JDJEDGE2001 EARLY BIRD SAVINGS TO EXPIRE JULY 31 P.133

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

July 2001 Volume: 6 Issue: 7

JAVADEVELOPERSJOURNAL.COM

Sept 23–26, 2001 New York, NY

SHOW PROGRAM INSIDE!

web services EDGE conference & expo

JDJ EDGE conference & expo 133

SYS-CON RADIO

Interview with David Litwack 10

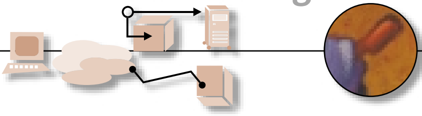
Sneak Peek Interview with James Gosling 122

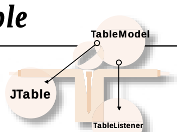
RETAILERS PLEASE DISPLAY UNTIL AUGUST 31, 2001
$4.99US $6.99CAN

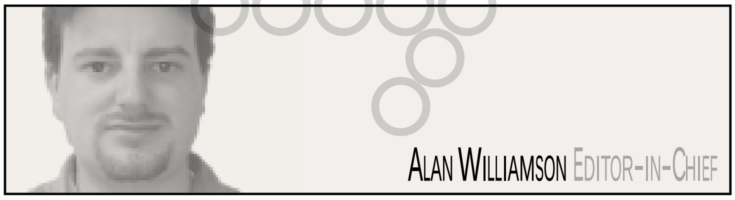
SYS-CON MEDIA

You Make the Decision

Written by Jon Stevens p22







ALAN WILLIAMSON EDITOR-IN-CHIEF

If You're Going to San Francisco . . .

Be sure to wear some flowers in your hair…as the classic Scott McKenzie song advised. What sound advice that is since last month thousands of Java developers made the annual pilgrimage to JavaOne at the Moscone Center. We were there, the ***JDJ*** crew, covering the event, talking to all who would listen, and listening to all those who spoke (flip on through this issue and you'll find the JavaOne show report from Ajit Sagar).

Now you'd be forgiven for believing that the whole show was devoted to Web Services, but I have to report that this wasn't the case. Sure the topic of Web Services came up many times, along with the whole .NET versus Java debate. But on the whole, the topic of conversation was much broader than that, encompassing many of the Web Services ideals. I think this was mainly due to the fact that JavaOne is so developer-focused; in many respects, it's a bit like preaching to the choir. We know all about open standards and the importance of not closing off any future possibilities.

I spoke to a lot of the application vendors and listened intently to their findings and what their clients were asking for. I was told that many of their clients are still using JSP and servlets and haven't yet discovered the true power of EJBs. I quizzed them as to why this may be the case and the answers that came back were varied, ranging from "lack of education" to "perceived to be complicated." This on the whole surprised me, as I had always presumed that if you purchase an application server, you're going to be doing more than just servlets/JSP. I wonder then how many companies would have saved themselves a small fortune by opting for something more along the lines of ServletExec or Apache Tomcat/JServ.

As if to support this, I discovered many companies touting JSP-related products. Many more than there were last year. There were tools to convert Visual Basic programs to JSP, JSP development environments, JSP reporting tools, JSP plug-ins, the list goes on. I guess this is validation of a technology, when an entire industry spawns up to support it. I'm just surprised at the number of JSP-related activities compared to EJB offerings. Is the world going JSP mad? Not if you read Jon Stevens feature this month on some of the idiosyncrasies of JSP. It was definitely an eye-opener for me.

Another great buzz around the floor was that of J2ME and the world that's beginning to open up for us there. I was introduced to a plethora of products that were designed to allow designers and developers to easily utilize this new, emerging platform. If only half of what I saw makes it to mass market, we're in for some beautiful applications. I'm not too sure though whether society as a whole is ready for J2ME and its implications. I've seen firsthand the stresses that carrying an always-connected device (Blackberry, for example) can impose on a person, and this is just for e-mail applications. It's just the tip of the iceberg; the best is still to come.

That said, when arranging to meet people, it was such a novelty to e-mail people as opposed to ringing their cell numbers. E-mailing people while on the move is definitely a useful tool, but something I think we need to be careful with. Where up till now we opted into the worldwide network of communications, in the future, we'll have to consciously opt out of the network. This is a major social shift and one that I'm sure will cause many a sleepless night.

Let me thank all those who stopped by and said hello. It was good to meet you and I look forward to continuing to serve you.

alan@sys-con.com

AUTHOR BIO

Alan Williamson is editor-in-chief of Java Developer's Journal. In his spare time he holds the post of chief technical officer at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Reach him at alan@n-ary.com (www.n-ary.com) and rumor has it he welcomes all suggestions and comments!

magicdraw™

The rational alternative

All 9 UML diagrams

Additional Features:

- Performs Java, C++ or CORBA IDL code round-trip engineering (code generation and reverse engineering); recognizes JavaDoc comments. This feature allows you to write code, reverse engineer, make changes to the model and re-generate the code without losing any implementation specific information.
- Supports UML 1.3 notation.
- Saves diagrams as bitmap PNG/JPG and scalable WMF/SVG/EPS/DXF formats.
- Provides XML interoperability – native model files are stored in XMI format.
- Integrated with Forte for Java (FFJ) IDE versions 1.0 and 2.0.

Standard Edition: $249

Professional Edition: $499

Visit us at Java One

Floating License . . $100 additional

Upgrade for a $69 annual fee!

No Magic... Just Technology That Works

Download fully functional demo at:

www.magicdraw.com

contact us at contacts@magicdraw.com

100% Pure Java Application

WRITTEN BY JAMES DUNCAN DAVIDSON

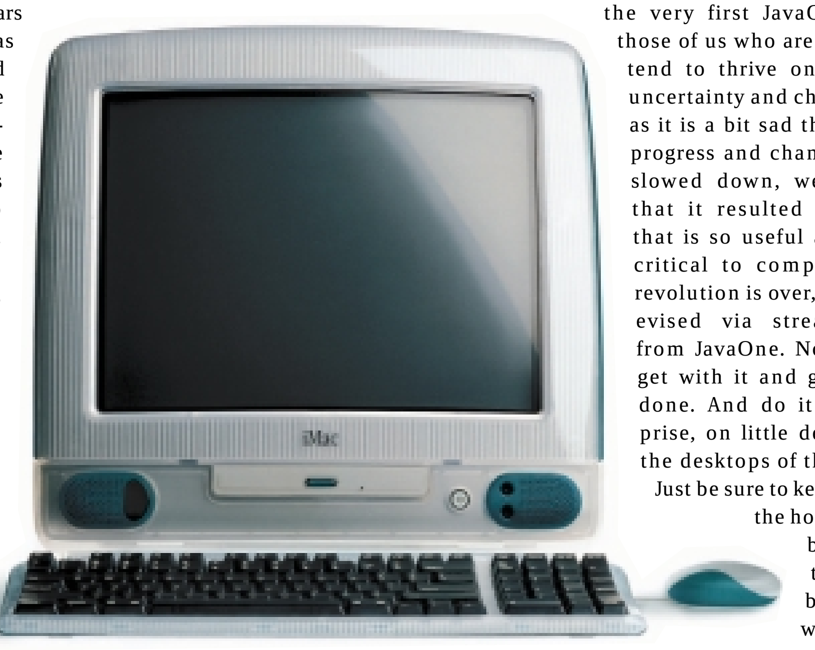
The Revolution Is Over

It was televised, now get to work

At this year's JavaOne conference in San Francisco, I came face to face with the reality that Java no longer occupies the position of being a disruptive technology. It is now an accepted, depended-on, stable, workhorse technology. Of course, this has been shaping up for years, but for those of us who work every day on the technology, it's hard to tell when this happens until the change is done. For me, the final moment of realization came during Monday's opening keynote. In fact, it came as I saw a slide that positioned the invention of Java as of equal importance to the microprocessor and networking.

Now, even though I love the platform and have spent the last six years working almost exclusively with and on Java, I don't agree with that statement. The invention of the microprocessor and of networking are in a totally different league than Java when it comes to affecting history. If James Gosling and the Oak team had not put together Java, then we would all be using some (probably not as good) alternative. But if the network had not been invented, or the microprocessor had not come to be, the world would be a much different place.

That said, just the fact that the claim was made is a statement all its own. Six years ago, at the first JavaOne, it was not clear that the technology would live up to any of its promises. Even two years ago when J2EE was announced, it was beyond any expectation that the APIs would unify the application server space to the degree they have. Now, it's simply expected that an app server will support J2EE. And over the last few years we've seen an amazing upswing in the use of Java in the small device market. We have seen Java-enabled cell phones ship like crazy in Japan, and we're on the verge of seeing them sweep over the domestic U.S. market. And let's not forget the zillions of Java-enabled smart cards out there.



Even the desktop is getting a boost. With Apple's shipment of Mac OS X, there is once again a desktop OS shipping with the latest version of Java built in. And Apple's implementation is not a simple port, but a complete integration of Java into the OS. Apple has even done quite a bit of innovative work that they plan to give back to Sun. This work includes the sharing of the core libraries between virtual machines and the utilization of hardware acceleration in Swing. After a long dry spell on the desktop while J2EE and J2ME take the limelight, J2SE is poised to make a comeback.

Java has matured to the point where the question is no longer, "Why should I use it?"; it is now, "How should I use it?" Anybody who lived through the years where we had to work hard to convince people that Java was ready for prime time is surely grateful that Java has come so far. And it has taken a lot of hard work by a lot of talented people to get to this point. In the process, Java has matured and stabilized. Instead of radical change every 12 months, we now see gradual changes over the span of years. Java has moved away from being a revolution to simply being useful. And that is a good thing.

Truth be told though, this is a bittersweet moment for those of us who have been there since the very first JavaOne. After all, those of us who are early adopters tend to thrive on the chaos of uncertainty and change. But even as it is a bit sad that the pace of progress and change in Java has slowed down, we're all happy that it resulted in something that is so useful and, yes, even critical to computing. So the revolution is over, and it was televised via streaming media from JavaOne. Now it's time to get with it and get some work done. And do it in the enterprise, on little devices, and on the desktops of the world.

Just be sure to keep your eyes on the horizon; there will be other revolutions that will be interesting to watch.

duncan@x180.net

AUTHOR BIO

James Duncan Davidson was the original author of Apache Tomcat, the official reference implementation of the Servlet API; and Apache Ant, a Java and XML build tool. He was also the specification and project lead at Sun for the 2.1 and 2.2 versions of the Servlet API as well as the specification lead for the Java API for XML Parsing 1.0 and Java API for XML Processing 1.1.

Interview with David Litwack

President and CEO of SilverStream

Interviewed by Dave Johnson

<jdj>: **SilverStream has been on the cusp of J2EE and Enterprise Java in Internet environment development. This has helped SilverStream grow from an application server company to a J2EE frameworks company and, now, a consolidation of all these efforts, a Web Services company. So, what's going on at SilverStream this year?**

<litwack>: Web Services is an important technology. However, the really important thing that's happening is that the industry is going through a once-in-a-decade transformation to a new application paradigm – a new way of building applications.

I'd like to draw a comparison with client/server. Client/server wasn't a specific technology. It was the culmination of a set of technologies – interfaces, relational databases, networks, personal computers – that resulted in a new way of building applications. And that way of building applications was very important because it was a clean separation of information from the user interface. It transformed the way we use computers in businesses from what had been predominantly an administrative or clerical function, to having the appropriate repository of information available to everybody that had a computer on their desktop. That was the revolution.

What's happening today is we have a set of very important technologies – Java, J2EE, XML, SOAP, WSDL, HTTP, the Internet – that have culminated in a level of maturity where we can think about building the applications in a completely different way. The GartnerGroup calls that a services-oriented architecture, and Web Services is certainly a key piece of that. The important thing about a services-oriented architecture is a clean separation of the transaction, information, or service from the audience to which it's supposed to be delivered. Put another way, it turns the traditional design paradigm backward, the traditional being: Who is the audience you're delivering the application to?

In a services-oriented architecture you shouldn't know or have any design criteria that involves knowledge of who the audience is. That is so important today because, through the Internet, we can deliver applications to people we've never met or to devices that haven't been invented yet. We can deliver applications inside or out-

JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD

JEREMY ALLAIRE, TED COOMBS, ARTHUR VAN HOFF, JIM MILBERY, GEORGE PAOLINI, KIM POLESE, SEAN RHODY, RICK ROSS, AJIT SAGAR, BRUCE SCOTT, RICHARD SOLEY, ALAN WILLIAMSON

DEPARTMENT EDITORS

FOUNDING EDITOR/CHIEF CORPORATE EDITOR:	SEAN RHODY
EDITORIAL DIRECTOR:	JEREMY GEELAN
EDITOR-IN-CHIEF:	ALAN WILLIAMSON
EXECUTIVE EDITOR:	M'LOU PINKHAM
ASSOCIATE ART DIRECTOR:	LOUIS F. CUFFARI
MANAGING EDITOR:	CHERYL VAN SISE
EDITOR:	NANCY VALENTINE
ASSOCIATE EDITORS:	JAMIE MATUSOW
	GAIL SCHULTZ
	BRENDA GREENE
ASSISTANT EDITOR:	GREGORY LUDWIG
TECHNICAL EDITOR:	BAHADIR KARUV, PH.D.
PRODUCT REVIEW EDITOR:	JIM MILBERY
INDUSTRY NEWS EDITOR:	LEE PARSEGHIAN
J2EE EDITOR:	AJIT SAGAR

WRITERS IN THIS ISSUE

ROBERT ANDREASEN, BILL BALOGLU, LIZ BLAIR, JASON BRIGGS, JAMES DUNCAN DAVIDSON, JOSH ECKELS, JEREMY GEELAN, JEFF HEATON, BOB HENDRY, BILLY HO, DAVE JOHNSON, BILLY PALMIERI, AJIT SAGAR, JON STEVENS, HELEN THOMAS, ALAN WILLIAMSON, BOBBY WOOLF, BLAIR WYMAN

SUBSCRIPTIONS:

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT

SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM

COVER PRICE: $4.99/ISSUE

DOMESTIC: $69.99/YR. (12 ISSUES)

CANADA/MEXICO: $79.99/YR. OVERSEAS: $99.99/YR.

(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $10/EA., INTERNATIONAL $15/EA.

EDITORIAL OFFICES:

SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

WORLDWIDE DISTRIBUTION BY:

CURTIS CIRCULATION COMPANY

730 RIVER ROAD, NEW MILFORD NJ 07646-3048 PHONE: 201 634-7400

side a firewall. So a services-oriented application meets the needs of e-businesses in the Internet age because it allows us to build applications that can be flexibly deployed to unknown future audiences.

<jdj>: **Can you describe SilverStream's progression in leadership roles through the advent of the application server and Enterprise Java, the frameworks that you've pioneered, and now the culmination of all these technologies with your framework and application servers under the Web servers' realm?**
<litwack>: We look at the services-oriented architecture. The history of SilverStream is the history of the pieces of a services-oriented architecture. SilverStream was one of the first, if not the first, pure Java-based application services in the market. Five years ago we were building one of the largest Java applications in the world. Then when J2EE came along we quickly hopped on the bandwagon and again were one of the first Java applications services to be fully J2EE certified and compliant with our 3.7 release, which has been shipping for several months now.

For the past two years we've been building a set of engines or class libraries generically on top of J2EE, which involves a lot of the pieces that are necessary in a Web Services environment. These include XML mapping and a transformation engine that provides activity and integration to a broad variety of back ends such as mainframes, Web sites, or MQ series. This is important as most won't be invented from scratch. Most Web Services will be existing applications that are Web Services enabled. By creating an environment in which we can connect to back ends, we can provide an XML request response metaphor. We're already doing the essence of Web Services and, of course, by being fully J2EE, we have the environment for building new Java-based applications as well.

When I think of Web Services there are two unanswered problems that the standard doesn't answer. One is, "Where does the XML come from?" And the other is, "What do you do with the XML once you have it?" With our XML information and integration, and with our Java-based J2EE capability, we answer the question, "Where does the XML come from?" That's the service producer's side.

On the service consumer side it's delivering up the XML in a variety of ways. The key thing is to take the information delivered up from Web Services and apply the notion of relevance to the particular circumstance.

Web Services are great but we don't want to deliver them in the same way to everybody. We want to tailor them depending on who the users are–what their jobs are, whether they're high-rate individuals, whether they're partners or salespeople, whether they're inside or outside the firewall. We may increasingly tailor Web Services to whether it's the weekend or the week, or what device they want. Even if they get far out with GPS and wireless devices, we may want to tailor the service depending on where they are, because with GPS devices we'll know their location within 10 meters. These are all parts of the services-oriented puzzle.

> "the industry is going through a once-in-a-decade transformation to a new application paradigm – a new way of building applications"

At JavaOne we announced the release of the cap piece of all of this, the lock piece of the puzzle. It's our Web Services engine that we've built using a combination of our existing Java-based Web technology and things such as WSDL compilers to generate very high-performance Web Services out of any Java class. Not only is this high performance, the Web service effectively gets generated as a servlet that can then be flexibly deployed in any J2EE server?

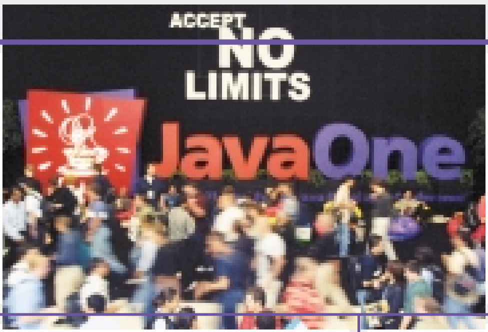
On the client side it can be accessed either from a SOAP client with a SOAP stub that will generate, or through a RMI stub so the Web service can be accessed as a straight remote Java object. I think this will appeal to Java developers because they can create a Java class, generate a remote Java object in the client, and the entire Web service looks like a straight Java capability.

I also want to mention our plans to generate a Visual Basic and C# client, because when you create that Web service, part of being a Web service is accessibility from any kind of client, and Web Services is an agnostic technology.

<jdj>: **What's in store for SilverStream, and Web Services in general?**
<litwack>: Over the next few months we'll be delivering a full product line called SilverStream eXtend. It includes the engines I've just described – Web Services, XML integration, workflow, business rules, and content management – all running on the major J2EE servers.

We'll also be shipping our Web Services producer product, which is the creation of Web Services.

For me, the new Java applications are existing back-end systems. The Web Services consumer product includes personalization, user customization, and customization to a variety of wireless devices as well as an integrating workbench that's oriented around the notion of services-oriented applications. In addition, we're beginning to provide vertical application-level frameworks, the first of which is shipping Silverstream eXtend with actual application framework code that demonstrates to different industries the way they can take advantage of Web Services, to have the various constituencies in a business communicate electronically in an e-business.

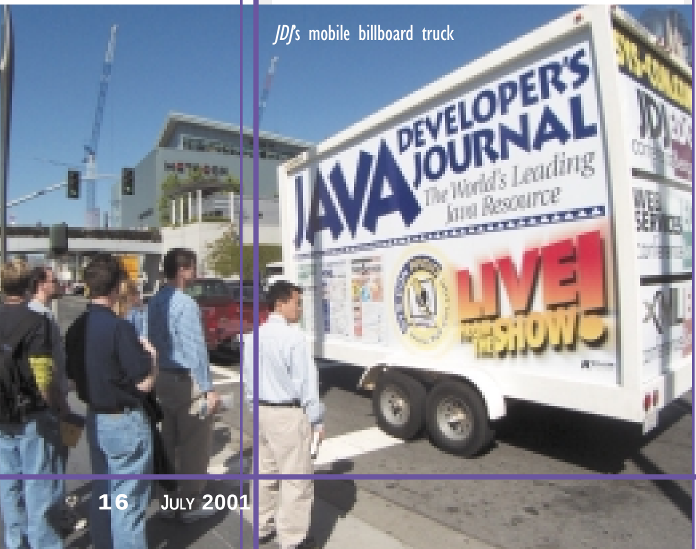


JavaOne 2001

This is the last day of the sixth annual JavaOne Conference here in San Francisco. It has been an interesting five days. This is my fifth JavaOne conference. I partially attended last year's show, and attended most of the previous three shows here at the Moscone. My first impression – although there was a plethora of technologies and industry applications, I felt the show was lukewarm in comparison to last year's show. This is the disadvantage of having a base to compare against. If I hadn't attended this conference in the past few years, perhaps I would probably have been equally impressed by this one. Overall, it was still a pretty good show, with Java showing a much higher degree of maturity, especially in the J2EE platform.

The Themes

The conference revolved around three major themes – Web Services, wireless and small devices (J2ME), and J2EE. There was an air of euphoria around Web Services. Most of the vendors I talked to, whether they were application server vendors, development tools providers, or providers of software for mobile devices, had a story to spin around Web Services. This was similar to the atmosphere regarding XML a couple of years ago. There were more examples of actual prototypes in the wireless space. And Sun was definitely trying to convince everyone that JINI is alive and kicking. This is despite the fact that though the number of downloads and the community around this technology has shown substantial growth, real-world applications were conspicuous by their absence.


JDJ's mobile billboard truck

Web Services was definitely a main theme for the conference. A large majority of the vendors were presenting their take on Web Services. It was interesting to get the perspective on this technology from vendors ranging from application server vendors, through development environment providers to providers of software for mobile devices. Opinions ranged from Web Services as a panacea for all ailments to skepticism about the business models that they could be applied to in the near term. Much of the information presented in the sessions about Web Services was introductory, which is not surprising, as this is a fairly new paradigm. Under the umbrella of Web Services, UDDI, XML, SOAP, and .NET technologies were major themes for discussion and presentation. Of course, there was a Web Services spin around all three editions of the Java Platform – J2SE, J2ME, and J2EE.

Much of the J2ME and wireless application vendors represented the European community. Telecom has always found early adopters and standardization in Europe as compared to the U.S. Therefore this trend is not surprising. The mobile devices market shows signs of taking off. Several J2ME-powered devices were displayed in the underground corridors between Moscone North and Moscone South. These included mobile phones, set-top boxes, and PDAs. Of course, the concepts have been displayed as prototypes in the previous conferences. Some of these devices have become popular in consumer services.

The maturity of J2EE was apparent. And with it, the problems that organizations have faced in adopting this platform. As companies become more careful about spending, there are valid concerns about adopting a platform that requires expensive peripherals. J2EE by itself is a spec. The actual products are offered by third-party vendors and the complete suite of applications servers, commerce servers, personalization products, workflow engines, and so on, adds up to quite a formidable sum. This can put a damper on J2EE enthusiasts. However, there were many cases of real-world applications, which discussed the trade-offs and benefits of using different aspects of J2EE. Server-side Java computing has definitely come of age. There was a wealth of information on design patterns, tips and tricks, J2EE battle scars, architecture and design, amd more.

Keynotes

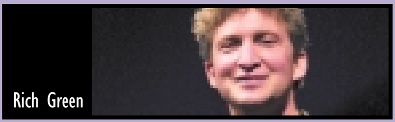
Probably one of the reasons we saw no obvious Microsoft-bashing was because Scott McNealy didn't have a keynote. Don't get me wrong. McNealy is a fantastic speaker and it's always a pleasure to listen to him, but he is also one of the most obvious Microsoft-bashers. The keynotes centered around Web Services, client-side mobile devices and networked applications, and J2EE. I noticed that there were no "celebrity" keynotes. No Douglas Adams at this conference. Also, the majority of the keynotes were from Sun representatives. In fact, there were only three keynotes from Sun's partners – Oracle (Larry Ellison), Nokia (Pekka Ala-Pietila), and BEA (Bill Coleman).

James Gosling

Ed Zander

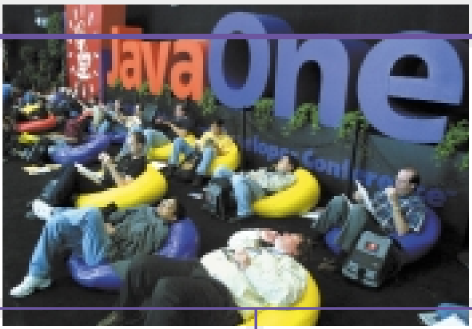
Larry Ellison

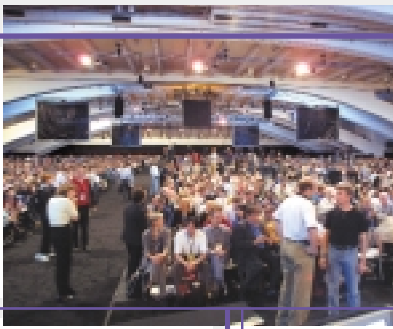
Simon Phipps


Rich Green

Show Report

WRITTEN BY AJIT SAGAR





No Microsoft Bashing?

Surprisingly, there was less of Microsoft bashing than in the previous conferences. No Bill-Gates-Apple-in-the-face type gimmicks. To me this indicates a couple of things. One is that the Java platform technologies have matured to a stage where proponents of Java don't need a target to lash out at. Secondly, the emerging markets powered by XML, Web Services, and wireless technologies level the playing field to the extent that one camp has no choice but to acknowledge the other's presence. In most of the interviews that we conducted at the **SYS-CON Radio** booth, the vendors indicated a neutral stance; they confirmed the fact that the Microsoft as well as Sun and other Java-supporters have substantial presence in Web Services. Most vendors are planning to support both. Thus we will probably see true layers of abstraction created over existing technology camps.

The Program

There was a variety of information available through the seven tracks that comprised the sessions of the conference. For me, Monday and Tuesday were disappointing as there weren't many sessions that went beyond basic introductions to new technologies. Most of this stuff can be picked up from white papers on the Internet. However, Wednesday through Friday was where the presenters actually got into the meat of things. Thursday and Friday were the best days for real-world examples of enterprise applications, J2EE design and implementation, and J2EE vis-á-vis. Web Services. In fact, I was pleasantly surprised to see that the attendance was quite impressive. The last two sessions I attended were packed, including Ed Roman's session on J2EE versus .NET. I think this is a clear indicator that many of the serious developers stuck around to get to the real meat of things.

My conversations with different attendees lead me to believe that a lot of the sessions were either introductory or marketing pitches. Several of my colleagues from San Francisco attended the latter half of the conference, because that was of most use to them. The BOFs were more suited to people looking for an informal forum to get specific information.

One of the very useful facets of the conference was that Sun actually held the Java certification exams on site, as well as several developer competitions. There was also a wealth of information available on the Java certification process.

The Pavilion

The Pavilion had a wide variety of exhibitors with demos and showcased products. Most vendors I talked to thought that the show was a little slower than last year. Congruent with the themes for the conference, J2EE application servers and IDEs, messaging products, mobile communications software, and Web Service's frameworks were the norm. Sun had arranged their booth in a rectangle around all the other vendors. This was a better arrangement than what I have seen earlier, where it was hard to find other vendors without running into a Sun booth.

"The conference revolved around three major themes – Web Services, wireless and small devices (J2ME), and J2EE."

IT Woes

The slump in the economy had obviously affected the show. Though the impact on technology was less obvious, the affect on the conference was quite apparent. Some clear indications of the result of the slowing economy were:

- The attendance in 2000 topped out at 25,000. In fact, Sun's press release from last year claims that this limitation was only because of the Moscone Center's fire marshal limits. Well, no such claims were made this year. Tuesday's JavaOne Today, the conference's daily newsletter, had the attendance tagged at 17,000 – about 32% less than last year.
- There were definitely less gimmicks. No Palms or BlackBerry devices at half-price. The vendors had obviously cut back on spending, as there were fewer giveaways such as T-shirts at the show.
- On Monday, there seemed to be a smaller crowd of people flocking around Moscone. However, the tempo picked up in the next few days.
- I had a chat with the cab drivers – the best sources of information. The consensus was that business was much slower this year. The fact that I could easily book the Crowne Plaza hotel at Union Square on the Friday before the conference, as well as frequent flyer airline tickets, indicated to me that the conference would be a little smaller than the previous year.

But All in All ...

All said and done, it was a fairly good conference. Not the best JavaOne, but one certainly worth attending.

SYS-CON Media

As usual, we had the **SYS-CON Radio** booth set up at the entrance of the show. We interviewed a wide variety of vendor representatives. The interviews are available at www.sys-con.com/java.

SYS-CON Radio

Interview with James Gosling

Interview with David Litwack

AJIT SAGAR J2EE EDITOR

J2EE Has Come a Long Way

I just got back from JavaOne in San Francisco this weekend. My humble opinions on the conference are presented elsewhere in this issue of ***JDJ***. As expected, one of the main themes of JavaOne this year was the J2EE platform and related technologies. Over the last two years, since Sun announced the three editions of the Java Platform, J2EE has come a long way.

The products offered by third-party vendors are mature – and a large part of the mundane development activities are abstracted from the programmer. With the Java Connector Architecture (JCA) specification around the corner, integration with Enterprise Information Sources (EIS) also becomes much clearer and more manageable.

Openness Has Its Advantages

J2EE is definitely more open than any of the other enterprise software platforms available in the industry. However, this openness also means that Sun, as the owner of the specifications, doesn't really provide the entire product suite needed to build the enterprise applications. In short, J2EE is basically a bunch of software specs and APIs. While this is a good thing, in that the specs and APIs are not owned by a single vendor (i.e., Sun), this does mean that application developers have to depend on infrastructure support from third-party vendors, such as IBM, BEA, ATG, as well as Sun.

Openness Is Costly

This in itself is not a bad thing. Open APIs and vendor neutrality are what we all look for as enterprise application developers. However, when you get around to designing the architecture for an enterprise application, there are several tools and products needed to create the building blocks for the application. For starters, in order to use the basic features of J2EE, you need a J2EE application server. Some folks would argue that applications built on JSPs and servlets only require a servlets engine. However, for true *n*-tier distributed applications, you need the EJB container. EJBs are the nexus of J2EE.

Even if you were developing applications in the pure Web paradigm using servlets and JSPs, if you wanted to save the costs associated with the complexities of development and maintenance of infrastructure services – such as threading and persistence – you would have to pick an industry-strength application server.

And the story doesn't stop here. For messaging, you have to pick a messaging provider, such as MQSeries or Tibco. For commerce services, you need a commerce server, such as the one offered by BEA, ATG, or Macromedia. For the complete application, you need to add a personalization server, a workflow environment, a security solution, and a service provider for enterprise connectivity. Add an IDE, designing tools, and administration to the mix, and you're looking at a pretty hefty sum for a basic application. This can run into thousands for pure development.

J2EE development is expensive. And, because of the complexity associated with the platform, so is the training. With new APIs emerging every month, and updated versions of the existing ones, training becomes a major issue. The problem associated with cost may have been dismissed during the boom times, but corporations are now questioning the viability of swallowing such costs in an economy trying to bounce back from a slump.

In Search of Less Expensive Solutions

Although Microsoft technologies offer a less expensive solution, the openness and enterprise-presence of these technologies has yet to be proven. An alterna-

–continued on page 116

ajit@sys-con.com

AUTHOR BIO
Ajit Sagar is the J2EE editor of JDJ*, and the founding editor and editor-in-chief of* XML-Journal*. A lead architect with VerticalNet Solutions, based in San Francisco, he's well versed in Java, Web, and XML technologies.*

J2EE INDEX

WRITTEN BY BILLY HO

J2EE Is a Solid Foundation for Any Application Server . . .

