

# J2SE vs J2ME for Embedded Systems

## Some History

In 1998, Sun Microsystems announced two specifications intended to support the use of the Java platform in embedded systems. The PersonalJava specification defined a standardized subset of the then-current Java 1.1 Virtual Machine (VM) and libraries considered useful for embedded devices. Even though it was a subset, most of the basic networking, file I/O, and graphics functions available at the time in desktop Java were included. Sun promoted PersonalJava for use in networked consumer appliances such as web-phones, set-top boxes, and handheld computing devices.

The second wasn't so much a specification as it was a licensing option. EmbeddedJava allowed a Sun licensee to define a custom subset of the Java libraries suitable for their application requirements to reduce memory usage below that which PersonalJava required. EmbeddedJava licensees were not permitted to offer a Java programming environment to their customers or other third parties because the custom subset would have been non-standard. Sun did not want to break the promise to developers of Java's Write Once Run Anywhere (WORA) philosophy.

At the JavaOne conference in June 1999, Sun announced the Java 2 Micro Edition (J2ME) platform, beginning with the K Virtual Machine (KVM) that could run simple Java applications in 128 Kbytes of RAM. Sun worked with 3Com to sell discounted Palm V organizers loaded with the KVM to conference attendees. Over the next three years, a set of specifications for configurations and profiles to accommodate different memory footprints and features emerged to round out the definition of J2ME. Although Sun didn't say so outright, it was clear that J2ME would replace PersonalJava and EmbeddedJava as the standards available to Sun licensees. Today, PersonalJava and EmbeddedJava have reached EOL (End of Life) status at Sun and are no longer supported.

## J2ME

Two baseline configurations are defined in J2ME today: The Connected Limited Device Configuration (CLDC) runs on devices with as little as 160 Kbytes of memory. The Connected Device Configuration (CDC) runs on devices with at least two Mbytes of memory. By comparison, the current Java 2 Standard Edition (J2SE) version 1.4.2 for Windows desktop systems requires at least 32 Mbytes of physical RAM. To achieve such a radical reduction in memory compared to desktop Java, CLDC removed many of the Java library packages, the Just In Time (JIT) compiler, and core capabilities such as floating-point data types, object finalization, Java Native Method Interface (JNI), and debugging support. Some of those features are added back in CDC, but graphics, Remote Method Invocation (RMI) and Java Database Connectivity (JDBC) require additional profiles or optional packages on top of CDC. The additions are based on subsets of the J2SE 1.3 application programming interfaces (APIs). The J2SE HotSpot JIT compiler is not available, although Sun offers improved byte code optimization in recent versions of J2ME. For the most part, if you wrote your application to the old PersonalJava standard you should be able to adjust it to run under J2ME CDC plus the Foundation Profile (FP) for file I/O, networking, reflection, and utilities plus the Personal Profile (PP) for graphics.

## Embedded Markets

J2ME is an appropriate solution for certain memory-constrained devices where processor performance is limited and embedded applications are not very complex. Most of the applications are single threaded and act in response to human input. An application is considered acceptable if a new screen is painted within a few hundred milliseconds after the user presses a button, although with typical J2ME solutions with unpredictable garbage collection, such response is not guaranteed.

Embedded systems in other markets such as Network Infrastructure, Industrial Automation, Military/Aerospace, and Information Automation have more complex and demanding applications and

higher performance requirements for response and predictability. For these applications, developers want the full power of J2SE and the ability to utilize the thousands of open-source and commercial middleware libraries for database access, XML, cryptography, web services, servlets, Java Management Extensions (JMX), logging, regular expressions, Open Services Gateway Initiative (OSGi), CORBA, Java Message Service (JMS), Java Naming and Directory Interface (JNDI), and Secure Socket Layer (SSL), just to name a few. Table 1 lists some open source and freely distributable J2SE-based Java components.

