

No. 1 *i*-Technology Magazine in the World

# JDJ

AUGUST 2004 VOLUME:9 ISSUE:8

Feature:  
*Using Apache Cactus*  
 Testing server-side components  
 page 28

# JAVA DESIGN PATTERNS

## + FAST PERFORMANCE

### PLUS...

- ▶ **Walking the Tightrope of Microsoft-Java Interoperability**
- ▶ **The Blind Men, the Elephant, and App Server Migration**
- ▶ **Dynamic Sorting with Java**
- ▶ **Unlocking Microsoft Office Documents**
- ▶ **Embedding the Java Virtual Machine, Once and for All**
- ▶ **Web Conferencing Using the Java Media Framework**

RETAILERS PLEASE DISPLAY UNTIL OCTOBER 31, 2004

\$9.99US \$9.99CAN



**Oracle Platform**

---

# **Database 10g**

# **Application Server 10g**

**Common LDAP directory**

**Unified security model**

**Common administration**

**Automated space management**

**Engineered to work together**

**ORACLE®**

**[oracle.com/platform](http://oracle.com/platform)  
or call 1.800.633.0753**

# Putting the 'i' Back in i-Technology



Jeremy Geelan



**Editorial Board**

Desktop Java Editor: **Joe Winchester**  
 Core and Internals Editor: **Calvin Austin**  
 Contributing Editor: **Ajit Sagar**  
 Contributing Editor: **Yakov Fain**  
 Contributing Editor: **Bill Roth**  
 Contributing Editor: **Bill Dudney**  
 Contributing Editor: **Michael Yuan**  
 Founding Editor: **Sean Rhody**

**Production**

Production Consultant: **Jim Morgan**  
 Associate Art Director: **Tami Beatty-Lima**  
 Executive Editor: **Nancy Valentine**  
 Associate Editors: **Jamie Matusow**  
                           **Gail Schultz**  
                           **Jennifer Van Winckel**  
 Assistant Editor: **Torrey Gaver**  
 Online Editor: **Lin Goetz**  
 Research Editor: **Bahadir Karuv, PhD**

**Writers in This Issue**

Ryan Ackley, Calvin Austin, York Davis, Jeremy Geelan, Ted Goddard, Gunnar Grim, Rob Halleron, Michael Havey, Pramod Jain, Yayati Kasralikar, Kishore Kumar, Heman Robinson, Ajit Sagar, Avik Sengupta, Derek Spratt, Stanley Wang, Joe Winchester

To submit a proposal for an article, go to <http://grids.sys-con.com/proposal>

**Subscriptions**

For subscriptions and requests for bulk orders, please send your letters to Subscription Department [subscribe@sys-con.com](mailto:subscribe@sys-con.com). Cover Price: \$5.99/issue. Domestic: \$69.99/yr. (12 Issues) Canada/Mexico: \$99.99/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: 201 782-9638

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.

**©Copyright**

Copyright © 2004 by SYS-CON Publications, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without written permission. For promotional reprints, contact reprint coordinator Kristin Kuhnle, [kristin@sys-con.com](mailto:kristin@sys-con.com). SYS-CON Media and SYS-CON Publications, Inc., reserve the right to revise, republish and authorize its readers to use the articles submitted for publication.

Worldwide Newsstand Distribution  
 Curtis Circulation Company, New Milford, NJ  
 For List Rental Information:

Kevin Collopy: 845 731-2684, [kevin.collopy@editthroman.com](mailto:kevin.collopy@editthroman.com)  
 Frank Cipolla: 845 731-3832, [frank.cipolla@epostdirect.com](mailto:frank.cipolla@epostdirect.com)

Newsstand Distribution Consultant  
 Brian J. Gregory/Gregory Associates/W.R.D.S.  
 732 607-9941, [BJGAssociates@cs.com](mailto:BJGAssociates@cs.com)

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. SYS-CON Publications, Inc., is independent of Sun Microsystems, Inc. All brand and product names used on these pages are trade names, service marks or trademarks of their respective companies.

Ever since Nicholas G. Carr's now historic *Harvard Business Review* article, "IT Doesn't Matter," published in the May 2003 edition of *HBR*, it was only a matter of time before the wider world caught up with Carr's thesis. The article formed only a small part of Carr's broader exploration of the influence of information technology on business strategy contained in his book *Does IT Matter? Information Technology and the Corrosion of Competitive Advantage*, but it is the "IT Doesn't Matter" chapter that sticks in everybody's mind.

In it, Carr argued that while IT infrastructure is essential to competitiveness, particularly at the regional and industry level, it's no longer a source of advantage at the company level. In other words, it doesn't enable individual companies to distinguish themselves in a meaningful way from their competitors. While essential to competitiveness, argued Carr, IT has become inconsequential to strategic advantage. IT is best viewed (and managed) nowadays, he concluded, as a commodity.

Of course he never truly meant that IT "didn't matter"; he merely wanted the business community to understand that it could no longer rely on it as a source of competitive advantage.

Now that the JavaOne techfest has come and gone, but with the Linux-World Expo still to come, everyone and his dog is naturally busy commenting, interpreting, opining, and dissecting... so at *JDJ* we thought it might be useful to do a round-up of some of what is being said about the state of technology, the Internet, e-commerce, and all things related. We will publish them in next month's issue. Already we can reveal that one thing emerges above all else: technology is back – most especially Internet technologies such as search, storage, and security. It is these three items together that are putting the "i" back into i-technology in a big way.

How else can you explain why the (possible) market value of Google,

Inc., seems likely to be as high as \$36 billion, rivaling corporate IT stalwarts such as McDonald's Corp. and Sony Corp? Search is still very much the new frontier so far as the Internet is concerned. How else can you account for the growth of storage giants like EMC and security giants like Cisco, which as long ago as 2000 was being labeled "the quiet security giant" as the undisputed king of switches and routers began rapidly to make a name for itself in the security arena?

Without the Internet and the technologies related to it there would be no U.S. stock market uptick in process. So Bill Gates didn't really need to launch the broadside he did in his speech at Microsoft's CEO Summit on May 21 last year:

*And so when somebody says, to take the extreme quote from the Harvard Business Review article, they say IT doesn't matter, they must be saying that with all this information flow, we've either achieved a limit where it's just perfect, everybody sees exactly what they want, or we've gotten to a point where it simply can't be improved – and that's where we'd object very strenuously.*

Because that wasn't ever Carr's point. What his *HBR* article was arguing was that we're at the stage in the business/technology cycle where any technological improvement in the management of information will be quickly and broadly copied, rendering it meaningless for competitive advantage. Not that information technology, the Internet, and all kindred phenomena no longer matter. Far less that improvement is no longer possible. Software developers everywhere – and CIOs, CTOs, and CSOs too – certainly think it matters, perhaps more than ever. Just as Gates does, and Messrs. McNealy, Ellison, Dell, and Palmisano. That is why the "i" is so firmly back in i-technology. And why, in turn, the "P" is back in IPO. ☺

**Jeremy Geelan** is group publisher of SYS-CON Media, and is responsible for the development of new titles and technology portals for the firm. He regularly represents SYS-CON at conferences and trade shows, speaking to technology audiences both in North America and overseas.

[jeremy@sys-con.com](mailto:jeremy@sys-con.com)



**Relational  
database**

**Object-oriented  
development**



# GET THE RIGHT BACK-END FOR YOUR FRONT-END

If your back-end database isn't a good match for your front-end development, you need a new database.

Caché, the *post-relational* database from InterSystems, combines high-performance SQL for faster queries and an advanced object database for rapidly storing and accessing objects. With Caché, no mapping is required between object and relational views of data. Every Caché class can be automatically projected as Java classes or EJB components with bean-managed persistence. Plus, every object class is instantly accessible as tables via ODBC and JDBC.

That means huge savings in both development and processing time. Applications built on Caché

are massively scalable and lightning fast. They require little or no database administration. And Caché's powerful Web application development environment dramatically reduces the time to build and modify applications.

We are InterSystems, a specialist in data management technology for over twenty-six years. We provide 24x7 support to four million users in 88 countries. Caché powers enterprise applications in healthcare, financial services, government, and many other sectors. Caché is available for Windows, OpenVMS, Linux, and major UNIX platforms – and it is deployed on systems ranging from two to over 10,000 simultaneous users.



**Try a better database. For free.**

Download a free, fully-functional, non-expiring version of Caché or request it on CD at [www.InterSystems.com/match6](http://www.InterSystems.com/match6)

# JDJ contents

**JDJ Cover Story**

# Java Design Patterns for Long Lists

*Providing fast performance*

by Heman Robinson

**44**

**Features**



**10**

**Embedding the Java Virtual Machine, Once and for All**

by Stanley Wang

FROM THE GROUP PUBLISHER

**Putting the ‘?’ Back into i-Technology**  
by Jeremy Geelan ..... **3**

VIEWPOINT  
**Walking the Tightrope of Microsoft-Java Interoperability**  
by Derek Spratt ..... **6**

JAVA ENTERPRISE VIEWPOINT  
**The Blind Men, the Elephant, and App Server Migration**  
by Ajit Sagar ..... **8**

VIDEO/AUDIO  
**Web Conferencing Using the Java Media Framework**  
*Broadcast and receive media streams*  
by Yayati Kasralikar and Pramod Jain ..... **18**

CORE AND INTERNALS VIEWPOINT

**A Tail of Two Tigers**  
by Calvin Austin ..... **26**

INTERFACES  
**Dynamic Sorting with Java**  
*A reusable implementation*  
by York Davis ..... **32**

JNI  
**Calling Java from C**  
*A framework for easier JNI*  
by Michael Havey ..... **36**

DESKTOP JAVA VIEWPOINT  
**Swing Low, Swing High, Sweet Desktop**  
by Joe Winchester ..... **42**

DOI  
**A GUI Painter Friendly Table Component**  
*The principle of the column container*  
by Gunnar Grim ..... **48**

POI  
**Unlocking Microsoft Office Documents**  
*An open source alternative*  
by Ryan Ackley and Avik Sengupta ..... **52**

LABS  
**VERITAS i³ for J2EE**  
Reviewed by Rob Halleron ..... **60**

PRESSROOM  
**Industry News**  
*JDJ News Desk* ..... **64**

@ THE BACKPAGE  
**Unified Diversity**  
by Ted Goddard ..... **66**



**28**

**Using Apache Cactus**

by Kishore Kumar



Derek Spratt



## Walking the Tightrope of Microsoft-Java Interoperability

If you were one of the 14,000 developers walking the aisles of the JavaOne Worldwide Developers Conference in San Francisco in June, you likely picked up the buzz that surrounded the issues and opportunities presented by the recent agreement between Microsoft and Sun, and the settling of their outstanding litigation while pledging to work together to provide better support and platform interfaces for each other's technologies.

Customers likely had their say in the matter. It all boils down to the fact that Java is here to stay at the enterprise – “rip up and replace” hasn't gone over very well since the tech sector meltdown. Yet .NET is making steady inroads into the very same organizations that develop, deploy, and run Java applications – client and server apps alike.

There are a whole host of good reasons for mixing and matching platforms.

A simple example is the development of a front-end GUI using Visual Studio/.NET to create an app with the familiar look and feel of Microsoft Windows/Office that links to a Java enterprise app in the back office, and which can be developed quickly with minimal technical effort. As Microsoft's server-based technologies and applications grow exponentially, more and more Java-based client-side applications will have to work seamlessly with .NET in the back office as well. Heterogeneous computing environments are here to stay.

This isn't as hard to accomplish as you might initially perceive it to be. The goal of any application architect is to have the flexibility to build applications in way that optimizes performance and cost – the platform it's built on should be a secondary factor because high-performance third-party interoperability solutions are available and well accepted

by the major vendors in the market today. By late 2006 or early 2007, Microsoft is scheduled to release “Indigo”, a Web services interoperability framework that should further break down the barriers between Microsoft technologies and Java.

