

Java Micro Edition

A Micro Introduction

Johnny Tolliver

The name is *not* J2ME

- Since “Java 2” is no more, the official name is *Java Micro Edition*, or *Java ME*, not J2ME
- But you still see the J2ME name from time to time
- J2ME was first released ~1999, along with Java 2 SE and EE versions
- The “2” was dropped with the release of “Java 5” in 2005.
- Java (2) ME has evolved continuously since 1999 as devices have improved

Java ME is real Java

- **Same object-oriented language, garbage collection, class structure, multi-threading, thread synchronization, exception handling, etc. as “big” Java**
- **Has java.lang, java.util, java.io, etc.**
- **But... some of the library APIs are different**
 - **Some significantly different**
 - **No Swing, for instance, no Collections framework, no assertions, no Generics or annotations, no reflection, no JNI**
 - **No floating point in early versions**
 - **Kind-of like JDK 1.3 minus Swing and Collections**

Two broad ME “configurations”

- **Java ME has two broad divisions: CDC and CLDC**
 - **CDC is Connected Device Configuration**
 - **CLDC is Connected Limited Device Configuration**
 - **The “Limited” means lesser hardware and potentially limited connectivity**
 - **Think “cell phone”**
 - **CDC is more appropriate to advanced PDAs, “smart” phones, set-top boxes**
 - **The line between CDC and CLDC is distinct, though the line between what devices can support each is blurring**
 - **CDC is a superset of CLDC, adding back much from big Java that was removed for CLDC (e.g., Collections Framework is there, but still no Swing)**

Configurations and Profiles

- A *configuration* provides basic application services, but not enough to write an app.
- On top of configurations are built *profiles* which provide more APIs for user interface, networking, persistent storage, etc.
- For CLDC there is the Mobile Information Device Profile (MIDP) and the (newer) Information Module Profile (IMP)
 - IMP is like MIDP minus the user interface
- MIDP applications are called “MIDlets”
- CDC has the foundation profile, personal basis profile, and the personal profile (these give PDA functionality and more)

MIDP 1.0 hardware guidelines

- **Screen size: 96x54x1**
- **Memory: 128 KB (available)**
- **Sound: none**
- **Power: battery (i.e., not necessarily always on)**
- **Connectivity: intermittent, limited bandwidth**

- **(MIDP 2.0 increased memory to 256 KB and added limited sound capabilities)**

Multiple versions of everything

- All of these have had numerous versions
 - CLDC has 1.0 and 1.1
 - MIDP has 1.0, 2.0, 2.1 (3.0 is coming)
 - CDC has had a few iterations too (current version is 1.1.2, similar to J2SE 1.4.2 minus Swing)
- Plus there are various “optional packages” like the Bluetooth API, Location Services API, and Mobile Media (MMAPI), sometimes in various versions themselves
- Everything has a JSR number (82 for Bluetooth, 179 for Location, 139 for CLDC 1.1, etc.)
- As you can imagine the product matrix can be pretty confusing
- “Device Fragmentation”

CLDC 1.1 and MIDP 2.0 software additions

- **CLDC 1.1**
 - Floating point
 - Several classes look more like Java SE
- **MIDP 2.0**
 - Secure networking (HTTPS)
 - Sound API (audio only subset of the MMAPI)
 - GUI improvements, Game API, RGB images
 - Permissions and code signing
 - Push Registry
 - Record store sharing among MIDlets

JTWI tried to simplify the fragmentation problem

- **Java Technology for the Wireless Industry (ca. 2003) tried to unify this mess**
- **JTWI specified a minimum collection of the various pieces**
 - **CLDC 1.0 and MIDP 2.0**
 - **Wireless Messaging API 1.1**
 - **Plus “conditionally required” CLDC 1.1 and Mobile Media API 1.1**
- **I don’t think the handset manufacturers every really bought in to the concept**

So... introduce a new JSR...