J2EE provides a common set of processing constructs that provide common building blocks for creating e-commerce applications. In the case of connectors and containers, standardization relieves the development of infrastructure code, allowing application behavior to be declared rather than hardwired into the business logic.

Saying that two application servers are J2EE-compliant means that each interprets the deployment descriptors of the other and can reproduce the other's component containers. They provide the same services on which the application relies. What's not standard for application servers, though, are the components – the processing details and other content that provide a business process.

Standardization creates components that are more manageable. An organization that relies on J2EE doesn't want to create a new type of component to meet a previously unanticipated need. Having industry-defined component types and infrastructure dependencies avoids that.

The evolution of more deployable, more manageable Web applications won't stop with XML, prefab infrastructure services, or predefined component types. It's at the next level – codeless applications – that the differences between "standard" J2EE application servers are most telling. This is where implementation becomes a process of managing resources rather than of development or deployment. The focus is more inclusive, encompassing front- and back-end services, and the interface between them.

The goal is not just to provide an infrastructure server that one-of-a-kind applications invoke. It's to provide interoperable services so the distinction between these services and those provided by the infrastructure becomes as academic for IT as it is already for customers.

Mergers and partnerships require convergence of Web sites, which may not work technically, commercially, or even on an aesthetic level. Only when it addresses the totality of the application experience can the app server truly serve Web-ready applications.

Even if a partner's applications eventually demonstrate interoperability and cohesiveness, it may be too late. In an ideal world, no implementation would be needed, applications would plug into each other's URLs and work. The ultimate measure of a great application server is its ability to work within the Web environment. That requires interoperability, deployability, and manageability. Let's look at those characteristics.

- **Interoperability**: Sun provides tests and a test application against which vendors implement their servers as a model. But the real value of the branding program is that it gives organizations the opportunity to have their applications interoperate with their partners. They won't have to identify nonstandard code, write patches, or test servers.
- **Deployability**: Two obvious examples of this characteristic are hot swapping and synchronization. A third deployability shortcut is seamless, out-of-the-server support for different programming codes. Serving applications means the server should actually serve applications written in the various languages likely to be encountered in a Web environment – an environment likely to become even more heterogeneous.
- **Manageability**: A technology that allows you to plug-and-play application components over multiple servers takes you only partway. You also need a single, comprehensive, and simple-to-use, Java-based management console that lets the administrator configure every feature of the server and refresh all application components. It also calls for a public management API that lets application software do the same thing.

In a nutshell, the best test of a true application server, as opposed to merely an infrastructure server, is to predefine application services, not just infrastructure. An application server should provide vertically integrated, server-based, ready-to-configure applications. By adhering to J2EE branding standards, an application server puts itself into a solid starting position to accomplish this. But providing superior interoperability, deployability, and manageability is just as critical.

bho@sybase.com

AUTHOR BIO

Billy Ho, as vice president of product solutions at Sybase Inc., is responsible for the strategy and development of the company's core products, including Sybase Enterprise Portal, middleware, and database products worldwide. He earned a BS in Engineering at McGill University and an MS in computer science at CalTech.

J2EE ROADMAP

The Java2 Platform, Enterprise Edition defines the APIs for building enterprise-level applications.

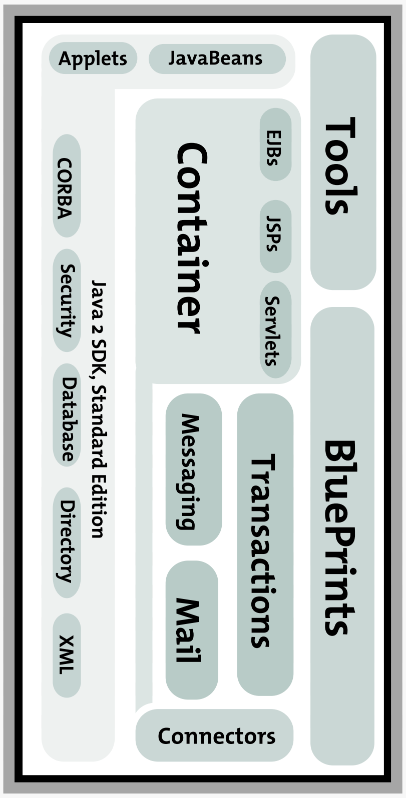
J2SE	v. 1.2
Enterprise JavaBeans API	v. 1.1
Java Servlets	v. 2.2
JavaServer Pages technology	v. 1.1
JDBC Standard Extension	v. 2.0
Java Naming and Directory Interface API	v. 1.2
RMI/IIOP	v. 1.0
Java Transaction API	v. 1.0
JavaMail API	v. 1.1
Java Messaging Service	v. 1.0

Useful URLs:
Java 2 Platform Enterprise Edition
http://www.java.sun.com/j2ee/

J2EE Blueprints
http://www.java.sun.com/j2ee/blueprints

J2EE Technology Center
http://developer.java.sun.com/developer/products/j2ee/

J2EE Tutorial
http://java.sun.com/j2ee/tutorial/

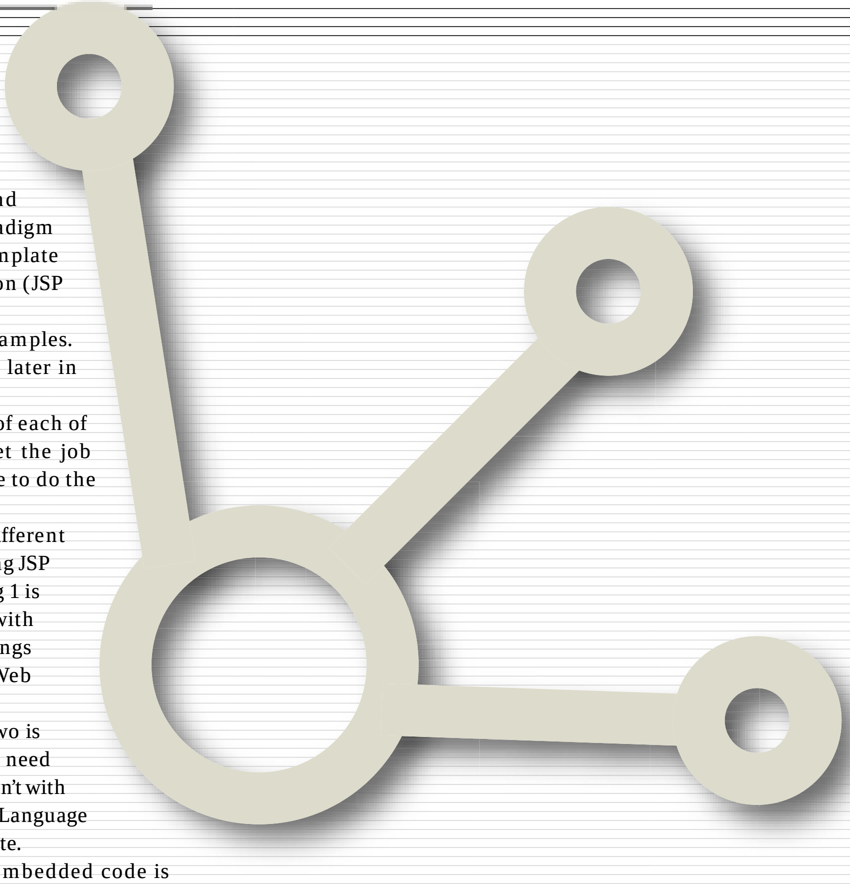


JSP You Make the Decision

Written by Jon Stevens

SAY HELLO

If you are currently looking for alternatives to JSP and you're tired of reinventing the wheel each time, then this article can provide you with the solutions you need to build your Web applications. We'll explore what it's like to develop a Web application using a couple of popular tools that are available today - and I'll use examples to show what it's like to use these technologies on a daily basis.



I'll start by comparing usage of Velocity and JSP. In both cases, we have a framework of reusable code that makes life easier for building Web applications by providing the "Model" and "Controller" portions of the MVC paradigm (Turbine and Struts). There is also a template language that provides the "View" portion (JSP and Velocity).

First let's start with some easy examples. We'll show more complicated examples later in the article.

We will not compare the feature set of each of these technologies as they both can get the job done. Instead we'll compare what it's like to do the job with each of the tools.

In the first example, we show two different approaches to doing the same thing using JSP and Velocity (see Figures 1 and 2). Listing 1 is an example of printing out a parameter with JSP, and Listing 2 with Velocity. (The listings for this article can be found on the ***JDJ*** Web site, www.JavaDevelopersJournal.com.)

The primary difference between the two is the way output is performed. With JSP you need to embed "code" within <% %> tags; you don't with Velocity. Simply use the Velocity Template Language (VTL) directly in any portion of the template.

The benefit (and detriment) of the embedded code is that the JSP code within a page won't show up when the file is simply loaded into the browser. However, there might be times when you want it to show up (for example, in debugging).

Another issue with JSP is that even the most basic examples start to blow the whole MVC paradigm right out of the water. The reason is that embedding HTML within Java code is a bad design decision; it makes it more difficult to modify the look and feel of an application at a later date. It also destroys the concept of MVC separation in which the View (HTML) display of the page is separated from the Model and Controller. For example, if just the word "Hello" had to be in bold, we would need to embed <b> </b> tags into the out.println() statement.

Of course, people in the know would recommend that we write JSP as in Listing 3, or with Struts as in Listing 4. Figure 3 shows the new JSP when the code in Listing 3 is loaded directly into the browser.

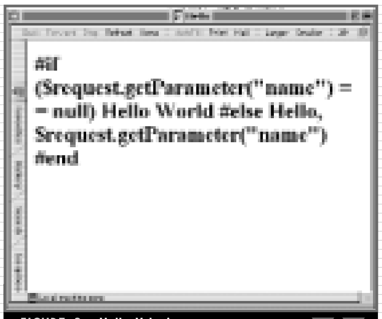
The point is, to make JSP "pure", you need to jump through hoops. Figure 3 looks similar to Figure 2, however, you still need to embed the necessary <% %> tags everywhere. More typing – more chances for mistakes! There's also a bit of a disconnect as to when the ";" needs to be added and when it doesn't. With Velocity, the rule is that you place a # before a directive and a $ before a member in the context.

As you can see from the example image, there's now a bit more information in the displayed page, except it's also missing all the logic that was used to build the page. If you view the equivalent Velocity template directly in the browser without it being rendered first, all the logic in the template remains visible. The advantage is that it's easier to debug problems.

JSP touts as an advantage that it takes an existing .jsp page and compiles it into a servlet for speed and efficiency reasons. This means that it first parses the .jsp page into a Java File, then uses javac or your favorite Java compiler (e.g., Jikes) to compile that servlet into a .class file, which is then loaded by the servlet engine.

The point is that using JSP is now a multistep process. The authors of JSP have done a good job of hiding this process behind the scenes in such a way that you don't even notice it. You simply edit the .jsp page in their favorite editor and then use their browser to call the page via a URI that starts the process of transforming it into a .class file.

FIGURE 1 Hello JSP


```
#if
($request.getParameter("name") =
= null) Hello World #else Hello,
$request.getParameter("name")
#end
```
FIGURE 2 Hello Velocity

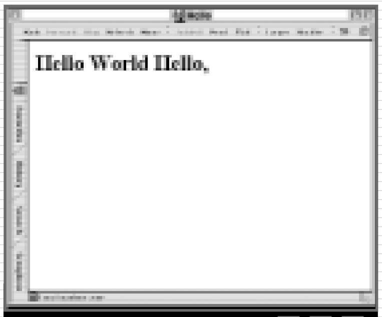

Hello World Hello,

FIGURE 3 Hello "Pure" JSP

This whole process of edit->transform->compile->load->run is really unnecessary and, in particular, a bad design. The reason is that it could be made much simpler.

Velocity simply loads templates, parses them a single time, and then stores an Abstract Syntax Tree (AST) representation of the template in memory, which can then be reused over and over again. The process is simply edit->parse->run. The benefit is that working with Velocity templates ends up being much faster and it also eliminates the requirement of having a javac compiler and temporary scratch directory hanging around. In Velocity, when the template changes, the existing cached template is simply replaced with a freshly parsed version.

Another advantage to Velocity's approach to templates is that the actual template data can be stored anywhere, including a database or remote URI. By using configurable template loaders, it's possible to create template loaders that can do anything you want.

Even without Turbine, Velocity offers several ways to deal with errors. Frameworks such as Struts and Turbine come in handy by providing ways of properly dealing with errors. However, since Struts is based on top of JSP, it also inherits the same amount of problems associated with JSP. The next section discusses it in more detail.

One final problem, shown in Figure 4, is that the JSP page catches only exceptions. What if the code within a JSP page throws another exception like OutOfMemoryError? The problem here is that OOME is based on throwable, not exception. Therefore, it's much more difficult to catch this exception with just a JSP page. Future versions of the JSP spec and implementations will improve on this.

Figure 4, provided by our friends at NASA, sends multibillion=dollar equipment through the heavens; it's a perfect example of why JSP needs better error handling.

Buffering is also another big issue since constantly writing to the output stream is not very efficient.

```
    <%@ page buffer="12kb" %>
<%@ page autoFlush="true" %>
```

These examples tell JSP to buffer the output 12KB and to autoFlush the page. Struts plus JSP has implemented the MVC model by providing the View portion through JSP templates. What part of the MVC model do you think those tags belong to? You guessed it, not the part where they're being used.

Velocity deals with this issue by allowing the developer to pass a stream into the rendering engine. If there's an exception thrown during rendering, it can be caught and dealt with. Buffering is also handled by passing a properly buffered stream to the parser. Again, if there's an error, another stream can be easily substituted for the output.

Listing 5 provides an example of the intermediate code generated by Tomcat 3.3m2.

Error Handling

This is a good one and a fundamental design issue with JSP. The question is: How many different types of errors can you get when using JSP? For example, because the JSP servlet is autogenerated from a .jsp text file and then compiled with a compiler, what happens when there's a generation/parsing error or a compile error? The unnecessary complexity of JSP actually increases the number of ways to get errors.

The ugliest aspect of all of this is that the errors are reported via two different mechanisms. The parser can throw one set of errors and the javac compiler can throw a whole different set of errors and, as a result of the layers of generation, errors from the compiler generally don't make any sense whatsoever. For example, can you tell me what this error is from?

```
 org.apache.jasper.JasperException: Unable to compile
class: Invalid type
expression.
                 out.println("JSP is great!")
                            ^
: Invalid declaration.
                 out.write("\r\n\r\n\r\n");
                          ^
2 errors
```

If you guessed that it was a result of a missing ";" after the first out.println(), you were correct. Now, put yourself in the shoes of someone who has never written or seen a line of Java code. Do you think that person could have figured out the error quickly and easily? Compound that with the fact that if the error had been on a less deterministic part of the file, it would be much harder to find the source of the error because there's a level of abstraction from the original .jsp file and an intermediate .java file that gets generated.

Again, Velocity doesn't suffer from these same problems because there's no intermediate step and no layers of abstraction.

```
  <%@ page errorPage="/error.jsp" %>
```

JSP also allows you to define an error page that's used if a throwable exception is thrown during the processing of a page. Doesn't this again break the MVC model? In other words, shouldn't the application framework be responsible for dealing with error messages?

```
  <% throw new Exception("oops"); %>
```

To throw an Exception somewhere in a JSP page, you need to first embed it within a statement. *Note:* In this specific case, if optimizations are turned on in the compiler, chances are the entire exception would be compiled out. Therefore, a more concrete object must be used instead of the "true." This can actually prove difficult if using a strict MVC model because instantiation of objects breaks the View.

```
  <%
  if (true) {
    throw new Exception("oops");
  }
%>
```

The reason is that JSP will generate an additional out.println ("\r\n"); after the Exception. When javac attempts to compile the page, another hard-to-debug error will be reported:

```
  org.apache.jasper.JasperException: Unable to compile
class for
  JSPC:\engines\jakarta-
tomcat\work\localhost_8080%2Fjsp\
  _0002ferrorMaker_0002ejsperrorMaker_jsp_3.java:75:
Statement not reached.
                 out.write("\r\n");
                 ^
```

Taking a direct quote out of Jason Hunter's book, *Java Servlet Programming* (I couldn't say it better myself):

> In fact, there are many such "gotchas" when using scriptlets with JSP. If you accidentally write a scriptlet instead of an expression (by forgetting the equal sign), declare a static variable inside a scriptlet (where statics aren't allowed), forget a semicolon (they're not needed in expressions but are needed in scriptlets), or write anything but perfect Java code, you're likely to get a confusing error message because the compiler is acting on the generated Java code, not on the JSP file. To demonstrate the problem, picture if <%= name %> were replaced by <% name %> in errorTaker.jsp. Tomcat generates this error:

```
org.apache.jasper.JasperException: Unable to compile class
for
  JSPC:\engines\jakarta-tomcat\work\localhost_8080%2Fjsp\
  _0002ferrorTaker_0002ejsperrorTaker_jsp_6.java:91:
Class name not found.
                 name
                 ^
```

Debugging an error like this often requires a programmer to look at the generated code to reconstruct what caused the error.

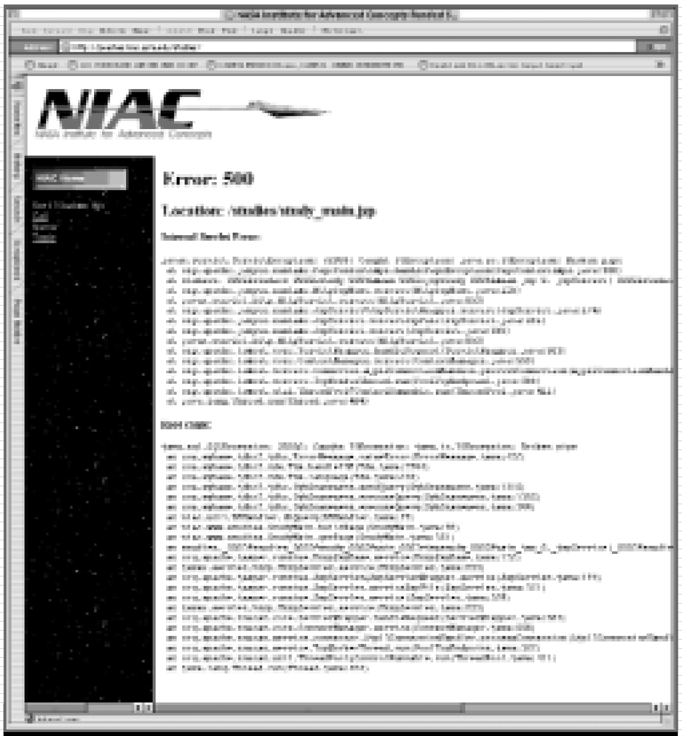
Velocity doesn't have these same problems since it doesn't allow the author to place any Java code within a template. The only things allowed in the template are Velocity Template Language (VTL) and method codes.

Everything else is considered "text" for literal output by the parser. The only place where you could run into trouble within Velocity is if there's a call to a method that throws an exception during runtime. For example, this VTL defines a string $foo, but attempts to call its substring() method on it would throw an IndexOutOfBoundsException:

```
  #set ($foo = "bar")

#set ($bar = $foo.substring(0,10))
```

When the exception is thrown, the parser will stop processing and throw that exception up the stack tree where it can be caught in the method that caused the parser to execute. At that point, the exception can be handled gracefully. Yes, this is something that's not easily debugged by a designer who doesn't know Java; it's easily debugged by a template engineer who has limited Java knowledge.


NIAC

Error: 500

Location: /studies/study_main.jsp

Internal Servlet Error:

[illegible]

Root cause:

[illegible]

FIGURE 4 The JSP page catches only exceptions

This is one of the benefits of using Turbine combined with Velocity: because of Turbine's design it's easy to deal with Exceptions in a consistent manner. It's also possible to get this same functionality by using Velocity's included VelocityServlet. The Exception will contain the line and column numbers in the .vm file where the error happened. Since there's no abstraction as with JSP, the line number and column matches up to the error. Also, the only tool that will throw the Exception is the Velocity parser, and that Exception will contain the location and error information pertinent to your actual template, not an intermediary file. No need to try to debug the cryptic javac messages that are a result of generated Java code. *Note:* Some application servers offer better handling of matching the line number in the JSP file to the error.

JavaBeans

JavaBeans are the way to use Java objects from JSP pages in order to follow the MVC design pattern. The point of doing this is to implement something similar to the Pull methodology, for example:

```
  <jsp:useBean id="name"
scope="page|request|session|application"
              class="className" type="typeName">
```

Examining the syntax of the above code, the first thing that pops up is the use of the scope attribute. How many HTML designers understand the programming concepts of scope? It's safe to suggest that a good portion of Web designers barely understand the concept of how a CGI works. By stating this, we're not trying to slight people. Instead, we're simply pointing out that design and software engineering are distinct skill sets. You wouldn't expect a Java programmer to select a print- and Web-safe color palette, would you?

The common response to an argument like this is that the designers should simply ignore these tags and let others define and implement them. The problem with that is that you've now given them the power to accidentally wreck your entire application in such a way that it's very difficult to debug because a complex scope issue might not show up right away.

The Java code:

```
public class HelloBean {
  private String name = "World";

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
}
```

The JSP code:

```
<jsp:useBean id="hello" class="HelloBean">
  <jsp:setProperty name="hello" property="*" />
</jsp:useBean>

<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
Hello, <jsp:getProperty name="hello" property="name"
/>
</H1>
</BODY>
</HTML>
```

The above code is a simple example of using a bean in a page. Pass the bean some properties and then retrieve the results. This is the right way to do things when using JSP. However, if we look at an example of the same thing in Velocity, the extra amount of typing that you need to perform to simply retrieve a property seems a bit much. Of course, there are always GUI-based, drag-and-drop tools to make typing a thing of the past. Really.

There are several commercial solutions available that provide a nice drag-and-drop view for developing with JSP and Struts. However, many of these tools are still first generation. They typically address only parts of the problem and require digging down into the nitty-gritty stuff when things become difficult or even impossible to do with the GUI. Often these tools also produce code that isn't optimized for heavily hit sites, and getting an existing application to scale sometimes requires a complete rewrite. Another item to note here is that these are costly ($1000/seat) development tools. In this economy, who really has the money to spend on these tools?

The Java code:

```
context.put ("hello", new HelloBean());
```

The Velocity code:

```
$hello.setName("*")
<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
Hello, $hello.Name
</H1>
</BODY>
</HTML>
```

This code example creates the HelloBean() object and then places it into the context. Then, during runtime execution of the template, that object is available as a $variable that uses the JavaBean specification to do introspection on the object. For example, Velocity uses bean-style introspection to permit the method call to be shortened from $hello.getName() to simply typing what's shown above.

When Velocity is combined with Turbine, the HelloBean object can be added into the context as a configuration option or it can be added at any point in the processing. This is what provides the "scope" of the object in the context.

Another "gotcha" with using JavaBeans in JSP is again quoted from Jason's book:

> One thing to watch out for: On some servers (including Tomcat 3.2) if you have a bean with a scope of "session" or "application" and you change the bean class implementation, you may get a ClassCastException on a later request. This exception occurs because the generated servlet code has to do a cast on the bean instance as it's retrieved from the session or application, and the old bean type stored in the session or application doesn't match the new bean type expected. The simplest solution is to restart the server.

Sample Application

Velocity developers believe that you shouldn't have to specially code your applications to work around issues that are related directly to Java.

In other words, one of the strong arguments of the JSP/Struts community is to say something to the effect of: "This is a poor example of using JSP." While this may be true, the fact is that nearly every example available is a poor example of using JSP. This goes back to the statement: embedding Java code in your page is a bad thing. Yes, we all know that now.

The truth is, it's hard to use the tool correctly. Struts is doing an excellent job of making it easier and attempting to show the right way, however, it's simply hiding the ugliness of the original design of JSP.

Object-oriented design dictates that you extend a class to add functionality to the base class. The publicly available methods in the base class are still available to the classes that extend it. Putting Struts on top of JSP doesn't fix the warts in JSP. It simply hides them until your developers find them (see Listing 6).

Listing 7 translates Listing 6 to Velocity.

Taglibs

Taglibs are intended to be the savior of JSP. They're billed as the way to extend JSP so that there's a nice MVC abstraction while still adding functionality to the base "language." Struts has concentrated a good portion of its efforts here by providing a nice taglib library.

The advantage of using taglibs is that they allow you to extend the "language" syntax of JSP so you can provide the things that are missing from it, but are available in Java. In other words, instead of encouraging people to embed Java code within their pages, this has been abstracted to encouraging people to embed XML tags in their page. How is this any better than what ColdFusion did with their product? Sure, plenty of people are using ColdFusion, but is that the best way to do things?

Listing 8, borrowed directly from the Struts documentation, shows how logic would be embedded into your JSP page.

Listing 9 shows the same example using Velocity.

Of course Listing 9 could be written even cleaner as:

```
  #if($number == 7)
  You guessed right! You win a high speed blender!
#elseif( $number < 7 )
  lower
#else
  higher
#end
```

This really falls into a preferences situation. In other words, which syntax would you prefer to use? It seems as though the amount of typing required to implement the taglibs approach would be a major deciding factor for many people. One reason is that the more typing that needs to be done, the more chances for errors. Listing 10 provides another example taken from the Struts documentation.

It's clear from Listing 10 that the whole strict MVC model has been broken again because a call to java.util.ArrayList and creating an Object is embedding Java code within a template. Compound that with the need to place that ArrayList into the pageContext makes it even more confusing. In addition, the designer has to remember to use cryptic code at the top of the file that references a taglib document as well as declares a prefix attribute. Why do things need to be so overly complicated?

The same example using Velocity would be written like this:

```
  #set ( $list = ["First", "Second", "Third",
"Fourth", "Fifth"] )

#foreach ( $item in $list )
  Element Value: $item<br />
#end
```

Moving on with examples, we come back to the previous sample application from Jason Hunter's book that was shown before. This time it has been implemented entirely within the Struts framework (see Listing 11).

At this point, we now have a combination of standard JSP tags as well as Struts-specific tags. The use of Struts has appeared to clean things up significantly with regards to embedded scriptlets. Note that quite a few of JSP's warts are still shining through. Is it as easy to understand as the Velocity version?

Velocity provides a simpler solution designed around a few core template tags. A scripting language is similar to PHP, which may require months of usage to completely master. A template language can be mastered in just a few hours. This is what makes Velocity a template language instead of a scripting language.

For example, Velocity has the following core template tags (otherwise known as "directives"):

```
#if
#else
#elseif
#foreach
#set
#parse
#include
```

It's possible to add more #directives to Velocity either by adding them directly to the parser or through an API. It's also possible to use a neat feature of Velocity called *Velocimacros*, documented at http://jakarta.apache.org/velocity/user-guide.html#velocimacro.

Some people suggest that the benefit of using JSP and Struts is that they simply extend HTML, something designers already understand. This is a powerful argument. However, the reality is that HTML is not a template language; there's no logic in HTML. For example, it's not possible to embed conditional statements (as in Listing 1) into HTML.

What this means is that taglibs allow developers to infinitely extend HTML to become much more than it previously was, almost like inventing another scripting language. This is reminiscent of the early browser wars in which each company was implementing their own browser tags. Netscape went so far as to invent the <blink> tag. What did we learn from that? Is that really a good thing? Sure, standardizing taglibs is a great idea. Will it ever become widely adopted? We sure hope so.

One last point to make about using HTML syntax is that this really pigeonholes JSP and Struts into simply being tools for creating dynamic HTML/XML/WML (tag markup) code. Velocity on the other hand is designed to take any type of text (Java, SQL, HTML, etc.) as input and output anything as a result of running it through its processing engine.

Embedded Usage

The following is an example of using Velocity within an application.

```
      OutputStreamWriter writer = new
OutputStreamWriter(output, encoding);
    Template template = RunTime.getTemplate(filename);
    template.merge( context, writer );
```

In other words, the above is translated into:

1. Create a Writer.
2. Ask the Velocity RunTime to retrieve the Template.
3. Merge the Context with the Template.

Velocity templates can contain *any* type of data – XML, SQL, HTML, plain text, internationalized text – anything. The parser is very robust in terms of parsing out only what it cares about and ignoring the rest. As just demonstrated, it's possible to embed Velocity's template engine into any other Java code. The advantage of this is that it provides users with a single template system for all sorts of uses.

For example, in the PetStore example on the J2EE Web site, http://java.sun.com/j2ee/blueprints/sample_application/index.html, there's a JavaBean that sends an e-mail confirmation when an order is complete. The unfortunate part about this example is that in order to change the contents of the e-mail, you need to edit Java code. Of course, this example could have been done differently to use a JSP-based template as the base format. However, the implementation of this is more difficult than simply creating a Java object that resides in the context and can render the template as arguments.

Another example of this is the Texen and Anakia tools that come with Velocity. The advantage again is the ability to use Velocity as an embedded tool to produce other useful applications that are not bound strictly to a Web application. While it's possible to do this with the JSP engine, it's significantly more difficult.

Implementation

One of the touted advantages of JSP is that it's a "standard" and quite a few people like to hold this in high regard. So much so that they refuse to use any technology that's not "standard." Digging into the reality of this statement reveals that the important correct terminology is that JSP is a "Sun Standard Specification," not strictly a "standard." This is important because JSP is really no more "standard" than Microsoft ASP or the PHP Group's PHP product. In other words, whatever tool you happen to be using becomes the "standard."

A small group within the Java Community Process (JCP) defines what JSP is. There's a fairly high barrier to joining the JCP because an NDA must be signed, the project leads must approve your entry, and in some cases a fee must be paid. You could even go as far as to say that the JSP specification is really a proprietary product of Sun.

It's important to note at this point that I'm a member of the JSR-053, which defines the Servlet and JSP specifications.

Inside JSR-053 it's clear that not everything is done in the open and decisions are made behind closed doors. Of course, the participants could object, but Sun still is the binding force behind the decisions.

JSP is both a specification and an implementation. There are various corporations (as well as open source) implementations of the specification. The JSP specification isn't a particularly easy thing to implement. There are many complex systems involved, some of which even require special hooks into the servlet engine in order to obtain optimal performance.

The JSP specification is relatively new (it's still in a 1.x phase). This means that the specification also has several places in it that are not as well defined as others, leaving some of the specific details to be determined by the implementation. Not only does this mean that there are plenty of places to make mistakes, but it also means that JSP template code might not behave the same across implementations. This makes testing JSP-based applications across several different containers a nightmare, especially if there are esoteric bugs (meaning bugs that show up only under extreme conditions) in one and not the other.

Part of the reason for creating the Jakarta Project and having Sun release the source code to Jasper (the JSP reference implementation) is to encourage vendors to adopt a single base for their source code. Unfortunately, this has not happened. There are compatibility testing suites available; however, there's no policy in place requiring vendors to pass the tests. Nor is there a place that shows vendors who don't pass the tests in order to publicly humiliate them into submission.

The user and developers of the velocity project are the ones who define the Velocity specification. They've created a comprehensive testing suite that's used for regression testing to ensure that all changes to Velocity won't break existing templates. If you or your company ever want to join the development of Velocity and help determine its direction, that can be easily done by simply joining a mailing list and contributing to the development effort. This means that no one corporation will ever own the Velocity specification and that it will always work with your templates within any servlet engine.

Hosting

JSP is great for hosted environments (e.g., shared development environments, ISPs, and large companies) in which many different kinds of people who are putting pages on the servlet engine may not know much more than HTML. One reason it's so nice is that with the HTML'ish syntax, JSP is so easy to learn.