While SOAP/XML-based Web services fit neatly into the “loosely connected” SOA systems arena, they aren't typically high performance, which is often of concern for enterprises today. Indigo will rely on Web services, so how much of the problem will really be solved? There's a difference between basic interoperability and high-performance interoperability. Therefore, developers need to choose the right balance between cost, performance, and future-proofing their apps.

Some of the Microsoft-Java interoperability options available on the market today will not work with Indigo due to their reliance on a piece of the *System.Runtime.Remoting* namespace called channels and formatters. The issue is that the entire channels/formatters subsystem will be removed from Indigo. Therefore any software written using those features will not run under Indigo. If future support for Indigo is important for your customers, plan accordingly.

The software industry needs choice and continuous innovation. The foundations for future advancements in computing are based on these concepts. For this reason the Sun-Microsoft settlement is perhaps one of the best pieces of industry news to date in 2004. Nevertheless, the resulting heterogeneous computing environments do present their challenges to systems architects and developers. There is a growing list of interoperability options available to them that can and should be explored and leveraged. ☛

**Derek Spratt** is the founder of Intrinsyc and currently serves as its president and CEO. Mr. Spratt was also a cofounder and CEO of Consequent Technologies, a cofounder and EVP of PCS Wireless, Inc., the VP and business unit manager of Nexus Engineering, and the product development manager in Motorola's Wireless Data Division. He also takes a keen interest in supporting the nonprofit sector and has provided financial and advisory support to the BCT Social Venture Partners, BC ScienceWorld, and the Sierra Legal Defense Fund.

[dspratt@intrinsyc.com](mailto:dspratt@intrinsyc.com)

### President and CEO:

**Fuat Kircaali** [fuat@sys-con.com](mailto:fuat@sys-con.com)

Vice President, Business Development:

**Grisha Davida** [grisha@sys-con.com](mailto:grisha@sys-con.com)

Group Publisher:

**Jeremy Geelan** [jeremy@sys-con.com](mailto:jeremy@sys-con.com)

### Advertising

Senior Vice President, Sales and Marketing:

**Carmen Gonzalez** [carmen@sys-con.com](mailto:carmen@sys-con.com)

Vice President, Sales and Marketing:

**Miles Silverman** [miles@sys-con.com](mailto:miles@sys-con.com)

Advertising Sales Director:

**Robyn Forma** [robyn@sys-con.com](mailto:robyn@sys-con.com)

Director, Sales and Marketing:

**Megan Mussa** [megan@sys-con.com](mailto:megan@sys-con.com)

Associate Sales Managers:

**Kristin Kuhnle** [kristin@sys-con.com](mailto:kristin@sys-con.com)

**Beth Jones** [beth@sys-con.com](mailto:beth@sys-con.com)

**Dorothy Gil** [dorothy@sys-con.com](mailto:dorothy@sys-con.com)

### Editorial

Executive Editor:

**Nancy Valentine** [nancy@sys-con.com](mailto:nancy@sys-con.com)

Associate Editors:

**Jamie Matusow** [jamie@sys-con.com](mailto:jamie@sys-con.com)

**Gail Schultz** [gail@sys-con.com](mailto:gail@sys-con.com)

**Jennifer Van Winckel** [jennifer@sys-con.com](mailto:jennifer@sys-con.com)

Assistant Editor:

**Torrey Gaver** [torrey@sys-con.com](mailto:torrey@sys-con.com)

Online Editor:

**Lin Goetz** [lin@sys-con.com](mailto:lin@sys-con.com)

### Production

Production Consultant:

**Jim Morgan** [jim@sys-con.com](mailto:jim@sys-con.com)

Lead Designer:

**Tami Beatty-Lima** [tami@sys-con.com](mailto:tami@sys-con.com)

Art Director:

**Alex Botero** [alex@sys-con.com](mailto:alex@sys-con.com)

Associate Art Directors:

**Louis F. Cuffari** [louis@sys-con.com](mailto:louis@sys-con.com)

**Richard Silverberg** [richards@sys-con.com](mailto:richards@sys-con.com)

Assistant Art Director:

**Andrea Boden** [andrea@sys-con.com](mailto:andrea@sys-con.com)

### Web Services

Vice President, Information Systems:

**Robert Diamond** [robert@sys-con.com](mailto:robert@sys-con.com)

Web Designers:

**Stephen Kilmurray** [stephen@sys-con.com](mailto:stephen@sys-con.com)

**Matthew Pollotta** [matthew@sys-con.com](mailto:matthew@sys-con.com)

### Accounting

Financial Analyst:

**Joan LaRose** [joan@sys-con.com](mailto:joan@sys-con.com)

Accounts Payable:

**Betty White** [betty@sys-con.com](mailto:betty@sys-con.com)

Account Receivable:

**Shannon Rymza** [shannon@sys-con.com](mailto:shannon@sys-con.com)

### SYS-CON Events

President, SYS-CON Events:

**Grisha Davida** [grisha@sys-con.com](mailto:grisha@sys-con.com)

Conference Manager:

**Lin Goetz** [lin@sys-con.com](mailto:lin@sys-con.com)

### Customer Relations

Circulation Service Coordinators:

**Edna Earle Russell** [edna@sys-con.com](mailto:edna@sys-con.com)

**Linda Lipton** [linda@sys-con.com](mailto:linda@sys-con.com)

JD| Store Manager:

**Brunilda Staropoli** [bruni@sys-con.com](mailto:bruni@sys-con.com)



The right Java, whatever the gig.

**Borland® JBuilder.®** The #1 Java™ tool in the world for a reason. Pick the size that fits your needs. Automate the routine stuff. Handcraft the unique. Have an active voice at every stage in the process. Move faster, and make every project a hit. Whether your application is headed to the Web, enterprise, or mobile, just pick the feature set you need. And start rockin'!

Customizable code editor • Refactoring • Local and remote debugging • Integrated unit testing • Two-way visual Struts designer • JSP™ tag library/framework support • XML and Web Services • Mobile application development • Advanced build and configuration management with Apache® Ant • Visual EJB™ designer • Two-way deployment descriptor editor • Archive builder • Integration with all major J2EE™ application servers

[go.borland.com/j6](http://go.borland.com/j6)

Made in Borland® Copyright © 2004 Borland Software Corporation. All rights reserved. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. • 21715.9

**Borland®**  
Excellence Endures™

Learn from the experts at the premier event for technical education – the 2004 Borland Conference – <http://connect.borland.com/borcon04s>

# The Blind Men, the Elephant, and App Server Migration



**Ajit Sagar**  
Contributing Editor

**T**he six blind men\* who attempted to describe the elephant eventually described it only from their perspectives – the parts and not the whole. The same malady can be found lurking in one of the problems that faces many organizations that have adopted J2EE as their platform of choice: the migration of these applications between J2EE application servers – be it vendors or versions. The number of migration initiatives that have come up in the past few years is substantial. There are several reasons for this:

- Java, as ever, is rapidly evolving.
- Although the splitting of Java into three platforms (J2EE/J2ME/J2SE) happened a few years ago, it took a while for the app servers to catch up and provide the necessary support.
- The number of mainstream app server vendors has died down from a few tens to single digit numbers within the short span of a couple of years.
- Since the platform on which the core product is written has moved on, there is no choice but to move. Often the support for an existing version is cut off.
- The drivers for migration are not merely limited to app server vendors and software. Many companies are recognizing the need to shift to open source and Linux platforms as a more viable alternative. So migration can involve one or many of several dimensions – versions, vendors, operating systems, hardware, related third-party vendors, etc.

For most organizations already using a J2EE application server, the upgrade to another version is not difficult, if planned properly. However, the complete migration of several enterprise applications is not trivial. Therefore, it's critical that adequate planning be done in advance so that the external factors (besides code migration) have minimal impact on the actual upgrade. The strategy and

planning for such initiatives is very complex. The complexity is multiplied due to the number and profiles of stakeholders in the equation. People tend to view migration from a narrow perspective due to the limited visibility each individual has into the entire process. There are several stakeholders involved in such migration initiatives, including:

- Developers who think of migration in terms of the application code changes
- Administrators who think of migration in terms of production runtime
- Product architects who think of migration in terms of the impact on design and product features as well as the product roadmap
- Development managers who think of migration in terms of the resources available, existing deadlines, etc.
- Technical support and services who think of migration in terms of the infrastructure and capacity planning
- Executive management who think of migration in terms the cost, the risk, and the impact on the LOB (Lines of Business)

A typical migration requirement that is prevalent in the industry today is migration from IBM WAS 3.5 to WAS 5.0. This is not unexpected. IBM has finally caught up with the latest version of the J2EE platform, but they took their time doing it. IBM's support for EJB 2.0 came nearly a year after competing vendors, such as BEA, had provided the same. From a technology viewpoint, a migration from 3.5 to 5.x involves code migration, application redesign, total repackaging for deployment, and migration of the entire development environment from VAJ to WSAD – just to name a few key factors. Add the integration with MQ at IBM clients and throw mainframes into the mix, and the prospect of migrating 20–50 enterprise applications becomes very formidable.

The best way for companies to tackle this type of a tech initiative is

to include a planning phase during which several aspects of migration are addressed, some of which are:

- Dependencies between applications in order to bundle and sequence the applications to minimize disruptions
- Training, especially if the development team is shifting IDEs
- Shared code libraries, which feed into the bundling
- Third-party APIs that may have incompatibilities with the new version of the app server/Java platform
- Integration with in-house utilities

IBM provides a Redbook that serves as a “how-to” guide for such a migration (<http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246910.html>). However, the other aspects of migration, such as the ones mentioned earlier, cannot be covered in a generic migration guidebook. Someone has to define application characteristics, dependencies, etc., and define a viable strategy for the migration of each application, as well the migration of all the applications in a fixed time frame. Then a team needs to manage the migration to ensure it's done in the proposed manner. The dollars spent up front in such an effort are a fraction of the amount of money that will go down the drain if these parameters are not accounted for.

To do this in a planned fashion, the best recourse is to engage a team that works solely on this planning initiative, across the applications in the scope of the migration. Such a team needs to operate outside all the applications and deliver an analysis that addresses the needs of each application. This is your seventh (seeing) “man” who can paint the true picture of the elephant. ☺

\*THE AMERICAN POET JOHN GODFREY SAXE BASED HIS POEM, “THE BLIND MEN AND THE ELEPHANT” ([WWW.WORDFOCUS.COM/WORD-ACT-BLINDMEN.HTML](http://WWW.WORDFOCUS.COM/WORD-ACT-BLINDMEN.HTML)) ON A FABLE THAT WAS TOLD IN INDIA MANY YEARS AGO.

**Ajit Sagar** is a senior technical architect with Infosys Technologies, Ltd., a global consulting and IT services company. He has been working with Java since 1997, and has more than 15 years' experience in the IT industry. During this tenure, he has been a programmer, lead architect, director of engineering, and product manager for companies from 15 to 25,000 people in size. Ajit has served as *JDJ's* J2EE editor, was the founding editor of *XML-Journal*, and has been a frequent speaker at SYS-CON's Web Services Edge series of conferences. He has published more than 75 articles. [ajitsagar@sys-con.com](mailto:ajitsagar@sys-con.com)





**New in RELEASE 4:**  
**Greatly enhanced Text View**  
**XML-aware file merging**  
**Additional database support for:**

- IBM® DB2®
- Sybase®
- MySQL®



Look into **xmlspy® 2004**, the industry leading XML development tool from Altova. Discover everything new in **xmlspy** – the standard for modeling, editing, transforming and debugging all XML technologies. With advanced functionality like graphical WSDL editing, SOAP debugging, Java code generation, and support for all major databases, Web Services are now a reality and interoperability issues are a thing of the past. See for yourself how **xmlspy 2004** builds developer productivity.

**Download xmlspy® 2004 today: [www.altova.com](http://www.altova.com)**

**ALTOVA®**

[www.altova.com](http://www.altova.com)

All other trademarks are the property of their respective owners.

