Developing Wireless Applications using the Java™ 2 Platform, Micro Edition

Bill Day
Technology Evangelist
Sun Microsystems, Inc.
www.billday.com

Java™ technology has come a long way. Originally conceived for digital TV development, it moved onto our desktops and Webtops, then into our servers, and now, coming full circle, back into devices. Java technology will be nowhere more prevalent in the coming months and years than in the small wireless gadgets we all carry in our pockets and wear on our belts, our ever present companions, our mobile phones, PDAs, and handhelds.

I am writing this to help people new to the possibilities of wireless development using Java technology. I will discuss the Java 2 Platform, Micro Edition (J2ME™ Platform). I will outline where we are today with J2ME, and discuss where the industry and wireless community are headed. If you have lingering questions about J2ME and how it relates to other wireless technologies, this article should help to clear things up.

I discuss the fundamentals of Java technology for mobile devices and wireless application development. I give an overview of the Java 2 Platform, Micro Edition (J2ME), and discuss J2ME Configurations and Profiles used to develop Java applications for mobile phones, PDAs, and two-way pagers. Hopefully, this paper will give you an overview of the technologies that will help you to get started developing wireless applications using J2ME.

I also address some of the business issues in the emerging world of wireless data and how technical and business issues intermingle as developers, service providers, and operators around the world build out next generation mobile and wireless networks. Whatever your background or interest in wireless Java technology, by the end, you should be ready, willing, and able to get started using J2ME today.

The Network is the Computer™

Computing is ubiquitous. We still have one-computer-to-many-people (mainframes) and one-person-to-one-computer (PCs) scenarios deployed in organizations around the world today, and we will for many years to come. Increasingly, however, the most interesting applications and devices revolve around one-person-to-many-computers models. These many computing devices— be they what you carry, wear, drive, or have sitting on your kitchen counter at home— will be ever present, ready to work for you, and networked to the world beyond.

What devices are interesting for programmers and engineers today? I'd argue that all of the most interesting devices marry *computing+networking.*

A mobile phone's value comes in its inherent wireless networking capabilities. Likewise, the great thing about Palm PDAs that led to their rapid acceptance and financial success is the fact that you can synchronize ("network", in a sense) with your laptop or PC. Even PCs themselves were constrained to the offices of the few when they only represented a source of computing power: The arrival of Internet and Web networking on the PC scene drove PCs out from the offices of technical and

business people and into everyday settings such as our homes, our childrens' schools, and our community organizations.
Sun has been saying for years now that networking and computing power realize their true potential only when they are seemlessly married together. As the world continues to standardize around Java technology for portable applications, XML-based markups for portable data, and standard Internet Protocol (IP)-based protocols for interoperable and scalable networking, Sun's mantra, "The Network is the Computer™", has never been more true.

Java technology, XML, and IP work very well together, indeed. Java technology was created for ubiquitous, multi-device computing. The Java language gives programmers a consistent way to write code that can be executed over a wide range of devices. The Java language is a natural choice for XML and server-side development, and using Java technology for network programming continues to be the de facto choice for development projects the world over.