However, if you look at that statement in more detail it becomes apparent that this might not be such a good thing. Consider the following snippet of Java code:

```
   Hashtable strings = new Hashtable();
  int i=0;
  while (true)
  {
      strings.put ("dead"+i, new
StringBuffer(999999));
  }
```

This creates a Hash table and then goes into a tight loop. At the same time it creates a new StringBuffer object that has a default size of 999999, creating what amounts to a memory leak in the Java Virtual Machine (which all the hosted applications share).

As soon as all the memory is gone, every single hosted application will start to receive the dreaded "OutOfMemoryError." The reason why this is so bad was explained earlier. Essentially, JSP pages can't catch OOME errors and the entire servlet engine will suddenly become useless, all because of a few lines of badly written Java code.

Remember, it's a bad idea to put Java code into a JSP page. Tell that to the 14-year old kid who is being hosted on your ISP's servlet engine and really doesn't care that others might be affected by these actions.

Velocity doesn't have this issue because there's no while loop in the Velocity Template Language. The only looping construct in Velocity is a #foreach and that loops over an array that has a finite amount of elements. It's also possible to disable the #foreach directive and limit the amount of recursive nesting that's possible.

Currently, there's nothing in JSP or Struts that prevents people from embedding Java code in the page. As a result, Sun has developed a way to validate JSP pages so that they conform to a specification that you can define. In the end, this results in the need to remove the ability to place scriptlets into the templates. However, scriptlets are a major part of JSP – and the examples people give show that this effectively will break nearly every template in existence.

Conclusion

We hope that you enjoyed this little excursion into the land of alternatives. Every developer and user in the world has his or her own opinion on the "right way" to do things. The goal of this document is not to force you to believe that the Velocity way is the "Right Way," but to let you see the differences between the two approaches and develop your own opinions about the tools that are in use today.

Some people may disagree strongly with this article because it challenges conventional thinking as well as the way that they may be doing things today. It's difficult to show people who believe strongly in something that there might be another solution out there that solves problems in a different way. So please send us your feedback .

AUTHOR BIO

Jon Stevens is an Apache Software Foundation Member and the cofounder of the Java and Jakarta Apache projects. He is a cofounder of the Velocity and Turbine projects. He has been developing Web applications since 1994 in various languages and platforms. In 1998, he started to build the Turbine framework in an effort to develop open source tools to help developers build complex Web applications.

jon@latchkey.com

Accelerating Java Web Application Environments with Dynamic Content Caching

The rise of dynamic Web sites

WRITTEN BY HELEN THOMAS

E-business sites are increasingly utilizing dynamic Web pages since they enable a much wider range of interaction than static HTML pages can provide. Dynamic page generation, also known as dynamic scripting, allows a Web site to generate pages at runtime, based on various parameters.

Delaying content decisions until runtime provides several advantages to Web sites, including greater personalization and interactivity. For instance, a site might recognize and greet a returning customer based on his login (or perhaps from a cookie stored on his machine), e.g., "Welcome, John Smith." A more elaborate customization scheme might provide John with a set of recommendations, based on his past purchases.

Still another advantage of the dynamic content generation paradigm is that it allows greater control over content. Updates to content can be made in a single place (e.g., a database), rather than to every single HTML file, as is done in the case of static pages. This aspect allows a site to truly separate presentation from content.

Java is the development environment of choice for dynamic sites for a number of reasons. The Java platform simplifies enterprise application development by utilizing standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without the need for complex programming.

These features are part of the Java 2 Platform, Enterprise Edition (J2EE) standard, which provides many features including portability, support for Enterprise JavaBeans (EJB) components, Java Servlets API, and JavaServer Pages (JSP). Another advantage of using the Java development platform is that there is significant vendor support for Java-based scripting technologies among the high-end application server products.

The Impact on Server-Side Performance

While the dynamic scripting paradigm has provided the above-mentioned benefits, it has simultaneously resulted in serious performance problems due to the increased load it places on the site infrastructure. Each request for a dynamically generated page invokes a program on the application server, which in turn contacts various resources in order to generate the requested HTML page. The specific delays associated with dynamic content generation include (but are not limited to):

1. Delays due to accessing persistent storage (e.g., databases, file systems)
2. Network delays due to accessing remote resources (e.g., legacy systems, databases, external data feeds)
3. Data formatting and transformation delays (e.g., XML-to-HTML transformations)
4. Interaction bottlenecks (e.g., database connection pools)
5. Overhead of JVM garbage collection
6. Overhead of script execution

As user load increases, the effects of these delays are compounded, significantly impairing site scalability.

Performance Acceleration Solutions

In order to mitigate this list of content generation delays, a number of Dynamic Content Caching solutions have emerged. The basic idea behind these solutions is to reuse content that has already been generated to service a request; thus the redundant work in content generation is eliminated. The benefits of deploying such solutions are threefold: faster response times, higher throughput, and lower infrastructure requirements.

Dynamic Content Caching solutions can be categorized into two broad types: 1) application-specific solutions and 2) purpose-built solutions.

Application-Specific Caching Solutions

Application-specific solutions include caching functionality as an "add-on" feature. For instance, a num-

"Dynamic Content Caching is a valuable **technology that should** be built into any new Java site's design and architecture"

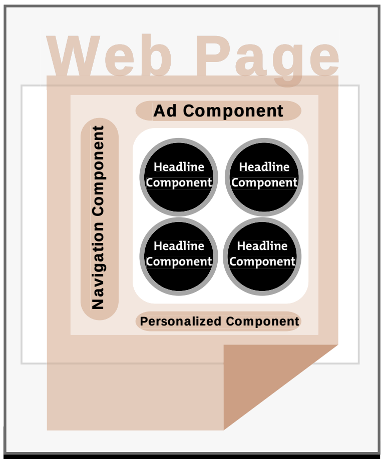


FIGURE 1 Dynamically generated news page

ber of application server vendors have recently incorporated JSP cache tagging into their products. One such vendor is BEA, with the introduction of WebLogic Server Cache Tags in WebLogic Version 6.0.

JSP cache tagging solutions allow page and fragment level JSP output to be cached. Fragment-level caching is enabled by marking or tagging the script code that produces cacheable output. When the script executes, if a tagged fragment is found in cache, then the script code that generates the fragment is bypassed.

A key advantage of application-specific caching solutions is that they are cost-effective since the caching features are included with the product. There are a few drawbacks to be aware of, however. For instance, JSP cache tags are only capable of presentation-layer caching; thus these solutions provide little value for sites based on the Model View Controller (MVC) design paradigm, as virtually all of the "real" work is done within the servlets and business components.

Another disadvantage of these solutions is that the caching process competes with the numerous other application server tasks for both CPU and memory resources. Furthermore, clustered implementations require multiple caches, creating issues in maintaining cluster-wide cache coherency.

Purpose-built solutions are typically standalone products designed specifically to address the delays caused by dynamic page generation. They can be further classified into two categories: page-level and component-level caching.

Page-Level Caching

Page-level caching solutions intercept requests for specific HTML pages and cache the resulting output. For instance, consider the following request, which displays a page corresponding to a particular product category in an online catalog:

```
"http://www.xyz.com/catalog.jsp?cat
alogID=100"
```

As long as the URL uniquely identifies the page that will be displayed, a page-level cache can store the resulting content to fulfill future requests. Vendors of Java-based solutions in this category include Oracle and Persistence Software.

There are a number of advantages associated with page-level caching solutions. Since the entire page is cached, these solutions can completely offload work from the application server, as the request never reaches that resource. In addition, these solutions allow some degree of personalization (e.g., in cases where users sharing the same preferences are served the same content), and they are relatively simple to implement.

On the other hand, there are several limitations to consider. By far, the most significant drawback is that caching only applies when the URL uniquely identifies a page. This is often not the case in dynamic Java-based sites, where complex business logic in the script determines the page that is created and served. In addition, on highly personalized sites page-level caching results in low cache hit rates since each page instance is unique to a user.

Component-Level Caching

A component-level caching solution stores and serves dynamic page components (e.g., stock quotes, product descriptions, weather reports) that are reusable across multiple-user sessions, obviating significant work for the application server. For instance, consider a dynamically generated page in a news site, as shown in Figure 1.

The page consists of multiple components: Ad, Navigation, Headlines, and Personalized. While each page instance may be unique due to the Personalized component, it is likely that much of the content (e.g., Navigation, Headlines) will be the same for multiple requests during a given time interval. Component-level solutions capitalize on such reusability by caching and serving these components, eliminating the need for the application server to create them again and again.

Implementation of component-level caching requires identification of the specific components that cause performance bottlenecks. The corresponding application code that generates such bottleneck components is tagged, which involves inserting an API into the script. When the script executes, the API instructs the application server to check for the component in cache before generating any content.

The goal is to fulfill a high majority of requests from cache, thus producing site-wide infrastructure efficiencies. The only vendor in the component-level caching category is Chutney Technologies.

With component-level caching, cache invalidation policies are vital. Components must always be kept up-to-date, so that "stale" data is never served. Several strategies for cache invalidation exist, the most popular of which are time-based, event-based, observation-based, and on-demand.

Time-based policies are the most straightforward, and basically flush components at pre-set time intervals. Event-based invalidation is keyed on an external occurrence, such as a database update made by another application. In this case, the database is programmed to send a notification to the cache when a component's value changes. Observation-based policies are utilized by sites where updates to data sources occur inside the scripts, such as with exchange or auction sites.

Finally, on-demand invalidation allows keywords or regular expressions to be used to remove cached components at the site administrator's discretion. To maximize flexibility, a component-level cache allows the use of all of these methods concurrently to accommodate the data's unique characteristics.

There are several advantages to using component-level caching. Such solutions deliver high cache hit rates since most components have high reusability levels, if only for a brief time period. This factor enables even highly personalized sites to achieve significant performance benefits.

In addition, component-level caching solutions typically reside on a separate machine, eliminating the resource contention problems that are inherent with application-specific solutions. This "separate-box" architecture allows for enterprise-class scaling, as a single component-level cache can interface with a rack of application servers. Also, unlike application server cache tagging, component-level caching can be applied within nested script calls at the business logic layer, where much of the work is done in Java-based architectures.

AUTHOR BIO

Helen Thomas is a cofounder of Chutney Technologies and a technical expert in the area of decision-support databases. She received an M.S.E. degree in operations research and industrial engineering from the University of Texas at Austin, and a B.S. degree in decision and information sciences from the University of Maryland at College Park. She is currently a Ph.D. candidate in the DuPree School of Management at Georgia Tech.

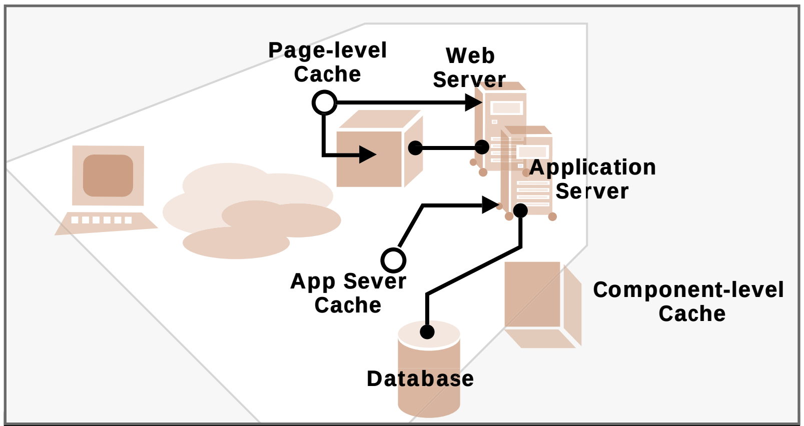


FIGURE 2 Performance acceleration solutions in Website architecture

A disadvantage of component-level caching solutions is that they require identification of the specific components that cause the most severe delays. This is either accomplished through site-specific knowledge or load profiling tools. After these components are identified, code-level integration is necessary to insert the appropriate caching tags within the scripts. Therefore, component-level caching's main tradeoff is one of implementation simplicity for maximized performance.

Putting the Solutions into Context

Figure 2 provides a simplified schematic indicating where each of these solutions fits into an end-to-end Web site architecture. Application-specific solutions, such as JSP cache tags, are part of the application server process, and therefore reside on the application server machine itself. Page-level caching solutions interact with the Web server, and may reside on either the Web server machine or a separate system. Component-level caching solutions interact with the application server, and reside on a separate machine.

Conclusion

Java is the technology of choice for enterprise e-business Web sites. Dynamically generating pages is highly advantageous but causes strain on a site's infrastructure There is a set of solutions (collectively known as Dynamic Content Caching) that are designed to alleviate these resource issues. A specific solution should be chosen based on a site's architecture, development plans, and business objectives, but some general guidelines do apply:

- Application-specific caching solutions are best suited for sites that are not clustered and where content is primarily generated at the presentation layer.
- Page-level caching solutions are most appropriate for sites where seamless integration and simplified management are critical.
- Component-level caching solutions are optimal for highly personalized sites that will perform code-level integration to achieve enterprise-class scaling.

Summing up, Dynamic Content Caching is a valuable technology that should be built into any new Java site's design and architecture.

helen.thomas@chutneytech.com

BUILD TO SPEC!

Written by Liz Blair

Follow J2EE specifications to 'Write Once, Run Anywhere' on the server

Many J2EE 1.2-based applications and components are emerging in the marketplace as the J2EE platform matures. Application portability is one of the most important benefits offered by the J2EE platform. Through the J2EE Java Pet Store sample application, the J2EE Blueprints team has developed a set of best practices for ensuring application portability across J2EE-compatible application servers.

This article is the first in a series of J2EE application and component portability recommendations for the J2EE development community. This series will focus J2EE application portability in light of maturing J2EE technologies, especially J2EE 1.3 and EJB 2.0.

This first article presents several portability guidelines for the J2EE platform, version 1.2, but includes references to the upcoming version 1.3 where appropriate. This article assumes that the reader is familiar with basic J2EE 1.2 platform terminology.

What Is Portability?

Portability is an important goal for application developers who want to maintain freedom of choice in technology. It's important to understand how J2EE application portability fits into the Java portability picture. Java technology provides several levels of portability, each supported by a particular specification:

- ***Source-code:*** Allows a single source base to provide identical results, regardless of CPU, operating system, runtime system, or compiler. The Java language is highly source-code portable because its definition (the Java Language Specification) clearly spells out such details as byte order, memory management behavior, and the size and format of its primitive types.

- ***CPU:*** Allows a compiled program to execute identically on computers with different CPUs. The Java Virtual Machine provides CPU portability by specifying a virtual instruction set to which Java (or other language) source code compiles, isolating the compiled code's behavior from the underlying CPU.
- ***OS:*** Allows a developer to write a program that accesses system-level resources in a uniform way, regardless of the underlying operating system. The Java Platform provides OS portability by specifying a "virtual operating system" that gives developers a unified model of system services.
- ***J2EE application:*** Allows J2EE application developers to write client/server components and applications that deploy and execute identically regardless of the underlying server implementation or vendor. The J2EE Platform Specification defines how the various J2EE platform technologies (servlet containers, EJB containers, transactions, security, and so on) must behave, so J2EE application developers can depend on consistent behavior across vendor implementations.

This series focuses on J2EE application portability tips and guidelines. The J2EE Platform Specification defines a contract between J2EE product providers, who create J2EE servers, APIs, and so on, and J2EE component and application developers, who use those J2EE products to create solutions. Both sides of the contract must understand their responsibilities in order to realize the portability benefits of the J2EE platform. This article provides some detailed tips for ensuring that your J2EE components and applications will be portable between different J2EE vendors' products.

J2EE Product Compatibility

J2EE product providers produce Web and application servers, servlet and EJB containers, databases, platform APIs, and development tools. The J2EE Platform Specification defines the behavior of each type of J2EE product, so J2EE component and application developers can depend on consistent results. Several tools are available to help J2EE product providers create products that meet the platform specification requirements.

The J2EE Compatibility Test Suite (CTS) helps J2EE product providers check that their products meet the requirements of the J2EE platform specification. The J2EE Compatibility Test Suite is a standard set of over 5,000 test cases derived from the J2EE specification requirements. Products that pass the CTS are considered J2EE-compatible. The tests check specific requirements for each of the J2EE technologies. For example, the EJB container tests check that the EJB instance life cycle behaves as defined in the EJB specification.

The J2EE Reference Implementation (RI) provides J2EE server developers with an example implementation of a J2EE application server. The J2EE RI demonstrates that the specifications can indeed be implemented. The RI also gives application server vendors a working server they can use to check the correctness of their implementation. The J2EE RI is required to pass 100% of the CTS, since its purpose is to demonstrate the complete J2EE specification.

Application Developer Tools

Good tools play an important role in helping the J2EE application developer write portable applications. Providing powerful, easy-to-use tools is one of the ways that product providers can serve their customers, and thereby gain a competitive advantage in the marketplace.

Several kinds of application development tools provide support for application portability. Code generation tools generate code for various parts of an application, based on higher-level specifications provided by the developer. Verification tools ensure the correctness of deployment descriptors. Sample code and references help developers understand how to use application components. Let's look at an example of each of these types of tools.

Code generation tools automatically generate source code based on what the developer has specified or has already coded. Such tools improve portability by creating code that adheres to J2EE specifications. For example, a tool called EJBGen (v1.1.9 – an EJB 2.0 code generator) automatically generates the EJB Home and Remote interfaces for an existing EJB class. In addition to eliminating mindless coding, EJBGen improves portability and code quality by ensuring that the methods of the EJB class and its home and remote interface throw the proper set of exceptions as required by the EJB 2.0 specification. See the list of resources at the end of the article for more on EJBGen.

Verification tools ensure that the contents of a deployment descriptor (e.g., web.xml or ejb-jar.xml) are both well formed (meaning that the descriptor meets the basic requirements of XML) and valid (meaning that the descriptor matches the structure defined by its DTD). A deployment tool may choose not to deploy an application if it detects a problem in a deployment descriptor. The Sun RI provides a tool known as the "verifier" that verifies the contents of a WAR, JAR, or EAR file. The verifier runs a series of checks (based on requirements of the J2EE specifications) against the J2EE component or application provided. The verifier allows the developer or deployer to fix errors at, or even before, deploy time.

A good code example is worth a thousand words. Samples and references provide good examples to demonstrate how individual J2EE components, applications, and servers all come together. Many code snippets and tutorials for particular technologies are only a Web search away, but developers need to understand how to use these technologies together: they need to be able to see "the big picture".

The Sun J2EE Blueprints program provides developers with a free sample application, the Java Pet Store. The Java Pet Store is an end-to-end, B2C, e-commerce application that gives the J2EE application developer an example from which to learn. The J2EE Blueprints program also offers design discussion explaining best practices for J2EE application design, and provides a set of design patterns for building robust, scalable, portable J2EE applications. The Java Pet Store sample application can be downloaded for free from http://java.sun.com/j2ee/blueprints.

J2EE Application Portability Tips

The rest of this article presents several specific technology usage tips for maximizing J2EE application portability. Each tip may also have other benefits, but the focus is on portability between J2EE product providers' products. This section assumes a working knowledge of J2EE 1.2 technology.

Know When (and When Not) to Use Serializable Fields

As an Enterprise JavaBean provider, you are responsible for ensuring that all of a bean's instance fields are serializable, so that the state of the bean can be preserved when the bean is passivated.

While fields of primitive types like String and int are serializable, reference fields (fields whose value is a reference to a class instance) are serializable only if the reference's class implements java.io.Serializable and all of that class's nontransient fields are serializable.

For example, consider an entity bean called AccountEJB, which has a field identifying the account owner, and another field recording the account contact information (see Listing 1).

Since the ContactInformation class is serializable, and all of its fields are serializable, then AccountEJB's info field is also serializable, and the requirements are met.

You must mark all fields of nonserializable types as transient. Fields marked as transient are ignored by the JVM's serialization machinery.

For example, a database connection, represented by java.sql.Connection, is not serializable, so it is marked transient:

```
public class AccountEJB {
 <o:p>
    // ...
    private transient Connection dbConnection = null;
 <o:p>
    // ...
}
```

For more information on serialization and passivation, refer to Section 6.4.1 of the EJB 1.1 Specification (available for download from the resource list).

Close Database Connections Cleanly

When retrieving items from a database, open a connection through a Connection instance, and then execute an SQL statement through a Statement instance. When you're done, remember to close your Statement before closing your Connection (see Listing 2).

What happens if you try to do this the other way around? Some JDBC 2.0 driver implementations will throw an exception if you try to close a Connection with open Statement instances. This is a portability issue because different JDBC implementations handle this detail differently.

Be Careful When Writing Deployment Descriptors

If your J2EE-compatible application server provides tools to construct deployment descriptors for you, you should use those tools. Hand-written deployment descriptors are prone to errors such as misordering of elements, duplicate element entries, and simple typos.

If you still want (or need) to write your own descriptors, many application servers provide tools that validate a deployment descriptor against a DTD, ensuring that the descriptor's contents are both well formed and valid.

The verifier tool shipped with the J2EE Reference Implementation will catch only some errors in a descriptor, but not all of them. You can supplement the verifier with a small program that uses a validating parser to check a descriptor's validity against its DTD (see Listing 3).

Use Valid Encodings for Deployment Descriptors

Ensure that the static deployment descriptors for your Web and EJB components include a valid encoding declaration. For example, if you're using encoding ISO-8859-1, the deployment descriptor should begin with the following line:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

In an encoding declaration, the value ISO_8859_1 is not an acceptable substitute for ISO-8859-1.

Refer to Section 4.3.3, "Character Encoding in Entities," of the W3C XML 1.0 Recommendation for more details on XML character encodings.

Using Methods Correctly Is Vital

When an entity bean is activated through a callback to ejbActivate(), it is returned to the ready state so that business methods can be called on it. The activation method, however, is not responsible for loading the bean's fields from a resource; that is the job of ejbLoad().

Similarly, when an entity bean is passivated through a callback to ejbPassivate(), it's returned to the pooled state. The passivation method, however, is not responsible for storing the bean's fields into a resource; that is the job of ejbStore().

Even if you tried accessing a resource from ejbActivate() or ejbPassivate(), a container implementation may reject the access, on the grounds that the EJB 1.1 Specification does not allow such access from either of these methods. Using these methods correctly is vital for application portability.

For more information on these methods, refer to Sections 9.1.5 and 9.1.6 of the EJB 1.1 Specification.

Throwing Exceptions from Entity Beans

The EJBGen tool helps developers write EJBs, and in particular ensures that EJB method signatures express the appropriate thrown exceptions. If you do not use such a tool, then you should double check the EJB method signatures, especially the thrown exceptions, in your component's EJB class, Home and Remote interfaces.

Here is a summary of the EJB 1.1 Specification exception requirements for EJB Home, Remote, and EJB class method signatures:

Remote Interface:

- Throws clause must include RemoteException.
- All exceptions defined in the throws clause of the matching method of the EJB class must be defined in the throws clause of the method of the remote interface.

Home Interface – Session:

- All exceptions defined in the throws clause of the ejbCreate method must be defined in the throws clause of the matching create method in the Home interface. The throws clause must include CreateException.

Home Interface – Entity:

- All exceptions defined in the throws clause of the ejbCreate/ejbPostCreate methods must be defined in the throws clause of the matching create method (the set of exceptions must be a superset of the union of exceptions defined for the ejbCreate/ejbPostCreate methods).
- All exceptions defined in the throws clause of an ejbFind method of the EJB class must be included in the throws clause of the matching find method of the Home interface. The throws clause of the finder method must include FinderException.

The J2EE RI 1.2.1 verifier tool checks for method signature exceptions.

Packaging

How to package an application seems to be an area of confusion among developers in general.

Packaging is an important portability consideration. Packaging requirements and options are described in the Servlet 2.2, EJB 1.1, and J2EE 1.2 specifications. The Servlet and EJB specifications do a good job of describing what each application component package (WAR, JAR) should do and the J2EE specification describes how to pull it all together into an application package (EAR).

However, some concrete examples describing how you can package your application make the specifications more comprehensible. If you don't understand the specifications regarding application packaging, your application might not be portable.

The following scenarios present increasingly complex deployment situations, describe options for packaging them, and provide some considerations and notes on deployment for each scenario. Keep in mind that WAR file and EAR file formats are different.

SCENARIO #1: SINGLE EAR, WAR REFERENCES EJB

This first, simple scenario shows how to reference an Enterprise JavaBean in a single EAR file from a single WAR file.

```
 <o:p>
appA1.ear contains:
        x.war           References client view of
ejbA1.
        ejbA1.jar
```

Packaging Options:
- Include ejbA1_clientview.jar in the WEB-INF/classes directory of x.war
- Expand contents of ejbA1_clientview.jar into the WEB-INF/classes directory of x.war
- Include ejbA1_clientview.jar in your own specific WAR directory (for example, "clientviews"). Then the WAR can refer to the clientviews directory JAR file via its Class-Path entry in the manifest.mf.

War File Notes:
- The Web server should automatically recognize JAR files in the WAR file's WEB-INF/classes directory. A bug in the J2EE RI 1.2.1 causes JAR files located in the WEB-INF/classes directory not to be recognized.
- If a WAR file needs to reference an EJB, the WAR should do so via the client view of that EJB. A client view of an EJB contains all of the classes that a client program needs to access a referenced EJB.
- A packaging don't: don't try to access an EJB client JAR file from the WAR's manifest.mf Class-Path entry when that EJB client JAR does not exist in the WAR file. Remember, a WAR file has a different file format than a JAR file. You can't access EJB client view JAR files that are not local to the x.war file via the manifest.mf Class-Path entry.
- When a Web application's web.xml file references ejbA1 via an <ejb-ref> element, the Web container is required to be able to find the ejbA1_clientview.jar file automatically, assuming ejbA1.jar's deployment descriptor includes an appropriate <ejb-client-jar> element. See EJB 2.0 Specification, section 23.4.

SCENARIO #2: SINGLE EAR, EJB REFERENCES ANOTHER EJB

This scenario shows how to reference one EJB from another within a single EAR file.

```
 <o:p>
appA1.ear:
        x.war
        ejbA1.jar       References client view of
ejbA2
        ejbA2.jar
```

Packaging Options:
- Include ejbA2_clientview.jar in ejbA1.jar's Class-Path manifest.mf entry
- Include ejbA2_clientview.jar expanded in ejbA1.jar file.

EJB JAR File Notes:
- If an EJB has a client view available, it should specify the name of the client view JAR file in the <ejb-client-jar> entry of its ejb-jar.xml file.
- A packaging don't: EJB JAR files cannot reference a WAR file using a manifest.mf Class-Path entry. Remember, a WAR file has a different file format than a JAR file.

SCENARIO #3: SINGLE EAR, EJB REFERENCES UTIL CLASSES

This scenario shows how to reference utility classes from within a single EAR file.

```
 <o:p>
appA1.ear:
        x.war
        ejbA1.jar       References classes in
yUtil.jar
        yUtil.jar
```

Packaging Options:
- Include yUtil.jar in ejbA1.jar's Class-Path manifest.mf entry
- Expand yUtil.jar's contents into ejbA1.jar file.

Utility and Implementation-Specific Notes:
- A packaging don't: If you have common utility classes, don't duplicate the code in your application components. Instead, create a simple util.jar file and include the JAR file as needed (see above options for WARs and EJB JARs).
- If you've got implementation-specific code (i.e., deploy/undeploy), but want to provide a way to minimize the impact to different application servers, following these steps will help to minimize the effort to deploy the application on different servers:
  1. Provide an interface for a common look and feel.
  2. Provide your own server-specific implementation classes.
  3. Document how to use and rebuild the implementation-specific code. This is a crucial detail!
  4. Put it in its own JAR file; for example, xAdapter.jar
  5. Include xAdapter.jar file as needed (see above options for WARs and EJB JARs). For example:

```
6.          public interface Foo {
7.              public doBar();
8.      <o:p>
9.          }
10. <o:p>
11.         public class FooImpl implements Foo {
12.             public doBar() {
13.                 // Your server-specific code
        here
14.             }
        }
```

SCENARIO #4: INTER-EAR REFERENCES

In this case, there are several EAR files, and an EJB references another EJB in a different EAR file (see Listing 4).

Packaging Options:

- Include ejbA2_clientview.jar in x.war WEB-INF/classes directory
- Include ejbA2_clientview.jar expanded in x.war WEB-INF/classes directory
- Include ejbA2_clientview.jar in your own specific WAR directory ("clientviews", for example). Then the WAR can refer to the clientviews directory JAR file via its Class-Path entry in the manifest.mf.

SCENARIO #5: MULTI-EAR, UTILITY CLASSES REFERENCED FROM EAR AND WAR

In this scenario, several EARs exist, and reference utility classes from both EAR and WAR files.

```
 <o:p>
appA1.ear:
        x.war          References classes in util.jar
 <o:p>
appA2.ear:
        ejbZ.jar       References classes in util.jar
 <o:p>
```

util.jar

The util.jar file must be present in both application EAR files. The appA1.ear application includes util.jar in its WEB-INF/classes directory. The appA2.ear application includes util.jar and is referenced by ejbXZ.jar.

Conclusion

This article has presented some tips that should help you create more portable J2EE applications. The J2EE Blueprints team is interested in hearing about your experiences with the J2EE platform. Please send feedback and comments to j2eeblueprints-feedback@sun.com. Special thanks to the J2EE development community for their continued commitment to the J2EE platform and for developing J2EE applications and components.

Resources

1. *J2EE Blueprints:* http://java.sun.com/j2ee/blueprints
2. *Download the J2EE Platform Specification, as well as other developer support documentation and software:* http://java.sun.com/j2ee/download.html.
3. *EJBGen*: http://beust.com/cedric/ejbgen.

Author Bio

Elizabeth Blair is a staff engineer with Sun Microsystems where she leads the Java 2 Platform, Enterprise Edition Blueprints implementation team. She contributed to the recent Java Pet Store sample application with an emphasis on the tiers for EIS and EJB architecture. In the past, Liz has worked on the Compatibility Test (CTS) suite for the J2EE platform, the J2SE platform, and the WABI (Windows Applications Binary Interface) projects.

liz.blair@sun.com

Listing 1

```
public class AccountEJB implements EntityBean {
 <o:p>
    private String userId;
    private ContactInformation info;
    // ...
}
 <o:p>
public class ContactInformation implements Serializable {
 <o:p>
    private String telephone;
    private String address;
    // ... etc ...
}
```

Listing 2

```
Connection c;
PreparedStatement ps;
ResultSet rs;
 <o:p>
try {
    c = dataSource.getConnection();
    ps = c.prepareStatement(statement);
    rs = ps.executeQuery();
 <o:p>
    // Do stuff with 'rs'.
}
catch (SQLException se) {
    // Handle this exception.
}
finally {
    if (rs != null)
        rs.close();
 <o:p>
    // Close statement before connection.
    if (ps != null)
        ps.close();
    if (c != null)
        c.close();
}
```

Listing 3

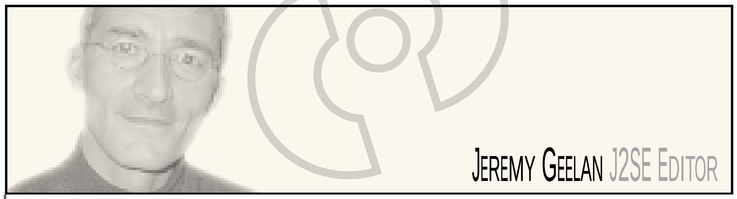
```
import java.io.File;
import java.io.StringWriter;
import java.io.PrintStream;
 <o:p>
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
 <o:p>
public class XMLChecker {
 <o:p>

    public static boolean parseFile(File file, PrintStream out,
                                    boolean validate) {
        try {
            Document document = null;
            DocumentBuilderFactory docBuilderFactory = null;
            DocumentBuilder docBuilder = null;
            try {
                docBuilderFactusingory =
DocumentBuilderFactory.newInstance();
                docBuilderFactory.setValidating(validate);
                if (docBuilderFactory != null) {
                    docBuilder =
docBuilderFactory.newDocumentBuilder();
            using    }
            } catch (ParserConfigurationException pce) {
                out.println(pce);
            }
            if (docBuilder != null) {
                document = docBuilder.parse(file);
            }
            return true;
        } catch (java.io.IOException ioe) {
            out.println("Caught IO exception" + ioe);
        } catch (org.xml.sax.SAXParseException spe) {
            out.println(spe);
        } catch ( org.xml.sax.SAXException se) {
            System.out.println(se);
        }
 <o:p>
        return false;
    }
}
```

Listing 4

```
<o:p>
 <o:p>
appA1.ear:
        x.war          References classes in ejbA2.jar
        ejbA1.jar
        yUtil.jar
 <o:p>
appA2.ear:
        z.war
        ejbA2.jar
        zUtil.jar
```

Download the Code!
www.JavaDevelopersJournal.com



Inscrutable Java

One of the joys of Java is that it never seems to lose its capacity to surprise. This has been true ever since its debut at one of the earliest DEMO conferences, one of the most dynamic and interactive gigs in the whole technology industry.