# Embedding the Java Virtual Machine, Once and for All

by Stanley Wang

*TTL is a portable C++ JNI template library that allows you to embed a JVM within another program. An example of this might be found inside a browser that needs to support Java plug-ins. First I'll discuss the traditional way to embed a JVM within a native program, and then talk about the generic solution offered by JTL.*

To apply the solution provided by JTL, you need to know the basics of Java, C++, and JNI. However, to understand JTL's design you should be familiar with modern C++ techniques, such as template programming and the Boost library ([www.boost.org](http://www.boost.org)).

All the sample code in this article is written for Microsoft Windows, but it can easily be ported to other platforms.

## The Traditional Way to Embed the JVM

Although there are lots of resources on how to embed a JVM, most of them are based on Sheng Liang's classic book, *Java Native Interface: Programmer's Guide and Specification*. Figure 1 shows the basic procedure.

If the application has other threads that are using the JVM as well, the communications among these threads will look like Figure 2.

Listing 1 shows the typical code (all the error checking and exception handling code in this article has been omitted for clarity). This code works fine as long as your program is simple enough; however, there are three major drawbacks.

**Issue 1:** The JVM is a global variable, which is not acceptable in some cases.

**Issue 2:** The threads in which `jvm` and `env` are used and the thread that launches `jvm` couple tightly. For example, there will be a potential timing issue that's clearly shown in Figure 2. The `AttachCurrentThread(...)` must be called after the launcher thread initialized `jvm`, and `DetachCurrentThread()` must be called before `jvm` has been destroyed; otherwise this creates unnecessary communications overhead between threads. It gets worse when you don't know which thread is the launcher thread.

Of course the VM can always be created in one thread when the native application starts. However, this is a pure waste if there is no Java client. The JVM should not be created if there are no requests.

**Issue 3.** The function calls to `AttachCurrentThread(...)` and `DetachCurrentThread()` should be paired so the JVM can get a chance to free the local references. However, it's too easy to not have the `DetachCurrentThread()` called by the programmer (because of multiple flow paths) or by exceptions. A better approach would be to encapsulate the JVM into a class or classes.

## Thin JVM Wrapper

It's natural to wrap the `JavaVM*` pointer into a member variable, say `jvm`. Because only one JVM in each process can be created (until JDK 1.4.2), the obvious class design is to make `jvm` static. Listing 2 shows the first attempt.

Here the method's signatures are not important so let's focus on the class design. The thin wrapper approach is simple and straightforward; however, it's not much better than the C-like code in Listing 1. It only gets rid of the explicit global variable `jvm` (the static member variable is still kind of global though). Issues 2 and 3, mentioned in the previous section, still need to be addressed.

## Singleton JVM

The Singleton design pattern is described in *Design Patterns* by Erich Gamma, et al, as "Ensure a class only has one instance, and provide a global point of access of it." In other words, a Singleton class is a class for which no more than one instance can exist at runtime. This is what the JVM behaves like. We can apply the Singleton pattern to the JVM class design.

Andrei Alexandrescu has provided all kinds of singleton implementations in his book *Modern C++ Design* and the Loki library. Loki has a singleton manager class template, `SingletonHolder`, that looks like Listing 3.

The template parameter `T` in Listing 3 is the target class, in our case `CJavaVM`. Other template parameters specify the policies that manage the singleton. As explained in Alexandrescu's book, "A policy defines a class interface or a class template interface. The interface consists of one or all of the following: inner type definitions, member functions, and member variables." Policy classes are not intended for stand-alone use and their member functions are often static. In `Loki::SingletonHolder`, the



**Stanley Wang** is a lead software developer at Vcom3D, Inc., and the author of JTL. He received his MS in computer science from the University of Florida. He is interested in system programming, generic programming, database systems, and computer graphics.

[stanleyycwang@yahoo.com](mailto:stanleyycwang@yahoo.com)

# We have a problem.

J2EE application problems can grind your business to a screeching halt, devouring resources and devastating your quality of service.

Why hunt and peck, trying to recreate the problem and arguing about who's to blame?

AppSight breaks through the wall between the place problems are found and the place they're solved, so your team can pinpoint root causes faster than ever.

We're talking user blunders, configuration problems, performance issues, all the way down to code errors —

All without taking your application offline.

With AppSight,  
it's problem solved.



CreationPolicy determines how the singleton instance is created and destroyed, and the ThreadingModel policy determines whether the singleton is living in a multiple threaded world.

Listing 4 provides the redesigned CJavaVM class and its usage with Loki::SingletonHolder. A difference between CJavaVM and CJavaVM2 is that all members in CJavaVM2 are not static. Listing 4 looks fancier than Listing 1. However, it does not solve the real problems. Issues 2 and 3 are still unresolved. For example, if there are any other threads trying to make a JNI call, it's still a requirement that the JVM launcher thread has already called `jvm.StartJavaVM()`. This is because CJavaVM2's constructor does nothing. If `StartJavaVM()` can be moved into CJavaVM2's constructor and `DestroyJavaVM()` can be moved into CJavaVM2's destructor, then the timing issue will be solved, which is not easy to do.

The major difficulty is that Loki::SingletonHolder's template parameter CreationPolicy only calls the target class T's default constructor. In other words, CJavaVM2's constructor cannot look something like this: `CJavaVM2(std::string jvmpath, JavaVMInitArgs& args)`. It must be `CJavaVM2()`. Period.

We are not out of bullets though. The "Every problem can be solved by adding another layer of indirection" idiom applies here.

### Policy-Based JVM Class

The solution is to make CJavaVM2 policy-based too. Listing 5 shows the new version.

In CJavaVM3, a new policy class JavaVMLauncher is introduced and declared as follows:

```
class JavaVMLauncher {
public:
    static void LaunchJavaVM(void** ppLib, JavaVM** ppJavaVM);
    static void DestroyJavaVM(void* pLib, JavaVM* pJavaVM);
}
```

`ppLib` is a pointer to a handle returned by `LoadLibrary` on Windows or `dlopen` on Solaris. To make `JavaVMLauncher` more generic, the void pointer is used here. The pointer to pointer method is used for `ppLib` and `ppJavaVM` in `LaunchJavaVM()` because `JavaVMLauncher` does not contain any member variables. It's the caller to provide the placeholders for the objects pointed to by `ppLib` and `ppJavaVM`. `DestroyJavaVM()` simply accepts the cached `JavaVM*` and `lib` handle to destroy the JVM and free the library.

Now we can invoke the JVM as in Listing 6, which is pretty much what JTL does. In JTL, there is another class template called `JavaVMMgr` that is the placeholder for the `JavaVM*` and `pLib`. All the JVM calls will be delegated to the `JavaVMMgr`.

### Exception Handling

The `LaunchJavaVM()` method in `JavaVMLauncher` can fail; for example, if the user does not have a JRE installed on his machine. When this happens, `LaunchJavaVM()` can return a flag or error code. In JTL, it will simply throw a `jtl::load_javvm_error` exception.

### Multithreading, Extending Loki, and boost::thread

Remember in Listing 3, `Loki::SingletonHolder` has a template parameter `ThreadingModel` that's used to synchronize the calls to `MakeInstance()`. However, under different

platforms the thread implementation and API are different. To make the JVM class design platform portable, it needs to use a portable threading framework. JTL uses `boost::thread` because it's simple and easy to use, and it will likely be part of the next version of the standard C++ library.

Though Alexandrescu provides us with the excellent `SingletonHolder` class, JTL does not use it directly for two main reasons: not all compilers support `Loki` well and `Loki::ThreadingModel` is only implemented to work with Windows. `jtl::SingletonHolder` does some extension to `Loki::SingletonHolder` to achieve the platform-portable goal. However, `jtl::SingletonHolder` uses the same template parameters (and in the same order) as `Loki::SingletonHolder`, therefore the user can easily switch to `Loki`'s version if someday `Loki` provides us with a platform-portable `ThreadingModel`.

Another thing needs to be mentioned for `ThreadingModel` – it should be able to work with both the single-thread model and multiple-thread model. If there's only one thread, there's no sense in doing the synchronization. To achieve this generic solution, JTL provides two class templates, `MultipleThreadModel` and `SingleThreadModel`, as shown in Listing 7.

These two classes simply provide type definitions. In `MultipleThreadModel`, JTL uses the `boost::mutex` and `boost::mutex::scoped_lock` as the synchronization primitives, while in `SingleThreadModel` it uses `boostex::faked_mutex` and `boostex::faked_mutex::scoped_lock` as the synchronization primitives. The `faked_mutex` and `scoped_lock` are simple extensions to `boost::mutex` and `scoped_lock`. They're empty classes and provide only the required interface methods, which are implemented as `noop`. This is a common trick in generic programming. (ATL programmers will recall the `CComFakeCriticalSection` used in `CComSingleThreadModel` and `CComMultiThreadModel` that does the same trick.)

### JNIEnv Smart Pointers

In Listing 6, we solved issues 1 and 2; we no longer have a global variable, and we don't have a timing issue when calling `AttachCurrentThread()` and `DetachCurrentThread()`. However, issue 3 – how to ensure that `DetachCurrentThread()` is called – is still unresolved.

When doing JNI programming, most of the time we're dealing with the `JNIEnv*` pointer rather than the `JavaVM*` pointer. The most important characteristic of the `JNIEnv` pointer is that it has thread affinity (it's only valid in its associated thread). It would be nice if we could get the `JNIEnv` pointer in an arbitrary context. To achieve this and solve issue 3, JTL provides three smart pointers: `simple_env_ptr`, `auto_env_ptr`, and `thread_env_ptr`. They're all class templates.

Before talking about these smart pointers, let's review the scenarios in which the `JNIEnv` pointer is used:

- In the launcher thread:** In this case the `ThreadingModel` template parameter should be `SingleThreadModel` to avoid unnecessary thread synchronization, even though there may be multiple threads.
- In multiple threads:** `AttachCurrentThread()` and `DetachCurrentThread()` will be called only once for each thread.
- In multiple threads:** `AttachCurrentThread()` and `DetachCurrentThread()` can be called multiple times for each thread (see below).

**Preventing Errors Through Defect Tracking**

Have you ever considered using a defect tracking system as an error prevention tool? Here is what you can do to prevent the same errors from re-occurring in your software lifecycle:

With every release of a new product, use a defect tracking system to record all of the errors that were reported. Then analyze these errors and construct a list that groups the defects into similar categories. By grouping them, you will be able to find any correlations between the errors.

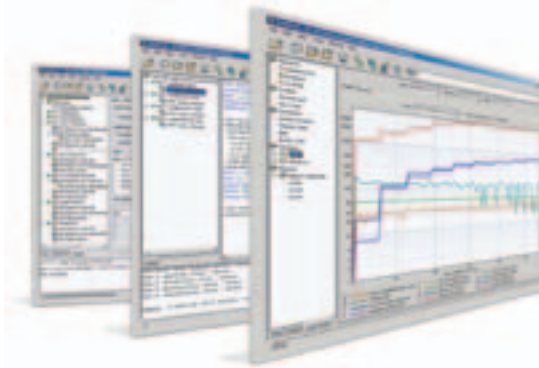
When you find a correlation, verify whether or not the root cause of the errors originated from the same place. The errors you find may or may not have the same root cause. Regardless, you should be able to locate the points in your software lifecycle at which these errors occurred, and you should also be able to determine the exact cause of these similar errors.

After locating and determining the cause of your errors, you will be able to implement preventative practices that will keep you from making the same mistakes over and over again. Give this procedure a try and see if it works for you. You should see a significant reduction of errors in the next release of your product.

In essence, the procedure described above is a fundamental part of the Parasoft Automated Error Prevention (AEP) Methodology. You can find more information about Automated Error Prevention at [www.parasoft.com](http://www.parasoft.com).

— Adam Kolawa, Ph.D.  
Chairman/CEO of Parasoft

**It's automated. It's fast. And it's the most versatile Web Services testing tool ever.**



**Introducing Parasoft® SOAPtest®**

The simple fact is, no other Web service testing tool can do what SOAPtest can do. Or do it as fast and reliably. From ensuring functionality and interoperability to telling you how your Web service is performing, SOAPtest automates all critical testing processes across the entire lifecycle of your Web service.