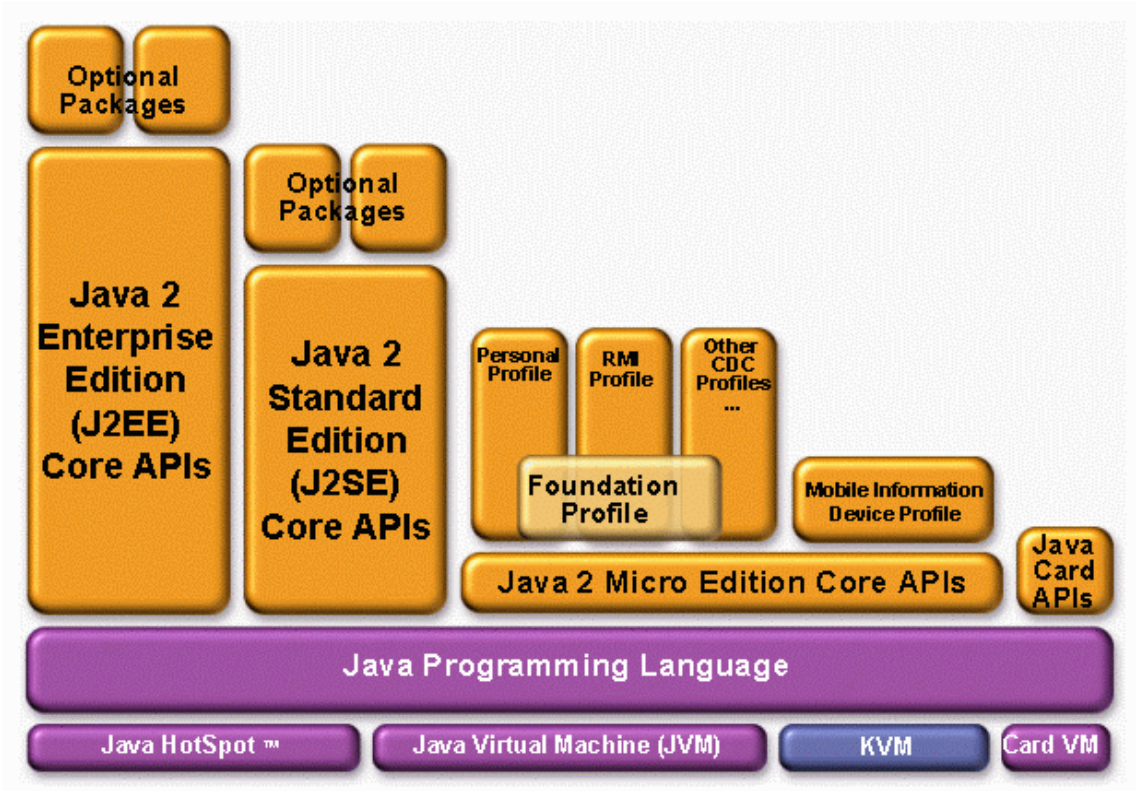
I will discuss how Java technology for mobile devices addresses ubiquitous computing in three sections in this article:

- First, I will discuss J2ME technology and how it fits within the Java platform overall.
- Next, I will discuss the markup technologies used in mobile devices (including but not limited to Java technology) to enable Internet access from them: WAP, XHTML, and CHTML, among others.
- The last part of the paper list resources for more in depth information.

One last thing before we start: a shameless plug. I maintain the *J2ME Archive* (http://www.billday.com/j2me) to help developers learn about J2ME. This archive is my attempt at a comprehensive resource for Java technology based wireless development. The J2ME Archive contains example applications and tools from people around the world. Most of the examples include source code and links to the authors' email addresses. If you are having trouble finding a link to more information on some particular part of the J2ME, I would encourage you to browse the J2ME Archive.

J2ME and The Java 2 Platform

You may have already seen a diagram something like this:



Several configurations and device profile specifications are now available, including our focus here, the J2ME CLDC and the Mobile Information Device Profile (MIDP).

There are some interesting development and packaging issues raised by the extremely small footprints (memory, CPU, etc.) of mobile and wireless devices. How do you fit useful applications into resource constrained devices? As a developer, you know that you can't write a full-blown desktop application and load it into a phone with a Java Virtual Machine and expect it to work — it just won't fit. Even if it *does* fit, it will be painfully slow, as you are using all sorts of things that aren't fast on these constrained mobile devices.

J2ME addresses this problem by providing for a standard subset of the Java 2 Platform. (When I say “Java 2 Platform” here, I am properly referring to the Java 2 Platform, Standard Edition (J2SE). The J2SE is basically a superset of the J2ME. Java technology for servers, the Java 2 Platform, Enterprise Edition (J2EE) is in turn a superset of the J2SE.) Portability of your application and service code across various J2ME-based devices is insured through the use of J2ME Configurations and Profiles.

What are Configurations and Profiles?

Fundamentally, a **configuration** is a specification. A configuration defines the minimum Java libraries and VM capabilities that a developer can expect to find on all mobile devices implementing the configuration specification.

A J2ME configuration defines a minimum Java platform for a family of devices. A configuration can cover a broad range of devices. A configuration specification is aimed at devices with similar requirements of memory size and processing power. To ensure portability across the device range, configurations cannot contain any optional features.