The formula, for those unfamiliar with DEMO, is that each year the organizers conduct over 1,000 individual interviews with companies that represent the best and the brightest new ideas to hit the technology markets. These 1,000 are whittled down to about 75, and those 75 get to do a presentation at DEMO that year...in front of the "influencers" who will be helping to shape technology opinions in the coming 12 months.

Imagine having been there when the Java "applet" was first presented. Is there anyone *out there*, any reader of ***JDJ,*** for example, who claims today to have sensed six whole years ago the possible impact of this new programming language? If so speak up, we'd love to hear from you in the next issue!

It was a surprise that Java turned out to be far more than just applets. It was a surprise that Java evolved into an entire platform. My own secret view is that Java simply surprises every one of us, almost all the time.

And the surprises continue. In the extraordinarily relaxed and modest chat at JavaOne that James Gosling, the "Father of Java," had live on **SYS-CON Radio** with ***JDJ*** editor-in-chief Alan Williamson and ***JDJ*** editorial panel member Blair Wyman, he evidenced similar surprise...in this case, it was at the fact that in 2001 Java was all set to make its appearance in 100 million Japanese telephone handsets. It's "kinda cool" to think that Java has found its way into such a mass application.

As developers, readers of ***JDJ*** will of course be thirsty for more. Thirsty to hear that, maybe at DEMO 2002, Sun Microsystems – or, more likely, some independent software development shop – has come up with some new twist, some new evolutionary leap, to the Java story.

Maybe we can't afford the time to sift through 1,000 or even 1,200 different companies in order to achieve a sense of what's over the *i*-tech horizon. And maybe we lack the connections that obtain us a place at DEMO so that we can see firsthand the 75 demonstrations that make the cut.

But surely we can all afford the time to keep our eyes on Sun. Which is why upwards of 17,000 Java developers happily got themselves to the Moscone Center this year, to JavaOne. And why **SYS-CON Radio** managed to record and transmit live interviews over the Web not just with James Gosling, but also with over 100 *i*-tech whizzes, from the CEOs of some of the major software giants to the CTOs of some pretty left-of-field startups.

Among the latter category, for all we know, was a future Sun. Anyone who wants to hear the interviews can tune into www.sys-con.com/java. Or, better still, come to **JDJEdge 2001**, the conference and expo produced by **SYS-CON Events**, and hear a great many of the same *i*-tech professionals firsthand, either in the keynote addresses, the expert panels, the FastTrack sessions, or the main presentations.

The conference is being held September 23–26 at the Hilton New York and, well, with Java you just never know what new product announcements will be made, or what strategic twists of fate may go down.

Can you afford not to be there? Come join prominent executives, technologists, analysts, VCs, and entrepreneurs, all of whom will be there to rub shoulders with the developers who make the whole Java ecosystem work. You'll meet the editors behind ***JDJ*** as well as James Gosling, who is giving a keynote address, and a host of other Java luminaries. Register today!

jeremy@sys-con.com

AUTHOR BIO

Jeremy Geelan, editorial director of SYS-CON Media, speaks, writes, and broadcasts about the future of Internet technology and about the business strategies appropriate to the convergence of business, i-tech, and the future.

J2SE INDEX

Designing for Object Serialization

It's a lot more than just implementing Serializable

Written by Bobby Woolf

Often objects need to be serializable. This can be as easy as simply declaring that a class implements the Serializable interface. But just because an object says it's serializable doesn't necessarily mean that it will serialize (and deserialize) successfully. This article will explore what serialization is, how to prove that a purportedly serializable object can really be serialized, and how to redesign a nonserializable class to make it serializable.

The first question is: What is serialization and why do objects need to be serializable? The Javadocs aren't much help; they say what java.io.Serializable does, but not what serialization is. To learn what it is, you must read the "Java Object Serialization Specification" in Sun's Object Serialization documentation (http://java.sun.com/products/jdk/1.2/docs/guide/serialization/). According to the spec, "The key to storing and retrieving objects in a serialized form is representing the state of objects sufficient to reconstruct the object(s)." So serializing an object converts it to a form that can be stored easily, and the stored form can easily be converted back into a Java object again. The new object is essentially a copy of the original. Furthermore, the object being serialized typically refers to other objects, a graph of objects that must be serialized and deserialized so that the entire structure is copied.

Why do objects need to be serialized? The simple answer is persistence and distribution. One of the goals of serialization is to "support simple persistence of Java objects," so object persistence schemes that simply want to store objects as BLOBs tend to use serialization and store the serialized form. (GemStone/J is an exception; it persists objects without serializing them, so it can persist any Java object, even those that don't implement Serializable.)

In the EJB technology, the javax.ejb.EntityBean and javax.ejb.SessionBean interfaces extend Serializable so that EJBs can easily be passivated and activated. To persist an entity bean field that is a complex object, the value is serialized.

Another common use of serialization is as part of object distribution. A distributed object is one that is instantiated in one Java Virtual Machine (JVM) but accessible from other virtual machines. If the object is not copied but is accessible remotely via Remote Method Invocation (RMI), the arguments and result of the remote invocations must be marshalled (java.rmi.MarshalledObject), which implements Serializable so that the arguments and result can be serialized.

When using the Java Message Service (JMS), if you know that the consumer of your message will be another Java application, you can send it an object using a javax.jms.ObjectMessage that contains the serialized object. Even a Throwable is serializable, so that it can be thrown from one VM to another like a marshaled object. Thus if you're going to implement objects that can be distributed or persisted, you're going to need to make them serializable.

Moving Around in the Byte Stream

Serialization is not unique to Java. BOSS, the binary object streaming system, was a prominent feature of VisualWorks Smalltalk since the early '90s. Whenever you have objects, you'll find a need to move them around using simple byte streams.

As common as serialization is, there are alternative ways to persist and distribute objects. Java serialization is practical only when Java code is being used for both the serialization and deserialization.

CORBA objects can be transferred from one object space to another, even when the two spaces use different languages (for example, to enable a Java application to exchange objects with a C++ application). XML converts an object to a form that can be read by any application with an XML parser. Object-to-relational frameworks like TopLink (www.webgain.com) and CocoBase (www.thoughtinc.com) convert objects into rows that can be stored in relational database (RDBMS) tables. Yet when you know that Java will be used both to convert the object and to unconvert it again, serialization is definitely the easiest way to do so.

You'll rarely write your own serialization code. Rather, the frameworks you're using – such as RMI, EJB, and JMS – will use serialization and will expect your objects to be serializable. But to understand how to make your objects serializable, it helps to understand a little about how serialization works.

Implementing Serialization

Serialization is implemented in the java.io package by two main types: ObjectOutput (implemented by ObjectOutputStream) and ObjectInput (implemented by ObjectInputStream). Ultimately, serialization and deserialization are performed by two methods:

- ***void::ObjectOutput.writeObject(Object obj):*** Serializes the object into bytes
- ***Object::ObjectInput.readObject():*** Deserializes the bytes back into an object, a new instance of the same class as the original object (whose type is upcast to Object)

This code will serialize some object, obj, into a byte array:

```
public byte[] serialize(Serializable obj)
throws IOException {
   ByteArrayOutputStream stream = new
     ByteArrayOutputStream();
   ObjectOutput serialStream = new
     ObjectOutputStream(stream);
 serialStream.writeObject(obj);
 return stream.toByteArray();
}
```

This code will deserialize the byte array back into an object:

```
public Object deserialize(byte[] bytes)
throws IOException, ClassNotFoundException {
 InputStream stream = new ByteArrayInputStream(bytes);
 ObjectInput serialStream = new
  ObjectInputStream(stream);
 return serialStream.readObject();
}
```

Thus when you're using a framework that does serialization, these two pieces of code are essentially what the framework is doing.

A class can customize its serialization by implementing writeObject(ObjectOutputStream) and readObject(ObjectInputStream). An object can even use Object::writeReplace() and Object::readResolve() to specify an instance of an entirely different class to be serialized instead of itself. However, most classes don't need to implement these methods and just let the stream classes do the heavy lifting.

As if Serializable weren't enough, it has a specialized subtype called Externalizable. The difference between serialization and externalization is a bit like the difference between container-managed persistence (CMP) and bean-managed persistence (BMP) in EJB. With Serializable, the object delegates serialization to the container and its implementations of ObjectOutputStream and ObjectInputStream. With Externalizable, the object manages its own serialization, ignoring what the container's streams would otherwise try to do.

Externalizable declares two methods – writeExternal(ObjectOutput) and readExternal(ObjectInput) – that are solely responsible for saving and restoring the object's contents. Externalizable isn't used much, so I won't discuss it further here.

Books on improving the performance of Java code point out that you can usually improve your objects' serialization performance by implementing the serialization code yourself. This is because the serialization streams are ignorant of your particular class's implementation and use lots of reflection to pick an object apart and get its (often private) state. Your custom code – knowing how the class is implemented and having direct access even to private variables – is bound to be more efficient. On the other hand, implementing your own serialization code is tricky and difficult to maintain. You should generally only do it for large classes that are serialized frequently, so that the performance improvements will be significant enough to justify the implementation effort.

Let's look at a simple example of serialization that doesn't involve a more complex framework like RMI or EJB. The DataMapper (www.xprogramming.com/datamapper/datamapper_1.htm) is a framework for converting record files into Java objects and vice versa (which itself is an alternative to serialization and XML for converting objects into an easily transferable and persistable form). The mappings for doing the conversion are implemented via a format map, a tree of FieldFormat objects.

Sometimes it is useful to have a process in one virtual machine define a map, then have a process in another virtual machine use the map, perhaps storing the map in between. To be able to move the map between VMs, it needs to be serializable, which means that the FieldFormat graph – and everything those objects reference – needs to be serializable. The remainder of this article will discuss the redesign necessary to take the working format map design and modify it to make the maps serializable as well.

The Principle Is Simple

In principle, making objects serializable is pretty easy: just change the class to implement Serializable.

This class isn't serializable:

```
public class MyClass {
 . . .
}
```

whereas this makes the class serializable:

```
public class MyClass implements java.io.Serializable {
 . . .
}
```

When you implement Serializable, you don't even have to implement any new methods because Serializable doesn't declare any. So then what good does implementing Serializable do? Serializable does not implement or declare a class's serializable behavior; the default implementation for that behavior is already implemented by ObjectOutputStream.defaultWriteObject() and ObjectInputStream.defaultReadObject(). (It would have just been easier to put this code in Object as default implementations of readObject(ObjectInputStream) and writeObject(ObjectOutputStream) that subclasses could override or extend. But since objects delegate this responsibility to the stream classes, new subclasses of ObjectOutputStream and

ObjectInputStream can impose their own serialization approaches and default implementations.) Thus all objects already know how to serialize themselves (or at least the serialization streams already know how to serialize any objects that don't otherwise implement their own serialization); implementing Serializable just enables the serialization behavior.

Serialization Surprises

This seems to imply that all objects are naturally serializable and that the default implementations are sufficient. It turns out that's not always the case. The problem is especially difficult to detect because even classes whose objects will not serialize successfully still compile just fine as if they will serialize. Typically, when a class says it implements an interface but does not have the necessary code to do so, it will fail to compile and the compiler will indicate what code the class is missing. But in the case of serialization, you can change any properly compiling class to implement Serializable and the class will compile. But just because the class compiles doesn't mean it will serialize.

Also remember that an object doesn't just get state from its class, but can also inherit state from its class's superclasses. Serialization only serializes the state in a class that says it's serializable. If a superclass declares state, but doesn't implement Serializable, then the superclass's state won't be serialized. This can result in an object whose serialized state may only represent part of its total state. The state declared by its class gets serialized but the state declared by the superclass does not.

But just because the class compiles doesn't mean it will serialize

Another common pitfall is forgetting that for an object to be serializable, all of its parts must be serializable. Serialization will recurse through the entire object structure, serializing each object in the structure. If any of those objects are not serializable, the serialization will fail.

A risk to keep in mind with serialization is that it's potentially a huge security hole. A class may be carefully designed to hide its state, but a serialized version of an object lays its state out for anyone who wants to read it. It can also enable a malicious client to bypass a class's constructors and manufacture new instances with virtually any possible state stuffed inside it. Thus classes with especially sensitive variables should be designed to avoid serializing those variables. If the whole class must be secure, it should block serialization completely by not implementing Serializable and perhaps even implementing writeObject(ObjectOutputStream) and readObject(ObjectInputStream) to throw NotSerializableException.

When Objects Don't Serialize

So how do you find out that your objects won't serialize? The compiler won't tell you; it'll let anything implement Serializable. It's not until you actually try to serialize an object that you'll find out whether or not it's really serializable.

When you try to serialize an object that's not serializable, Java will throw some sort of ObjectStreamException. Typically it'll be a NotSerializableException, which indicates that you tried to serialize an object that doesn't implement Serializable. The stack trace is like the code shown in Listing 1.

Interestingly, NotSerializableException doesn't necessarily mean that the object inherently is not serializable. It just means that the object doesn't explicitly say that it can be serialized. Often all that's needed is to change the object's class to implement Serializable and the object will now serialize successfully.

Still, though, how do you know whether or not an object will serialize? The compiler won't tell you. You could run all the code in your application in every possible way, but that tends to be difficult and is a kind of overkill just to make sure that you can serialize everything you need to. What we need is some sort of serialization tester that we can use as part of our normal testing procedures. It will make sure that the objects we're producing that are supposed to be serializable and say that they're serializable really are serializable.

A Serialization Tester

As it turns out, I implemented just such a serialization tester for the DataMapper. It's the class bw.dm.test.ObjectSerializer, and although it's bundled with the DataMapper's testing code, it can be used to test the serialization of any code. It's a very small class; the whole thing is shown in Listing 2. The code should look very familiar at this point. The class just embodies the serialization and deserialization code shown earlier in this article.

So, to test an object structure and verify that it can really be serialized and deserialized successfully, you just do this (assuming that MyClass extends/implements Serializable):

```
MyClass obj1 = // create an instance of MyClass
MyClass obj2 = (MyClass)
    ObjectSerializer.serializeAndDeserialize(obj1);
```

Two things should happen here:

1. obj2 should be a copy of obj1, such that they're equal and can be used interchangeably.
2. Perhaps more important, serializeAndDeserialize should not throw any exceptions.

For convenience, the ObjectSerializer converts any exceptions the serialization throws (IOException or ClassNotFoundException) into RuntimeExceptions. As long as no exceptions get thrown, the serialization has most likely worked. On the other hand, if the object cannot be serialized successfully, you'll know right away.

Making the Map Serializable

I already had tests written for the DataMapper that created different types of maps and tried them to verify that they work. Now that I had ObjectSerializer, I modified each test to create the map, serialize, and deserialize it, and then use the copy for the rest of the test. As long as no exceptions were thrown and the copy worked the way the original was supposed to, that was good enough for me.

Fortunately, my tests were now able to reproduce the NotSerializableException all too easily. I had assumed that since my code implemented Serializable and it compiled successfully, serialization would work. The tester now confirmed that the serialization was not working.

Some problems were easy to fix. Basically, certain classes that needed to be serializable didn't implement Serializable, so making them implement Serializable fixed the problem. But in other cases, labeling a class as serializable didn't help because the class fundamentally isn't serializable. These classes either need significant redesign, or the map structure needs to be redesigned so that these nonserializable objects don't need to be serialized.

Nonserializable Objects

Some objects just cannot be serialized. These include some of the most fundamental classes in Java. While Class is serializable, Method and Field are not. And that makes sense. They represent particular members of a particular class. What if they were deserialized in a VM without the class in its classpath, or if the class didn't contain the member? Likewise, a Thread (an object representing the execution of a program) is Runnable, but it isn't Serializable, which is good, because if you copy a thread into a new VM, or persist it and reload it later, what's it supposed to do after it deserializes?

So it makes sense that objects like these aren't serializable, but that's a problem for the DataMapper. It has objects like FieldAspectAdapter that uses a pair of Methods to get and set a value in the object being mapped. It has to store Methods, which are not serializable, so how can FieldAspectAdapter be?

The DataMapper gets around this problem with a FieldAdapterProxy class. An implementation of the Proxy pattern, it's a FieldFormatAdapter that is serializable. A similar example is MethodSpec, a class that makes it easy to convert the parameters necessary to specify a method into the Method itself, but also a proxy designed to be serializable, a distinct advantage over Method.

Transient Variables

Serializable proxies for nonserializable objects don't completely solve the problem, however. Eventually, the proxy has to point to its subject, and the subject (something like a ProtocolAdapter or a Method) still isn't any more serializable than it ever was. So how do these proxies really solve the problem?

Let's consider MethodSpec. It never actually stores the Method. It stores the receiver class, the method name, and the parameter types, but stores them all as Strings, which are very serializable. When asked for the Method, the MethodSpec quickly creates the Method from the Strings and returns it. By never storing the Method, MethodSpec has no problem being serializable even though Method isn't.

This won't work for FieldAspectAdapterProxy, however. As a FieldAdapterProxy, it has to be serializable. But it's a proxy for a ProtocolAdapter, which uses either a Field or a pair of Methods to do its work, which clearly isn't serializable. The FieldAdapterProxy could create the ProtocolAdapter for every use, but ProtocolAdapter is a pretty complex object that references several others, and probably shouldn't be re-created over and over even if it could be. So FieldAdapterProxy needs to create its ProtocolAdapter once and maintain a reference to it, but prevent it from being serialized.

A serializable object can prevent some of its references from being serialized by declaring them as transient. A standard instance variable declaration looks like this:

```
public class MyClass {
 private SomeType variable;
 . . .
}
```

To make the variable transient, do this:

```
public class MyClass {
 transient private SomeType variable;
 . . .
}
```

The serialization code in ObjectOutputStream.defaultWriteObject() serializes the object's state by serializing each of its references. But if a reference is transient, rather than write the variable's value, the stream writes the variable type's default value (false, 0, null, etc.). Then when the object is deserialized, the transient variable's value naturally gets set to this default value without any special processing by the input stream. In the end, the transient variable's real value is never written to the stream, which is an important security consideration if the variable is something sensitive like a password.

So while the FieldAdapterProxy maintains a reference to its ProtocolAdapter, it is a transient reference, so the stream will not attempt to serialize the ProtocolAdapter.

A more complex approach for specifying what parts of an object should be serialized is to use a special static serialPersistentFields variable. The value is an array of java.io.ObjectStreamField objects, each of which specifies the name and type of a nonstatic field to be serialized. These nonstatic fields don't necessarily have to exist in the current version of the class, so this approach is used for migrating serialized instances from one version of a class to another, even when the class's fields have changed.

Lazy Initialization

Yet transient variables don't entirely solve the problem either. They start out as null (or for a primitive type, a default value like false or zero). Yet they're expected to be valid object instances. So when does a transient variable's value get set, and how does the container object avoid setting the value over and over?

A good technique to use here is lazy initialization. Basically, you access the variable through a getter method that initializes the variable the first time, then just returns its value every time after that. The code looks like this:

```
public MyClass getLazyVariable() {
 if (lazyVariable == null)
   lazyVariable = this.defaultLazyVariable();
 return lazyVariable;
}
```

Here's what happens when lazy initialization is used with the transient variable:

1. The container object gets created or deserialized, so the

transient variable is null.
2. The first time the variable is accessed, it's null, so it gets initialized and the newly initialized value is returned.
3. After that, each time the variable is accessed, its value is no longer null, so the current value is returned immediately.
4. When the container is serialized, the transient value is ignored and goes back to null at deserialization.

Thus lazy initialization makes sure that a transient variable gets initialized before it's used but not reinitialized unnecessarily.

Proxy objects with transient variables that are lazy initialized are a good way to develop a serializable wrapper for an otherwise unserializable object.

Implementing a Proxy

For an example of transient fields and lazy initialization, let's look at another DataMapper class. FieldAspectAdapterProxy is a kind of FieldAdapterProxy, which, among other things, needs to be serializable. It's a proxy for a ProtocolAdapter, which is not serializable. (What all these classes do exactly is not too important in this example; you just need to keep your eye on which ones are serializable and which ones aren't.)

First, we have FieldAdapterProxy, which is serializable:

```
public abstract class FieldAdapterProxy implements
    Serializable {
 protected FieldFormat field;
 . . .
}
```

It has a variable, field, which is a FieldFormat. Since FieldFormat is serializable, we're okay so far.

Second, one of the main subclasses of FieldAdapterProxy is Field AspectAdapterProxy. As a subclass, it has to be serializable too. It looks like this:

```
public class FieldAspectAdapterProxy extends
FieldAdapterProxy {
 protected AspectSpec spec;
 protected ProtocolAdapter adapter;
 . . .
}
```

AspectSpec is also serializable, so it's okay. But ProtocolAdapter isn't serializable. (Obviously. If it were, we wouldn't need a proxy class for it!) Implementations of ProtocolAdapter are those classes like FieldAspectAdapter and FieldFieldAdapter that are implemented with reflection classes like Method and Field that aren't serializable. So when we serialize a FieldAspect-AdapterProxy, we're not going to be able to serialize its ProtocolAdapter.

This is where we want to make a variable transient, so that it won't be serialized. Thus we change FieldAspectAdapterProxy like this:

```
public class FieldAspectAdapterProxy extends
FieldAdapterProxy {
 protected AspectSpec spec;
 transient protected ProtocolAdapter adapter = null;
 . . .
}
```

Initializing the variable to null isn't really necessary, but it helps remind us what the variable's initial/default value will be.

Now the problem is that when a new instance is created, or an instance is deserialized, the adapter variable will be null. Let's add some code that uses lazy initialization to set the value when necessary (see Listing 3).

Now, the methods that use adapter call setupDomainAdapter() first to make sure that the variable is initialized. If I weren't sure which methods these were (they're actually just about every method in FieldAspectAdapterProxy), I could put the lazy initialization in a getter method like this:

```
protected ProtocolAdapter getAdapter() {
 if (adapter == null) {
  try {
   this.initializeAdapter();
  }
  catch . . .
 }
}
```

Then as long as all methods using the variable use the getter to access it, it'll get initialized. To help make sure methods don't access the variable directly (at least methods in subclasses), I could declare the variable to be private instead of protected.

Conclusions

To summarize:
- Object serialization is one of Java's fundamental standard features. It's used for enabling objects to be persisted and/or distributed.
- Serialization is accomplished via three types in the java.io package: Serializable, ObjectOutput, and ObjectInput.
- Making an object serializable may be as simple as making its class implement Serializable. But it may take more than that.
- A NotSerializableException is the main sign that an object is not serializable but needs to be.
- A simple serialization tester like ObjectSerializer removes much of the uncertainty about whether or not an object structure is really serializable.
- When an object cannot be designed for serialization, it can often be wrapped with a proxy that is serializable.
- When a serializable object, such as a proxy, has nonserializable parts, those parts can be excluded from the serialization process by declaring their variables as transient.
- Lazy initialization is a good way to make sure that a transient variable gets initialized at the right time and doesn't get initialized repeatedly.

Now, if you ever need to make a data structure of yours serializable, and a teammate advises, "Just implement Serializable," you'll know what you really need to do.

AUTHOR BIO
Bobby Woolf is an indepenpent consultant. He specializes in developing application architectures using various J2EE technologies and embeddable tools.

bobby.woolf@brokat.com and woolf@acm.org

Listing 1

```
java.io.NotSerializableException: bw.dm.impl.FieldAspectAdapter
 at
java.io.ObjectOutputStream.outputObject(ObjectOutputStream.java:845
)
 at
java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:342)
 at
java.io.ObjectOutputStream.outputArray(ObjectOutputStream.java:811)
 at
java.io.ObjectOutputStream.checkSubstitutableSpecialClasses(...:432
)
 at
java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:337)
 at
java.io.ObjectOutputStream.outputClassFields(ObjectOutputStream.jav
a:1567)
 at
java.io.ObjectOutputStream.defaultWriteObject(ObjectOutputStream.ja
va:453)
 at
java.io.ObjectOutputStream.outputObject(ObjectOutputStream.java:911
)
 at
java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:342)
 . . .
```

Listing 2

```
package bw.dm.test;

import java.io.*;

/**
 * Serializes and deserializes an object. Used to verify
 * that the object can be successfully serialized (and
 * deserialized).
 *
 * Simulates the typical serialization process of
 * serializing an object, persisting the data somewhere
 * (pehaps in a file), retreiving the persistent data,
 * and deserializing it to get the object back.
 *
 * DataMapper examples do not <EM>have</EM> to use
 * serialization unless they need it. But format maps
 * are expected to be serializible, and this is a good
 * way to test one to make sure that it is.
 *
 * @see java.io.Serializable
 * @see java.io.ObjectInput
 * @see java.io.ObjectOutput
 */
public class ObjectSerializer {

 public static Object serializeAndDeserialize(Serializable obj) {
  try {
   byte[] bytes = ObjectSerializer.serialize(obj);
   return ObjectSerializer.deserialize(bytes);
  }
  catch (IOException ex) {
   throw new RuntimeException("Serialization failed: " +
ex.getClass().getName() + " - " + ex.getMessage());
  }
  catch (ClassNotFoundException ex) {
   throw new RuntimeException("Serialization failed: " +
ex.getClass().getName() + " - " + ex.getMessage());
  }
 }

 protected static byte[] serialize(Serializable obj)
 throws IOException {
  ByteArrayOutputStream stream = new ByteArrayOutputStream();
  ObjectOutput serialStream = new ObjectOutputStream(stream);
  serialStream.writeObject(obj);
  return stream.toByteArray();
 }

 protected static Object deserialize(byte[] bytes)
 throws IOException, ClassNotFoundException {
        InputStream stream = new ByteArrayInputStream(bytes);
  ObjectInput serialStream = new ObjectInputStream(stream);
  return serialStream.readObject();
 }
}
```

Listing 3

```
public class FieldAspectAdapterProxy extends FieldAdapterProxy {
 protected AspectSpec spec;
 transient protected ProtocolAdapter adapter = null;
 protected void setupDomainAdapter() {
  if (adapter == null) {
   try {
    this.initializeAdapter();
   }
   catch . . .
  }
 }
 protected void initializeAdapter() throws . . . {
  adapter = new FieldAspectAdapter(field, spec);
 }
 . . .
}
```

Download the Code!
www.JavaDevelopersJournal.com

Using the JTable

Working with custom Table Models and JDBC

Part 2 of 3

WRITTEN BY
BOB HENDRY

In Part 2 I continue my discussion on the use of the JTable. (Part 1, "Mastering the JTable," can be found in the January issue of *JDJ*, [Vol. 6, issue 1].) I'll briefly review the three major classes you'll need while working with data within the JTable.

1. ***JTable:*** Controls the visual presentation of the data, however, it has limited control over where the data comes from. In the simplest of circumstances, the JTable can populate itself with data only if the data is static and doesn't come from a database. In the above case, it can be used without any supporting classes.

Usually the data to be populated into a JTable comes from a database. In this case, the JTable must work with the JTableModel class; rarely does it work alone.

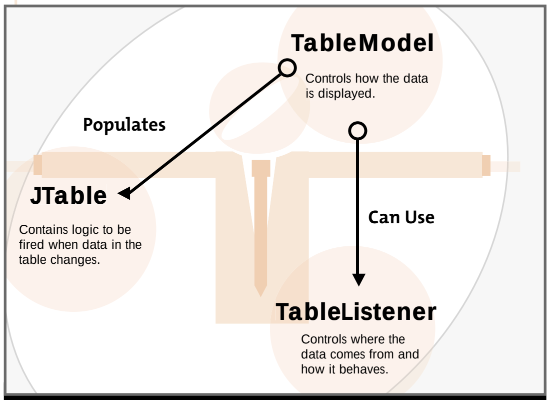

TableModel
Controls how the data is displayed.
Populates
JTable
Contains logic to be fired when data in the table changes.
Can Use
TableListener
Controls where the data comes from and how it behaves.

FIGURE 1 The relationship between a table, its model, and listener

2. ***AbstractTableModel:*** Controls where the data comes from and how it behaves within the JTable. Although the data can come from just about anywhere, this class is almost always used when the data comes from (via JDBC) a database. AbstractTableModel is used by an extension class that you define and is never used directly. It has some interesting default methods, similar to listeners, that automatically fire when data is altered. Knowing when and how these methods are fired is important to understanding how the AbstractTableModel works. I'll discuss them later.

3. ***TableModelListener:*** Not really a class but an interface that's used when you want some action to be performed while the user is interfacing with the data in the JTable. It's misnamed – in general, listeners are automatically fired when some action is performed on a control. A programmer-defined class that implements the TableModelListener isn't really a listener at all as it's never automatically fired. It must be manually fired by the programmer via the fireTableDataChanged method for the TableModel. Since TableModelListeners must be fired manually, they can't be considered a true listener. Not that they aren't valuable. I think the powers that be at Sun purposely made it this way so the programmer controls when the listener is fired.

The code placed in the listener can vary. Popular uses include adding code to update the database. *Note:* The use of this interface is optional. If you don't really care when data is changed, you don't have to use it.

Figure 1 illustrates the relationship between the three major players.

Part 1 focused on the use and implementation of the JTable. Part 2 covers the use of the table model in depth.

The AbstractTableModel Class

As stated earlier, the job of the AbstractTableModel is to provide the source for the data in the JTable and some built-in methods that are automatically fired when certain things happen within the Table (more on this later). This class isn't used directly but as an ancestor in a programmer-created class. When a programmer-defined class extends the AbstractTableModel class, in Javaspeak we're using a Table Model.

The Table Model can get the data from several places. I'll start with a simple Table Model example and get more complex as we go. Your Table Model might obtain its data in an array, vector, or hashtable, or it might get the data from an outside source such as a database (the most flexible and most complex). If needed, it can even generate data at runtime. In Listing 1 (Listings 1–6 can be found on the ***JDJ*** Web site, www.JavaDevelopersJournal.com), the Table Model populates itself with an array of strings. The resulting frame window is displayed in Figure 2.