**But don't take our word for it...** Go to [www.Parasoft.com/SoapTest](http://www.Parasoft.com/SoapTest) and try it for free—no commitment, no obligation. If you like it (and we suspect you will) just let us know. We'll be more than happy to help you and your development team get up and running.

For Downloads go to [www.parasoft.com/soaptest](http://www.parasoft.com/soaptest)

Call 888-305-0041 x3303 or email [buzz@parasoft.com](mailto:buzz@parasoft.com)



*Founded in 1987, Parasoft's clients include IBM, HP, DaimlerChrysler and over 10,000 companies worldwide.*

**Features**

- WSDL schema verification and compliance to standards.
- Automatic test creation using WSDL and HTTP Traffic.
- Data-driven testing through data sources (Excel, CSV, Database Queries, etc.).
- Scenario-based testing through XML Data Bank and Test Suite Logic.
- Flexible scripting with Java, Java Script, Python.
- WS-I Conformance: Basic Profile 1.0.
- WS-Security, SAML, Username Token, X.509, XML Encryption, and XML Signature support.
- MIME/DIME Attachment support.
- Asynchronous Testing: JMS, Parlay, Parlay X and SCP (SOAP Conversation Protocol) support.
- Identifies bottlenecks through SNMP, Windows, and JMX Monitors.
- Detailed Report generation in HTML, XML and Text formats.
- Real-Time graphs and charts.

**Benefits**

- Uniform test suites can be rolled over from unit testing to functional testing to load testing.
- Prevent errors, pinpoint weaknesses, and stress test long before deployment.
- Ensure the reliability, quality, security and interoperability of your Web service.
- Verify data integrity and server/client functionality.
- Identify server capabilities under stress and load.
- Accelerate time to market.

**Protocol Support**

- HTTP 1.0
- HTTP 1.1 w/Keep-Alive Connection
- HTTPS
- TCP/IP
- JMS

**Platforms**

Windows 2000/XP  
Linux  
Solaris

**Contact Info:**

Parasoft Corporation  
101 E. Huntington Dr., 2nd Flr.  
Monrovia, CA 91016

[www.parasoft.com](http://www.parasoft.com)

simple\_env\_ptr, auto\_env\_ptr and thread\_env\_ptr are corresponding solutions for scenarios 1, 2, and 3. All these smart pointers are derived from env\_ptr\_base, which is a placeholder for JNIEnv\* pointer and has other helper functions, such as operator->(), operator!(). By the way, all three JNIEnv smart pointers are not full-blown. For more details about smart pointers, please refer to Alexandrescu's book.

The simple\_env\_ptr is defined as the following:

```
template < ... >
class simple_env_ptr : public env_ptr_base {
public:
    //... typedefs for SingletonJVM;
public:
    simple_env_ptr() {
        SingletonJVM::Instance().GetEnv(&env_, jvm_version);
    }
    ~simple_env_ptr() { env_ = 0; }
};
```

In simple\_env\_ptr's constructor, the JNIEnv pointer has been initialized by calling GetEnv(). This works because the smart pointer is in the launcher thread.

Similarly, auto\_env\_ptr is defined as the following:

```
template < ... >
class auto_env_ptr : public env_ptr_base {
public:
    //... typedefs for SingletonJVM;
public:
    auto_env_ptr() {
        SingletonJVM::Instance().AttachCurrentThread(&env_, 0);
    }
    ~auto_env_ptr() {
        if(env_) {
            env_ = 0;
            SingletonJVM::Instance().DetachCurrentThread();
        }
    }
}; // auto_env_ptr
```

As you have seen, auto\_env\_ptr's constructor and destruct-

tor do the job of attaching and detaching the current thread. Thus we have solved issue 3 and we don't need to worry about the missing detaching call. However, auto\_env\_ptr may not work if there is more than one auto\_env\_ptr instance in the same thread. This is clear in the following snippet:

```
// auto_env_ptr ptr1, ptr2
ptr1;
// ...other stuff
{
    ptr2;
}
// ... from here all calls on ptr1 may be invalid
```

In this snippet, when ptr2 is out of scope, its destructor will be called, causing the current thread to detach from the JVM. This may cause problems because ptr1 is still active; however, all its local references may have been freed by the JVM because of the ptr2's detaching call.

To solve this problem, JTL provides thread\_env\_ptr, which is defined in Listing 8. thread\_env\_ptr has a counter that's a thread local storage variable. Whenever there is a new thread\_env\_ptr instance, the counter will be bumped by one. Whenever there is a thread\_env\_ptr out of scope, the counter will be decreased by one. When the counter is decreased to zero, the current thread will be detached from the JVM.

## Putting It All Together

Listing 9 illustrates how to use the JTL JVM invocation. You may provide your own JVMLauncher if the default one does not meet your requirement.

## Conclusion

JTL provides complete support for the JVM invocation. It also provides JNIEnv smart pointers to get the JNIEnv\* pointer in arbitrary context. Most of the classes in JTL are designed as class templates and they can easily be extended. JTL provides many template parameter implementations as well, which can be used as the default template parameter value under most circumstances.

All comments about JTL and this article are highly appreciated. ☺

### Listing 1: The traditional way to launch a JVM

```
typedef jint (JNICALL *pfnCreateJVM)(JavaVM** ppJVM, void**
ppEnv, void* args);
// global jvm, may be used by other threads
JavaVM* jvm;
... .. // other stuff
// Java VM launcher thread, maybe main thread, maybe not
JavaVMInitArgs vm_args;
JavaVMOption options[n];
... .. // fill in options
... .. // fill in vm_args
// get JNI_CreateVM address
HMODULE hModule = ::LoadLibrary(your_libpath);
pfnCreateJVM pfn = (pfnCreateJVM)::GetProcAddress(hModule,
"JNI_CreateJavaVM");
JNIEnv* env;
// launch jvm
pfn(&jvm, (void*)&env, &vm_args);
// work with env
... ..
// destroy jvm
jvm->DestroyJavaVM ();
jvm = 0;
::FreeLibrary(hModule);

// Java VM consumer thread: thread-1
```

```
void thread_fun() {
JNIEnv* env = 0;
// get env
jvm->AttachCurrentThread(&env, JNI_VERSION_1_2);
... .. // work with env
jvm->DetachCurrentThread();
}
```

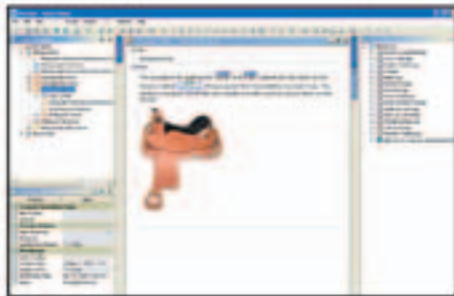
### Listing 2: Thin JVM wrapper

```
class CJavaVM { // JavaVM* wrapper
private:
    static JavaVM* jvm;
public:
    static bool StartJavaVM(std::string jvmpath,
JavaVMInitArgs& args);
    static bool DestroyJavaVM();
    static bool GetEnv(JNIEnv** ppEnv, jint version);
    static bool AttachCurrentThread(JNIEnv** ppEnv, void*
args);
    static bool DetachCurrentThread();
    static bool AttachCurrentThreadAsDaemon(JNIEnv** ppEnv,
void* args);
    ... .. // other methods
};
```

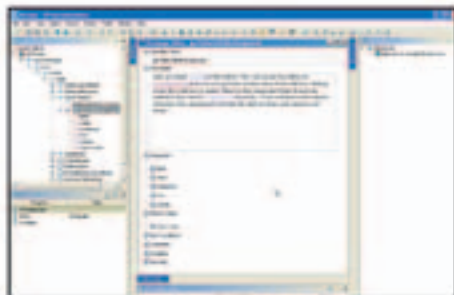
> Can't someone develop a single sourcing authoring tool that saves time, allows for collaboration and reduces headaches for developers and technical writers?

# Introducing Veredus™ 2.0

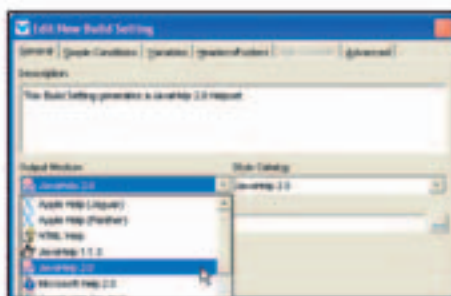
## Single sourcing made easy.



A single fully integrated editor for all outputs.



Veredus directly imports type libraries and Java™ source files and creates a template to easily add comments.



Output is as easy as choosing the output medium, style and where to store the output to create JavaHelp.

## VEREDUS

### API DOCUMENTATION

- > Produce documentation using terms and vocabulary common to writers and developers
- > Create documentation automatically by scanning Java source code and Microsoft Windows COM Objects
- > Automatically generate signatures of API properties and methods, return values, parameters, and when available, error codes
- > Re-scan developer code to merge changes into your Veredus project quickly and easily, keeping documentation accurate and up-to-date
- > Generate documentation in a variety of formats from your Veredus project such as JavaHelp and PDF

### SINGLE SOURCING

- > A single product that creates content once for multiple deliverables in content and format
- > Veredus saves you time, reduces headaches and creates W3C compliant output for customized printing, Web portals and a variety of Help formats

### HELP AUTHORING

- > Quickly create professional help systems and user manuals from a single source.
- > Change your output target without painful conversion processes
- > Produce output in any format: JavaHelp, Oracle Help, HTML Help, Apple Help, Microsoft Help, WinHelp and others
- > Translate the same information into different languages via XLIFF and a localization partner



Download a **FREE** trial of Veredus 2.0

Visit [www.rascalsoftware.com/java](http://www.rascalsoftware.com/java) or call 206.624.7300

**Listing 3: Loki::SingletonHolder class template**

```
template <
    typename T,
    template <class> class CreationPolicy =
    CreateUsingNew,
    template <class> class LifetimePolicy =
    DefaultLifetime,
    template <class> class ThreadingModel =
    SingleThreaded
>
class SingletonHolder {
public:
    static T& Instance();
private:
    static void MakeInstance();
    static void DestroySingleton();

    typedef typename ThreadingModel<T*>::VolatileType
    PtrInstanceType;
    static PtrInstanceType pInstance_;
    static bool destroyed_;
};
```

**Listing 4: CJavaVM2 with Loki::SingletonHolder**

```
// CJavaVM2.hpp
class CJavaVM2 { // JavaVM* wrapper
private:
    JavaVM* jvm;
    friend Loki::CreateUsingNew<CJavaVM2>;
    CJavaVM2();
public:
    bool StartJavaVM(std::string jvmpath, JavaVMInitArgs&
args);
    bool DestroyJavaVM();
    bool GetEnv(JNIEnv** ppEnv, jint version);
    bool AttachCurrentThread(JNIEnv** ppEnv, void* args);
    bool DetachCurrentThread();
    bool AttachCurrentThreadAsDaemon(JNIEnv** ppEnv, void*
args);
    ... .. // other methods
};

// main.cpp
typedef Loki::SingletonHolder<CJavaVM2> JVM;
int main(){
    JavaVMInitArgs args;
    std::string libpath;
    JNIEnv* env;
    ..... // fill in libpath and args
    CJavaVM2& jvm = JVM::Instance();
    jvm.StartJavaVM(libpath, args);
    jvm.GetEnv(&env, JNI_VERSION_1_2);
    ..... // working with env
    jvm.DestroyJavaVM();
}
```