Configurations are defined through the Java Community Process initiative (JCP), an open, industry wide process used to develop standard Java APIs. The JCP insures that Java technology implementers (such as device manufacturers) produce

compatible Java platform implementations, and thereby, that your Java applications will be portable across all implementations.

For example, the CLDC specifies that it targets devices with:

- 160KB to 512KB total memory available for Java technology
- Limited power (often battery)
- Limited, perhaps intermittent connectivity to a network (often wireless)
- Extremely constrained user interfaces, small screens

(The CLDC 1.0 specification is available for free download now, and Sun provides an implementation built using the KVM. Work has begun on the CLDC Next Generation spec. We will be discussing the CLDC and the KVM in greater detail later on.)

A **profile** is a collection of Java technology APIs that supplement a configuration to provide capabilities for a specific device or market type. Again, these are defined through the JCP and are compatibility tested by implementers to insure portability of your applications.

Version 1.0 of the MIDP specification is available for free download, and addresses:

- Devices implementing the J2ME CLDC
- Display toolkit, user input methods
- Persistent data storage using simple record-oriented database model
- HTTP-based networking using CLDC Generic Connection framework

The practical definition for developers is that a configuration specifies everything you need to have basic Java Technology support in a compliant device, while profiles give you industry-specific or specialized APIs on top of the underlying configuration. The CLDC and MIDP described above together specify both the core VM and Java language features plus the wireless- and mobile-specific APIs you need for mobile phones and pagers with small memory, power constraints, small screens, and limited user interfaces.

For more information on configurations and profiles, please refer to the J2ME homepage (URL in resources below).

Web Content for Mobile Devices

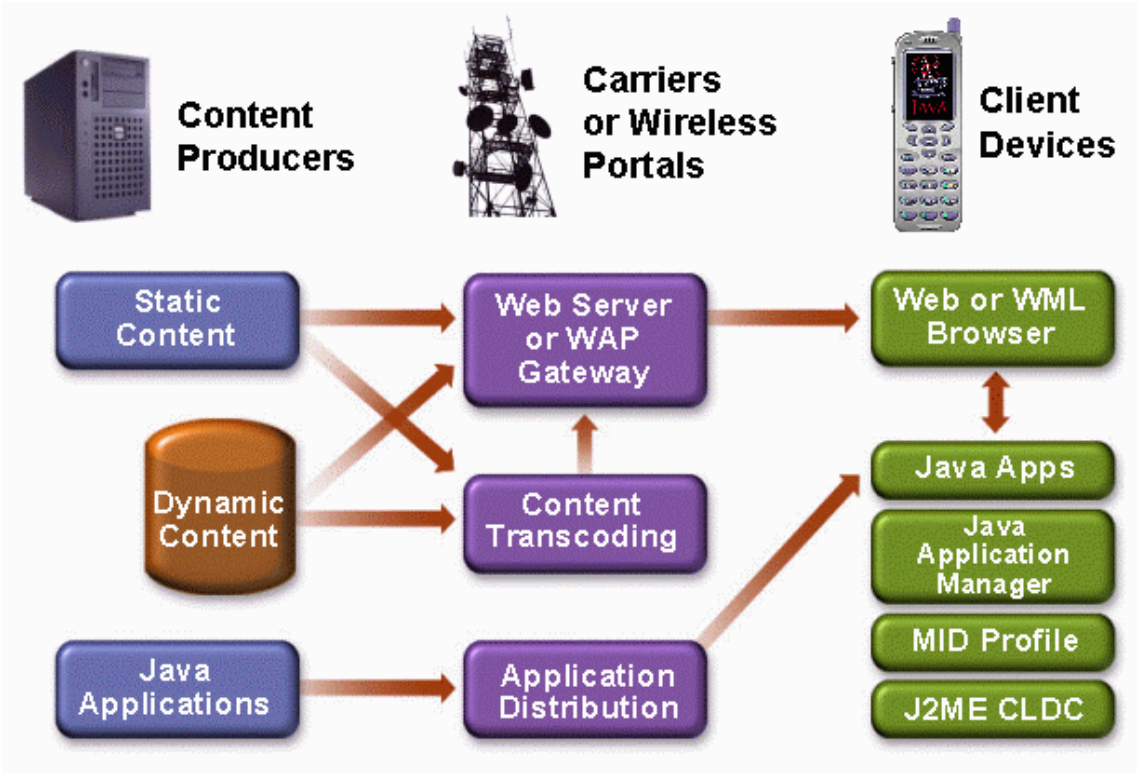
There are three options for getting static (markup) content to a mobile phone today:

- *CHTML (Compact Hypertext Markup Language)* — a subset of HTML used by NTT DoCoMo for its i-mode service in Japan. This has been deployed since February 1999, and has over twenty six million subscribers (and growing) as of late summer 2001.

