

MIDlet execution may sometimes appear to be slow in Series 40 devices. While mobile devices in general do not have infinite processing power, in Series 40 such problems can relate either to the way in which MIDlet code has been written or to the tasks which Java Virtual Machine performs during runtime. This article examines these causes and presents suggestions for reducing sluggish execution.

## Java Virtual Machine activity

To start with, it's good to keep in mind that certain functionalities in Series 40 platform take time during MIDlet start-up due to Java Virtual Machine:

? Security Checks ? the checksum and signature of applications need to be checked to determine what domain they are from and ensure they haven't been tampered with. This takes longer the larger the JAR file.

? Class loading - classes are stored in the JAR. Whenever a new class is referenced for the first time, the virtual machine must find that class in the JAR, unzip it into memory and initialize all the data relating to it. This can take time.

? Unnecessary method invocation - Breaking the code into timing methods is usual in Object Oriented software. This slows Java Virtual Machine because each method invocation requires the linker to find the class/method and jump to it.

## Code optimization: the right way with startApp()

In Series 40, it's important to pay attention to use startApp() method properly. All MIDlets start initially with a call to startApp() method which is called by the system thread. Any MIDlet which does not return from startApp() quickly will display the "Opening..." wait note for a long time even if the midlet calls setCurrent early in startApp(). This is because until startApp returns, the system thread is busy and the virtual machine cannot refresh the display.

The following code represents a poorly written MIDlet which displays a splash screen:

```
protected void startApp() throws MIDletStateChangeException {
    display=Display.getDisplay(this);
    display.setCurrent(mySplashScreen);
        .....
        Do a lot of stuff here.
        .....
}
```

The right way to write a MIDlet is to implement a thread to perform all the start-up so that startApp() returns quickly and the splash screen is displayed straight away.

```
public void startApp() {
    display = Display.getDisplay(this);
    display.setCurrent( mySplashScreen );
    System.out.println("Starting" + System.currentTimeMillis());
    new Start().start();
}
```

## Series\_40: Considerations regarding MIDlet startup and runtime execution

```
public class Start extends Thread {
    public void run() {
        .....
        Do a lot of stuff here.
        .....
    }
}
```

# Code optimization: general guidelines

Above we showed how to get the MIDlet to display the splash screen immediately after start-up quickly. With many MIDlets, there's a second problem; the splash screen is displayed for a long time. The way around this is to look carefully at your MIDlet and what it is doing. Add trace to see where the time is used, then optimize your code to run faster by doing the following:

- Reduce the number of classes and objects created (classloading takes time and so does instantiating objects).
- Avoid Object Oriented architectures which are too complex or deep.
- Strip unnecessary inheritance.

In addition, following optimization guidelines can be taken to speed up overall MIDlet execution:

- Obfuscation can reduce JAR size (and hence speed up loading a little)
- Some operations are very slow, and/or may result in threads being blocked - opening network connections, reading from files, creating multimedia resources, etc. Avoid performing such operations until required (i.e. just concentrate on getting to a point where the user can interact with the MIDlet!)
- Judicious use of multi-threading also improves performance - e.g. creating threads to update UI and to perform slow operations like network resource access or file parsing. As long as the UI thread is updating the screen, a MIDlet will be perceived as more responsive.
- If all else fails, a progress bar (or some similar distraction) is useful to at least keep the user informed about any expected delay.