**Listing 5: CJavaVM3**

```
template <class JavaVMLauncher>
class CJavaVM3 {
private:
    JavaVM* jvm;

    CJavaVM3 () { JavaVMLauncher::LaunchJavaVM(... /* to do
*/); }
    ~ CJavaVM3 () { JavaVMLauncher::DestroyJavaVM(... /* to
do */); }
public:
    bool AttachCurrentThread(JNIEnv** ppEnv, void* args);
    bool GetEnv(JNIEnv** ppEnv, jint version);
    bool AttachCurrentThread(JNIEnv** ppEnv, void* args);
    bool DetachCurrentThread();
    bool AttachCurrentThreadAsDaemon(JNIEnv** ppEnv, void*
args);
    ... .. // other methods
};
```

**Listing 6: JVM invocation with CJavaVM3**

```
// main.cpp
typedef CJavaVM3<YourLauncher> CJavaVM;
typedef Loki::SingletonHolder<CJavaVM> JVM;
int main(){
    JNIEnv* env;
    CJavaVM2& jvm = JVM::Instance();
    jvm.GetEnv(&env, JNI_VERSION_1_2);
    ..... // working with env
}
```

**Listing 7: JTL Threading model**

```
template <class Host>
class SingleThreadModel {
public:
    typedef Host volatile_type;

    typedef boostex::faked_mutex mutex;
    typedef boostex::faked_mutex::scoped_lock scoped_lock;
};

template <class Host>
class MultipleThreadModel {
public:
    typedef volatile Host volatile_type;

    typedef boost::mutex mutex;
    typedef boost::mutex::scoped_lock scoped_lock;
};
```

**Listing 8: thread\_env\_ptr**

```
template <... >
class thread_env_ptr : public env_ptr_base {
public:
    //... typedefs for SingletonJVM;
public:
    thread_env_ptr() {
        bool bOk = SingletonJVM::Instance().
AttachCurrentThread(&env_, 0);
        if(bOk) {
            if(0 == s_counterptr.get()) {
                s_counterptr.reset(new int(1));
            }else {
                ++(*s_counterptr);
            }
        }
    }
    ~thread_env_ptr() {
        if(env_) {
            env_ = 0;
            --(*s_counterptr);
            if(0 == *s_counterptr) {
                SingletonJVM::Instance().
DetachCurrentThread();
            }
        }
    }
private:
    static boost::thread_specific_ptr<int> s_counterptr;
};
```

**Listing 9: A complete JTL JVM invocation example**

```
// all necessary headers
typedef jtl::win::DefaultJavaVMLauncher JVMLauncher;
boost::mutex io_mutex; // synchronize std out

class Dummy {
public:
    jtl::thread_env_ptr<JVMLauncher> env_;
    void Test() {
        boostex::ThreadID::ThreadIdType tid = boost-
ex::ThreadID::get_current_threadid();
        boost::mutex::scoped_lock lock(io_mutex);
        if(env_)
            std::cout << "get JNIEnv* in thread:"
                << tid << std::endl;
        else
            std::cout << "can not get JNIEnv* in thread:"
                << tid << std::endl;
    }
};

void launch() // thread function
{
    Dummy dummy[3];
    for(int i = 0; i < 3; ++i)
    {
        dummy[i].Test();
    }
}

int main()
{
    boost::thread thrd(&launch);

    Dummy dummy;
    dummy.Test();

    thrd.join();

    return 0;
}
```



# Introducing tools you'll actually use.

We've all bought our share of shelfware. Seduced by visions of breakthrough performance and eye-popping results, we gleefully loaded the latest software, only to find it cumbersome and overly complicated. And so it was successfully installed ... on the shelf.

## Energy software tools for Java: Simple. Unobtrusive. And refreshingly useful.

At Enerjy, we believe that the purpose of software tools is to help developers create better applications faster, without getting in the way. That's why we designed our tools to integrate seamlessly within your IDE. They let you analyze your code, monitor performance, identify thread problems and memory leaks—all without breaking your stride.

Sure, we can trot out ROI studies, user stories, and application briefs that prove our software increases productivity, reduces development cycles, and reliably produces eye-popping results. But we'd prefer to let you draw your own conclusions.

So go to [www.enerjy.com](http://www.enerjy.com) and add Enerjy to your Java development today. If our tools don't become an indispensable part of your development process, return them for a full refund under our Anti-Shelfware Guarantee. Hey, your shelves are crowded enough as it is.

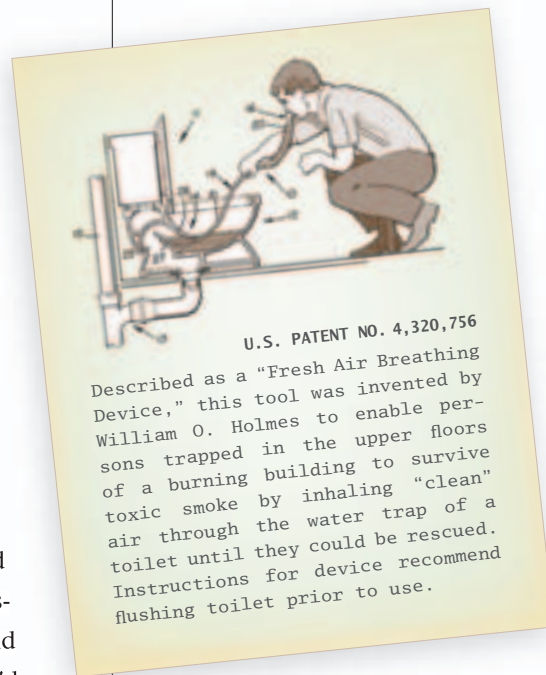
Visit our website to learn more about our Java tools:

ENERJY |  CODE ANALYZER Best practices audit tool

ENERJY |  PERFORMANCE PROFILER Improve application speed

ENERJY |  MEMORY PROFILER Reduce your memory footprint

ENERJY |  THREAD PROFILER Find, fix, and avoid thread problems



From the "Tools You're Not Likely Ever To Use" archive.

# ENERJY™

Java software tools

[www.enerjy.com](http://www.enerjy.com)

# Web Conferencing Using the Java Media Framework

by Yayati Kasralikar and  
Pramod Jain

## Broadcast and receive media streams

This article describes our experiences with developing a browser-based Web conferencing application with the following constraints:

1. HTTP protocol (port 80) to broadcast and receive video/audio
2. Broadcasters and receivers are not required to have public IP addresses
3. Multiple users, each capable of broadcasting to and receiving feeds from many users
4. Low-cost solution for continuous video/audio feed



**Pramod Jain** is president, Innovative Decision Technologies, Inc. (INDENT). INDENT has built several large-scale Java-based collaboration portals. Pramod has a PhD from the University of California, Berkeley.

pramod@indent.org



**Yayati Kasralikar** is lead programmer at Innovative Decision Technologies, Inc. (INDENT). He holds an MS from the University of Central Florida, Orlando.

yayati@indent.org

Java Media Framework (JMF) is used to develop the browser-based Web conferencing application. In this architecture, the client uses two JMF applets – one for capturing video/audio from a Webcam and the other for playing video/audio feed. The capture applet continuously captures video/audio feed for a specified length of time (e.g., 10 seconds) and saves it locally in a file. This file is uploaded to a Web server using an upload servlet. The media stream is stored in MSVIDEO (AVI) or QUICKTIME (MOV) format. The player applet then continuously fetches the media clips from the Web server. The player applet uses perfecting capability to play clips from the server in a continuous manner. The advantages of this approach are that it does not require expensive streaming servers, and it satisfies the constraints mentioned earlier.

The article will start with a brief explanation about real-time streaming, followed by an introduction to JMF, a description of the capture and player applets, and a comparison with other technologies.

### Introduction to Real-Time and Progressive Streaming

Real-time streaming of media allows users to play media as they receive it.

Users don't have to wait for the whole media file to be downloaded before watching. To enable real-time streaming, dedicated streaming media servers and streaming protocols, such as Real-Time Protocol (RTP), are required. RTP is an Internet standard for transporting real-time data. It uses the unreliable UDP protocol to transmit packets.

A variant of real-time streaming is progressive streaming, also called HTTP streaming because it uses the commonly used HTTP protocol and standard HTTP servers to deliver media files. Progressive streaming enables files to be watched as they are downloaded. When the client makes a request (HTTP) to the server for the media file, the file eventually gets stored in the client's memory buffer. The playback is allowed before the entire file gets downloaded. Most firewalls allow traffic over HTTP whereas RTP is not permitted by most firewalls. In our approach, we're emulating HTTP streaming.

### Introduction to Java Media Framework

The JMF API specifies a simple, unified architecture to synchronize and control audio, video, and other time-based data within Java applications and applets. JMF software, documentation, sample programs, and the source code can be downloaded from Sun's Web site at <http://java.sun.com/products/java-media/jmf>. In this section, we'll briefly discuss the basic concepts of JMF, including a few useful classes required to build a Web conferencing application:

1. The DataSource class is an abstraction that represents audio, video, or a combination of both. A data source can be a file or a stream and is constructed from the Manager and MediaLocator as follows:

```
DataSource ds = javax.media.Manager.createDataSource(mediaLocator);
```

Here, MediaLocator is a class that JMF uses to represent audio or video media location and is created as follows:

```
MediaLocator mediaLocator = new MediaLocator("vfw://0");
```

2. The Player class is used to play media files or stream media. A player is constructed from MediaLocator or the media URL as follows:

```
Player player = Manager.createPlayer(mediaLocator);
```

Once the player is realized (ready to play state), you can call player.start() to play the media. A realized player can be created from the DataSource:

```
Player player = Manager.createRealizedPlayer(ds);
```

3. A processor is a type of player. Besides playing the media, it can also output media through a DataSource to another player or processor. A processor is used to manipulate the data and convert the data from one format to another. It's created from the DataSource, MediaLocator, or a URL:

```
Processor processor = Manager.createProcessor(new URL("http://localhost/test.mov));
```

4. A manager is one of the most important classes of JMF. It handles the construction of players, processors, and DataSources, as we have seen earlier.

### Architecture Description

The architecture of our approach is shown in Figure 1. It implements a Web conferencing application over HTTP. The architecture has one



# 21 WAYS TO USE SPREADSHEETS IN YOUR JAVA APPLICATIONS

**A free offer for readers of *Java Developer's Journal*!**

**Formula One e.Spreadsheets Engine:**

Finally, there's a *supported*, Pure Java tool that merges the power of Excel spreadsheets and Java applications.

- 1 Automatically generate dynamic Excel reports. No more manual querying and cutting-and-pasting to create Excel reports!
- 2 Manage calculations and business rules on J2EE servers with Excel files. No more translating Excel formulas to Java code!
- 3 Embed live, Excel-compatible data grids in applets and Java desktop applications. No more static HTML or presentation-only data grids!

**Download this quick-read white paper and trial today!**



Download your trial and test our demos and sample code. See for yourself how the Formula One e.Spreadsheets Engine can help your Java application leverage the skills of Excel users in your business.

<http://www.reportingengines.com/download/21ways.jsp>



888-884-8665 • [www.reportingengines.com](http://www.reportingengines.com)  
sales@reportingengines.com

**Need to deliver reports from your J2EE application or portal server? Try the Formula One e.Report Engine!**



Build reports against JDBC, XML, Java objects, BEA Portal Server logs, BEA Liquid Data, and other sources visually or with Java code. It's embedded! No external report server to set up. Unlimited users and CPUs per license.

<http://www.reportingengines.com/download/f1ere.jsp>

centralized server and one or many distributed clients. The server has a Web server and a servlet container. Clients run two applets, one for capturing media and the other for playing the media.

The high level steps are:

1. The applet continuously captures video and audio streams from the Webcam. These streams are saved locally in a specified format as a file every few seconds. This file is uploaded to the server over HTTP using a file upload servlet. Uploading uses a separate thread. A significant loss of frames will result if the file upload is in the same thread as file capture. Note that a more efficient method would be to write these streams directly on the server using a socket. This is currently not possible because the DataSource class provided with JMF does not contain a method to get the InputStream. A custom InputStream-based DataSource can be developed (e.g., <http://www.extollit.com/isdsmjmf.php>).
2. A server gets a new file clip from the sender and stores it in a sender-specific directory. A counter, such as filename+i, is attached to the filename.
3. The JMF Player applet continuously downloads new files from the Web server. It uses JMF's perfecting capability to play these clips in a continuous manner. When the current clip is being played, a new instance of Player is created for the next clip and the next clip is downloaded from the server. This makes the playing of clips continuous, as the next clip to be played has already been prefetched. Note that the entire clip is downloaded by the player applet before playing it.