- *WML (Wireless Markup Language)* — emerging as the standard for delivering content in most of the world, although North America is still predominantly HDML (Handheld Device Markup Language, the proprietary precursor to WML). Part of the Wireless Application Protocol (WAP) stack defined by the WAP Forum.

- *HTML* — Many devices will still use HTML, or some subset of it. For example, there are quite a few HTML browsers that run on the Palm OS. There may be some massaging of the HTML before it gets to the device, but generally they just do their best to render it.

How does the wireless world work? How would you get content out to a device?



At the top left of the figure, you have content that comes out of simple, static files. Dynamic content is also generated in application servers and Web servers, largely based upon information extracted from a database and being modified dynamically using Java Servlets, Enterprise JavaBeans (EJBs), or other server-side technologies.

Increasingly, we see Java applications that sit on a Web server and move through the network to mobile devices. A Web server and/or WAP gateway system will distribute this content out onto the mobile phone or other device.

To get your content onto that web server you may have to go through a "transcoding" process. (This process may be referred to by other names, such as “transforming” or “translating”, but is essentially the same operation: Content in format A goes into the transcoder, which works on the content and emits roughly equivalent markup content in format B out.) Effectively, there is a service available that will take your database, or XML, or HTML, and turn it into WML, or CHTML, or whatever formats you need. In addition, if you are using standard Internet protocols, then you can use a web server to push it out over the wire. If it's a WAP connection, then you'll be going through a WAP server and gateway, and the gateway and transcoder will probably all be bundled together in one package.

Once the content (whether WML, CHTML, or some other markup format) is served out to the mobile device, it is rendered in a micro browser.

Again, increasingly we see Java applications moving out across the network, and there is an issue of distribution — how do the Java applications actually move through the network to the phone, how will operators and providers bill for applications and

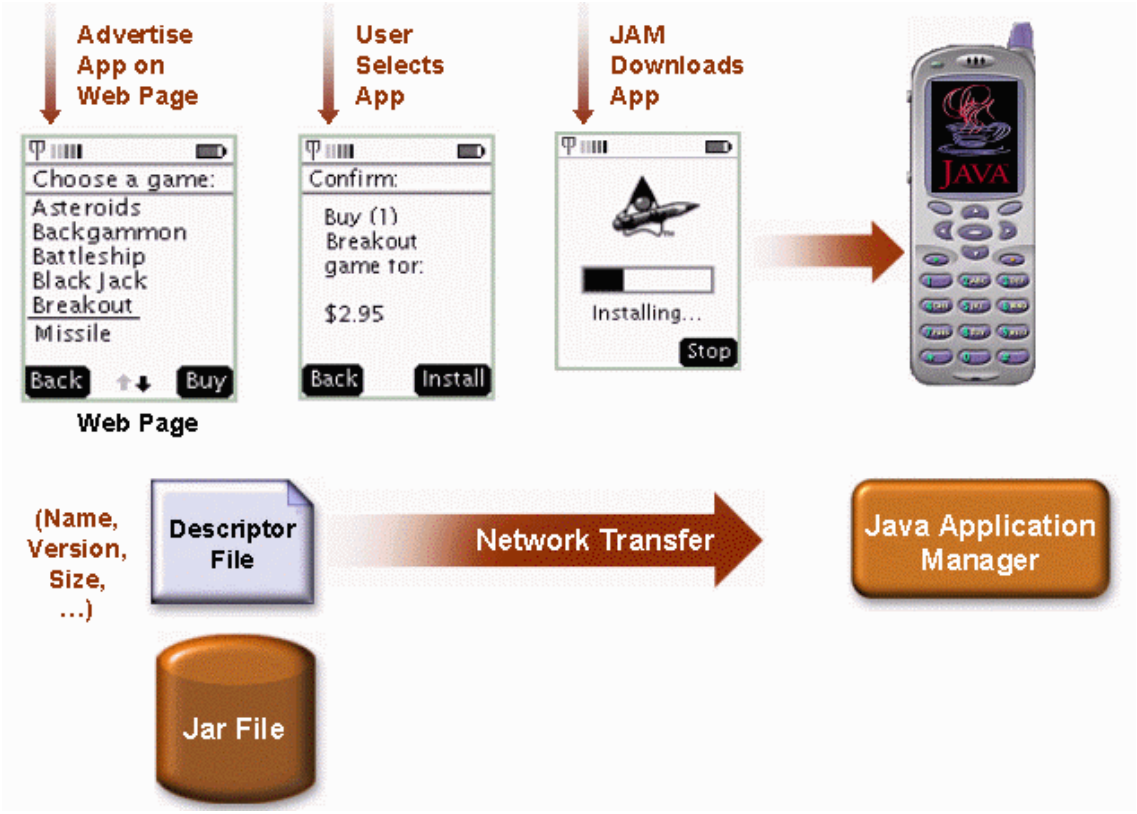
services, etc.? A number of companies are involved with defining these distribution technologies and business models, and defining the services that should be available.