Table Model Dissected

There are a few immediate advantages to using a Table Model instead of using the JTable alone. In addition to the data, the Table Model informs the JTable of the data type of the column, which leads to some desired formatting behavior. For example, since the JTable knows that a column is numeric, it'll right-justify it within the cell. Also, Boolean values will display as a checkbox (since there are only two possible values). Since the JTable is using a Table Model, it'll display the data in a user-friendly way. In Listing 2 the application doesn't use a Table Model. Notice the difference in how the data is represented (see Figure 3).

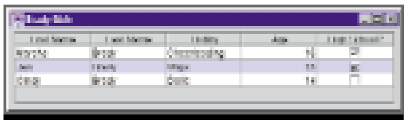


FIGURE 2 Result of Brady girls application

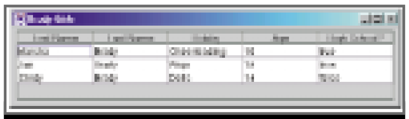


FIGURE 3 Brady girls app without Table Model

Table Model Methods

The methods shown in Listing 1 are used internally by the Table Model. Similar to listeners, these methods are fired when needed, and the programmer can choose to implement them at will. You can use all or none of them; Java doesn't care. The important point is that these methods are fired automatically and usually never by the programmer. Similar to other automatically fired methods, Java lets you decide what kind of code to execute. Java provides the method, but you (surprise!) decide the functionality. The automatic table methods are described below.

getColumnCount()

This method is fired once – at the time the JTable is created. Its sole purpose is to tell the JTable how many columns it has. If the data to be passed to the JTable is contained in an array, simply return its length (see Listing 1). If the data comes from a database, the ResultSetMetaData class can determine how many columns exist by using the following code:

```
// myResultSet is of the Java type
ResultSet and has already been
// filled with data in the program.
int li_cols;
ResultSetMetaData myMetaData =
myResultSet.getMetaData();
li_cols =
myMetaData.getColumnCount();
return li_cols;
```

getRowCount()

As with getColumnCount(), this method is fired once – at the time the JTable is created. Its sole purpose is to tell the JTable how many rows it has. If the data to be passed to the JTable is contained in an array, simply return its length (see Listing 1). If the data comes from a database, there are a few options. Later I'll show how to populate database data into a vector. The following code can be used to return the number of rows in a vector. I'll show you how to populate it later.

```
// Get the number of rows that will
be used to build the JTable.
allRows if of the Java type vector
return allRows.size();
```

getColumnName(int col)

After the Table Model fires off the getColumnCount() method, it calls this method for each column that's been found. For example, if the getColumnCount() method returned a seven (for seven columns), the Table Model will fire this method seven times, passing it the number of the column that it must resolve the column name for each time. The method returns a string that represents the column name. For example:

```
// colnames is an array of Strings
representing the column names for
// the JTable.
return colNames[col];
```

getValueAt(int row int col)

After the Table Model determines the number of rows and columns, it needs to populate them with data. This is the job of the getValueAt function, which is called for every column and row intersection (called a *cell*) that exists in the Table Model. If there are 30 rows with five columns, this method will automatically be fired 150 times – one for each cell. As in the previous methods, you determine which code is executed in this method. Remember, Java provides the method, you provide the functionality. Listing 1 returns objects from a two-dimensional array. The following code can be used if you're returning data from a vector.

```
// row and allrows are vectors
row = (Vector)
allRows.elementAt(row);
return row.elementAt(col);
```

getColumnClass(int col)

This method is automatically fired and gets the class for each column of the Table Model. The following code gets the data type of each column.

```
// Return the class for this column
return getValueAt(0,
col).getClass();
```

Next, the table compares the column's data type with a list of data types for which cell renders (more about renders in Part 3) are registered. The default classes are:

- ***Object:*** Left-aligned label that displays the object's string value
- ***Boolean:*** Displayed as a check box
- ***ImageIcon:*** Displayed by a centered label
- ***Number:*** Displayed as a right-aligned label

Remember, if you don't specify a table model for a JTable, Java will give it a "default" table model and treat the data types of all columns as the object data type. In this case, all the row data will display as strings and be left-justified.

isCellEditable(int row, int col)

This method determines which rows and columns the user is allowed to modify. Since this method returns a Boolean, if all cells are editable it simply returns a true. To prevent a JTable from editing a particular column or row value, it returns a false from this method. The following code enables only column one to display while allowing the rest of the columns to be modified.

```
// Make column one noneditable
while allowing the user to edit at
all // other columns.
If (col == 1){
   return false;
}
else{
   return true;
}
```

public void setValueAt(Object value, int row, int col)

When the user makes changes to an editable cell, the Table Model is notified via this method. The new value, as well as the row and column it occurred in, is passed as arguments to this method. If the original data is coming from a database, this method becomes important. As you'll see, data retrieved from a database is held locally within the Table Model, usually as vectors. When the user changes a cell value in a JTable, the corresponding data in the Table Model isn't automatically changed. It's your responsibility to add code in this event to ensure that the data in the Table Model is the same as the data in the JTable. This becomes important when code is added to update the database. The following code updates the data (held in an array of objects) in the Table Model with the new value that the user just entered in the JTable.

```
// Update the array of objects with
the changes the user has just
entered in a cell. Then notify all
listeners (if any) what column and
row has changed. Further processing
may take place there.
rowData[row][col] = value;
fireTableDataChanged();
```

Taking the Model Further

I have a promising example of Table

Models that use JDBC as their data source. By far the most powerful implementation of the JTable with a Table Model uses data from a database (via JDBC). The good news is that adding the JDBC element doesn't significantly change what we've already learned. Admittedly, it does add more complexity to our task, but it doesn't change the way we treat the JTable/Table Model relationship.

Where Does the Data Come From?

Any program written in Java uses JDBC to connect to a database. JDBC is a complex topic in its own right so I'll save the comprehensive overview for a later date. Here I'll explain the relationship between the Table Model and JDBC. The examples will use a Microsoft Access database that can be reached via a locally configured ODBC data source. For simplicity, the data contained within the database will be the same as the data in the previous examples that used arrays as the data source.

Creating the Table Model

The first step is to declare the Table Model and all instance variables that will be needed. All classes needed as well as a description of the instance variables are included in the code below:

```
// Table Model
import
javax.swing.table.AbstractTableMode
l;
import java.sql.*;
import java.util.Vector;

public class MyTableModel extends
AbstractTableModel {

Connection myConnect;   // Hold the
   JDBC Connection
Statement  myStatement; // Will
   contain the SQL statement
ResultSet  myResultSet; // Contains
   the result of my SQL statement
int         li_cols;     // Number
   of database columns
Vector      allRows;      //
   Contains all rows of the result set
Vector      row;          //
  Individual row for a result set
```

The next step is to add a constructor to the class. The code in the example constructor is broken into two functions. One secures a connection to the database while the other performs the retrieval. This division of functionality is intentional. When this class is instantiated we want to perform the database connection and the data retrieval. However, during the life cycle of this object, we're only interested in the data retrieval since the connection needs to be performed only once. That's why the connection logic is separate from the retrieval logic. For the sake of simplicity, this example is only interested in retrieving data. The actual update of the data will be a topic in Part 3. The code for the constructor is:

```
public MyTableModel() {
//connect to database
   try{
      connect();
      getData();
   }
   catch(Exception ex){
      System.out.print-
ln("Error!");
   }
}
```

The first method called within the constructor provides the connection to the database. The primary purpose of this method is to provide a connection to our ODBC data source. Since our connection (myConnect) is an instance variable, the database can be accessed as long as it's instantiated.

```
void connect() throws SQLException{
   try{

Class.forName("sun.jdbc.odbc.JdbcOd
bcDriver");
   }
   catch(ClassNotFoundException
cnf){
      System.out.println("Driver
does not exist!");
   }
   String url =
"jdbc:odbc:BradyGirls";
   myConnect =
DriverManager.getConnection(url);
}
```

The second method called in the constructor retrieves data from the database and puts it into a vector (see Listing 3). I won't go into detail about vectors as they're a topic in their own right. Vectors are basically "growable" arrays and can also hold any type of object. The function uses two vectors: one holds an individual row of the result set (result of the SQL statement), the other contains all the "row" vectors and is essentially the "vector of vectors." When we connect a JTable with this table model, the vectors will be used to provide the data to be displayed within the table.

The last portion of the Table Model involves the automatic Table Model methods. The code placed in these methods is important. It informs the JTable about the contents of the data and what the data looks like. The automatic Table Model methods are provided in Listing 4.

Building the JTable

If you followed the steps provided, you have a Table Model ready to be used. The next (and last) step is to write a Java program that will use it. Listing 5 instantiates a JTable based on the Table Model that's just been built. Listing 6 contains the completed Table Model.

Conclusion

You should now have a basic understanding of how custom Table Models work and how to use JDBC within a Table Model. We're still far away from what I'd consider a robust application, however. Next month I'll focus on listeners and updating the database with changes to the data.

bobh@envisionsoft.com

AUTHOR BIO

Bob Hendry is a Java instructor at the Illinois Institute of Technology. He is the author of Java as a First Language.

A Class for Reading Binary Files

The sides are even for the big-endian and the little-endian?

WRITTEN BY
JEFF HEATON

Java contains an extensive array of classes for file access. A series of readers, writers, and filters make up the interface to the physical file system of the computer.

The advantage to this sort of system of classes is that the programmer is freed from the overhead of dealing with the physical layout of files. The main disadvantage to this architecture is that the programmer is isolated from the physical details of how a file is stored. Java programs have a distinct, well-defined way in which they store data to files. Unfortunately, this complicates matters when dealing with files created by other languages.

This article presents a reusable class that deals with binary files. Methods are provided that allow the programmer to read a variety of standard numeric and string formats. Additional methods are provided that take into account signed/unsigned, little/big-endian storage as well as file alignment. Using this class, the programmer can read nearly any sort of binary file. An example program is provided that will read the header from a GIF file.

One of the first problems to overcome is reading an unsigned byte. Java treats nearly all types as signed. In order to do the mathematics later required to convert bytes into larger data types the bytes must be unsigned. A protected method is provided to read bytes in an unsigned form. Converting the byte to a short and then trimming all but the least significant eight bits does this. This is done with the following line of code:

```
protected short readUnsignedByte()
{
     return
(short)(_file.readByte() & 0xff);
}
```

Using the BinaryFile Class

The BinaryFile class can be seen in BinaryFile.java. To use the BinaryFile class create a RandomAccessFile class to the file that you would like to work with. This file can be opened for read or write access. Then construct a BinaryFile object, passing in your RandomAccessFile object to the constructor. The following two lines prepare to read/write to a file called "test.dat".

```
file=new
RandomAccessFile("test.dat","rw");
bin=new BinaryFile(file);
```

Once this is complete you can call the various methods provided to access different data types. The methods to access the various data types are prefixed with either read or write and then the type. For example, the method to read a fixed length string is readFixedLengthString.

“Java programs have a distinct, **well-defined way** in which they store data to files”

String Data Types

There are many ways that strings are commonly stored in a binary file. The BinaryFile object supports four different string formats. The null-terminated and fixed-width null-terminated types used by C/C++ are supported. Additionally, fixed-width and the length-prefixed string used by Pascal are also supported.

Null terminated strings are commonly used with C/C++ and other languages. In this format the characters of the string are stored one by one, with an ending zero character. This allows strings to be of any length. Strings stored in this format can contain any character, except for the zero character. Two types of null-terminated strings are supported.

The readZeroString and writeZeroString methods are used to read and write null terminated string. This is an unlimited length string that ends with a null (character 0). The readZeroString accepts no parameters and returns a String object. The writeZeroString accepts a String object to be written.

The readFixedZeroString and writeFixedZeroString methods are used to read and write fixed-length null terminated strings. This is the type of string most commonly used by the C/C++ programming language. The amount of memory held by this sort of string is fixed. But the length of this string can vary from zero up to one minus the amount of memory reserved for this string. In C/C++ this type of string is written as:

```
char str[80];
```

This means that the str variable occupies 80 bytes. But its length can vary from zero to 79. No matter how long this string is, it is always stored to a disk file as exactly 80 bytes.

The Pascal language uses length-prefixed strings. The Macintosh operating system is based on Pascal strings and as a result length-prefixed strings are commonly found in files generated from the Macintosh platform.

The readLengthPrefixString and writeLengthPrefixString methods are used to read and write length-prefixed strings. The writeLengthPrefixString accepts a string and writes it out to the file. The readLengthPrefixString returns a String object read from the file. Length-prefixed strings occupy their length plus one byte in memory.

The last, and simplest, string type supported by the BinaryFile object is the fixed-width string. A fixed-width string is simply an area of memory reserved for the string. The string occupies the beginning bytes of this buffer and any remaining space is padded with either zeros or spaces. It is not unusual to have to do a trim on a string just read in from this format. The readFixedString and writeFixedString methods are used to read and write fixed-width strings. The readFixedString method accepts a parameter to specify the length of the string and returns a String object read from the file. The writeFixedString method accepts a length parameter and a String object. The String object is then written to the file. If the string is longer than the specified length then the string is truncated. If the string length is less than the specified length then the string is padded.

04 00	Big-Endian
00 04	Little-Endian

TABLE 1

Numeric Data Types

In Jonathan Swift's *Gulliver's Travels*, the nations of Lilliput and Blefuscu find themselves at war over which end of a hard boiled egg to cut before eating. Lilliput preferred the Little Endian approach whereas Blefuscu preferred to start with the large end. An inane controversy, indeed, but one that mirrors our own computer industry.

DATA TYPE	SIZE	READ FUNCTION	WRITE FUNCTION
Byte	3-Bits	readByte	writeByte
Word	16-Bits	readWord	writeWord
Double Word	32-Bits	readDWord	writeDWord

TABLE 2

When an integer stored in memory occupies more than one byte, it is necessary to decide which byte to place first. Take, for example, the number 1025. This number would have to be stored in two bytes. The high-order byte would be four. The low-order byte would be one. This is because the integer division of 1025 by 256 is four, with a modulus of one. So we have the bytes of four and one. Is this stored as 04 00 or as 00 04? Computer scientists call the two notations little-endian and big-endian respectively. The same words as those used by Swift to describe the dilemma of the Lilliputians. The two systems can be seen in Table 1.

So which one is predominant in the industry? Unfortunately it's a near dead heat. Most of the UNIX variants and the Internet standards are big-endian. Motorola 680x0 microprocessors (and therefore Macintoshes), Hewlett-Packard PA-RISC, and Sun SuperSPARC processors are big-endian. The Silicon Graphics MIPS and IBM/Motorola PowerPC processors support both little- and big-endian. As a result, the binary file class presented in this article will handle both standards.

In order to accommodate the little- and big-endian numbers, integers are first read in byte by byte – and then converted into the correct data type. For numbers that are four bytes, the next four bytes from the file are read into the variables a, b, c, and d. Then to convert to big-endian or little-endian, the following equation is used.

```
result = ((a<<24) | (b<<16) | (c<<
8) | d);// big endian
result = ( a | (b<<8) | (c<<16) |
(d<<24) ); // little endian
```

In addition to the issue of little-endian or big-endian, numeric data types can be stored as signed or unsigned. Unsigned numbers are virtually unheard of in Java, but they are all too common in other programming languages. This causes there to be four major categories of numbers to be supported. Signed big-endian, unsigned big-endian, signed little-endian, and unsigned little-endian.

To accommodate these different systems, the methods setEndian and setSigned are provided. Set endian will accept either BinaryFile.BIG_ENDIAN or BinaryFile.LITTLE_ENDIAN. There is also a getEndian method to determine the current mode. The setSigned method accepts a boolean. *True* indicates that the numbers are signed. *False* indicates that the numbers are unsigned. There is also a getSigned method to determine the current mode.

Signed numbers are stored in a format called two's complement. Two's complement uses the most significant bit as a signed or unsigned flag. In all numbers, except zero, a value of one for this bit signifies a negative number. In the case of zero, which has no sign, this bit is set to zero. Positive numbers are stored just as they normally would be. Negative values are stored by subtracting their magnitude from one beyond the highest value that an unsigned number of that type would hold. For example –1 in a word would be stored as 0x10000 – 1, or 0xffff.

In addition to signed or unsigned, the BinaryFile object can also read a variety of sizes of number. The supported sizes are byte, word, and doubleword. The methods used to read/write these types are readByte/writeByte, readWord/writeWord and readDWord/writeDWord. A byte occupies just one byte of memory. The endian setting does not affect byte read/writes. A byte can be signed or unsigned. A word occupies two bytes of memory. Words can be little or big endian. Words can also be signed or unsigned. The double-word occupies four bytes of memory. A double word, like the word, follows the endian and signed modes.

Each of the numeric read/write methods deals in Java types that are one size bigger than the underlying data type. A byte is stored in a short, a word is stored in an int, and a double-word is stored in a long. This is done to accommodate the unsigned data types. The Java byte data type cannot hold values all the way to 255. Because of this, the readByte method returns a short and not a byte. The readByte command, when working in unsigned mode, can return numbers in the range of 0 to 255. That would overflow a Java byte, so a short is used instead. These different types can be seen in Table 2.

Alignment

Binary files are often aligned to certain boundaries, for example, "word aligned" or "double word aligned." This means that if one record only took up 10 bytes and the file is "double word aligned" then before the next record is written enough bytes must be written so that the record falls evenly on a double word boundary. The next double word boundary after 10 bytes would be 12. So two extra bytes must be written to accommodate the alignment requirement.

The BinaryFile object accommodates alignment requirements through the align method. The align method accepts one parameter that specifies the boundary to align to. This parameter is the amount of bytes that you wish to align to at this point. For example, if you were at file position 10, and you called the align method with a value of four, you would be moved to file position 12. Because 12 is the next even multiple of four after 10.

The align method works for both read and write operations. It is important to remember that the align method only alters the way in which data is written when it is called. Therefore it is likely that you will call the align method just after a record has been written.

The Example Programs

To test this program, I ran it on a variety of systems. I tested it on the little-endian platforms of Windows NT and x86 Linux. It was also tested on the big-endian platform of Sun. There are two example programs given. The first, seen in ScanGIF.java, reads the header of a GIF file. The second, seen in BinaryExample.java, opens a file named "test.dat" then proceeds to write several of the data types. The file is then closed, reopened, and the same data types are read back.

To read a GIF file header, the file is first opened and passed into a BinaryFile object. To match the format of a GIF file, the options of little-endian and unsigned are selected. The GIF file consists of a fixed with type, then a fixed with version, followed by a height and width. This is read in with the following method calls.

```
type = bin.readFixedString(3);
version = bin.readFixedString(3);
height = bin.readWord();
width = bin.readWord();
```

Using the BinaryFile object, Java programs can easily access a variety of binary file types. Perhaps in the future standards, such as XML, will make binary files obsolete. But for now, there are many such files out there that a Java program may need to be compatible with.

heatonj@heat-on.com

AUTHOR BIO
Jeff Heaton is a software designer for the Reinsurance Group of America(RGA), a college instructor, and a coauthor of Teach Yourself Visual C++ in 21 Days, Professional Reference Edition (Macmillan 1999).

JASON BRIGGS J2ME EDITOR

Modern Rituals

It's an odd sensation when you're wandering around and everyone immediately looks at your chest. (No, I don't have a strange growth protruding from my sternum.) I recently visited the Embedded Systems Show (ESS) in London. The offending item, attracting all the attention, was the printed badge visitors were expected to wear when entering the exhibition, displaying in large letters their names and – more important – their companies.

Like some sort of bizarre, primitive ritual, each badge was scrutinized by exhibitors to determine whether the visitor was worth talking to and, undoubtedly, where he or she ranked in the hierarchy. There was a sense of quiet desperation in the air as I watched each visitor mentally catalogued by the exhibitors. I wondered if my badge had read "Venture Capitalist," whether I would have been set upon like a tourist in a Turkish bus station or whether I would have just inspired some peculiar type of fawning-over-the-alpha-male behavior.

Perhaps this is just more evidence of the dot-com bust. I remember the European Consumer Trade Show a few years ago being the exact opposite; you were lucky to attract the attention of exhibitors at all – unless it was by the scantily clad models who were paid to take photos with you.

No, ESS was a completely different and somewhat alien environment – a psychologist's dream and a hardware-geek's paradise. However, for your average Java developer, only a few stands warranted immediate closer inspection.

NewMonics was announcing partnerships with Metrowerks (CodeWarrior) and Enea OSE Systems (Real Time Operating Systems) while at the show, and Espial was showing off a rather funky-looking, tablet-sized "Internet Appliance."

esmertec, inc. (www.esmertec.com) seemed to be the only company to make any particular effort to produce an interesting demo.

Demonstrating their Jbed virtual machine and Real Time Operating System was a foot-high perspex frame with a circuit board and magnet resting on top. A softball floated about an inch below the magnet. I mean, when you have a product that could conceivably be used in all sorts of devices, you could drop a Ferrari on a rotating platform, throw a monitor with your product in the front, and probably draw large crowds no matter how good or bad your product is. But it takes a special kind of geekness to think up a floating softball. esmertec now occupies a special place in my heart because of this.

Last, but not least, QNX (www.qnx.com) was situated near the back of the auditorium, next to another large, well-known operating system vendor. Micro-something, I think. QNX is another real-time OS vendor who first came to my attention about four years ago – with their 1.44Mb operating system on a floppy disk (a real-time OS, graphical user interface, browser, dialer, TCP/IP and sample apps, all on a single disk. A seriously impressive effort).

On the Java front, QNX was sharing the stand with OTI (an IBM subsidiary company), whose J9 virtual machine is included in QNX's Voyager Web Browser. (We'll try to take a look at it in the next few months.)

As far as I know, the 1.44Mb demo disk does not include Java – but I live in hope that they will somehow find a way to squeeze it on. Just for the hell of it, of course.

So, considering there were in excess of 120 companies exhibiting at the show, the percentage of Java-related technologies was relatively low. But I'm sure if you had put a similar selection of companies together this time last year, or the year before, you would no doubt be looking at an even smaller fraction.

In this month's issue of *JDJ*, you'll find a discussion on J2ME Cryptography (perfect for gadget-happy paranoiacs, looking at securing their data on handheld devices), a beginner's guide to MIDP development, and the essential API rundown – your J2ME cheat sheet.

As for me, I'm off to get my chest x-rayed... just in case.

jasonbriggs@sys-con.com

AUTHOR BIO

Jason Briggs works as a Java analyst programmer in London.

J2ME INDEX

Q: ***Is PersonalJava part of J2ME?***

A: The short answer is yes. For the long answer, we'll refer to Sun's FAQ for J2ME, which states that PersonalJava was the "first Micro Edition technology." Because PersonalJava has been around for a while now, you'll find more products with a version of it installed. But sometime this year (2001), Sun is expected to replace the existing PersonalJava technology – based on Java 1.1 – with a new release based on Java 2, and incorporate it into the J2ME concepts of configuration and profile components.

Q: ***Is all the Java API within J2ME?***

A: No. Even PersonalJava, which has the most complete coverage of the Standard Edition API, is still just a subset.

Q: ***What is a "midlet"?***

A: Actually, the correct word is MIDlet. A MIDlet is an application written for the Mobile Information Device Profile (MIDP). You might find these on mobile phones, PDAs – in general, small devices.

Q: ***Can I use threads? Is there a penalty?***

A: Yes you can use threads, unless you're writing a JavaCard applet. As for the penalties, it depends on how you want to use them and the environment you're working within. When developing for constrained devices always remember what resources you have available. If you're writing a MIDlet and create 100 threads to try to load 100 images simultaneously, there definitely will be a penalty – it undoubtedly won't work.

Q: ***Do I use AWT or Swing for my GUI?***

A: If you're developing a PersonalJava application, you have access to a modified version of AWT; "modified" meaning that a few java.awt classes/methods are optional, some have been changed, and there are some additions to the basic package.

You may be able to get Swing to work within a PersonalJava environment as well. A brief skim of the PersonalJava forums shows some success stories – and more than a few painful attempts.

None of the other J2ME "products" support AWT or Swing (for example, MIDP has the javax.microedition.lcdui package for user interfaces).

Q: ***Where can I find more information about wireless technologies?***

A: The back issues of ***JDJ*** are one place. For online information, check out the following URLs:
- http://developer.java.sun.com/developer/technicalarticles/wireless/
- *Bill Day's J2ME archive:* www.billday.com/j2me/
- *Sun's PersonalJava forum:* http://forum.java.sun.com/list/discuss.sun.personaljava
- *Sun's KVM forum:* http://forum.java.sun.com/list/discuss.sun.k.virtual.machine.kvm

Q: ***Where can I download J2ME emulators?***

A: You can download from the following Web sites:
- *The J2ME Wireless Toolkit:* http://java.sun.com/products/j2mewtoolkit/download.html
- *The MIDP reference implementation:* http://java.sun.com/products/midp/
- *CLDC:* www.sun.com/software/communitysource/j2me/cldc/download.html
- *CDC (and the Foundation profile):* www.sun.com/software/communitysource/j2me/cdc/download.html

Q: ***Where can I find devices that run J2ME?***

A: Move to another country. At the moment there are a limited number of countries where J2ME-capable devices have been released, especially for mobile phones. While you can probably find PDAs that support PersonalJava almost anywhere in the world, the same is not true for mobiles.

In Japan NTT DoCoMo has a number of phones from Panasonic, Fujitsu, Sony, and others (available only in Japan, of course). In the U.S., Motorola has a couple of J2ME-capable mobiles. For a more comprehensive list check out www.javamobiles.com/.

SYS-CON MEDIA

PUBLISHER, PRESIDENT, AND CEO
FUAT A. KIRCAALI fuat@sys-con.com

ADVERTISING

SENIOR VICE PRESIDENT, SALES & MARKETING
CARMEN GONZALEZ carmen@sys-con.com

VICE PRESIDENT, SALES & MARKETING
MILES SILVERMAN miles@sys-con.com

ADVERTISING SALES DIRECTOR
ROBYN FORMA roybn@sys-con.com

ADVERTISING ACCOUNT MANAGER
MEGAN RING megan@sys-con.com

ASSOCIATE SALES MANAGER
CARRIE GEBERT carrieg@sys-con.com

ASSOCIATE SALES MANAGER
CHRISTINE RUSSELL christine@sys-con.com

SALES ASSISTANT
ALISA CATALANO alisa@sys-con.com

EDITORIAL

EXECUTIVE EDITOR
M'LOU PINKHAM mpinkham@sys-con.com

EDITOR
NANCY VALENTINE nancy@sys-con.com

MANAGING EDITOR
CHERYL VAN SISE cheryl@sys-con.com

ASSOCIATE EDITOR
JAMIE MATUSOW jamie@sys-con.com

ASSOCIAT EDITOR
GAIL SCHULTZ gail@sys-con.com

ASSOCIATE EDITOR
BRENDA BREENE brenda@sys-con.com

ASSISTANT EDITOR
GREGORY LUDWIG gregory@sys-con.com

EDITORIAL INTERN
NIKI PANAGOPOULOS niki@sys-con.com

PRODUCTION

VICE PRESIDENT, PRODUCTION & DESIGN
JIM MORGAN jim@sys-con.com

ART DIRECTOR
ALEX BOTERO alex@sys-con.com

ASSOCIATE ART DIRECTOR
LOUIS F. CUFFARI louis@sys-con.com

ASSISTANT ART DIRECTOR
CATHRYN BURAK cathyb@sys-con.com

GRAPHIC DESIGNER
ABRAHAM ADDO abraham@sys-con.com

GRAPHIC DESIGNER
RICHARD SILVERBERG richards@sys-con.com

GRAPHIC DESIGNER
AARATHI VENKATARAMAN aarathi@sys-con.com

WEB SERVICES

WEBMASTER
ROBERT DIAMOND robert@sys-con.com

WEB DESIGNER
STEPHEN KILMURRAY stephen@sys-con.com

WEB DESIGNER
CAROL AUSLANDER carol@sys-con.com

WEB DESIGNER
PURVA DAVE purva@sys-con.com

ACCOUNTING

ASSISTANT CONTROLLER
JUDITH CALNAN judith@sys-con.com

CREDIT & COLLECTIONS
CYNTHIA OBIDZINSKI cynthia@sys-con.com

ACCOUNTS PAYABLE
JOAN LAROSE joan@sys-con.com

SYS-CON EVENTS

VICE PRESIDENT, SYS-CON EVENTS
CATHY WALTERS cathyw@sys-con.com

SALES EXECUTIVE, EXHIBITS
MICHAEL PESICK michael@sys-con.com

SALES EXECUTIVE, EXHIBITS
RICHARD ANDERSON richard@sys-con.com

JDJSTORE.COM
ANTHONY D. SPITZER tony@sys-con.com

Some of the more frequently asked questions about the various forums for J2ME are, "What is J2ME?" and "Is <so-and-so-product> a part of J2ME?" So this month, in an attempt to reduce the number of FAQs – and therefore offer a valuable service to search engines, intelligent agents, and the Java community – JDJ presents the Great J2ME API Rundown!

Here is where you'll find all the APIs that fall beneath J2ME's umbrella, and the packages within those APIs. In future issues, we'll include the API rundown in a summarized form and attempt to update the information whenever Sun moves the goalposts around.