At the start of playing and during the process of fetching a new clip, the player applet checks new file availability for  $n$  seconds before timing out.

### Computation of Parameters

First, we'll do an approximate mathematical analysis for bandwidth consideration and demonstrate the usability of our approach with a few special cases.

Suppose:

One second file clip size = oneSecFileSize bits  
 Time duration of each clip = cSec seconds  
 Upload Transmission rate = uRate bits per second  
 Download Transmission rate = dRate bits per second  
 Time to upload, tUpload = oneSecFileSize \* cSec / uRate  
 Time to download, tDownload = oneSecFileSize \* cSec / dRate

If the time to upload or download a clip is more than the time to play a clip, the player will wait and the receiver will see a break, i.e.,  $\max(tUpload, tDownload) > cSec$ . For the continuous playing of clips, the following condition must be true:

Max (1/uRate, 1/dRate) > 1/ oneSecFileSize  
 Min (uRate, dRate) > oneSecFileSize

According to the equation, the wait time between clips at the receiver does not depend on clip size. The only variable that matters for a continuous playback is the size of a one-second file and that the provided upload and download rates meet the above condition. Lag time between playing and capturing is:

$cSec + tupload + tdownload$

From the above equation, the maximum lag with no break in the feed is  $3 * cSec$ , and the minimum lag is  $cSec$ .

To get a Web conference that is as close to real time as possible,  $cSec$  should be reduced. Next, we will apply the above analysis to the following cases.

- **Both sender and receiver have a low bandwidth modem connection**

Let's assume the  $uRate = dRate = 20K$  bits/sec. In this case, the one-second file

size should be less than 20Kbits. If the clip size is 10 seconds, the maximum playback lag will be 30 seconds. We have observed that the minimum file size for transmitting a one-second video (with no audio) is 8Kbits using H263 encoding and 128x96 pixels video size. H263 encoding is ideal for a low-bandwidth environment because it produces smaller file sizes. The H263 encoder in the JMF 2.0 is capable of handling only limited video sizes (only 352x288, 176x144, and 128x96). We observed a minimum file size with the video and an 8-bit mono audio with an 8000Hz sampling rate to be 80Kbits.

- **Either the sender or receiver has a low bandwidth connection**

Let's assume that the lower rate is  $rate = 20Kbits/sec$  and the other rate is much higher. In this case the one-second file size should be less than 20Kbits, but the maximum playback lag is about 20 seconds if the clip size is 10 seconds.

- **Both sender and receiver have high bandwidth**

In this case better quality video can be transmitted. The playback lag will be the same as the clip size in seconds. JPEG encoding offers good quality video and is well suited to a high-bandwidth environment. File sizes can be decreased during JPEG encoding by lowering JPEG quality.

There are no easy guidelines to predict the exact size of the one-second clip; it depends on the video size, the audio sampling rate, video and audio encoding, the frame rate, and the file format. Users should experiment with using different values for these parameters and a variety of movement in the video to determine an approximate one-second file size.

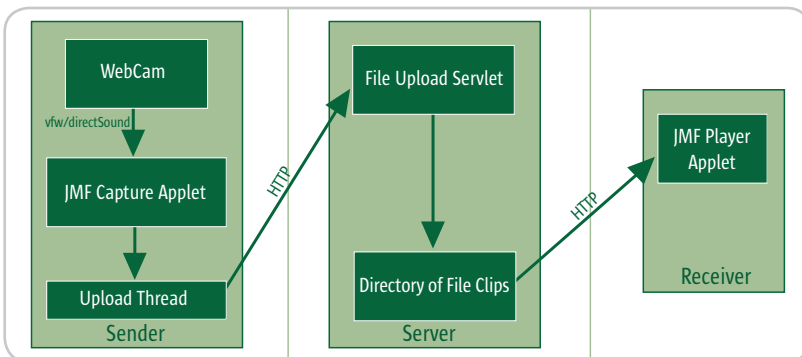


Figure 1 Architecture



**Ready for a  
stampede?**

**TANGOSOL COHERENCE™** provides even the most heavily burdened applications with predictable and linear scalability while shielding database systems from unnecessary load. Coherence powers some of the largest application server clusters, providing coherent data caching, stateful clustering and session management for clusters of up to hundreds of servers.

Try before you buy. Download free evaluation software at

[www.tangosol.com](http://www.tangosol.com)

Coherence™ is pure Java software that reliably and cost-effectively scales J2EE applications across any number of servers in a cluster. Coherence provides reliable coordinated access to in-memory data that enables safe data caching and user session management for applications in a clustered environment.

**Sales** +1.617.623.5782 or [sales@tangosol.com](mailto:sales@tangosol.com)  
**Support** [support@tangosol.com](mailto:support@tangosol.com) or <http://www.tangosol.net>  
**Pricing** US\$1,995 to US\$4,995 per Coherence production CPU  
Development licenses are offered free of charge.

### JMF Capture Applet

The high-level steps for developing capture applet are (see Listing 1):

1. A DataSource is created from the Webcam source using the MediaLocator.
2. A ProcessorModel is created from the DataSource, the format object specifying the video format, and the FileDescriptor object specifying the output file format.
3. A Processor is created from the ProcessorModel and the output DataSource is obtained from the Processor.
4. A DataSink object is created by first creating a MediaLocator for storing the media in a file.
5. Capture of the stream is started and the stream is saved for a specified duration into a file.

This process is repeated until the sender ends the session.

### File Upload

The File Upload uses the JUpload project (<http://jupload.sourceforge.net/>). It has two parts: the file upload thread at the client and the upload servlet at the server. The following are the steps for developing the File Upload thread (see Listing 2):

1. Create a socket connection with the server.
2. Create an HTTP POST request and an HTTP head and tail.
3. Create necessary IO stream objects.
4. Send an HTTP request to the server. Write the HTTP head, the clip file, and the HTTP tail to the server.

The Upload Servlet uses O'Reilly's multipart request executor ([www.servlets.com/cos/index.html](http://www.servlets.com/cos/index.html)) to upload the files. MultipartRequest is a utility class that handles multipart/form-data requests for file uploads.

### JMF Player Applet

The high-level steps for developing a player applet are (see Listing 3):

1. Construct two players from the URL of the media at the Web server. One player is for the current clip and the other is for the next clip.
2. Start the first player and fetch the next clip using the second player.
3. On the EndOfMediaEvent for clip *i*, start playing clip *i*+1. Destroy the visual component for the player of clip *i*, de-allocate the player, and create a new player for clip *i*+2. Prefetch the clip *i*+2 and add ControllerListener. Repeat these steps for subsequent clips.

This makes the playing of clips continuous, as there will be little or minimal delay between subsequent clips. Note that the entire clip is downloaded by the player applet before playing it.

### HTML Code for Sender and Receiver Applets

The HTML code for the sender applet is shown in Listing 4.

The HTML code for the receiver applet is shown in Listing 5. Note that this HTML page is generated dynamically with the appropriate senderID and current counter. If the receiver wants to receive multiple feeds, multiple applet entries are generated in HTML.

### Drawbacks

1. There is a lag between capturing and playing.
2. It involves expensive disk write operations.
3. Both receivers and senders must have JMF software installed.

### Future Enhancements

1. A sophisticated in-memory buffering mechanism to allow better video quality and efficient delivery by eliminating expensive disk writes

2. Extending the DataSource class to allow InputStream-based processing to save the media directly at the server and remove the need for a local buffer.
3. To package and deliver required dlls and registry files of JMF so that there's no need to install JMF software.

### Comparison to JMF-Based P2P Web Conferencing Using RTP

Thus far, we have described an HTTP-based approach that involves no real-time streaming. An alternative to the above approach is a peer-to-peer, RTP-based Web conferencing solution that can be developed using the JMF API. The source code for the RTP Server/Sender can be found at <http://java.sun.com/products/java-media/jmf/2.1.1/solutions/AVTransmit.html> and at <http://java.sun.com/products/java-media/jmf/2.1.1/samples/sample-code.html#RTPPlayerApplet> for the RTP Player applet. The RTP Server captures the media from the Webcam and streams it to receivers by specifying IP addresses and port numbers. The RTP Player listens on a specific port for streams coming from the sender's IP address.

The primary difference between the HTTP approach and the RTP approach is that RTP streams the feeds continuously to receivers without storing them in files locally or at the server. The disadvantages of the above RTP approach are:

1. Public IP addresses are required for both the sender and the receiver.
2. The senders and receivers should not be behind firewalls because RTP is not allowed by most corporate firewalls.
3. Also, as the number of participants increases, the number of ports also increases linearly. This makes user and port management challenging.
4. The default RTP implementation of JMF uses the unreliable UDP protocol, so delivery time and quality are not guaranteed – it may result in the dropping of frames or make frames out of sequence during transmission.

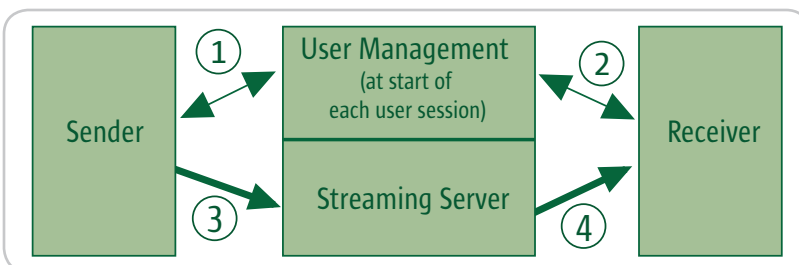


Figure 2 Web conferencing using streaming servers

Think .NET development is more productive than J2EE?

Think **again**.

Due to delivery pressures on our last project,  
we thought about moving to .NET.  
I suggested we stick with Java, but use ThinkCAP.

Think **better**.

**ThinkCAP**<sup>™</sup>

ClearNova's ThinkCAP is a comprehensive application platform that simplifies and accelerates the development and maintenance of J2EE-based business applications by 50 to 70%.

ThinkCAP's visual & intuitive designers bring high productivity to business developers (those with VB or PowerBuilder-like skills), content owners, and administrators while allowing J2EE architects & programmers to leverage its component infrastructure and build business logic using the tools and approaches they prefer. ThinkCAP utilizes existing infrastructure, web services, legacy systems, and business applications.

ThinkCAP saves organizations time and money—and lowers project risks. Applications are written faster and require less maintenance. Project teams utilize in-house skills and require less training. Existing infrastructure and application servers are leveraged.

With ThinkCAP you can build quality applications faster.

Learn more about ThinkCAP at [www.clearnova.com/thinkcap](http://www.clearnova.com/thinkcap)

**CLEARNOVA**  
APPLICATION DEVELOPMENT • SIMPLIFIED • ACCELERATED

Highly Visual Development Environment

MVC Framework with Page Flow & Actions

Advanced Data Aware Controls:  
Forms, DataViews, Queries, Navigations  
Workflows, Graphs, Treeviews, Grids, Tabs

Smart Data Binding<sup>™</sup> to data, objects, XML,  
sessions, or requests

Browser & server-side validation

Visual unit testing with RapidTest<sup>™</sup>

Service Flow Designer aggregates Web  
Services, EJBs, XML, and POJOs

Content Management engine & tools

Supports .NET clients

Integrated, seamless security

Use any app server or 3rd party tool

### Comparison to Web Conferencing Using Streaming Servers

The architecture of the Web conferencing system using commercially streaming servers is shown in Figure 2. Senders first register a unique broadcast/mount point with a user management component as shown by arrow 1. The sender then uses streaming protocol (for example, RTP or RTSP) to push the media stream to a centralized streaming server, as shown by arrow 3. Receivers first look for senders at the user management component (as shown by arrow 2) and obtain corresponding broadcast addresses. Receivers then request and receive media from the streaming server, as shown by arrow 4.