For the first deployments (LG Telecom in Korea in 2000, NTT DoCoMo in Japan and Nextel in the USA in early 2001), the exact details of provisioning, distribution, and business models have been somewhat network specific. Every network, and every operator, has its own little quirks and differences — even as they all aspire to follow the same standards with the coming of third generation wireless technologies (3G) starting in 2001-2002. To help address some of these differences, the MIDP expert group has produced a best practice document for "Over the Air" (OTA) provisioning. This document standardizes some of the details for developers. The MIDP expert group hopes to integrate this document into the MIDP Next Generation specification (work underway via the JCP as Java Specification Request, or JSR, 118). Even if many of the technical issues of deployment are agreed upon and standardized, though, prudent developers will need to look beyond just the technical issues.

Partner early and learn as much as you can about the business practices and business model of any given operator with whom you wish to deploy your applications. Pay close attention when provisioning and billing are discussed. Hopefully, Sun can help you with that — we've announced a Java Wireless Developer Initiative to help companies find potential partners in the industry. (Please refer to the Resources section below for more information on this initiative.)

Java Application Loading Process

Once the application is ready to be distributed, it needs a mechanism to physically load it into the phone. I use the term Java Application Manager (JAM) here for this mechanism. The MIDP spec loosely specifies that the manufacturer of the given compliant device must provide an application manager built into the phone that knows how to load the code over the wireless connection, where it should be stored, and how the user would launch the application. If you look at the MIDP specification, it clearly describes what is required of a device and a manufacturer in this situation, and where they have flexibility. The flexibility is in how they implement this mechanism; the MIDP specification tells them that they have to have one.



This is a schematic of how a JAM might work. The JAM would be smart so the consumer interested in buying (or downloading for free) your application doesn't waste their money or time.

In this scenario, a consumer goes to a Web page (or WAP page). The page lists your application as one possible choice for download.

If the consumer wants to buy your application, they select it from the Web page using their mobile handset. Selecting the application automatically downloads a descriptor file (on the order of hundreds of bytes) over the network and into their handset. Since the descriptor file is very small, it loads relatively quickly and cheaply across the wireless connection.

The descriptor file tells the consumer (and their phone) some basic things about the application. It can tell the phone what version this application is: If the consumer already has the same version of the application on their phone, they can be alerted to this fact so they don't buy it again. The descriptor can also include the size of the actual application, so that if the user only has 2K of space left and the application is 6K, it can pop up a window saying that the user doesn't have enough room to download and store this application. This is good for the consumer because they don't waste time or expense on their wireless connection downloading an application for which they don't have enough memory (they can of course delete something already on the phone to free up space, then download the new application).

Once the consumer is ready to download the application and the JAM has confirmed that there is enough space, the JAM downloads the application. The JAM will save the application in the device and then present it as a selection so that the user can launch and use the application.

Supporting WAP in J2ME Enabled Mobile Phones

WAP has received a lot of discussion of late. You may be interested in it, and if you are, you are probably wondering how WAP and J2ME related. Let's discuss WAP and some of the options that are being considered for WAP and J2ME interoperability. How are they complementary, and how do they work together?

