MIDP 2.0 implementation running on Symbian OS (v7.0s and later) additionally supports the following:

- Sockets
- Secure sockets
- Server sockets
- UDP datagrams

Push registry protocols supported for the auto-launch of a MIDlet in response to an incoming connection are

- Server Sockets
- UDP datagrams
- SMS
- Bluetooth (added in v 8.0)

The powerful networking support provided by CLDC/MIDP on Symbian OS offers interesting possibilities. Not only does it open up a myriad of opportunities to communicate with remote hosts, it also allows MIDlets to communicate with local native applications using the localhost loopback address (127.0.0.1). For example, support for client sockets, allied to the full multi-tasking support offered by Symbian OS allows a MIDlet to open a two-way communication channel with a C++ application running on the same phone. Furthermore support for the registration of server socket connections with the Push Registry allows a locally running native application to auto-launch a MIDlet via a localhost socket connection. For more information on programming the Push Registry see "[Programming Java 2 Micro Edition on Symbian OS](#)".

5. Errors and Debugging

5.1. Stdout, Stderr and debugging

In common with many MIDP phones the standard output and error streams are not normally accessible on Symbian OS based MIDP phones.

In the absence of System.out and System.err on-target diagnostics on products based on pre Symbian OS v8.1 releases usually involves creating a debug version of the application in which output that would be normally written to System.out and System.err is written to a LCDUI Alert, Ticker or StringItem. Due to the limitations of on-target debugging it is wise to perform on phone testing early in the application development cycle.

Version 8.1 of Symbian OS brings in support for on target debugging of Java applications, via the Universal Emulator Interface. Output to System.out and System.err is directed to the standard output streams on the host PC.

5.2. Error Messages

CLDC/MIDP Java Runtime Environments are implemented employing a combination of Java and native code via the Java 2 Standard Edition and JNI².

When runtime errors/exceptions are encountered they may originate from Java or native code. Errors or exceptions occurring in Java will return standard Java errors or exceptions. However, often abnormal conditions will be encountered in native code. Symbian OS does not currently support C++ exceptions, instead native function calls return integer values to represent the success or otherwise of the operation; a successful outcome would typically result in a KErrNone being returned, whereas failure would be indicated by anyone of a number of error codes (eg KErrNoMemory).

When abnormal events occurring in native code, the implementation will attempt to handle the event gracefully. When a native method encounters an abnormal situation it will return an error code to the calling Java method. The Java method will then throw an appropriate Java exception, but append the Symbian OS error code that resulted in the exception. For instance the following exception is thrown when a HTTP connection is attempted to a non-existent URL

```
java.io.IOException: SymbianOS error = -5120
```

where -5120 indicates DNS name not found (for a list of Symbian OS error codes see [Resources](#)).

It is not always possible to handle abnormal conditions encountered in native code gracefully. Critical error conditions from which the system finds no way of recovering will result in a native thread panic, causing the associated process to terminate. For example the following has been

² Of course CLDC/MIDP does not provide a JNI API to the application developer, here we are talking about implementation details.

seen on v7.0 or v7.0s based phones such as the UIQ Sony Ericsson P900 or Series 60 Nokia 6600.

```
Program MontyThread -8  
Reason code KERN-EXEC  
Reason number 3
```

In this case the Panic originates from “MontyThread”³ which is the VM thread with identifier 8. The Panic has been detected by the Kernel Executive, which has terminated the associated process. The reason for the Panic is indicated by the number 3. In this example to quote the SDK documentation “This panic is raised when an unhandled exception occurs. Exceptions have many causes, but the most common are access violations caused, for example, by dereferencing a NULL pointer. Among other possible causes are: general protection faults, executing an invalid instruction, alignment checks, etc.”

Another panic occasionally seen on v7.0s based phones is the following

```
App. closed  
jes-8f-javax.microedition.lcdui1@1...
```

This cryptic message indicates the Java Event Server (jes) thread has been panicked. This instance of the Java Event Server is used to process callbacks to the gui toolkit (LCDUI). Again a common cause of this panic would be a null pointer exception. However, note here we are again talking about the panic arising in native code as a result of an unexpected null native pointer. The cause of this may be obscure and does not necessarily indicate a null Java reference in the Java application.

6. Summary

In this paper we have presented information useful to Java developers programming CLDC/MIDP on Symbian OS, including details about the VM, Symbian’s MIDP implementation, networking support and debugging. By taking heed of the particular characteristics of Symbian’s CLDC/MIDP implementation, developers should be able to write more effective code, more quickly and make the most of the opportunities afforded by Java on Symbian OS.

We have tried to present generic information, only detailing device-specific features where essential. For more information about known issues with particular devices the reader is referred to the relevant developer support organisation for that product (see [Resources](#)).

7. Resources

Symbian Developer Network
[Symbian’s developer support organisation](#)

“Getting Started with MIDP development on Symbian OS”

³ “Monty” being a codename for the CLDC HI VM.

[Symbian Developer Network White Paper](#)

“The CLDC HotSpot Implementation Virtual Machine”
[Sun White Paper](#)

“Symbian OS C++ for Mobile Phones”
[Richard Harrison, Wiley \(2003\) ISBN 0-470-85611-4](#)

“Programming the MIDP Lifecycle on Symbian OS”
[Symbian Developer Network White Paper](#)

“Programming Java 2 Micro Edition on Symbian OS”
[Martin de Jode, Wiley \(2004\) ISBN 0-470-09223-8](#)

Comprehensive list of Symbian OS Error Codes
[Symbian Developer Network](#)

Known issues in the Nokia 6600 MIDP 2.0 implementation
[Technical Note, Forum Nokia](#)

Java J2ME for P900 and P910 Series Mobile Phones
[Developer Guidelines, Sony Ericsson Developer World](#)

[Back to Developer Library](#)

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.