In this approach we don't need to break the feed into smaller clips. Send-

ers use the encoder to stream the media to the server. The server then streams the media to receivers. The disadvantages of this approach are:

1. The architecture is heavy as it involves the use of costly and complex streaming servers, players, and encoders
2. It's not an open architecture. The architecture becomes specific to one particular system such as RealSystem, which makes it nonportable with solutions from other vendors.
3. Capture programs, a.k.a. encoders, are not readily available and are not browser based.
4. It uses special streaming protocols such as RTP or RTSP, which are not allowed through a firewall.

### Conclusion

The approach presented here offers a near real-time, low-cost Web conferencing solution. It allows multiple users to broadcast and receive media streams, it uses the HTTP protocol, and does not require the broadcaster and receiver to have public IP addresses. The source code for this article is available at [www.indent.org/jdj-jmf/](http://www.indent.org/jdj-jmf/).

### References

- *Java Media Framework (JMF)*: <http://java.sun.com/products/java-media/jmf/>
- *JMF RTP Support*: [http://java.sun.com/products/java-media/jmf/ 2.1.1/support-rtp.html](http://java.sun.com/products/java-media/jmf/2.1.1/support-rtp.html)
- Mack, S. (2002). *Streaming Media Bible*. Hungry Minds.

#### Listing 1: JMF Capture Applet

```
Step-1
videoDataSource = javax.media.Manager.createDataSource(video
oMediaLocator);
Step-2
videoOutputFormat[0] = new VideoFormat(VideoFormat.H263, new
Dimension(160,120), Format.NOT_SPECIFIED, null, 15);
outputType = new FileTypeDescriptor(FileTypeDescriptor.
QUICKTIME);
ProcessorModel processorModel = new ProcessorModel(videoDat
aSource, videoOutputFormat, outputType);
Step-3
videoProcessor = Manager.createRealizedProcessor(processor
Model);
if (videoProcessor != null) videoDataSource = videoProces-
sor.getDataOutput();
Step-4
MediaLocator dest = new MediaLocator("file://" + file);
DataSink filewriter = Manager.createDataSink(videoDataSourc
e, dest);
filewriter.open();
Step-5
filewriter.start();
videoProcessor.start();
```

#### Listing 2: File Upload Thread

```
Step-1
URL url = new URL("http://localhost:80/jmf/parserUpload.
jsp");
Socket sock = new Socket(url.getHost(), (-1 == url.get-
Port()) ? 80 : url.getPort());
Step-2
header.append("POST ");
header.append(url.getPath());
header.append(" HTTP/1.0\r\n");
header.append("Content-type: multipart/form-data; bound-
ary=");
header.append(boundary.substring(2, boundary.length()) +
"\r\n");
header.append("Content-length: ");
header.append(contentLength);
header.append("\r\n");
header.append("\r\n");
Step-3
dataout = new DataOutputStream(new BufferedOutputStream(soc
k.getOutputStream());
datain = new BufferedReader(new InputStreamReader(sock.get-
InputStream());
Step-4
dataout.writeBytes(header.toString());
uploadFileStream(files, dataout);
dataout.writeBytes(tail.toString());
```

#### Listing 3: JMF Player Applet

```
Step-1
URL mediaURL1 = new URL(videoDir + mediaFile + counter +
".mov");
counter++;
URL mediaURL2 = new URL(videoDir + mediaFile + counter +
".mov");
Player player1 = Manager.createPlayer(mediaURL1);
Player player2 = Manager.createPlayer(mediaURL2);
Step-2
player1.start();
player2.fetch();
Step-3
player2.start();
add(player2.getVisualComponent());
if (visualComp!=null) remove(visualComp);
visualComp = player2.getVisualComponent();
counter++;
URL mediaURL = new URL(videoDir + mediaFile + counter +
".mov");
player = Manager.createRealizedPlayer(mediaURL);
player.addControllerListener(this);
player.prefetch();
```

#### Listing 4: Sender HTML code

```
<html> <head>
<body>
    <applet code=SaveVideoApplet.class width=320
height=280>
        <param name=archive value="jmf.jar">
        <param name=counter value="0">
        <param name=uploadurl value="http://localhost/jmf/
upload.jsp">
        <!-- URL of the Upload Servlet -->
        <param name=sender value="test">
        <!-- sender id -->
    </applet>
</body></html>
```

#### Listing 5: Receiver HTML code

```
<html> <body>
<applet code=PlayerApplet.class width=320 height=280>
<param name=archive value="jmf.jar">
<param name=counter value="0"> <!-- counter at the
time of request -->
<param name=sender value="test"> <!-- sender id -->
<param name=rate value="1.0"> <!-- rate at which
player sees the video -->
<param name=videodir value="http://localhost/jmf/vid-
eos/"> <!-- server directory; all clips from sender will be
at http://localhost/jmf/videos/ -->
</applet>
</body></html>
```



MILLIONS OF LINES OF CODE  
THOUSANDS OF DEVELOPERS TRAINED  
HUNDREDS OF ENTERPRISE CLIENTS  
TENS OF TOP-RANKED INSTRUCTORS  
ONE COMPANY TO CALL

**SO YOU THINK YOU'RE A GOOD DEVELOPER?  
THAT'S COOL. BUT WHY STOP AT GOOD,  
WHEN YOU CAN TAKE IT TO THE NEXT LEVEL?**

Our passion is leading-edge technical skills transfer. Students tell us we're so good at our job because our instructors are not only excellent educators, they're also real-world architects with relevant enterprise experience. They're practitioners, they're authors, and they personally develop the most engaging courseware. But, we're not your typical talking heads, we provide consulting and mentoring for top-shelf technical teams. Our approach is practical, insightful, and challenging. The difference? **Results.** *Call us today at 888-211-3421 to discuss your specific requirements, or just visit us online.*



Training. Consulting. Mentoring.

[www.inferdata.com/jdjmag](http://www.inferdata.com/jdjmag)



**Calvin Austin**  
Core and Internals Editor

# A Tail of Two Tigers

I recently enjoyed reading *A Short History of Nearly Everything* by Bill Bryson. In his book, Bill goes back to basics and delves into the history of many things we take as facts. One memorable observation is a reminder that we are all just collections of trillions of atoms assembled in a unique configuration, a one off, never to be repeated again.

Given this cosmological randomness I'm at a loss to explain how we ended up with two very fine articles about JNI this month. The random skew doesn't end there. I recently presented a session at the JavaOne conference about the J2SE 5.0 release, code named "Tiger." At the same time, Apple was headlining the OS X.4 release at the WorldWide Developer Conference, barely a block away, and their release was also code named "Tiger."

However, this is where the parallel universe stops. While Apple users were bemoaning the lack of access to key information like bug reports, the Java community was treated to a cute tiger cub, new open source projects, Java3D and Project Looking Glass, and the arrival of the next update to the Java platform, now renamed J2SE 5.0.

I have attended all nine JavaOne conferences and this year's confer-

ence reminded me of some of the best. Packed with technical content and with nearly 15,000 attendees, it represented a renewal in the community. A renewal that will result in new tools, new books, and new products based on the J2SE 5.0 foundation. By the time you read this, the J2SE 5.0 JSRs will be heading into the final stages of the Java Com-



munity Process, the Final Approval Ballot. Although that marks the end of the JCP and engineering cycle, it also represents the first day of J2SE 5.0 in real deployments and ultimately the success of the platform.

You may be thinking, what's in it for me, the regular *JDJ* reader? Well, I will make two promises. First, expect the same high-quality technical articles that you have been used to. I've been reading and writing for *JDJ* for many years and wouldn't settle for anything

less. My second promise is that there will be a special focus on J2SE 5.0 throughout the year. Don't worry, there will still be room for content for older releases too.

There has been a fair amount of attention centered on the language features, like Generics, Metadata, and the enhanced for loop, to name a few. The language changes are

certainly a core part of the release; however, there are many other features that are just as useful yet not as well known. Features like performance and monitoring can be used with existing applications without changing a single line of code. Some features, like the new profiling API, will require a fair amount of porting. To help make that transition easier, I will be searching for case studies. My aim is to give you the

tools and techniques to ramp up to using J2SE 5.0. This is also your opportunity to help out fellow developers. If you are interested in writing some J2SE 5.0 material, please send your proposal to <http://grids.sys-con.com/proposal>.

Finally, I would like to thank Joe Ottinger for steering the *JDJ* ship for the past year. Joe suggested I use my J2SE 5.0 experience to guide the core section this year and I hope to make good on that suggestion. Let us make this a year to remember. ☺

“By the time you read this, the J2SE 5.0 JSRs will be heading into the final stages of the Java Community Process, the Final Approval Ballot”

A co-editor of *JDJ* since June 2004, **Calvin Austin** is the J2SE 5.0 Specification Lead at Sun Microsystems.

He has been with Java Software since 1996 and is the Specification Lead for JSR-176, which defines the J2SE 5.0 ("Tiger") release contents.

[calvin.austin@sys-con.com](mailto:calvin.austin@sys-con.com)

# Javavavoom!

**Introducing a high-performance database that's 100% Java.**

**Berkeley DB Java Edition**

Download at [www.sleepycat.com/bdbje](http://www.sleepycat.com/bdbje)

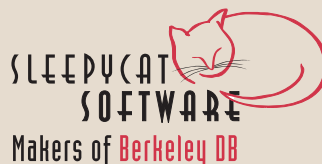
Finally there's a high-performance database that loves Java just as much as you do: Berkeley DB Java Edition (JE). Brought to you by the makers of the ubiquitous Berkeley DB, Berkeley DB JE has been written entirely in Java from the ground up and is tailor-made for today's demanding enterprise and service provider applications.



Berkeley DB JE has a unique architecture that's built for speed. The software executes in the JVM of your application, with no runtime data translation or mapping required. Plus Berkeley DB JE has been specifically designed to handle highly concurrent transactions, comfortably managing gigabytes of data. And because it's built in your language of choice, your organization enjoys shorter development cycles and accelerated time-to-market.

**Experience the outstanding performance of Berkeley DB JE for yourself.**

Download Berkeley DB JE today at [www.sleepycat.com/bdbje](http://www.sleepycat.com/bdbje). Register now, and you'll also receive a 15% discount on a commercial license purchased before November 30, 2004.



# USING APACHE CACTUS

## Testing server-side components

by Kishore Kumar

**A** *pache Cactus is part of the Jakarta project and is an open source framework for unit testing server-side Java code. It uses and extends the JUnit framework and facilitates unit testing of servlets, JSPs, Taglibs, EJBs, and filters.*

Testing server-side components is more complicated than testing client-side code because these components interact with a container and require access to many container-managed objects such as request and session. It's possible to make a mock-up of all the container-managed objects and test the components. These mock objects provide a "clean" environment for testing that is totally isolated from the container. The other approach is to use an in-container strategy. Using this, the test code runs on a real (not mock) container and uses real container-managed objects. Both approaches have their advantages and disadvantages. Cactus is based on the in-container testing and our discussion will focus on this approach.

I'll explore how to use the Cactus framework to write JUnit-based test classes for testing server-side components.

### Understanding Cactus – How It Works

Cactus tests are organized into Cactus TestCase classes. You can subclass and implement any of the three provided Cactus TestCase classes: ServletTestCase, JspTestCase, and FilterTestCase. Figure 1 explains the overall system.

Here, XXX is the name of the test. Unlike a JUnit test, the Cactus test runs in two different environments: client-side and server-side. The class-under-test is a server component like a servlet or a JSP. The following are the different steps that occur when a Cactus test (testXXX) is run:

1. The JUnit TestRunner executes the TestCase method runTest. If defined, the method beginXXX is executed. This method may be implemented to initialize a Web request (HTTP parameters and headers) to the server-side component-under-test.
2. Cactus opens an HTTP connection to a Cactus redirector proxy. All the parameters set up in step 1 are sent in the HTTP request.
3. The redirector acts like a proxy on the server-side for your tests. It creates a new instance of the TestCase class and executes the test. The TestCase class is instantiated twice: once on the client-side (by the TestRunner) and once on the server-side (by

the redirector proxy). The client-side instance is used to run the method beginXXX and endXXX and the server-side instance is used to run the test methods. The redirector also initializes the TestCase instance with server-side implicit objects (HttpServletRequest, HttpServletResponse, ServletContext,...) which are made available to the test methods.