- **JSR 248, aka the Mobile Service Architecture (MSA)**
- **A big collection of lots of Java ME JSRs**
- **MSA 1.0 released in December 2006**
- **This JSR had broad participation**
- **Might actually work**
 - **Alas, it also has required and optional parts**
 - **Plus, even if handset manufacturers build fully MSA-compliant devices, the carriers sometimes partially constrain the devices**

MSA Components

MSA:

- JSR 238 (Internationalization)
- JSR 234 (Multimedia Supplements)
- JSR 229 (Payment)
- JSR 211 (Content Handler)
- JSR 180 (SIP)
- JSR 179 (Location)
- JSR 177 (Security & Trust)
- JSR 172 (Web Services)
- JSR 226 (Vector Graphics)
- JSR 205 (Messaging)
- JSR 184 (3D Graphics)
- JSR 135 (Mobile Media)
- JSR 82 (Bluetooth)
- JSR 75 (File & PIM)
- JSR 118 (MIDP)
- JSR 139 (CLDC)

MSA Subset:

- JSR 226 (Vector Graphics)
- JSR 205 (Messaging)
- JSR 184 (3D Graphics)
- JSR 135 (Mobile Media)
- JSR 82 (Bluetooth)
- JSR 75 (File & PIM)
- JSR 118 (MIDP)
- JSR 139 (CLDC)

Beyond MSA

- **MSA 2.0**
- **JavaFX Mobile**
 - **Announced last week at JavaOne**
 - **Still not clear just what it is**
 - **Appears to be SavaJe rebranded and somehow integrated with JavaFX Script**
 - **That means most of Java SE, including Swing, on a handset**
 - **Not at all clear if handset manufacturers will actually support it**
 - **Can make a “jPhone”**

jPhone



Compared to iPhone



So how do you actually *do* this stuff?

- A *very* limited overview follows
- NetBeans has great mobile tools
 - Helps with fragmentation problem
- Manufacturers/carriers also offer their own SDKs (Nokia, Motorola, Sprint, RIM/BlackBerry)
- For Palm and Windows Mobile devices, you'll probably want IBM's J9 JVM and the IBM WebSphere IDE
- Some caveats:
 - I've only done CLDC/MIDP, no CDC
 - I've done only basic GUIs, no games
 - What I *have* done includes Location API, Push Registry, multi-threading, networking (HTTP, UDP, sockets), serial I/O, some Bluetooth

Getting started

- A MIDlet class must *extend MIDlet*
- MIDlets don't have a `main()` method
- They get started and stopped by the underlying OS's JAM (Java Application Manager) or AMS (Application Management Software)
 - AMS calls *startApp()*, *pauseApp()*, and *destroyApp()* whenever it feels like it
 - Generally when the user presses buttons on the phone to start or stop the app
 - Or when the phone rings
 - Or when you close the flip...
 - Somewhat like Java applets running in a browser with *init()* or servlets running in an application server

MIDlet UI

- The *Display* class keeps track of what's shown on the device's screen
- You can get an instance of *Display* and change what's shown on screen by setting the current *Displayable*
 - Actually a concrete subclass like *Form*, *TextBox*, *List*, ...
- Only one *Displayable* shows at a time
- You'll probably also want your MIDlet class to implement *CommandListener*
 - Create *javax.microedition.lcdui.Command* objects, add them to a *Displayable*, and set the *Displayable*'s command listener to *this*
- Your app's UI is a series of *Displayable* objects, shown one at a time
 - There are also Alerts and Tickers and titles

Persistent Storage

- ***Small* databases called “record stores”**
- ***javax.microedition.rms***
- **Store records of byte arrays, indexed by ID**
- **addRecord(), getRecord(), deleteRecord(), setRecord() methods**
- **Persistent across MIDlet runs**
- **Can be shared between MIDlets (in MIDP 2.0)**
- **And did I say *small*? 8KB**

Generic Connection Framework

- All I/O is through the GCF
- Call the static `Connector.open()` with a connection parameter (like a URL) and get back a `Connection` subclass depending on the parameter
 - `Connector.open (“http://some.address”)` returns an `HttpConnection`
 - `Connector.open (“socket://some.address:port”)` returns a `SocketConnection`
 - `Connector.open (“socket://:port”)` returns a `ServerSocketConnection`
 - `Connector.open (“comm:COM0;baudrate=19200;<options>”)` returns a `CommConnection` (for serial I/O)
- GCF is really neat
 - At one time there was JSR 197 to add GCF to “big” Java
 - Completed ~2003, never implemented in any Java SE release

That's all, folks!