Connected, Limited Device Configuration (CLDC) – Version 1.0

CLDC contains the following packages:

java.io	input and output through data streams
java.lang	fundamental classes
java.util	collections, data and time facilities, other utilities
javax.microedition.io	generic connections classes

You can find more information on CLDC at:
http://java.sun.com/products/cldc/

Connected Device Configuration (CDC) – Version 0.2

CDC contains the following packages:

java.io	input and output
java.lang	fundamental classes
java.lang.ref	reference object classes
java.lang.reflect	reflective information about classes
java.math	BigInteger support
java.net	networking support
java.security	security framework
java.security.cert	parsing and management of certificates
java.text	used for handling text, dates, numbers and messages
java.text.resources	contains a base class for locale elements
java.util	collections, date/time, miscellaneous functions
java.util.jar	reading Jar files
java.util.zip	reading Zip files
javax.microedition.io	connections classes

Look for more CDC information at:
http://java.sun.com/products/cdc/

Mobile Information Device Profile – Version 1.0

MIDP builds on CLDC and contains the following packages:

java.io	
java.lang	CLDC, plus an additional exception
java.util	CLDC, plus timer facilities
javax.microedition.io	networking support based upon the CLDC framework
javax.microedition.lcdui	for user interfaces for MIDP applications
javax.microedition.rms	persistent data storage
javax.microedition.midlet	defines applications and interactions between app and environment

The products page for MIDP is at:
http://java.sun.com/products/midp/

Foundation Profile – Version 0.2

The Foundation Profile builds on CDC and contains the following packages:

java.io	see CDC
java.lang	see CDC
java.lang.ref	see CDC
java.lang.reflect	see CDC
java.math	see CDC
java.net	see CDC
java.security	see CDC
java.security.cert	see CDC
java.security.acl	access control lists
java.security.interfaces	interfaces for generating keys
java.security.spec	key specifications, and algorithm parameter specifications
java.text	see CDC
java.text.resources	see CDC
java.util	see CDC
java.util.jar	see CDC
java.util.zip	see CDC
javax.microedition.io	see CDC

The profile products page is at:
http://java.sun.com/products/foundation/

PersonalJava Specification – Version 1.2a

PersonalJava will eventually be superseded by the Personal Profile, but for now the specification contains the following packages:

java.applet	full support from JDK1.1.8
java.awt	modified from JDK1.1.8

• Note: there is an extra method for PJ for double-buffering in java.awt.Component

java.awt.datatransfer	full support
java.awt.event	full support
java.awt.image	full support
java.awt.peer	modified
java.beans	full support
java.io	modified
java.lang	modified
java.lang.reflect	modified
java.math	optional – may or may not be supported
java.net	modified
java.rmi	optional
java.rmi.dgc	optional
java.rmi.registry	optional
java.rmi.server	optional
java.security	modified
java.security.acl	unsupported
java.security.cert	some classes required, some optional
java.security.interfaces	required if code signing is included
java.security.spec	required if code signing is included
java.sql	optional
java.text	full support
java.text.resources	modified
java.util	modified
java.util.jar	required if code signing is included
java.util.zip	modified

Additional PersonalJava specific packages are:

com.sun.awt	for mouseless environments
com.sun.lang	a couple of error & exception classes
com.sun.util	for handling timer events

For more information on the PersonalJava Application Environment:
http://java.sun.com/products/personaljava/

Java TV – Version 1.0

Java TV contains the following packages (in addition to PersonalJava):

javax.tv.carousel	access to broadcast file and directory data
javax.tv.graphics	root container access and alpha blending
javax.tv.locator	referencing data and resources
javax.tv.media	controls and events for management of real-time media
javax.tv.media.protocol	access to generic streaming data in a broadcast
javax.tv.net	IP datagram access
javax.tv.service	service information access
javax.tv.service.guide	supporting electronic program guides
javax.tv.service.navigation	services and hierarchical service information navigation
javax.tv.service.selection	select a service for presentation
javax.tv.service.transport	information about transport mechanisms
javax.tv.util	creating and managing timer events
javax.tv.xlet	communications interfaces used by apps and the app manager

Get off that couch and check out the JavaTV page at:
http://java.sun.com/products/javatv/

Java Embedded Server – Version 2.0

JES contains the following packages:

com.sun.jes.service.http	servlet/resource registrations
com.sun.jes	.http basic authentication
service.http.auth.basic	management of users and their access
com.sun.jes.service.timer	for handling timer events
org.osgi.framework	consistent model for app. dev., supports dev. and use of services
org.osgi.service.device	detection of devices
org.osgi.service.http	http access of resources
org.osgi.service.log	logging facility

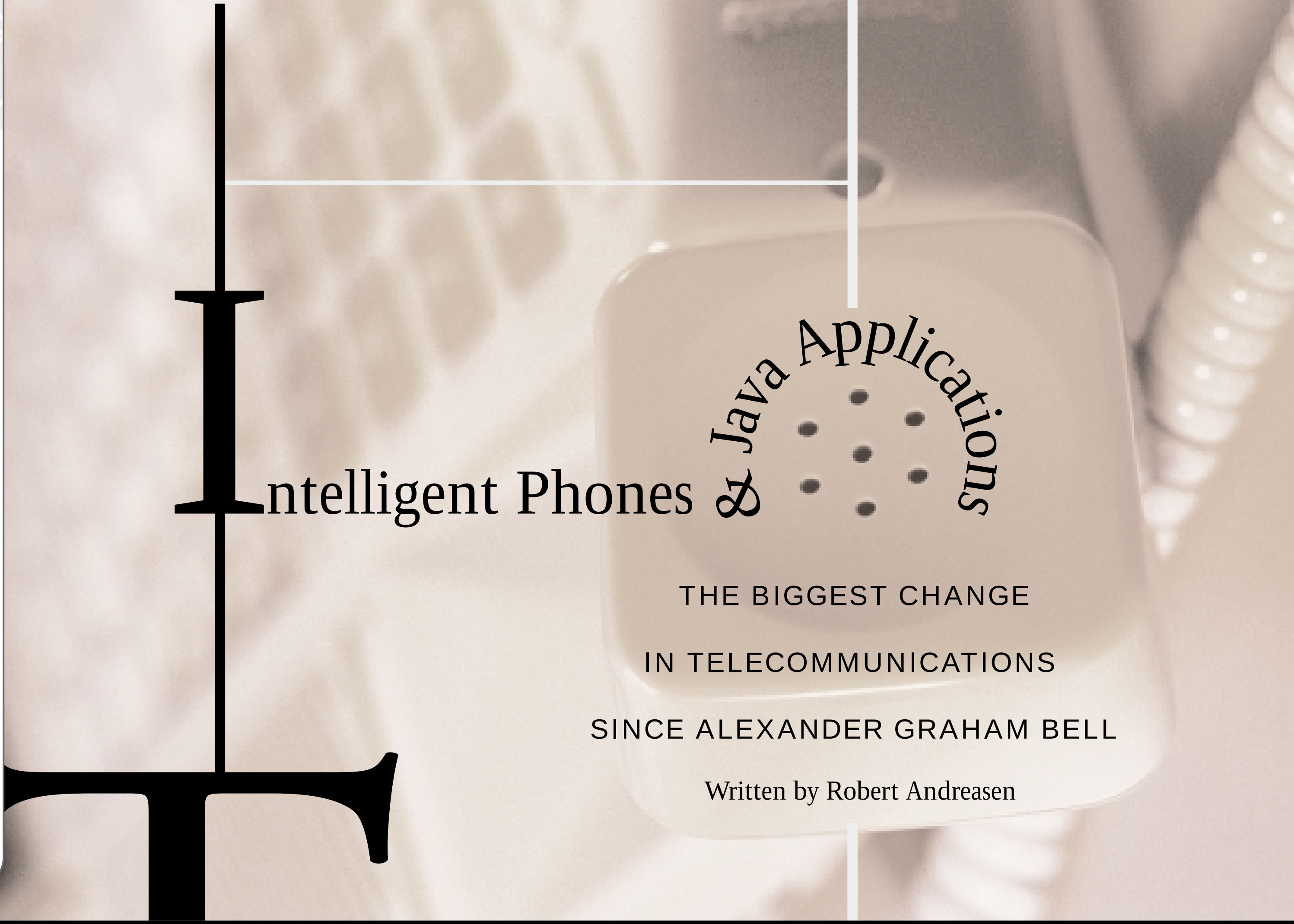
You can find more information on the Java Embedded Server at:
http://www.sun.com/software/embeddedserver/

Java Card – Version 2.1.1

Java Card contains the following packages:

java.lang	fundamental classes
javacard.framework	core functionality of a JC Applet
javacard.security	security framework
javacardx.crypto	extension package with security classes and interfaces

Next time you use that American Express Blue card, you may want to know how it works, so take a look at:
http://java.sun.com/products/javacard/



Intelligent Phones & Java Applications

THE BIGGEST CHANGE IN TELECOMMUNICATIONS SINCE ALEXANDER GRAHAM BELL

Written by Robert Andreasen

T
e simple words, r.
Watson, come here. I want
you, transformed
ommunications in ways
hat Alexander Graham
ell probably never
magined. Although
elecommunications has
hanged dramatically since
hat first call in 1876,
here are still many aspects
f our current phone
ystem that would seem
amiliar to Bell.

Throughout its history the telephone has continued to be a relatively simple device connected by wire or fiber optic cable to an intelligent core. While the intelligence at the center has changed from operators at a switchboard to smart switches, the phone has remained a simple terminal at the periphery of the network. And, with this traditional host-terminal model, any change in phone services has to be programmed into the central switch.

Intelligent, Java-based IP phones, such as the xpressa phone offered by Pingtel Corporation (see Figure 1), are beginning to transform the historical host-terminal model. The debut of intelligent phones creates potentially revolutionary change because it's much easier to program changes at the periphery than at the core.

With the old closed-system model it typically took two years and $2 million to develop, test, and deploy even a minor enhancement to the system. Furthermore, adding intelligence at the periphery offers enhanced scalability because processing tasks can be moved from the central server to the individual phones.

The Internet demonstrates how quickly change can occur when you eliminate central control over all innovation. The Internet provided a set of standards and a network for connecting any machine to any other machine, but the actual content and innovation often happened at the periphery.

Once developers recognized the potential of the Internet, the race was on. Suddenly anyone with a Web site could transform the network in new and unexpected ways. The development of portals, e-commerce sites, peer-to-peer file sharing, massive distributed computing solutions, messenger services, and streaming media all came from innovation at the edge. And there's no indication that the transformation has stopped or even slowed down.

Now, the same type of transformation is possible for telecommunications services. The combination of Java with an intelligent IP phone means the fun can finally begin in telecommunications as well. It opens up the phone system to a whole array of individualized services – things that you and your organization may want, even if there isn't a widespread audience for them.

By moving intelligence to the edge of the network, you can expect to see much greater innovation in phone services. End users will no longer be dependent upon their service providers for enhanced services, because they'll now be able to personalize their own phones in previously unimaginable ways.

This new phone architecture also leverages the same shareware phenomenon that has driven the development of thousands of new applications for the Palm PDA. When a product provides open APIs and developer kits, users benefit from the creativity of a wide community of developers. While this community often creates applications that meet their own specific needs and desires, many of these applications are eventually adopted by a wider group of users.

Java is central to this transformation because it's portable, flexible, widely adopted, and relatively easy to use. Thus, Java opens up the whole field of telecommunications to modification by a large community of developers. In addition, Java's ability to remotely load and execute apps allows new applications to be downloaded from the Web and installed by any user, anywhere in the world, without IT support.

This article covers the basics of Java programming for this new generation of intelligent IP phones. We discuss the technical background, the programming environment, and then generate a Java application for call filtering and routing – so you can see the code in action.

What's Possible

While we're discussing call filtering in this article, there are actually a wide variety of applications that can be developed for these new phones. With a Java phone you can create applications for everything from enhancing business productivity with simplified phone conferencing, speed dialing, enhanced caller ID, and transfer wizards to having fun with user-selected music from an MP-3 server and games such as head-to-head Tetris that can be played with other people. These applications represent only a small sample of what can be created for these new Java-enabled phones.

In addition, an IP phone's connection to the data network lets the phone become far more than just an instrument for transmitting voice. For example, you can integrate the phone with your PC so that you can dial from your Microsoft Outlook Contacts database, create and maintain phonebooks and a speed-dial directory, launch customer relationship management applications based on caller ID, and more.

Best of all, there are whole classes of applications that can be developed that haven't even been thought of yet. Remember, when you switch the intelligence to the periphery, the change comes rapidly and unpredictably. So the actual applications developed for the phone will be dependent upon what users want to accomplish.

Call Filter and Routing Application

Have you ever been bothered by telemarketers when you were trying to get something done? Ever wanted to accept certain phone calls while directing others to voice mail so you can deal with them later? Or transfer some calls to another person without picking up the phone? Or redirect calls to your cell phone so you get them as you travel about? The application we're about to describe and create allows you to do all this and more.

The call filtering application is a variation on the concept that many people use for automatically filtering and routing their e-mail. Much as an e-mail program looks at the sender's address or the subject line of an e-mail to determine what to do, an intelligent phone looks at the user ID, domain name, and a variety of other factors to determine what to do with an incoming call.

We chose call filtering as the subject of this article because it's easily understood by most people, and is a useful application that involves programming concepts that can be applied to a variety of other applications.

Programming Environment

While writing the new call filtering and routing application, we'll take advantage of a programming environment that includes an application framework for defining and installing applications, the Session Initiation Protocol (SIP), Simple Telephony API (STAPI), and programming hooks that control the phone's behavior.

The application framework for the Pingtel xpressa phone resembles the Java applet model used in Web browsers. Applications, or xpressions as they're called on Pingtel devices, must be packaged into a .jar file and contain both a class extending an application and a property file defining parameters and settings. Much like applets placed on a Web page, new xpressions are installed by reference and their URL is added to an application list on the phone.

The application class defines three methods in which developers can add logic – the onLoad, onUnload, and main methods.

```
public static void onLoad()
```

The onLoad method is invoked when the phone's class loader physically loads an xpression. Developers are expected to perform one-time initialization and add hooks and listeners. Depending on configuration settings, the xpression can be loaded at startup or on demand.

FIGURE 1 The Pingtel xpressa phone

```
public static void onUnload()
```

The onUnload method is called immediately before the application framework unloads an xpression. Developers should remove hooks and listeners, and perform any cleanup that may be needed.

```
public void main(String argv[])
```

The main method is invoked when the xpression is executed. Applications can be configured to execute on startup or when selected from an application launcher.

Session Initiation Protocol

SIP is a very flexible HTTP-like protocol used by IP phones for call control. It defines how phones should communicate with one another to initiate calls, conference calls, transfer parties, drop calls, and perform other functions. Generally, developers don't need a working knowledge of SIP because they can use APIs such as JTAPI or STAPI to control a phone. However, as we build the call filtering and routing application, we'll utilize SIP addresses.

SIP addresses are similar to "mailto:" and "telnet:" addresses. These addresses identify a specific user at a given domain or device. Ignoring URL parameters, ports, and display names, an SIP address follows the form:

```
sip:userinfo@host
```

Where "sip" represents the protocol, "userinfo" represents the user's name, and "host" represents the domain or device.

Simple Telephony API

STAPI, a simplified version of Java Telephony API (JTAPI), was designed by Pingtel to help developers easily create new Java applications for the IP phone. While developers can use JTAPI on an xpressa phone, JTAPI was originally designed for larger systems, such as call centers, and is fairly complex for a single end point. Therefore, STAPI is generally recommended for xpressa developers and is used throughout this article.

The call filtering and routing application performs its tasks before the phone rings and a call is actually completed. To accomplish this, the call filter hook works with a STAPI address that provides information about the caller and domain.

STAPI addresses can be expressed as either a PAddress or a PSIPAddress. A PAddress class defines an abstract address that represents addresses including traditional E164 phone numbers, SIP addresses, and addresses for other protocols. A PSIPAddress class extends a PAddress and adds methods for querying and changing the different parts of an SIP address.

Programming Hooks

The xpressa phone provides hooks that allow programmers to extend and alter the default behavior of the phone. The hook framework allows developers to supplement caller ID, alter the behavior of incoming calls, define dynamic dial plans, and provide advanced call filtering.

Hooks are callback routines that can be chained together sequentially. Whenever a decision needs to be made, such as how to filter a call or alert the user to an incoming call, a hook chain is walked. Each hook is executed in turn until a hook takes an action. Once a hook takes an action, there's no need to continue walking the hooks and the chain is considered terminated.

All hooks share a common interface and therefore look alike. A hook receives a hook-specific piece of data, checks the data's state, and then takes an action. In the call filter example the caller ID is the state information and the three possible actions are accepting, rejecting, or redirecting the incoming call.

```
public interface Hook {
 public void hookAction(HookData data) ;
}

public class HookData {
 public void terminate()
 public Boolean isTerminated()
}
```

The Hook interface and HookData class are fairly simple. Whenever a hook takes an action, the terminate() method is invoked on the data object. Once terminated, the hook chain is no longer walked.

```
public class CallFilterHookData extends HookData {
 public void accept()
 public void reject()
 public void redirect(PAddress address)
 public PAddress getAddress()
}
```

The CallFilterHookData class provides three possible actions – accept, reject, and redirect – along with a method to get the incoming caller ID in the form of an address. In this case the hook will be walked sequentially to determine if it should accept, reject, or redirect the call. Once an action is taken, the hook is terminated.

Building the Framework

We begin by building a very flexible framework for the call filtering application. This framework allows us to define any number of conditions, actions, and rules that will determine

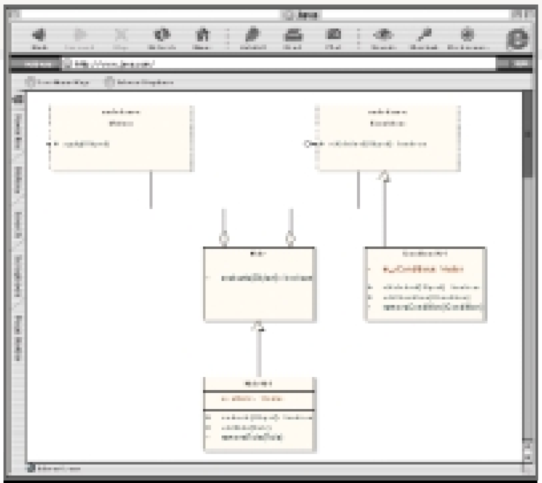


FIGURE 2 The general framework for a rules-based inference engine

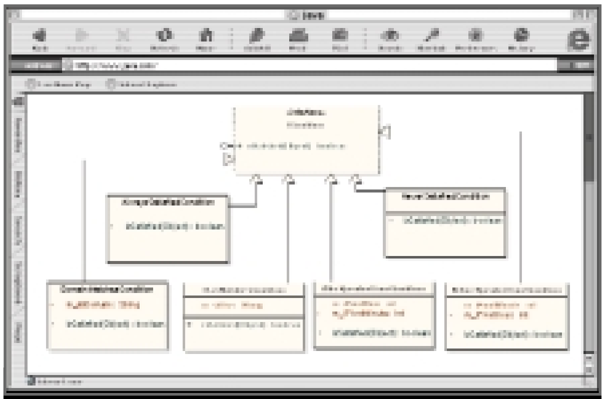


FIGURE 3 The original call filtering framework

the behavior of the phone when it receives an incoming call.

Figure 2 shows the general framework of a simple rules-based inference engine. When a rule is evaluated, it checks the condition that you specify and, if satisfied, applies the rule. The basic building blocks of the inference engine are shown below and in Listing 1; they demonstrate the structure of a condition, action, rule, condition set, and rule set. (Listings 1–12 can be found on the ***JDJ*** Web site.)

```
public interface ICondition {
    public boolean isSatisfied(Object object);
}
```

The ICondition interface defines a condition that can either be satisfied or not. The condition is handed an application-specific object that affects the decision-making process. The condition statement returns true if satisfied and false if not.

```
public interface IAction {
    public void apply(Object object);
}
```

The IAction interface defines an action that can be applied. Like the ICondition interface, an application-specific object is supplied to the implementer, which is expected to apply the action directly to the object parameter.

The Rule class (see Listing 1) combines IAction and ICondition interfaces into an entity that can be evaluated. The action and condition can be provided during construction or set later using the mutator method setRule. When the evaluate method is invoked, the action is applied if the condition has been satisfied.

While the diagram and code are relatively simple, the possibilities can be as simple or complex as you want because you can define any number of individual conditions, actions, and rules. In addition, you can create condition sets that contain one or more conditions and rule sets that contain one or more rules. You can also chain together these condition and filter sets so that multiple factors will be evaluated sequentially.

A ConditionSet class (see Listing 2) acts as a container for multiple conditions and also implements the ICondition interface. This composite design allows us to easily build complex conditions by ANDing them together. We could also add ORing abilities by introducing another composite container class. A RuleSet (see Listing 3) is a collection of rules that are processed sequentially until the first action is applied.

Building the Conditions

We'll now take the general framework and make it more specific by defining the conditions that we want to evaluate when someone calls us.

We'll begin by defining two universal conditions – AlwaysSatisfied and NeverSatisfied – that will make it easier to program certain features on the phone. We'll then add four specific conditions – UserMatches, DomainMatches, BeforeSpecifiedTime, and AfterSpecifiedTime – that will allow us to create a phone filter with wide applicability.

The initial six conditions listed in Table 1 have been selected to provide a usable filter and demonstrate the concepts. However, these are not the only conditions that could be established. In fact, you could define many other conditions, including any variable that can be measured and evaluated by the phone, such as whether you're currently on the phone.

Because an IP phone is part of a network, you can also evaluate any information located on it. For example, this call filtering application can look for information in your Microsoft Outlook Contacts database or evaluate data from a corporate database.

The connection of the phone to the network opens up some interesting possibilities. For example, a condition could evaluate information in a customer service database and the call filter could direct incoming calls to specific people depending upon what products that customer owned. It could also prioritize calls from customers who had purchased a premier service contract or automatically escalate calls to an expert if the call came from someone who had already called with an urgent problem. All of the conditions controlling the phone's actions could be dynamically generated and evaluated as the database entries changed.

Let's begin by defining our two universals – the AlwaysSatisfiedCondition and the NeverSatisfiedCondition.

```
public class AlwaysSatisfiedCondition implements
ICondition {
    public boolean isSatisfied(Object object) {
        return true ;
    }
}
```

The universal AlwaysSatisfiedCondition is particularly useful for creating default rules. For example, Rule1 will fire if Condition1 is satisfied, Rule2 will fire if Condition2 is satisfied, and Rule3 will always fire if none of the previously defined rules are satisfied.

```
public class NeverSatisfiedCondition implements
ICondition {
    public boolean isSatisfied(Object object) {
        return false ;
    }
}
```

Although the NeverSatisfiedCondition is not used in this application, its concept and potential uses complement the AlwaysSatisfiedCondition.

The four other conditions specified in Table 1 are defined in the Java code shown in Listings 4–7. Particularly interesting is the code for the UserMatchesCondition and DomainMatchesCondition in Listings 4 and 5. These routines use the call filter hook infrastructure and SIP address information to determine the caller ID or domain address.

The UserMatchesCondition code (see Listing 4) brings

AlwaysSatisfied	A universal condition that can force the application of a rule
NeverSatisfied	A universal condition that can prevent the application of a rule
UserMatches	Determines if the caller's phone number matches a user-specified phone number
DomainMatches	Determines if the caller's IP address matches a user-specified IP address
BeforeSpecifiedTime	Determines if the current time is before a user-specified time
AfterSpecifiedTime	Determines if the current time is after a user-specified time

TABLE 1 The six conditions used in the call filtering application

AcceptCall	Puts the incoming call through to the phone and causes the phone to ring
RedirectCall	Transfers the incoming call to another user-specified phone number
RejectCall	Transfers the incoming call to voice mail

TABLE 2 The three actions defined for the call filtering application

together the discussion of SIP addresses and the call filter hook infrastructure to determine if a designated user matches the user who placed an incoming call.

To determine the caller ID, we safely cast the generic object parameter into the CallFilterHookData object, then pull out an abstract PAddress. While the Pingtel phone supports only SIP now, it could support other call-control protocols; this abstraction requires a cast from a PAddress to a PSIPAddress. We then compare the user field to determine if they match.

The DomainMatchesCondition (see Listing 5) is identical to the UserMatchesCondition except that we're checking the domain address instead of the user name.

The AfterSpecifiedTimeCondition (see Listing 6) is satisfied when the current time is greater than the pivot time supplied in the object's constructor. There's some code required to pull out the hour and minute components of the time/date, store them, and then compare them to the pivot time. The BeforeSpecifiedTimeCondition (see Listing 7) works in the same manner as the AfterSpecifiedTimeCondition, but is satisfied when the current time is less than the pivot time.

Building the Actions

The call filtering application will be able to perform the three possible actions defined in Table 2. For each incoming call the phone will accept the call, redirect it to another number, or reject it by transferring it directly to voice mail.

The Java code in Listing 8 defines the three possible actions – accept, reject, and redirect the call. As part of the IAction interface the actions are given an object data parameter to work with. Since we specify this object parameter when evaluating the rule, it can be safely cast to a CallFilterHookData object and manipulated as needed.

Figure 3 shows the original conceptual framework along with the conditions and actions that we've defined. You could easily add additional conditions or actions to this same framework.

In Figure 3 note that the same framework can be applied to many different types of apps that could be built for a Java-enabled phone. Most phone applications will have a very similar structure consisting of conditions, actions, and rules. For example, we could easily transform this call filtering application into a distinctive ring application by simply changing the existing set of actions into ones that played different sound files depending on the caller or the time of day (see Figure 4).

Building the Rules

We've now defined the conditions we want to evaluate and the potential actions that can be applied. However, we haven't tied together the conditions and actions. It's the function of the rules to combine these two classes of objects and then tell the program which action to take when a condition or condition set is met.

The rules that could be defined are almost infinite. We can set up the filter to compare phone or IP addresses, or to act differently depending upon the time of day, or if you're already on the phone. We can create rule sets that string rules together so that the execution is sequential and different actions are taken depending upon the results.

Let's look at a specific example of call filtering rules that a user might want to apply. In this example, it's two days before the end of the quarter and Martha Smith, the regional sales manager for Ultimate Widget Corporation, is in the midst of negotiating new contracts with her two biggest customers – Global, Inc., and Star Enterprises. She needs to spend most of her time on these deals in order to close them before the end of the quarter.

In the morning Martha decides to set up her call filter for the day. She wants to forward all her phone calls directly to voice mail except the calls from these two customers and from her husband Dan. She sets up the call filtering app to accept calls from Pete's office phone at Star Enterprise and from her husband Dan on any of his office, cell, or home phones. In addition, because she's negotiating with many decision-makers at Global – ranging from the department manager to the CFO and president – she wants to take any call that originates from the domain name global.com.

In the afternoon, Martha has to leave the office to visit another customer. Therefore, after 3:00 p.m. Martha wants all phone calls from Dan and Global to go to her cell phone and all calls from Star Enterprises to be transferred to her vice president of sales.

To make it easier to understand, let's create a table showing the rules we want to implement before we start programming. Table 3 shows four different types of conditions – a specified user on a particular phone, a specified user on many phones, any user from a specified domain, and all other users.

We can now build the specific rules that will implement Martha's call filter. In this example we hard code the rules to demonstrate the Java programming behind the application. In actual use, the call filtering app would have a user interface that automatically writes these rules based upon user input. You can see the entire source code for the generic call filtering application, including the user interface, by visiting the ***JDJ*** Web site.

Each of our call filter rules relies on the CallFilterHook implementation given in Listing 9. This class defines our CallFilterHook implementation and performs two important functions for each call – it initializes the RuleSet described in this article and evaluates the RuleSet for each incoming call.

Initialization is performed in the constructor of the object. First, we create the two redirect addresses that are part of the After 3:00 p.m. rules. Please note the try/catch blocks around the creation of these addresses and recall that a PSIPAddress is really a URL. Next we define and add the Before 3:00 p.m. rules, After 3:00 p.m. rules, and finally the default/catch-all rule. With the exception of the default/catch-all rule, all the rules make use of helper classes defined later in this article.

The hook framework invokes the HookAction method when a new call is detected. Our implementation verifies

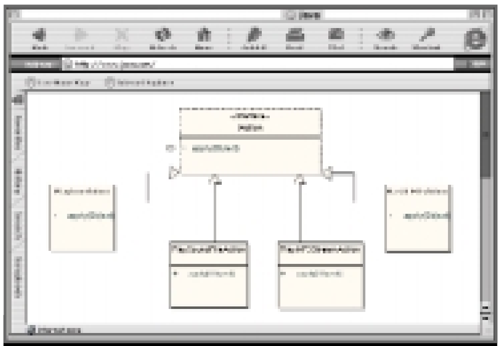


FIGURE 4 A call filtering application

BEFORE 3:00 P.M.		AFTER 3:00 P.M.	
CONDITION	ACTION	CONDITION	ACTION
pete@star.com	Accept	pete@star.com	Redirect to VP of sales
dan.smith@*	Accept	dan.smith@*	Redirect to 781-555-1234@gw.widget.com
*@global.com	Accept	*@global.com	Redirect to 781-555-1234@gw.widget.com
@	Reject	*@*	Reject

TABLE 3 The original call filtering framework

that the HookData object parameter is really a CallFilterHookData object and evaluates the RuleSet configured in the CallFilterHook's constructor.

As you can see in Table 3, we need to define six rules to handle the calls from three different users during two different time periods. The Java classes that define these six call filtering rules are given in Listing 10. Each of these helper classes relies upon the CallFilterHook implementation given in Listing 9.

Table 4 summarizes the call filtering rules represented by the six Java classes in Listing 10. This table makes it easier to understand each of these classes because it shows the condition set that each rule will evaluate and the action it will take if the condition set is satisfied. It also lists the AlwaysSatisfied rule that will be invoked if none of the first six condition sets are satisfied.

Installing the Hook

As the final step we need to install the CallFilterHook using the phone's hook manager. The CallFilterApp (see Listing 11) determines where the xpression will begin execution and provides a place to install the hook. Developers can define when their application will be loaded and executed through a properties file bundled in their .jar file (see Listing 12).

For this example, we'll load the application on phone startup and execute the application on demand. This means that installation will be performed by the onLoad method when the phone is powered up, and removal will be performed by the onUnload method when the phone is shut down.

Normally, the main() method would be invoked when the user selected the call filter's icon from the phone's application launcher. However, we haven't provided an implementation for the main() method here because it's part of the user interface that we haven't discussed. You can find the full source code, including the user interface, on the ***JDJ*** Web site.

Conclusion

We've demonstrated a simple Java application that can be used to filter and route calls on an intelligent Java-enabled IP phone. Most significantly, the application shows the ease with which new features can be added to a phone without relying upon a central programming environment to produce whatever apps people want. The two-year, $2 million wait for new phone functionality is a thing of the past with a Java-enabled phone.

We also showed a simple application of this filter to make Martha's life more focused and productive at a key juncture in the quarter. However, we've only hinted at the power of this phone. Remember that an IP phone is another device on a network, which means it can gather and evaluate information from anywhere on the network, be it LAN, WAN, or Web. For example, it's currently possible to set up your e-mail client or server to reject any e-mails that arrive from an address that's listed in one of the Internet databases of known spammers.

With the application we described here, you could set up your Java-enabled phone to eliminate all those telemarketing phone calls that interrupt your day at the most inconvenient times. Theoretically, an IP phone could refer to a similar Internet database that contains the phone numbers of all known telemarketers. Imagine, by referring to an up-to-the-minute database your phone could automatically generate a busy signal for any telemarketer trying to sell you a time-share condo or new programming tool.

If you really wanted to give some telemarketers a taste of their own medicine, you could change the actions to redirect each incoming telemarketing call to another telemarketer's phone number. You've now made your day more productive, and you have the satisfaction of knowing that these telemarketers will be spending their days trying to sell products to each other.

As you can see, an intelligent, Java-enabled IP phone opens up enormous possibilities for Java application developers. For the first time, many different types of applications can be easily developed and implemented for such a phone. Whole new classes of services can be easily built and deployed, creating a wealth of opportunities that were never imagined when Alexander Graham Bell first asked for help 125 years ago.

For More Information

The complete source code for the generic call filtering application is available at www.JavaDevelopersJournal.com. In addition, you can learn more about SIP, STAPI, and Java extensions for telephony and get more information about Java-based IP phones by visiting www.pingtel.com. A software developers kit for the Pingtel phone is available free at http://appdev.pingtel.com.

RULE	CONDITIONS SATISFIED	ACTION TAKEN
AcceptUserAtDomainBeforeTime	• The user matches the configured user • The domain matches the configured domain • It's before the configured time	Accepts the incoming call
AcceptUserBeforeTime	• The user matches the configured user • It's before the configured time	Accepts the incoming call
AcceptDomainBeforeTime	• The domain matches the configured domain • It's before the configured time	Accepts the incoming call
RedirectDomainAfterTime	• The domain matches the configured domain • It's after the configured time	Redirects the incoming call to 781 555-1234@gw.widget.com
RedirectUserAfterTimeRule	• The user matches the configured user • It's after the configured time	Redirects the incoming call to 781 555-1234@gw.widget.com
RedirectUserAtDomainAfterTime	• The user matches the configured user • The domain matches the configured domain • It's after the configured time	Redirects the incoming call to VP of sales
AlwaysSatisfied	• All calls that don't match any of the other condition sets	Rejects the incoming call

TABLE 4 A call filtering application

AUTHOR BIO

Robert Andreasen is the technical lead for Java applications at Pingtel. He has been writing applications for computers since 1993 and for phones since 2000, when the Pingtel xpressa was first available for independent developers.

bandreasen@pingtel.com

J2ME Cryptography

Implement a simple password-based encrypted database program

WRITTEN BY
JOSH ECKELS

The absence of standard and familiar Java APIs presents one of the biggest obstacles when developing for Java 2 Micro Edition (J2ME). Since J2ME targets much smaller devices, it lacks many libraries and features that are normally available in larger Java installations.

On Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE), the Java Cryptography Architecture and Java Cryptography Extension help streamline the process of adding security to a project. The J2ME platform, however, lacks crypto APIs.

Of course, many J2ME devices, such as Palm handhelds, aren't suited for common cryptographic functions. Their constrained RAM and processors can make operations, such as public key cryptography, infeasible. Still, there's a real need for security on small devices, and they can perform a useful subset of cryptographic operations.

This article explores the use of cryptography on J2ME devices and walks though an implementation of a simple password-based encrypted database program. The database holds user login information and can be used to manage accounts for different machines and Web sites. Although the example targets J2ME for Palm handhelds, its design maximizes portability.

Ciphers

Ciphers are algorithms that use a key to transform a message (plaintext) into an encrypted message (ciphertext) or vice versa. Symmetric ciphers rely on a single key for both encryption and decryption. DES, TripleDES (also called DESede), and Blowfish are symmetric ciphers. All three operate on 64-bit input blocks and produce 64-bit output blocks.

Modes are applied to ciphers to define their behavior. In the simplest mode, Electronic Code Book (ECB), each input block always produces the same output block. Although this makes implementation simple, the one-to-one mapping of input to output leaves the ciphertext vulnerable to attacks. Patterns in the encrypted data become apparent in the ciphertext if there are repeated input blocks, making cryptanalysis much easier.

The cipher block chaining (CBC) mode strengthens the ciphertext by making each block of ciphertext dependent on both the corresponding block of plaintext and all previous plaintext blocks as well. It uses an XOR feedback register to modify the plaintext before encrypting and after decrypting. At the start of an encryption or decryption operation, the XOR register is set to a randomly generated initialization vector (IV) that's transmitted or stored along with the ciphertext. It's not terribly important that the IV is truly random, but it should be different each time plaintext is encrypted with the same key.

Since plaintexts might be any length, extra bytes must be added to pad the input to a multiple of the block size in length. Public Key Cryptography Standard #5 (PKCS#5) padding is the most common padding scheme for symmetric block ciphers.

Choosing appropriate ciphers for J2ME requires special care, since most J2ME devices have relatively slow processors. Table 1 shows some rough benchmarks for a PalmV. The key setup column gives the time it takes for an algorithm to initialize itself with a key. The encryption column shows the amount of time it takes to encrypt one block (8 bytes) of data. Naturally, different implementations of the algorithms will have different performance. The data in Table 1 is by no means a definitive performance analysis.

For these implementations, TripleDES and Blowfish take significantly longer to initialize than DES, but Blowfish encrypts data faster than DES or TripleDES. Depending on the application, either key setup or encryption might dominate the running time. Choosing an appropriate cipher always involves weighing the trade-off between performance and the desired level of security.

Hashes, Passwords, and Keys

Hash algorithms are one-way functions that turn an arbitrary message of bytes into a fixed-length digest, or fingerprint, usually not much more than a dozen bytes long. Good hash algorithms should be collision-resistant, making it computationally infeasible to derive the original message from the digest. Even a minor change in the input should drastically change the resulting fingerprint.

ALGORITHM	KEY SETUP (SEC)	ENCRYPTION (IN MSEC)
DES	1.3	100
TripleDES	3.8	290
Blowfish	34	80

TABLE 1 Symmetric cipher performance

MD5 and SHA1 are two commonly used hash functions.

Applications commonly use hash functions to transform user-supplied passwords into symmetric keys. Several standards define techniques to convert passwords, but PKCS#5 specifies a simple technique: convert the password string to a byte array and use it as the input to a hash algorithm. The resulting digest serves as the key.

Running the password through a hash function serves several purposes. First, the hash function transforms variable length passwords into uniform-length hashes. Also, character strings follow certain patterns and good symmetric keys should be as close to random bytes as possible. Hashing the password removes the patterns, making cryptanalysis much harder.

Using a password-based key means that the key is only as secure as the password. If the number of likely passwords is smaller than the total number of keys, it's more efficient to search through all possible passwords than to attack the key directly. It's pointless to use a cipher with a huge key space unless users choose long passwords or passphrases. DES has a key space of 64 bits, of which only 56 are actually used. That's roughly equivalent to randomly selecting nine characters from the set upper- and lowercase letters and numbers. Users are unlikely to choose passwords longer than that for J2ME programs since text entry is typically inconvenient.

The dictionary attack is a common way to attack password-based systems. In it an attacker computes a list of keys from a dictionary of likely passwords. Once the attacker has obtained the encrypted data, he or she uses each potential key to decrypt the ciphertext until one of them produces meaningful plaintext.

A technique called *salt* helps guard against dictionary attacks. Salt is a set of random bytes that are stored unencrypted along with the ciphertext and are added to the password before it's hashed. It prevents an attacker from precomputing a list of keys since the password dictionary generates a different set of keys for each different salt value. Salt doesn't completely defeat a dictionary attack, but it does make it much more computationally intensive.

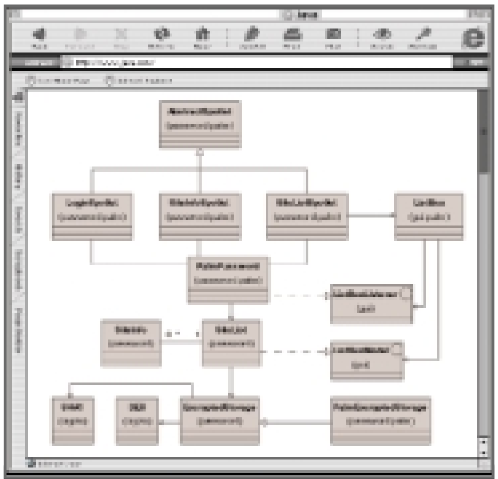


FIGURE 1 Class diagram

Implementing an Encrypted Password Database

Deploying on J2ME requires a virtual machine, configuration, and profile. Sun's KVM is the most compact virtual machine that fulfills the J2ME requirements. Configurations define a set of APIs for use on certain classes of devices. For example, the Connected Limited Device Configuration (CLDC) targets compact devices, such as cell phones and PDAs. Profiles extend the configuration and add functionality applicable to a particular domain. As of this writing, the Mobile Information Device Profile (MIDP) is the only profile that's been finalized. It contains functionality intended for cell phones and two-way pagers.

The primary design goal for this application is to make it as simple as possible to port to different J2ME configurations and profiles. As presented, it targets the CLDC and a PalmOS-specific profile. It contains two device-specific groups of functionality: the GUI and the storage mechanism. The design accommodates different GUIs by separating the application's business logic and presentation. It solves the device-dependent storage problem using the Template pattern. An abstract superclass relies on methods in the concrete subclasses to store the data.

Another important goal is maintaining acceptable performance on the target device. Therefore, the design encapsulates the usage of the computationally intensive algorithms in a single class. Swapping in alternate hash functions or encryption algorithms makes it possible to customize the program for the performance characteristics of different devices. For this application, SHA1 and DES provide reasonable performance and are as secure as the expected passwords (see Figure 1)

Cryptographic Algorithms

The supplied code includes DES and SHA1 implementations in the crypto package. Both take care of padding the input data if needed. The SHA1 class uses the padding defined by the SHA1 algorithm and DES uses PKCS#5 padding and CBC mode. The algorithms' inner workings are beyond the scope of this article. Other sources, such as *Applied Cryptography* by Bruce Schneier, discuss the algorithms and cryptanalysis in detail.

The DES class throws a CipherException if it encounters a problem encrypting or decrypting. Possible problems include invalid parameters (like an IV that's not 8-bytes long or attempting to decrypt an array that's not a multiple of 8 bytes in length) and invalid padding after decryption. Normally, using the wrong key or IV will result in improperly padded plaintext.

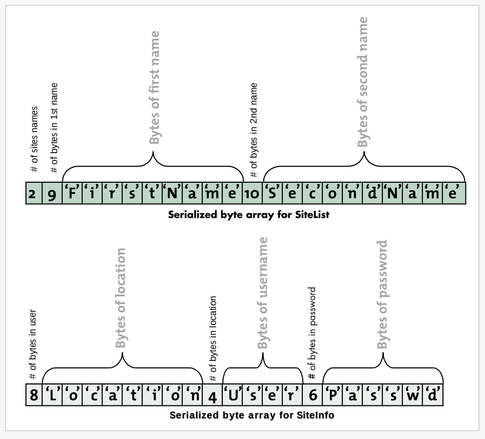
Data Objects

Two data classes, SiteList and SiteInfo, encapsulate the user's information. SiteList keeps track of all the SiteInfos, each of which stores an individual site's profile. Since J2ME doesn't support serialization natively, the classes handle saving and restoring their own state in their load() and save() methods. They use simple byte-array structures as shown in Figure 2.

The SiteInfo class wraps four strings: the site's name, location, username, and password. Each setter method trims the string to ensure it will be less than 127-bytes long (Java's maximum byte value). The class handles serializing its location, username, and password.

The SiteList class keeps track of all the SiteInfos in the system. It holds all the SiteInfos in a vector and is responsible for storing their name strings when it's serialized.

Since hashing the user's password, setting DES up for decryption, and decrypting the data are all computationally intensive operations, it's important to decrypt information only as needed. Keeping the site names in one byte array allows the application to start up faster because the application needs to display only the site names at startup. It can wait until the user selects a particular site to load and decrypts the rest of the SiteInfo data.



# of sites names
# of bytes in 1st name
Bytes of first name
# of bytes in 2nd name
Bytes of second name
2 9 'F' 'i' 'r' 's' 't' 'N' 'a' 'm' 'e' 10 'S' 'e' 'c' 'o' 'n' 'd' 'N' 'a' 'm' 'e'
Serialized byte array for SiteList
# of bytes in user
Bytes of location
# of bytes in location
Bytes of username
# of bytes in password
Bytes of password
8 'L' 'o' 'c' 'a' 't' 'i' 'o' 'n' 4 'U' 's' 'e' 'r' 6 'P' 'a' 's' 's' 'w' 'd'
Serialized byte array for SiteInfo

FIGURE 2 Serialized byte array for SiteInfo

Persistent EncryptedStorage

The abstract class EncryptedStorage handles the application's cryptography. It turns the password and salt into a DES key and deals with encrypting and decrypting the serialized SiteList and SiteInfos. When saving data it first creates an initialization vector. Then it encrypts the plaintext bytes and concatenates the IV with the ciphertext. EncryptedStorage then uses the Template pattern and calls an abstract method to store the combined array in the device's storage system. The class gets random bytes from java.util.Random, which produces bytes that are random enough for salt and IV values, but not for keys. Listing 1 shows the code that does the hashing and encrypting.

In this case, PalmEncryptedStorage subclasses EncryptedStorage and talks directly with the Palm's storage system, the database. The Palm-specific classes include com.sun.kjava.Database, a class that permits access to an application's database on the Palm, stored in a set of records. At startup the PalmEncryptedStorage checks if it has already created a database. If not, it makes a new one, generates salt, and stores it in record 0. The IV and ciphertext for the SiteList go in record 1. SiteInfos get stuffed into subsequent records. The SiteInfo at index 0 in the SiteList is kept in record 2, index 1 is kept in record 3, and so forth.

Listing 2 illustrates how the PalmEncryptedStorage interacts with the Palm's database. Three values identify each database: a creator ID (managed by Palm Computing at www.palmos.com/dev/tech/palmos/creatorid/), a database type ID, (separates different applications from a single creator ID), and a database type string (shown in the list of databases in the Palm's memory-usage list).

The class's constructor tries to open an existing database. If it can't find one, it creates a new one, opens it, and stores the 8-byte salt value. The setRecordBytes() method sets the bytes of the specified record index. If a record at that index already exists, it's overwritten; if no record exists, a new record is appended.

User Interface

A class called com.sun.kjava.Spotlet draws the GUI and handles user input. Event handling is very different from Swing's event handling. A Spotlet must register itself to receive user input events and only one can be registered at a time. For stylus events, the Spotlet gets the (*x*, *y*) coordinates and must interrogate each GUI widget to see if it occupies those pixels and then acts appropriately. For example, if a user clicks on a text field it should get the focus. For text input, the Spotlet must check if any of the GUI components have the focus and then pass it to that component. Listing 3 is a code snippet from SiteListSpotlet that does event handling.

In this program an abstract class, AbstractSpotlet, draws the familiar Palm UI tab at the top of the screen to show the user which application is running. It also keeps a reference to the PalmPassword object so that the Spotlet can interact with the rest of the application. There are three subclasses, one for each screen in the UI.

The first is the LoginSpotlet. It gets the password from the user and informs him or her if the password was not correct. SiteListSpotlet shows the list of sites and permits the user to either select one to display or create a new entry (see Figure 3). The SiteInfoSpotlet allows the user to view and edit a site's information and then save it (see Figure 4).

Unfortunately, the Palm-specific classes don't include a list GUI widget. The ListBox class provided works like list boxes in other Palm applications. It's backed by a ListBoxModel that supplies the data to be displayed and sends selection events to a ListBoxListener. It calculates the number of elements that can fit vertically, based on the size of the box, and draws as many elements from the ListBoxModel as can fit. It uses the Palm-specific ScrollBar widget to handle scrolling, shifting the elements displayed in response to user input. If the user selects one of the elements in the list, the ListBox notifies its ListBoxListener that an element in the list was selected.

"there's a real need for **security on small devices,** and they can perform a useful subset of cryptographic operations"

The PalmPassword class contains the main method for the application and acts as a controller for the system. It instantiates the three Spotlets that all keep a reference to the PalmPassword object. When the Spotlets get events from the user, they call methods in the controller to process the request and display other Spotlets as needed.

Building and Running the Application

Building an application for J2ME devices takes a few more steps than building for J2SE. J2ME runtime environments work with the same Java compilers as J2SE. Two additional steps are needed though: preverifying the class files and building the preverified files into a Palm application. Preverification checks each class file to make sure it's a valid Java class file, and stores verification information in it. This minimizes the amount of verification the KVM needs to do, improving runtime performance.

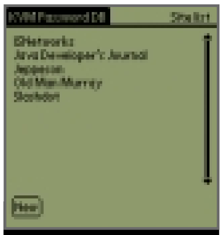


FIGURE 3 SiteListSpotlet

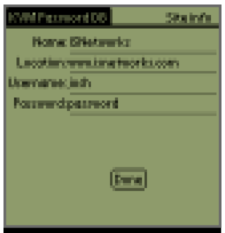


FIGURE 4 SiteInfoSpotlet

Author Bio

Josh Eckels is a software engineer at ISNetworks. His work focuses on security-related development in Java. His recent projects include an S/MIME library for JavaMail, a JCE provider, and PKCS#10 and PKCS#12 implementations. Josh has a BS in computer engineering from Northwestern University.

Setting up an IDE or a build tool to handle the process will save a lot of time, as will installing an emulator for the target device on the development workstation. Sun provides detailed installation and usage instructions for the J2ME libraries and utilities at http://developer.java.sun.com/developer/technicalarticles/wireless/palm/. The page also describes how to get a Palm emulator and transfer a ROM image from a Palm device.

Unfortunately, the section labeled "The J2ME CLDC Development Environment" contains a couple of errors. So run the following commands from the c:\j2me_cldc\tools\palm\ directory instead:

```
md classes
javac src\palm\database\*.java -d
classes
copy
src\palm\database\DefaultTiny.bmp
classes\palm\database
copy src\palm\database\Wrapper.prc
classes\palm\database
```

The build.bat file included with the source compiles the source code, preverifies the compiled classes, stores the results in ..\preverified, and builds them into a Palm application. Be sure to edit the first line of the batch file to point to the proper directory. After installing the KVM as documented by Sun, simply drag-and-drop PalmPassword.prc onto an emulator or HotSync it to a Palm device and run it.

Summary

The code provided with this article can be downloaded from the ***JDJ*** Web site. It gives a solid base for cryptography in J2ME and shows how to separate device-specific code from the main application logic, both for the GUI and for persistent storage. In addition, it provides an implementation of password-based encryption without relying on other APIs. Different hash and encryption algorithms can easily be swapped in to increase the cryptographic security of the data.

Although the example program is functional, it's still far from feature-complete. As presented there's no way to delete a SiteInfo from the list or sort the list. The application could give the user some sort of progress bar during the password hashing and key setup. More fields might be added to let the user input more information about a site, or the ListBox could be changed to display more than just the site's name. Also, if something goes wrong with encryption or decryption, it would be nice to inform the user instead of just swallowing the exception.

A conduit that synchronizes data with an application on the user's desktop machine might also be useful. Because all the device-specific code is kept isolated, it would be simple to create a Swing-based application that could read the encrypted data and allow the user to manage the password database.

It's important to remember that cryptography doesn't automatically make a system secure. For this application, a poorly chosen password will reduce the security significantly. Also, it's necessary to exit the program after use or the passwords will be vulnerable to anyone who happens upon the device. Even the best security is still only risk management.

josh@isnetworks.com

Listing 1

```
public void init( String password ) {
  byte[] toBeHashed = concat( getSalt(),
    password.getBytes() );
  byte[] hash = mSHA1.digest( toBeHashed );
  byte[] keyBytes = new byte[ 8 ];
  System.arraycopy( hash, 0, keyBytes, 0, 8 );
  mDES.setKey( keyBytes );
}

private byte[] encrypt( byte[] plaintext )
throws StorageException {
  try {
    byte[] iv = getRandomBytes( 8 );
    byte[] ciphertext = mDES.encrypt( plaintext,
      iv );
    return concat( iv, ciphertext );
  }
  catch( CipherException e ) {
    throw new StorageException( e.getMessage() );
  }
}
```

Listing 2

```
public PalmEncryptedStorage() {
  mDatabase = new Database( DB_TYPE_ID,
    CREATOR_ID, Database.READWRITE );
  if ( !mDatabase.isOpen() ) {
    Database.create( 0, DB_TYPE_STRING,
      CREATOR_ID, DB_TYPE_ID, false );
    mDatabase = new Database( DB_TYPE_ID,
      CREATOR_ID, Database.READWRITE );
    mDatabase.addRecord( getRandomBytes( 8 ) );
  }
}

private void setRecordBytes( int recordNumber,
  byte[] bytes ) {
  if ( mDatabase.getNumberOfRecords() <=
    recordNumber ) {
    mDatabase.addRecord( bytes );
  }
  else {
    mDatabase.setRecord( recordNumber, bytes );
  }
}
```

Listing 3

```
public void penDown( int x, int y ) {
  if ( mListBox.contains( x, y ) ) {
    mListBox.handlePenDown( x, y );
  }
  else if ( mNewButton.pressed( x, y ) ) {
    mPalmPassword.showSiteInfo( new SiteInfo() );
  }
}

public void penMove( int x, int y ) {
  if ( mListBox.contains( x, y ) ) {
    mListBox.handlePenMove( x, y );
  }
}
```

Download the Code!
www.JavaDevelopersJournal.com

Hello World

A Beginner's Guide to Writing Applications for the MID Profile

There's nothing truly unusual about writing a MIDlet

WRITTEN BY
JASON BRIGGS

The first thing you're likely to see, upon sitting down to learn a new language, is the ubiquitous "Hello..." application. My father bought me the *TRS-80 Basic for Kids* book when I was 8-years old, and I'm pretty sure that the first example was either "Hello World" or "Hello, my name is...."

As the years progressed and I acquired different computers (BBC, Commodore, Macintosh, PC, etc.) and learned different languages (Assembler, Cobol, Pascal, Modula-2, C, C++), it's always been "Hello World" in some form that I've ended up writing for my first exercise with the language.

If you're sitting down to Java for the very first time, I suggest you initially start with the excellent Java Tutorial (http://java.sun.com/docs/books/tutorial/), and then come back to this article. However, for the rest of us – and in the very best tradition of Hello World applets, applications, servlets, and EJBs – may I present the Hello JDJ MIDlet... at the very least, so I won't be lynched by the "Secret Society for Ensuring that the First Thing Anyone Learns Is "Hello World."

In the Beginning There Was...

If you haven't done so already, download and install the latest version of the Java Development Kit (1.3.0 or higher) and also download and install the J2ME Wireless Toolkit. You can currently find the toolkit at http://java.sun.com/products/j2mewtoolkit/download.html

Note that the MIDP reference implementation is also usable, but the WTK has customizable skins (see Definitions), so it's a bit more fun. If you intend to use Sun's Forte Integrated Development Environment for your development efforts, then you can also install the WTK integrated with Forte. However, for the purposes of these examples, we'll work from the command prompt – and we'll also assume a couple of standard directories, namely:

```
 c:\jdk1.3  –  the root directory
for the JDK
 c:\j2mewtk  –  the root directory
for the toolkit
```

And Then There Was Code...

There's nothing truly unusual about writing a MIDlet (MID applications that run on the K virtual machine). There are a couple of new packages to import, a couple of new methods to implement, but apart from that there's nothing really frightening – if you've written applets and applications in the past.

So let's look at our first MIDlet, defined in a file called HelloJDJMIDlet.java (see Listing 1).

Looking at the important lines of the code, we see that lines 1 and 2 import the necessary classes for the MIDlet – the MIDlet super class itself, and all the classes in the javax.microedition.lcdui package (lcdui being LCD user interface).

Line 6 declares a display object, which is the controller for display and input devices for the system.

Line 7 declares a TextBox, which we will use to display a message to the user.

Line 10 declares and instantiates a Command object, which defines an action containing a label ("Exit"), a type (specifying what the command will be used for), and a priority (describing the importance of the command in comparison to other commands on the screen).

Lines 11–18 declare and instantiate a command listener (used to handle events) – this listener checks to see whether an event occurred on the command defined in line 10, and if so, exits the MIDlet.

Line 22 retrieves the single instance of the Display object for the MIDlet. (*Note:* there is only one instance of a Display per MIDlet, hence we call the getDisplay() method, passing *this* as the parameter.)

Line 23 instantiates the TextBox with a title ("Simple MIDlet"), text contents ("Hello JDJ!"), a maximum size (256) and input constraints (TextField.ANY). Input constraints are used to restrict the user's input – in this case, we're not restricting the input at all.

Line 26 is the beginning of one of the methods required by a MIDlet – startApp() is called when the MIDlet is made active.

Lines 27 and 28 add the command *bye* to the text object, and then set the command listener.

Line 30 sets the current displayable object of the display to be the TextBox.

Finally, lines 33 and 36 define the other two methods required by a MIDlet: pauseApp() is called – rather obviously – when the MIDlet is paused, and destroyApp(), when the MIDlet is required to release its resources and shut down.

One thing to note about this MIDlet is that unlike an applet, the MIDlet is not, itself, a displayable object; therefore, you would not necessarily want to define a paint() method in the MIDlet code as you might in an applet. An applet extends from java.awt.Panel (and, further up the chain, from java.awt.Component), which means it's a displayable component – in MIDP, objects that can be placed on the screen extend from the javax.microedition.lcdui.Displayable abstract class (whereas MIDlet extends from java.lang.Object).

Compiling the Code

If you're anything like me, when working from a text editor and command prompt (rather than using a development environment, which does everything for you), you end up creating batch files to automate processes – because we developers are a lazy lot. And why not?

In this case, I've created a working directory on my C drive (dev\midp) with two subdirectories below it called classes and tmpclasses. HelloJDJMIDlet.java is located in the midp directory.

```
c:\dev\midp
         |
         |------ classes
         |
         |------ tmpclasses
```

To compile the MIDlet, we use the Java compiler as we would with any other Java code:

```
c:\jdk1.3\bin\javac -g:none -d tmp-
classes -bootclasspath
c:\j2mewtk\lib\midpapi.zip -class-
path tmpclasses;classes
HelloJDJMIDlet.java
```

Where:

- ***-g:none:*** Means we want to generate no debugging info
- ***d:*** Specifies the output directory for our files
- ***bootclasspath:*** Forces the compiler to use the midpapi.zip file instead of its default bootclasspath.
- ***classpath:*** Defines where other supporting classes might be found

So in tmpclasses we should end up with two files: HelloJDJMIDlet.class and HelloJDJMIDlet$1.class (this is the result of compiling the CommandListener inner class, used for handling events – see lines 11–19 of Listing 1).

Preverification

Before running an application, all classes must be preverified. This prepares them for use in the KVM (K virtual machine). To preverify the files we have just created:

```
c:\j2mewtk\bin\preverify -classpath
c:\j2mewtk\lib\midpapi.zip -d
classes tmpclasses
```

We should now have preverified copies of both files in the classes directory.

Application Descriptor and Application JAR

Each suite of applications to be run in a MIDP-capable device will have a descriptor and a JAR file. The descriptor defines a number of parameters that describe the environment and individual MIDlets in the suite. The JAR file contains the classes, plus a manifest file with some extra information to a standard Java JAR. If you don't know anything about JAR files and manifests, I recommend looking at the following tutorial: http://java.sun.com/docs/books/tutorial/jar/basics/index.html

The next example shows some of the information required for the descriptor we'll be using for our Hello test (note, the line numbers are not necessary in the actual file itself):

1. MIDlet-Name: JDJ
2. MIDlet-Vendor: Java Developers Journal
3. MIDlet-Version: 1.0.0
4. MicroEdition-Configuration: CLDC-1.0
5. MicroEdition-Profile: MIDP-1.0
6. MIDlet-1: HelloJDJMIDlet,, HelloJDJMIDlet
7. MIDlet-Jar-Size: 1770
8. MIDlet-Jar-URL: jdj.jar

Line 1 names the suite.

Line 2 names the vendor of the suite.

Line 3 is the version of the suite.

Line 4 defines the Configuration that is required by this suite (in this case, the Connected, limited Device Configuration 1.0)

Line 5 defines the Profile required by the suite (Mobile Information Device Profile 1.0)

Line 6 describes the MIDlet included in this suite. The information required is the MIDlet name, the name of a PNG image in the JAR to be used as an icon for this MIDlet, and finally the name of the class. If there was more than one MIDlet, there would be other lines beginning MIDlet-2, MIDlet-3, MIDlet-4, and so on.

There are two items missing from the descriptor. These are MIDlet-Jar-Size and MIDlet-Jar-URL. I include them separately (below), because the above information is used in both the descriptor and the JAR manifest file.

The steps to package the application so far are:

1. Create a manifest file (for example: manifest.mf) with the above information (minus the line numbers) in the classes directory
2. JAR the class files and include the contents of the manifest file. For example:

```
cd classes
C:\jdk1.3\bin\jar -cfm jdj.jar man-
ifest.mf HelloJDJ*.class
```

Where:

- ***jdj.jar:*** The name of the JAR file we are creating
- ***manifest.mf:*** The manifest file whose

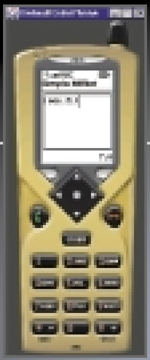


FIGURE 1 MIDlet running in the emulator

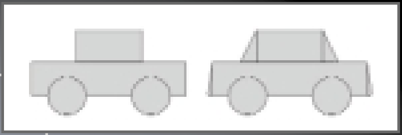


FIGURE 2 Two "cars" drawn with simple shapes

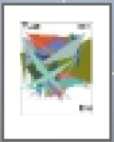


FIGURE 3 Scan conversion of a convex polygon

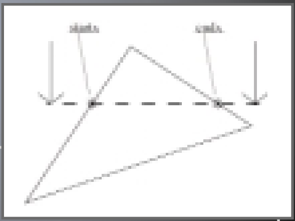


FIGURE 4 Polygons drawn on the emulator

contents we want to include
- ***HelloJDJ*.class:*** Grabs Hello JDJMIDlet.class and Hello JDJMIDlet$1.class

3. Create an application descriptor (jdj.jad) – duplicating the manifest file and including the information from lines 7 and 8.

Line 7 is the size of the JAR file, in bytes.

Line 8 is a URL from which the JAR can be loaded. (As we are delivering this application locally, I've just entered the URL as jdj.jar.)

Running the MIDlet

At the end of this process, we should have two files:
1. An application descriptor – jdj.jad
2. An application jar – jdj.jar – containing the class files and a manifest

We are now ready to test the application. There are a couple of ways to run the emulator, but as we're following a command-line driven approach, we will continue using this method:

```
c:\jdk1.3\bin\java -cp
c:\j2mewtk\lib\kenv.zip -
Dkvem.home=c:\j2mewtk
com.sun.kvem.midp.Main
DefaultColorPhone -descriptor
jdj.jad
```

Where:
- ***cp:*** Ensures the classpath points to the kenv.zip file
- ***Dkvem.home:*** The only required parameter for the emulator, the home directory of the toolkit
- ***DefaultColorPhone:*** The device for the emulator to use (DefaultGrayPhone, DefaultColorPhone, MinimumPhone, or Pager)
- ***Descriptor:*** The name of the application descriptor to use

The emulator should display a list of applications in the JAR file (there is only one in this case), and when you select the application, you should see something similar to Figure 1.

Of course, just to make life a bit more difficult, Sun has just released an "early access" version (1.0.2) of the toolkit, which works slightly differently from the previous release. So if you've downloaded this version, use the following command to run the app:

```
c:\j2mewtk\bin\emulator -
Xdevice:DefaultColorPhone -
Xdescriptor:jdj.jad
```

Where:
- ***Xdevice:*** Specifies the device to use
- ***Xdescriptor:*** The descriptor of the application to run

A More Advanced Example — Graphics

Graphics operations in a MIDlet are similar to those available in Java Standard Edition, such as drawImage(), drawRect(), drawArc(), and fillRect(), fillArc(). There are, however, a couple of notable exceptions. Methods such as draw3DRect(), drawOval() and, interestingly, drawPolygon() are missing from the graphics class. While drawing 3D rectangles and ovals can be quite easily accomplished by using either a combination of normal rectangles and/or lines, and with the latter, a call to drawArc() – the omission of polygons is a little more difficult to overcome.

Look at Figure 2 for an example of the problems caused by this lack of polygon drawing capability.

The left car is constructed from two rectangles for the body and two arcs for the wheels.

The right car is constructed from two rectangles and three triangles for the body, and again two arcs for the wheels.

web services EDGE conference&expo
October 24–25, 2001
XML EDGE conference&expo 2001
October 22–25, 2001
Santa Clara Convention Center Santa Clara, CA
FREE Registration
Register for one...attend both
Fundamentally Improving the Speed, Cost & Flexibility of Business Applications
Many New XML Products Will Hit the Market in Q3 of 2001
See them first at XMLEdge2001
REGISTER ONLINE TODAY SAVE $300
Opening Keynoter
Charles Goldfarb
The Father of XML
SYS-CON MEDIA
Owned & produced by SYS-CON EVENTS
WEB SERVICES CEO PANEL
Litwack CEO SilverStream
O'Connor President Sonic Software
Kutay CEO AltoWeb
O'Toole Chairman Cape Clear
Slama CEO Shinka Technologies
Morris CEO IONA Technologies
www.SYS-CON.com

Two things should be plainly obvious from this diagram:

1. The car on the right is slightly more realistic than the car on the left.
2. My artistic abilities are severely limited.