Name	Description	URL
Xerces2	XML Parser	<a href="http://xml.apache.org/">http://xml.apache.org/</a>
Xalan-Java	XSLT Processor	<a href="http://xml.apache.org/">http://xml.apache.org/</a>
WebServices SOAP	SOAP v1.1	<a href="http://ws.apache.org/soap/index.html">http://ws.apache.org/soap/index.html</a>
Tomcat	Java Servlet and JavaServer Pages	<a href="http://jakarta.apache.org/">http://jakarta.apache.org/</a>
Jetty		
Collopy		<a href="#">ons/</a>
Oper		
Osc		
JSSE		
JNDI		
JCE		
Cryp		
Cryp		
Oper		
WBE		
Servi		
mx4j		
Hype		
SQL		
Hibe		
Rhin		
JacO		
Com		
Oper		
JXTA		
Javal		
Spee		
JDBC		<a href="#">ivers</a>

## An Embedded J2SE Example

Many of these J2SE-based software components are useful in the embedded Java domain, especially as embedded systems begin communicating and collaborating with desktop and enterprise systems. For example, Web Based Enterprise Management (WBEM) Services is a great technology for developing distributed management applications spanning enterprise, desktop and embedded devices. The Desktop Management Task Force (DTMF) and Storage Networking Industry Association (SNIA) have worked together for the past two years to combine DTMF's Common Information Model (CIM) and WBEM Services to manage storage networking systems such as Fibre Channel switches, disk arrays, and tape libraries. Unified management of all the resources in a Storage Area Network is critically important to IT managers and the first products conforming to the new Storage Management Initiative Specification (SMI-S) are now shipping. The Java-based SMI-S reference implementation requires J2SE. Some storage

vendors run a proxy on a J2SE desktop or server machine to translate between the SMI-S standard and device-level management protocols. This is clearly less efficient than running SMI-S directly on the managed devices.

## Purpose-Built Embedded Java

At this point you might be asking yourself, “Why doesn’t Sun offer an embedded J2SE?” Sun’s own implementation of J2SE was not designed for embedded systems. It was designed to run on desktops and servers with lots of processor power and large amounts of memory and disk. In contrast, complex embedded systems need a purpose-built Java with:

- J2SE compatible libraries so embedded developers can use the Java component-ware available for the desktop and enterprise.
- Configuration options to select tradeoffs in startup time, performance, and memory footprint by

J2MI  
code  
Netw  
comr  
need  
solut

### Cle

When  
sourc  
by no  
IBM  
Outp  
comp  
neve  
speci.  
without

PERC<sup>®</sup> is a purpose-built clean room embedded Java VM and libraries from Aonix, Inc. Here are some key features:

- PERC is compatible with J2SE. This lets embedded developers use XML parsers like Xerces and Servlet containers like Tomcat from The Apache Software Foundation. It works with the Java Management Extension (JMX), Java Cryptography Extension (JCE) and Java Secure Socket Extension (JSSE) reference implementations from Sun. It works with Java Database Connectivity (JDBC) drivers for Oracle, SQL Server, DB2, and many others. It also runs the SMI-S reference implementation mentioned earlier in this article.
- PERC lets developers mix and match static linking and dynamic loading of Java application classes, as well as execution of interpreted, JIT-compiled, and AOT-compiled code in the same VM to get optimal tradeoffs between startup time, execution performance and memory footprint.

The JIT and AOT compilers provide execution speeds up to 20 times the speed of interpreted code.

- PERC runs on VxWorks, Linux, Windows, OSE, QNX, and LynxOS. It supports PowerPC, Intel x86, XScale, and ARM processors. The PERC porting layer provides a clean abstraction between the VM and the underlying RTOS, making new ports easy to implement and stabilize. Only a small amount of C code differs from one RTOS port to another.
- Thanks to its patented real-time garbage collector (GC), the PERC virtual machine supports predictable real-time behavior. In typical configurations, the garbage collector is preemptible within 100 microseconds and configured to defragment the allocation pool over a period of a few seconds. For advanced GC control, a pacing agent can monitor memory allocation behavior and automatically adjust timing and ratios of CPU-time increments allocated to GC.
- PERC includes support for industry leading Java IDEs and debuggers using the Java Debug Wire Protocol (JDWP). PERC is bundled with useful plugins for the popular open-source Eclipse IDE

Many  
J2SE  
comp  
board  
PERC  
embe

tion.  
+  
n  
he  
iC,  
line  
asses  
le  
er-  
ded  
a 1.0  
ks.