WML, WAP's markup language, works well for certain kinds of applications, such as text-centric applications like text weather reports, stock quotes, etc. But there are a number of weaknesses with WAP which in-device Java technology can overcome.

WAP-based applications require a constantly available network connection, since the applications themselves reside on a server and require frequent use of the wireless connection. Because WAP logic executes on the server-side (with the exception of some simple scripting that can run in the device), WAP is not very good at graphics-intensive applications such as games, interactive charting, etc. WAP also has known security issues in the WAP gateway, where protocol conversion occurs.

Java technology can run on devices which are disconnected from the network. Because Java technology allows for disconnected operation, the developer can choose how to split their application and service logic between the device and servers. This allows developers to create graphical applications, including color games, office applications, and more. Java technology also provides for a robust, well-tested security model, with many J2ME based devices supporting end-to-end secure HTTP (SSL, standardized as "Transaction Layer Security" or TLS).

And perhaps best of all, the Java language is a fully developed and well understood programming language and executes safely and portably on various mobile and wireless devices. You can write your application not only for the handful of physical

devices for which you have emulators or physical access, but also for the countless other devices that implement J2ME, including the many that you will never see (or may not even know exist). This is a critical strength for the Java platform, so I'll repeat it: You can write your J2ME application once with the certainty that it can deploy on the hundreds, or even thousands, of compliant devices available to consumers.

In a sense, then, Java technology is very good at many of the things in which WAP is lacking: Java techology and WAP are complementary technologies.

One option for using the two together is to have a WAP browser in the phone, and to have a VM that implements the CLDC and MIDP specifications. This virtual machine might be the KVM, or it might be another VM, so long as it is compliant with the J2ME CLDC and MIDP specifications. The WAP browser and the J2ME environment could then call each other via an API between the two. When the JAM wants to run a Java application, it has the WAP browser do the download via its standard mechanisms, and then the JAM manages the installation of the Java application and its execution by the JVM. If the JVM wants to put some text up in the device screen, it tells the browser that, "This is what I'd like you to display" via the standard callback mechanism.

Hopefully, what we will see in the longer term are WAP and other micro browsers written in the Java programming language themselves. (In fact, there are several Java technology based WAP micro browsers available already; refer to the J2ME Archive for examples.) In most of the non-Java technology browsers in phones now, if you want a new version or a new feature you have to buy a new phone. A micro browser based on Java technology could give you extra flexibility in that you could *update* your WAP browser itself, on-the-fly, ieven over the air. You could keep the same phone and download browser updates as they become available. This would benefit consumers by keeping their handset costs down and benefit operators by allowing them to add new features to their service more easily and quickly.

The WAP Forum, the standards group that sets WAP and WML standards, has had and continues to have discussions about a WAP browser-Java API, and how Java technology should work with WAP technology. Openwave (a merger of Software.com and Phone.com, the latter a principal founder of the WAP Forum and one of the world's leading WAP supporters) announced a major partnership with Sun to integrate J2ME and WAP in future releases of their micro browser and server technologies. The moral: WAP and the Java platform complement each other and work well together today, and things will only get better in the future.

The Future of Markup in Wireless Devices

WAP's WML, Compact HTML, and other markup languages are converging towards XHTML.

XHTML embodies a rewriting of HTML as an XML-based markup language. The core of XHTML, known as *XHTML Basic*, has been designed to be renderable on any device with an XHTML browser. If you can get your markup based content into XHTML Basic, your content should be deliverable on any operating system, platform, and device that has an XHTML browser.

The WAP Forum is involved with the World Wide Web Consortium (W3C)-led XHTML effort. (The W3C sets a number of Web related standards, including the standards for HTML and other markups for use over the Web.) Sun is a member of both the W3C and the WAP Forum and is one of the many companies working hard to bring WAP back into the standard Internet Java technology+XML+IP fold. Long term, wireless

technologies including WAP and Compact HTML are converging with standard Internet technologies, including the Java platform, XML, and IP. Some time relatively soon, you will be able to write one application using the Java language and XHTML and deliver it over IP-based protocols to any device.