So, for our first graphics example in MIDP we will attempt to provide the missing polygon routine. To do this we will need to create a displayable component that we can paint on. We'll call our displayable component PolygonScreen and it will extend from the Canvas class in the javax.microedition.lcdui package:

```
public class PolygonScreen extends
javax.microedition.lcdui.Canvas {

…

}
```

In addition, we will need to provide a paint() method. This works in exactly the same fashion as it would in an applet: the paint method takes a Graphics object (one slight difference here: the class is javax.microedition.lcdui.Graphics, and not java.awt.Graphics), for which you can then invoke any of the various draw methods (see Listing 2).

Lines 2–4 just check to make sure we actually want to paint at the moment.

Line 6 sets the current drawing color to a random color using a function called randRange(), listed below.

Lines 8–13 set each value in an integer array to a random number to create a list of coordinates for a triangle – in the form (*x*,*y*),(*x*,*y*),(*x*,*y*).

Finally, line 15 calls drawPoly() with the coordinates array, the number of coordinates (six because it's a triangle) and the Graphics object.

The supporting methods for the paint method are shown in Listing 3.

The method randRange() returns a random number not greater than *n*. The "guts" of the drawPoly() method are discussed below.

Drawing a Polygon

Way back in 1994, Chris Egerter (www.egerter.com) wrote a tutorial on drawing filled polygons, using C code for his example. Hunting through – literally – a pile of source code I've built up over the years, I found a copy of this source and have ported it to Java with a few minor changes to get things working. The drawPoly() routine is used for painting convex polygons, and not for concave polygons (see Definitions). The process – called *scan conversion* – calculates the left and right edges for the polygon shape based on the current *Y* coordinate (placing the results into two integer arrays), and then draws a horizontal line between these two points with the following method:

```
g.drawLine(startx[y], y, endx[y],
y);
```

See Figure 3 for a simple illustration of how this process works.

The other addition to the application is PolygonEngine.java:

```
public class PolygonEngine imple-
ments Runnable {

…

}
```

This adds simple thread handling so we can provide animation in our MIDlet. PolygonEngine is constructed by passing the PolygonScreen component as a parameter. The run() method repeatedly sleeps an animation thread for a few milliseconds before calling the repaint() method on the screen component (see Listing 4).

The final result is an application that continually paints colored triangles on the screen until the user decides to exit (see Figure 4) – completely useless, of course, apart from situations where you might want a visual demonstration of the graphics performance of a particular device.

Summary

That's a basic introduction to writing applications for the MID profile. We've discussed how to write a simple MIDlet, compiling and packaging, and looked at some of the graphics operations available. Next time we'll take a look at the packages we have not covered yet – persistence and IO – and look at how we might interact with a server in our apps.

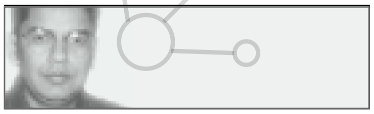
Definitions

- ***Polygon:*** A closed figure made by joining line segments, where each line segment only intersects two others.
- ***Convex polygon:*** If, for any two points that lie within a polygon, a line segment connecting them is also inside the polygon, then it is convex.
- ***Concave polygon:*** Any polygon that is not convex.
- ***Skin:*** A customizable user interface for an application.

Author Bio
Jason Briggs is a Java developer. He is currently thinking about designing a Java-enabled, hand-held device that will sell in the billions (of course) – and be used to brainwash the world's population – allowing him to complete his 10-year plan for global domination.

jasonbriggs@sys-con.com

J2EE EDITORIAL



J2EE Has Come a Long Way

–continued from page 18

tive for companies is to consider cheaper solutions in the Java world. Open source products – such as Enhydra from Lutris and Jserv from Apache – are equally high-quality products. And all applications may not require the degree of scalability, performance, and robustness that vendors like BEA, iPlanet, and WebSphere tout with their product suites; however, when choosing the vendor, you need to look at their financial status. You don't want to be stuck with an unsupported product or one that gets tagged with a high sticker price because the vendor got bought out.

A colleague of mine mentioned that he heard from one of the product managers at a leading application server vendor that the vendor was considering bundling the application server products with their hardware and selling this as a package. This will certainly bring costs down. Plus the costs and effort associated with training, installation, administration, and maintenance of these products will be reduced considerably. But, in that case, you lose the benefits of vendor neutrality. You are back to the issue associated with buying a Windows PC with IE bundled in the package.

Consider the Costs and Trade-offs

My intention here is not to discourage you from developing applications on J2EE. I architect solutions on J2EE and I think it is one of the best choices available for enterprise development. Java is a great platform and J2EE has matured rapidly as an environment for developing enterprise applications. I just want to encourage you to think carefully about costs and trade-offs associated with J2EE-based development.

The best approach is to build incrementally as opposed to offering a solution that tries to solve world hunger.

Listing 1

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class HelloJDJMIDlet extends MIDlet {

   private Display display;
   private TextBox text;

   // provide a command to exit the app
   private Command bye = new Command("Exit", Command.SCREEN,
12);
   private CommandListener commandListener = new
  CommandListener() {
      public void commandAction(Command c, Displayable s) {
         if (c == bye) {
            destroyApp(true);
            notifyDestroyed();
            return;
         }
      }
   };

   public HelloJDJMIDlet() {
      display = Display.getDisplay(this);
      text     = new TextBox("Simple MIDlet", "Hello JDJ!", 256, 0);
   }

   public void startApp() {
      text.addCommand(bye);
      text.setCommandListener(commandListener);

      display.setCurrent(text);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }
}
```

Listing 2

```
public void paint(Graphics g) {
   if (!running) {
      return;
   }

   g.setColor(randRange(255), randRange(255), randRange(255));

   triangle[0] = randRange(w);
   triangle[1] = randRange(h);
   triangle[2] = randRange(w);
   triangle[3] = randRange(h);
   triangle[4] = randRange(w);
   triangle[5] = randRange(h);

   drawPoly(triangle,6,g);
}
```

Listing 3

```
   private final int randRange(int n) {
      int r = random.nextInt() % n;
      if (r < 0) {
         r += n;
      }
      return r;
   }


   private final void drawPoly(int coords[], int numcoords,
Graphics g) {

   ...

   }
```

Listing 4

```
public void run() {
   int n = 0;

   stopped = false;

   while (!stopped) {
      now = System.currentTimeMillis();
      if (nextTime < now + 15)
         nextTime = now + 15;
      try {
         runner.sleep(nextTime - now);
      }
      catch(Exception e){}
      nextTime = System.currentTimeMillis() + 50;

      screen.repaint();

   }
}
```

Download the Code!
www.JavaDevelopersJournal.com

SHOP ONLINE AT **JDJSTORE.COM** FOR BEST PRICES OR CALL YOUR ORDER IN AT **1-888-303-JAVA**

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

JDJStore.COM

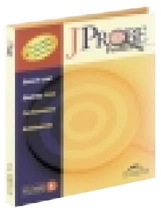
GUARANTEED BEST PRICES FOR ALL YOUR JAVA-RELATED SOFTWARE NEEDS

SYBASE
SQL Anywhere Studio v7.0

Sybase® SQL Anywhere Studio is a comprehensive package that provides data management and enterprise synchronization to enable the rapid development and deployment of distributed e-business solutions. Optimized for workgroups, laptops, handheld devices and intelligent appliances, SQL Anywhere extends the reach of a corporation's e-business information anywhere business transactions occur.

SQL Anywhere Studio (base w/ 1 user) v7.0 . . . . . $\$348^{99}$

SITRAKA
JProbe Profiler Developer v2.8.1



JProbe Profiler helps you quickly eliminate performance bottlenecks caused by inefficient algorithms in your Java code. JProbe Profiler combines a visual call graph interface, a unique Garbage Monitor, and advanced data collection technology to provide you with highly accurate performance diagnostics, including line-by-line results.

JProbe Profiler w/ Standard Support (includes JProbe Memory Debugger) . . . . $\$498^{99}$

VISUALSOFT
JSmartGrid 1.0

JSmartGrid 1.0 is the smallest Swing grid component (162 KB) specifically designed for Java 2 Platform. v1.3.0.JSmartGrid 1.0 is a 100% Java Swing Component, offering developers an extensive API with scalable presentation abilities that can be easily integrated into applications built for the Web, databases, and application servers. It allows developers to build powerful tables, grids, or spreadsheet-like user interfaces, and can be embedded in any eBusiness and B2B application.

JSmartGrid 1.0 . . . . $\$179^{99}$

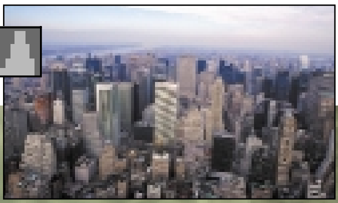
W W W . J D J S T O R E . C O M

web services EDGE conference & expo

WEB SERVICES INTERNATIONAL DEVELOPER CONFERENCE & EXPO

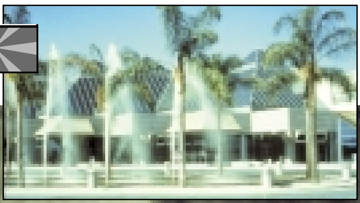
Hilton New York, NYC
Sept 24–25, 2001

EAST COAST



Santa Clara Convention Center, CA
Oct 24–25, 2001

WEST COAST



The First Major Conference & Expo Focusing Exclusively on Web Services

Integrating applications with Web Services

Web Services, in its infancy, promises to be the next major impact on the business model of the new millennium.

The Web Services Edge Conferences in New York and California provide the venue to explore the implications of Web Services and their impact on business process.

The Expo will bring together leading technology vendors and their business partners to showcase the products and services necessary to integrate applications and create e-business applications.

Now is the time for companies to become educated on the opportunities Web Services create for more efficient enterprises.

Web Services Edge is the venue to get you up to speed!

Get Up to Speed with the Fourth Wave in Software Development

Real-World Web Services: Is It Really XML's Killer App?

Demystifying ebXML: A Plain-English Introduction

Authentication, Authorization and Auditing: Securing Web Services

Wireless: Enable Your WAP Projects for Web Services

The Web Services Marketplace: An Overview of Tools, Engines and Servers

Building Wireless Applications with Web Services

How to Develop and Market Your Web Services

Integrating XML in a Web Services Environment

Real-World UDDI

WSDL: Definitions and Network Endpoints

Implementing SOAP-Compliant Apps

Deploying EJBs in a Web Services Environment

Swing-Compliant Web Services

Conference Program and Registration at sys-con.com/webservices

To reserve exhibit space, call 201-802-3004

Steve Benfield, a leading industry expert and frequent speaker at business and technology events, is CTO of SilverStream.

SteveBenfield Conference Tech Chair

SYS-CON EVENTS
SYS-CON.COM

Event Sponsors: Gold Sponsor: PointBase Silver Sponsor: SilverStream Corporate Sponsor: sitraka xml-j

Media Sponsors WebServices Journal, JAVA Developer's Journal, XML Journal, WebSphere Developer's Journal, webtechniques, Business & Technology wireless, ColdFusion Developer's Journal

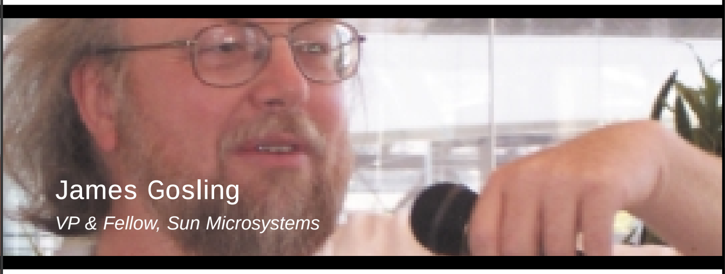
Next Month in *JDJ*...

Interview with James Gosling

At JavaOne this year, *JDJ*'s editor-in-chief, Alan Williamson, and Blair Wyman, got hold of James Gosling, the father of Java, for an informal chat. It was a discussion that led to many areas of Java including how James was impressed that Java is beginning to mature.

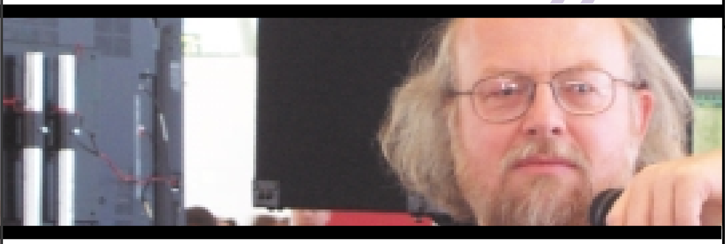
Read James's views on open sourcing Java:

"it's been open sourcing from the beginning"


James Gosling
VP & Fellow, Sun Microsystems

What's James been up to in this weather?

"I've been working on the TIGER compliment for several months now"



Plus...

J2ME
J2SE
J2EE
Home

DOT.COM
• buyer's guide
• java forums
• mailing list
• java jobs
• java store

MAGAZINE
• advertise
• authors
• customer service
• editorial board
• subscribe

CONTENT
• archives
• digital edition
• editorial
• features
• interviews
• product reviews
• source code

CONFERENECES
• jdj edge
new york, ny
sept 23–26

SEARCH JDJ
Search

JDJ SPECIALS
JDJStore.com
$1,210.99
ColdFusion 4.5

BESTSELLERS
1. Visual Cafe Standard Edition v4.0 from WebGain $97.99
1. Intr@Vision Foundation from Dallas Development Systems $5499.00
1. Deetz's Net Tools Pro IX from Laurence Deetz $19.50
1. JRun Server v3.0 Enterprise 1CPU License from Macromedia $4,673.99

What's Online...

July 2001

JDJ Online

Check in everyday for up-to-the-minute news, events, and developments of the Java industry. Can't get to the newsstand on time? Visit www.javadevelopersjournal.com and be the first to know what's happening in the industry. Check out what you've been missing.

JDJEdge Conference and Expo2001, Hilton New York, September 23-26, 2001

Opening day keynotes will be delivered by James Gosling and Alan Baratz.

Make sure you have a seat in New York this fall when ***JDJEdge*** brings you everything you'll need to be ahead of what's to come in the Java world.

Join us at the largest East Coast Java developers Conference and Expo offering cutting edge Java multiple simultaneous tracks, night school, and an accelerated weekend program.

So check out **JavaDevelopersJournal.com** daily for updates and conference news and register now to be a part of the experience.

JavaOne 2001 Radio Interviews

Tune in to SYS-CON Radio and hear the industry's top movers and shakers, interviewed by ***JDJ*** editors. Listen to James Gosling, the father of Java, ***JDJ*** Advisory Panel members, CEOs, product managers, and other top executives as they share their insight and opinions on today's hottest issues.

JavaDevelopersJournal.com Developer Forums

Join our new Java mailing list community. You and other IT professionals, industry gurus, and ***Java Developer's Journal*** writers can engage in Java discussions, ask technical questions, talk to vendors, find Java jobs, and more. Voice your opinions and assessments on topical issues, or hear what others have to say. Monitor the pulse of the Java industry!

JavaBuyersGuide.com

JavaBuyersGuide.com is your best source anywhere, anytime on the Web for Java-related software and products in more than 20 mission-critical categories including application servers, books, codes, IDEs, modeling tools, profilers, and more. Check the Buyer's Guide for the latest and best Java products available today.

Source Code

No need to key in code from the print magazine – here you can download source code and examples from the current issue and back issues, anytime.

Click for a FREE 30-day trial AltoWeb
INQMY A subsidiary of SAP AG
M-1 Mobile App Server aligo Download Now!
JClass
Get SonicMQ™ PROGRESS SOFTWARE
KADA SYSTEMS
MERANT DataDirect
TopCoder Compete for Cash www.topcoder.com

JDJ NEWS

Everlasting Systems Introduces SGIL Software Suite for Software Internationalization

(*San Francisco, CA*) – Everlasting Systems, Limited announced at the 2001 JavaOne Conference the debut of the Software Globalization by Internationalization for Localization (SGIL) Software Suite in the North American region.

The SGIL Software Suite features a software globalization framework with a set of components, APIs, and utilities that assess the internationalization status of software applications, then enable and optimize software globalization by internationalization for localization.
www.everlastingsystems.com

CocoBase Enterprise O/R Optimized for VisualAge for Java

(*San Francisco, CA*) – THOUGHT, Inc.'s CocoBase Enterprise O/R is now optimized for IBM's VisualAge for Java IDE. Customers using CocoBase with VA now have the ability to gain substantial time and labor savings through generating scalable and high-performance, enterprise-ready container and bean-managed persistence (CMP/BMP) in addition to JavaServer Pages, with its dynamic mapping capability.
www.thoughtinc.com

preEmptive solutions Releases DashO Embedded Edition

(*Cleveland, OH*) – preEmptive solutions, Inc., has announced the availability of the embedded edition of DashO, their tool for post-compilation packaging of Java byte code.
DashO-EE includes obfuscation, optimizing, and size-reduction capabilities.
www.preemptive.com

QA-Systems Unveils QStudio Java 1.4:

(*The Netherlands*) – QStudio Java, the code inspection and quality analysis solution for Java development, can now be seamlessly integrated with JBuilder 5. QStudio Java 1.4 comes in three versions: Lite, Pro, and Enterprise. A trial version can be downloaded from
www.qa-systems.com

HiT Allora Extends Sun's Forte for Java, JBuilder and Visual Basic

(*San Jose, CA*) – HiT Software's Allora now includes code wizards for Sun Microsystems' Forte for Java, release 2.0, Borland's JBuilder Version 4.0, and Microsoft's Visual Basic Version 6.0. Allora wizards work in conjunction with its runtime XML-RDB integration engine, and support all major relational databases (RDBs) including Oracle, IBM DB2, and Microsoft SQL Server.

Allora code wizards integrate into the menus of the IDE application for easy access. A sequence of GUI dialogs guides the developer to provide necessary and optional parameters.
www.hitsw.com

Rational and Sun Announce First End-to-End LifeCycle Development Solution

(*Lexington, MA and Palo Alto, CA*) – Rational Software has integrated its suite of application development tools with Sun's Java and J2EE technologies, as well as the Forte for Java IDE, creating a comprehensive, cross-platform set of e-development tools for developers writing to the Solaris operating environment.

With this environment, developers can build on the platform they're using to deploy their software, thereby avoiding the cost of additional testing and transition time.
www.rational.com
http://sun.com

Zero G Ships New Versions of PowerUpdate

(*San Francisco, CA*) – Zero G Software has announced the immediate availability of PowerUpdate Server, PowerUpdate Pro, and PowerUpdate Now! suite of software products and services that deliver a complete software updating solution.

PowerUpdate runs on AIX, HP-UX, Linux, Solaris, and Windows (95, 98, Me, NT, and 2000), and is fully compatible with most commercial installation and deployment solutions.
www.ZeroG.com

Empirix Releases e-TEST Suite 5.1

(*Waltham, MA*) – Empirix Inc. (formerly RSW Software and Hammer Technologies) has introduced e-TEST suite 5.1, with added functionality that accelerates Web application test and monitoring.

Other new features include enhanced Web reporting, extended support for testing Java applets deployed with the Java Virtual Machines (JVM) from Sun Microsystems and Microsoft, and enhanced support for WAP applications, including support for the Nokia WAP simulator.
www.empirix.com

PointBase Unveils PointBase Micro

(*Mountain View, CA*) – PointBase announced an industry first with PointBase Micro, a new fast, ultra-small SQL relational database with a footprint of less than 45KB. Written entirely in Java, the database is optimized for Sun's J2ME and J2SE, and uses a subset of JDBC technology as the API, making it ideal for next-generation Java technology–based database applications for occasionally connected and mobile devices, such as smart phones and PDAs.
www.pointbase.com

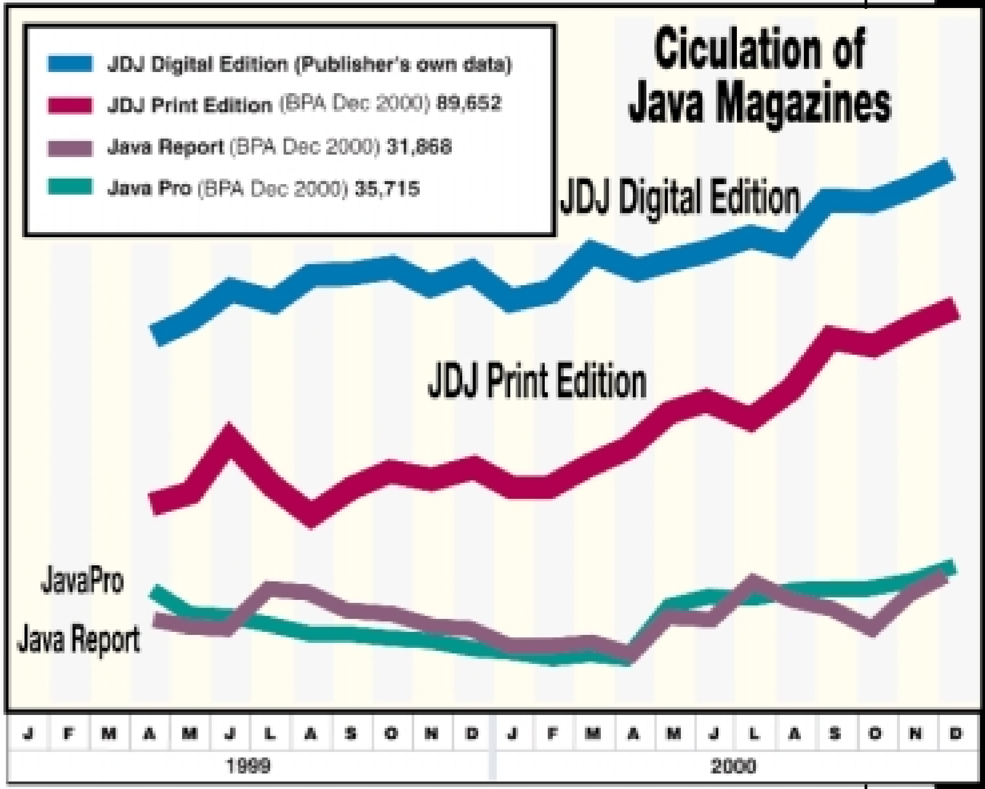
Computer Associates Delivers Comprehensive Solutions for J2EE

(*San Francisco, CA*) – Computer Associates (CA) International, Inc. announced delivery of a comprehensive portfolio of e-business management solutions to build, deploy, integrate, manage, and visualize J2EE technology–based applications.

CA also announced it has become a J2EE licensee, extending its relationship with Sun and furthering its commitment to provide management solutions for Sun's Java and J2EE technologies as a development environment for e-business.
http://ca.com

It's a no-brainer

Why would you advertise anywhere else?



www.JavaDevelopersJournal.com
(201) 802-3020

SYS-CON MEDIA

carmen@sys-con.com miles@sys-con.com

Your ad in ***JDJ*** reaches twice as many recipients as all other Java pubs combined for 1/2 the cost...*

*independent survey footnoted

The Big Interview

Selling yourself in a squeezed market to a prospective employer takes some know-how

WRITTEN BY
BILL BALOGLU &
BILLY PALMIERI

In our May column we examined the résumé as that critical first step in the hiring process. The most effective résumés are filled with specific, relevant details about your skills and experience, what you've done, and when and where you've done it.

It's common for your first interview to be a preliminary phone screening or a conversation between you and a third-party recruiter who's providing candidates for the position, a recruiter or recruiting manager within the company that's hiring, or the hiring manager.

Be prepared to answer questions about your technical expertise and work experience, as well as more personal, career-focused questions.

Phone interviews for a contract position may be more focused on your hardcore technical skills. They're typically hiring you to come in and get the job done on time and right the first time around.

For a full-time position, the company is deciding whether or not it should make a long-term investment in you. Expect these interviews to also include questions about your long-term work history, what motivates you, and your long-term goals.

Be prepared to explain why your last job ended – or, if you're currently employed, why you want to leave. Be honest and direct without offering too much information. Responses to these questions reveal a lot about a candidate.

With recent market changes and corporate IT budgets freezing up, contract consulting opportunities are becoming more and more scarce. Many long-term contractors are now looking for full-time jobs. If you're one of them, be prepared to address your reasons for wanting to make that transition – and why you won't be tempted to leave at the drop of a new contract position.

Other topics should be your availability, your hourly rate or salary, and any other opportunities you may have at the interview stage as well as any offers still pending.

If the interviewer fails to bring up any of these questions, be sure to offer the information. Too often candidates, recruiters, and hiring managers waste a lot of time and energy going through the interview process only to find that there's no way you could start when they need you to, you're way off on the issue of salary requirements, or some other deal-killing issue that could've been prevented or cleared up on the first phone call.

Do Your Homework

The single most important advantage that turns job applicants into "new hires" is their level of preparation before the interview. In the Internet age, there's no excuse for not familiarizing yourself with the company you're applying to or the position you're applying for.

Target Your Résumé

Candidates whose job search consists of a mass résumé spam are easy for hiring managers to spot and are the first ones to be eliminated from consideration.

Preparing for the Face-to-Face Interview

When you're invited in for an interview, ask how many people you'll be meeting with and get their titles, and names (and how to pronounce them), and find out how long you should expect to be there. If you'll be meeting with an executive, see if his or her bio is posted on the company Web site.

Face to Face

Your goal is to convince the interviewer(s) that you're the best person for the job. Make a clear case for your technical skills and experience, but also convey that you are confident, capable, and easy to work with.

Many highly skilled engineers expect to be hired on the basis of their skills alone, but lose out on opportunities because they don't make a good personal impression. In addition to finding the technical skills required for the job, hiring managers are choosing who they want to come to work with every day.

If you're unsure if you answered the question to their satisfaction, ask – and be prepared to clarify or elaborate if necessary.

There is currently a movement away from experience-based interviews (asking people to repeat what's on their résumé) to behavioral interviews. In a behavioral interview, you're likely to be asked about problems you faced in the past and how you solved them.

You may be given a hypothetical situation and asked how you would handle it. While employers agree that behavioral interviews are more effective in revealing how the candidate thinks and solves problems, they do require a strong ability to communicate.

If English is not your first language, expressing abstract ideas in a behavioral interview may be a challenge. You might want to refer the interviewer to references who could speak to your past performance.

Be prepared but flexible: everything you say should be relevant to the job you're interviewing for. However, during the course of the interview, the manager may think of another position that would fit you even better. Be open to discussing other options as well and don't fall apart if the plan changes during the interview.

Interview Don'ts

- Don't tell the manager how they need to do their project or their job differently.
- Don't pitch a new business plan for the entire company.
- Don't oversell yourself.
- Don't burn a bridge. If it becomes clear in the interview that you're not a good match for the job, stay pleasant and professional so they'll remember you later on when the ideal position for you does materialize.
- Don't offer personal information. By law interviewers can't consider it, so don't offer it.

Before concluding your interview, ask what the next step will be and when they expect to make a decision. Sending a follow-up note will further boost your chances.

billb@objectfocus.com

billp@objectfocus.com

AUTHOR BIOS

Bill Baloglu is a principal at ObjectFocus (www. ObjectForcus.com), a Java staffing firm in Silicon Valley. He was a software engineer for 16 years prior to his position at ObjectFocus. Bill has extensive 00 experience, and has held software development and senior technical magagement positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal of ObjectFocus. Before that he was at Renaissance Worldwide, a multimillion-dollar global IT consulting firm where he held several senior management positions in the firm's Silicon Valley operations.

Cubist Threads

Perception is reality

WRITTEN BY
BLAIR WYMAN

I recently returned from an iSeries and AS/400 conference in the amazing city of New Orleans. Thankfully, my job doesn't call for a lot of traveling, but twice a year my management unleashes me on whatever unsuspecting city happens to be next on the list of locations for this conference.

So far my impact on the various conference environs has been relatively minor, mostly limited to temporarily depleting the local supply of crispy bacon and well-done wheat toast. My incurable vanity, though, enjoys the notion that I'm making some impact as I proselytize for the Java programming language with the iSeries customers I meet at these events.

Making the case for Java on the iSeries is somewhat of an uphill battle, I must admit. Server-side Java is a great fit there, as the benchmarks show, but making the case, to long-time RPG programmers, for switching to Java is a tough row to hoe. What, in an already productive programming environment, can motivate as fundamental a change as that of your primary programming language? Other than by "management edict," I'm not sure how many from our wonderful and august body of iSeries and AS/400 developers will be swayed to Java by force of argument alone.

The conference was, by any measure, a wonderful success. More than 3,000 folks attended the five-day conference, hearing about all the goodies in the new release of the iSeries system. We just started shipping our largest release ever, V5R1, with more than 4 million lines of new code. Operating system development is definitely a horse of a different color, compared with applications development. I remember the fun I had writing a Mandelbrot set engine that exploited my then new 8087 math coprocessor. While significant, it was maybe a thousand lines of code, at most. Those were definitely simpler days.

On the JVM front, there isn't that much that was new in this release: some enhancements to the garbage collector (parallelizing the sweep phase) and some improvements in our JIT (basically a port of the JIT we get from the IBM Tokyo Research Lab) are probably the biggest hitters in the JVM support. Many of these improvements will be provided for earlier releases after the fact – lots of our customers stay a release or two behind for a while, as the new releases shake out. So far, V5R1 has "solidified" nicely, though the customer experience is the ultimate proving ground.

On the surface, a description of this conference probably sounds a lot like any number of technical conferences. The attendees went to education sessions and labs, polished up their programming techniques, heard the latest news about enhancements and new functions, accumulated vast hoards of "trash and trinkets" at the exposition, and rubbed elbows with their peers at receptions, dinners out, and nice catered get-togethers. What's unique about this particular computer conference, called COMMON, is its remarkable longevity.

When this conference of IBM midrange computer users was first held, I was barely beginning to do some work in the area of debugging. Of course, since the year was 1960, the debugging I was doing involved riding my steel-wheeled skateboard over migrating swarms of red ants on the neighbor's sidewalk. However, even then, I was toying with optimization techniques for the mass eradication of tiny insect life: flat-bottomed "thongs" (as we called the traditional open-toed South Dakota summer footwear, complete with irritating strap between toes one and two) turned out to provide the maximum Kills Per Thwap (KPT) factor. I guess that while I may have been destined to be a programmer, I'd never have made a very good Buddhist.

What's that? Yup, I said, "1960." Phones had dials. TVs had dials. Radios had dials. (Aren't you glad you used dials?) I was an utterly oblivious 3-year old when this conference first convened; it has been going strong ever since. (Hopefully, my oblivion has been tempered in the meantime, though the jury is still out.)

What, Dear Reader, were you doing 40 years ago? In many cases, I suspect the answer is a cause for mystical speculation, since you were... um... not. For those of you old enough to remember 1960, what do you remember about the computers of the day? How many of those computers are still being produced exactly as they were then? (Outside the air traffic–control arena, my guess is that the answer is a big, fat zero.)

So what is it about a conference that can give it such longevity? What, among the evanescent loyalties in our mercurial industry, can survive 40 years of evolution and revolution? Perhaps I'll leave my speculation on the One True Answer for another time – at least I got to use the word "evanescent."

blair@blairwyman.com

AUTHOR BIO
Blair Wyman is a software engineer working for IBM in Rochester, Minnesota, home of the IBM iSeries.

Java Dudes