It is not yet clear how far in the future this Java technology+XML+IP nirvana is, but we can say that WAP 2.0 will use XHTML as its markup and TCP/IP as its transport. The first XHTML specification (called a "recommendation" in W3C parlance) is already available from the W3C, so we now have manufacturers implementing it for use in real devices. We will see thousands, then millions, of devices using XHTML in consumers' hands in the not too distant future. Plan for it and start thinking in terms of Java technology+XML+IP based development and deployment now.

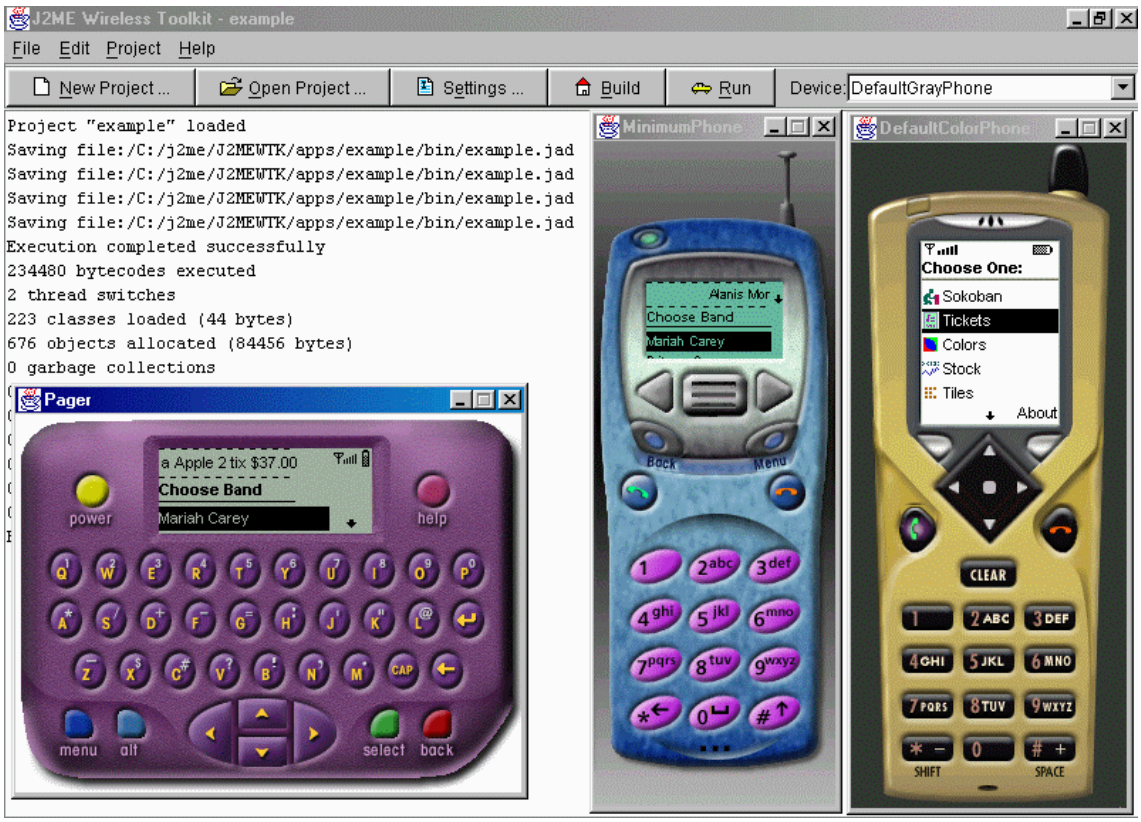
What's next?

Where is J2ME today? The J2ME CLDC 1.0 specification and reference implementation are both freely available from the Sun CLDC Web site. The CLDC Next Generation effort has started via the JCP's JSR 139. The J2ME MIDP 1.0 specification and reference implementation are also freely available from the Sun MIDP Web site, and the development of the MIDP Next Generation technology has also begun, via the JCP's JSR 118.

Sun is keen to work with others in the industry to develop open technology standards. The use of the Java Community Process for J2ME CLDC and MIDP development is a very good example of how we are developing mobile and wireless device technologies with the entire wireless community. Sun also continues to work with the GSM and 3G standards groups, and was very pleased to have J2ME accepted into GSM and 3G standards for mobile phone handset technologies.

The J2ME CLDC and KVM have been ported to many different platforms and devices, including Motorola phones and two way pagers, Research in Motion (RIM) wireless handhelds, and Palm PDAs. Millions of J2ME CLDC powered mobile phones are used daily by consumers in Japan and Korea. Many more consumers are using J2ME powered Motorola handsets on the Nextel network in the United States, and network tests and commercial deployments are coming in the months ahead from the likes of Vodaphone Airtouch, Sprint PCS, Telefonica, and many other operators. Symbian has pledged to continue support for Java technology in their EPOC OS, with the first EPOC-based, Java technology enabled Nokia Communicator, the Nokia 9210, released in early 2001.

Best of all, there are a number of free J2ME development tools available today. Sun provides a J2ME Wireless Toolkit, which contains a full J2ME CLDC and MIDP implementation. The toolkit can emulate the look and feel of various generic phones and two way pagers. It can run in a standalone mode (as shown below), or you can plug it into the freely downloaded Forte for Java, Community Edition IDE (also from Sun) for a full blown J2ME development environment.



There are quite a few other tools. Available J2ME development tools include:

- Sun's J2ME Wireless Toolkit (optionally plugs into Forte for Java IDE).
- MIDP reference and MIDP for PalmOS, CLDC/KVM SDK (support for Win32, Solaris, and Linux)
- Motorola/Metrowerks J2ME tools for CodeWarrior, RIM BlackBerry IDE, Zucotto WHITEboard SDK, and more
- Any IDE or standard Java tool, including J2SE SDK (JDK 1.2.2 or 1.3)
- Other tools from the J2ME archive (*Spotter* utility, *WBXML* package, etc.)
- J2ME tools CD, available now as part of the Java Jumpstart CD pack from the Sun Developer Connection (SDC).

For more examples, and to download your own copies, visit the J2ME Archive.

If you are interested in wireless development, and in the Java technology+XML+IP world coming very soon to devices of all sizes near you, I would strongly encourage you to get started prototyping and building J2ME applications today. The specifications are here, the tools are available, and opportunitites abound to deploy your applications in real networks today. It is a great time to become a J2ME wireless developer! Get started today!

Resources

If you are interested in developing for J2ME, be sure to check out the *kvm-interest* mailing list. Many of the Sun engineers, other expert group engineers, and real-world J2ME developers actively participate in Q&A and discussion on J2ME development on the kvm-interest list.

To subscribe, send e-mail to *listserv@java.sun.com*, and in the body of the message type: *subscribe kvm-interest*. Because this is a very active mailing list,

you might want to subscribe in digest mode; to do this, include the following line: *set kvm-interest digest*

Web Sites of Interest

The J2ME page can be found at:
http://java.sun.com/j2me

Java Wireless Developer Initiative:
http://java.sun.com/wireless

Download the CLDC specification and SDK at:
http://java.sun.com/products/cldc

Download the MIDP specification and reference implementation at:
http://java.sun.com/products/midp

Download the Sun J2ME Wireless Toolkit:
http://java.sun.com/products/j2mewtoolkit

J2ME tutorial articles:
http://java.sun.com/jdc/technicalArticles/wireless

J2ME and Wireless 101 Webcasts:
http://java.sun.com/jdc/onlineTraining/webcasts

Read the specs, dowload the emulators, SDK, and tools, and access a large number of example applications with source code from the J2ME Archive:
http://www.billday.com/j2me

jGuru.com J2ME FAQ:
http://www.jguru.com/faq/J2ME

Got questions? Email Sun's Technology Evangelists:
evangcentral@sun.com

Or visit our homepage:
http://www.sun.com/developers/evangcentral

About the Author

Bill Day is a Technology Evangelist at Sun Microsystems. He created the *J2ME Archive* to help developers build applications and services using Java technology.

Bill manages jGuru's J2ME and Java Media FAQs and writes about software development for numerous publications. He has contributed feature articles to CNN.com, JavaWorld, Dr. Dobb's Journal Software Careers, SunWorld, Gamasutra, and Sun's Java site. Bill is also a Sun Certified Programmer for Java 2 Platform and teaches Java and Wireless development as an extension instructor for the University of California Berkeley.

More information is available from Bill's Web site:
http://www.billday.com