4. The setUp method is executed. If required, implement this method to define a test fixture.
5. The redirector executes the test method (testXXX).
6. The test method usually instantiates the component-under-test and invokes the methods that need to be tested. It uses JUnit assert API (assertEquals, assert,...) to verify the result.
7. The tearDown method is executed. If required, implement this method to do clean up.
8. If the test fails, the redirector proxy handles the exception thrown from testXXX.
9. If an exception has been raised, the proxy returns the exception information back to the client side.
10. If no exception has occurred, the method endXXX(org.apache.cactus.WebResponse) or endXXX(com.meterware.httpunit.WebResponse)\* is executed if it is defined. This method may be implemented to verify the response from the server-side component.

\* This signature is used for HttpUnit integration

### Cactus Redirectors

Cactus provides three redirectors: ServletRedirector, JspRedirector, and FilterRedirector. Cactus TestCase uses a corresponding redirector implementation (for example, ServletTestCase uses ServletRedirector). The implicit objects that are created and initialized in a Cactus test instance depend on the specific redirector. A servlet redirector initializes a servlet test case with servlet API objects. On the other hand, a JSP redirector initializes a JSP test case instance with JSP API objects.

### Writing a Cactus Test

To write a Cactus test, complete the following steps:

1. Implement a subclass of a Cactus TestCase implementation. Subclass the ServletTestCase class if your component-under-test is a servlet or subclass the JspTestCase class if your component-under-test uses JSP API objects. If your component-under-test is a servlet filter, extend your test from FilterTestCase.



**Kishore Kumar** works as a Java architect at US Technology (www.ustricom). He specializes in J2EE applications.

kishore\_kumar@usswi.com

```
public class TestSampleServlet extends ServletTestCase
{
}
```

2. Implement standard JUnit methods. As in a normal JUnit test, define the following JUnit methods in your test case class:

- A constructor with a single string parameter, which is the test name that needs to be executed when the test is run.
- A method suite to collect the test into a JUnit TestSuite object. Running the TestSuite will run all contained tests. A convenient TestSuite constructor can create a suite object that contains test case instances for every method starting with "test" in a given class.

```
public TestSampleServlet(String testName)
{
    super(testName);
}
public static Test suite()
{
    return new TestSuite(TestSampleServlet.class);
}
```

- Override the method setUp to initialize a test fixture and the method tearDown to clean up the fixture. These are executed at the server side and all the server-side implicit objects are available to these methods.

3. Implement the testXXX method. In the test method, you will:

- Instantiate the component-under-test. Since the test case extends Cactus TestCase, server-side implicit objects are defined and initialized with valid values. These are available to the test methods through TestCase instance members.
- Call the method to be tested.
- Perform JUnit standard asserts (assertTrue, assert, ...) to verify the result.

```
public void testXXX()
{
    SampleServlet servlet=new SampleServlet();
    // session is an implicit object defined in cactus TestCase class
    session.setAttribute("name","value");
    String result=servlet.doSomething(request);
    assertEquals("some value",result);
}
```

4. Implement the method beginXXX to initialize the HTTP request to the server.

5. Implement the method endXXX to verify the HTTP response from the server.

### Testing Servlets

You need to subclass and implement a ServletTestCase to test a servlet. The ServletTestCase provides the following implicit objects: request (HttpServletRequest), response (HttpServletResponse), session (HttpSession), and config (ServletConfig). The ServletTestCase uses the ServletRedirector as the proxy to servlet tests.

```
public void beginXXX(WebRequest request)
{
    request.addParameter("param1","value");
}
```

```
}
public void testXXX()
{
    ServletToTest s=new ServletToTest();
    s.init(config);
    s.methodToTest();
    assertEquals("some value", session.getAttribute("result"));
}
public void endXXX(WebResponse response)
{
    Cookie cookie=response.getCookie("someCookie");
    assertEquals("some value",cookie.getvalue());
}
```

### Testing JSPs

Testing JSPs covers the following: verifying the result of JSP processing (HTML) and unit testing JSP tag libraries. Unit testing tag libraries are discussed later.

You can still have your test case class extend from ServletTestCase if your test does not use any of the JSP API objects (like PageContext).

The Web response can be easily verified by implementing the method endXXX in a ServletTestCase subclass.

```
public class SimpleTest extends ServletTestCase
{
    [...]
    public void testXXX()
    {
        RequestDispatcher rd=config.getServletContext().
            getRequestDispatcher("test.jsp");
        rd.forward(request,response);
    }
}
public void endXXX(org.apache.cactus.WebResponse webResponse)
{
    // Assert Result
    [...]
}
```

Cactus also integrates HttpUnit into the framework. The HttpUnit implementation of the WebResponse object (com.meterware.httpunit.WebResponse theResponse) can be used to verify the HTML response.

```
public void endXXX(com.meterware.httpunit.WebResponse theResponse)
{
    WebTable table = theResponse.getTables()[0];
    assertEquals("rows", 4, table.getRowCount());
    assertEquals("columns", 3, table.getColumnCount());
    assertEquals("links", 1, table.getTableCell(0, 2).getLinks().length);
}
```

### Testing JSP Tag Libraries

For testing JSP Tag libraries, extend your test case class from JspTestCase. In addition to the servlet implicit objects, JspTestCase provides the following implicit objects: out (JspWriter) and pageContext (PageContext). These implicit

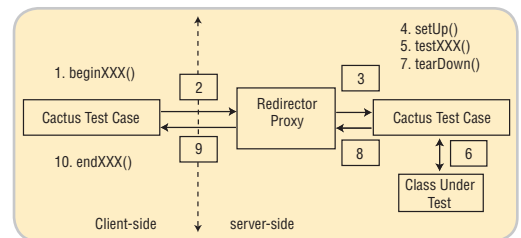


Figure 1 Cactus System

objects are made available to the `setUp`, `tearDown`, and `textXXX` methods as instance variables of the `JspTestCase` class.

To test the tag handler, use the implicit objects provided by the `JspTestCase` to set up initial state for the test. Then create and initialize your custom tag using the `pageContext` implicit object. After setting up the tag, call the tag life-cycle methods in the correct order and verify the results. The tag's output can be inspected in the `endXXX` method.

Complete the following to set up the custom tag for testing:

1. **Create the custom tag and initialize it with the `pageContext` implicit object:**

```
MyTag tag=new MyTag();
// pageContext is available as an implicit object to JspTestCase
tag.setPageContext(pageContext);
```

2. **Set the tag's attributes:**

```
tag.setNum1("10");
tag.setNum2("11");
```

3. **Set the parent tag (optional):**

```
Tag.setParent(enclosingTag);
```

The "enclosingTag" will have to be instantiated and set up as well. This will allow the tag to successfully call the method `getParent`.

4. **Create the `BodyContent` object (optional):** If the tag processes its body, call `pageContext.pushBody()` to obtain a `BodyContent` and the corresponding `pageContext.popBody()` after the tag completes execution.
5. **Set up `page state` (optional):** Set up appropriate objects into the request or `pageContext` for use by the tag.

Once the tag has been set up, test the tag by calling its relevant life-cycle methods and using JUnit assert API to verify the results.

#### Verifying Individual Methods

You can verify a tag that conditionally includes its body based on some values:

```
tag.setValueThatResultsInIncludingBodyContent("Correct Value");
assertEquals(Tag.EVAL_BODY_INCLUDE,tag.doStartTag());
```

#### Verifying Tag Output

The custom tag output can easily be verified in the `endXXX` method of the test case.

#### Testing Iteration Tags

You can test a tag that repeats its body output a number of times as shown:

```
// [...] set up tag state
int count=0;
do
{
    count++;
} while (tag.doAfterBody() == Tag.EVAL_BODY_AGAIN);
assertEquals(EXPECTED_RESULT,count);
```

#### Testing Body Tags

For testing tags with body content you must replicate the life cycle of the tag in your test code. Use the `pageContext` implicit object to obtain and release a `BodyContent` object:

```
tag.setPageContext(pageContext);
tag.doStartTag();
// if the doStartTag method return EVAL_BODY_TAG
BodyContent bodyContent=pageContext.pushBody();
tag.setBodyContent(bodyContent);
tag.doInitBody();

bodyContent.println("Sample content");

tag.doAfterBody();
tag.doEndTag();

pageContext.popBody();
```

Implement the `endXXX` method to verify whether the tag returns the expected body content or not.

#### Testing Filter

Your test case class should extend from `FilterTestCase` when you want to test servlet filters. Cactus automatically provides the implicit object `filterChain` (`FilterChain`), in addition to all the servlet implicit objects, to the `setUp`, `testXXX`, and `tearDown` methods. In your test method, do the following:

1. Instantiate the `Filter` class.
2. Set up the required request parameters.
3. To simulate the next filter in the filter chain, define a mock filter chain inner class.
4. Invoke the `doFilter` method. Pass the mock filter chain object as a parameter to `doFilter` if you need to verify if the filter is returning to the correct filter in the chain.

#### Testing EJB

EJBs can be tested from any of the Cactus redirectors. From the `testXXX` method, obtain the home reference to your EJB, create an instance of it, invoke the method to test, and finally assert the result.

## Running Cactus Tests

Cactus provides a `ServletTestRunner` to run the Cactus tests using a browser. In addition to the Cactus redirectors, you'll also need to map this servlet in the `web.xml` file of your Web application. Once the Web application is deployed, you can run your application using the URL `http://server:port/webapp/ServletTestRunner?suite=SimpleTestServlet`. This assumes that the class `SimpleServletTest` has a static method `suite` that will provide the test runner with a `TestSuite` that it can run. The test runner will return the test results as XML data to the browser.

## Summary

This article explored the concepts of using the Cactus framework to write unit tests for testing servlets, JSPs, filters, TagLibs, and EJBs. ☺

## References

- *Apache Cactus*: <http://jakarta.apache.org/cactus>
- *JUnit Framework*: <http://junit.org/index.htm>



Shrink my **development** time.  
Give me the technology to **deliver** it  
**Faster. Better. Easier.**



**Faster. Outsmart your development deadlines with AMD64 technology.**

**Better. Direct Connect Architecture lets you do more.**

**Easier. Your platform choice is simpler, since AMD64 technology excels across a wide variety of application workloads.**

**Register at [developer.amd.com](http://developer.amd.com) and enter a drawing for a chance to win an AMD64 system. See official rules for details and eligibility requirements.**

# Dynamic Sorting with Java

## A reusable implementation

by York Davis

Those familiar with the `java.util.Comparator` interface of the Java API realize its capabilities for sorting a collection of objects based on an attribute of the objects in the collection. This works well when there is only a single field in which sorting is required. When more complex sorting is necessary, the limitations of sorting on a single field become obvious. What about the situation in which a user desires the functionality to sort selectively on any field in object collection? This article describes an implementation of the `Comparator` interface that along with the reflection API allows an object to be sorted dynamically on any of its publicly accessible fields.

### Problem Statement

Let's describe the problem a bit more specifically. A collection of employee Transfer Objects (`EmployeeTO` class) exists. (For a description of the Transfer Object design pattern consult a software design pattern book.) Each `EmployeeTO` in the collection is a data container object for a single employee's information. For this example, our simplified `EmployeeTO` object contains only three pieces of data – employee ID, last name, and salary.

A Human Resources application also exists that uses this collection to display a list of all employee data to HR application users. The users of this system have the following requirements:

1. Allow sorting on any field on the report
2. Allow control of the sort order

### Simple Solution

Before delving into our dynamic sorting solution that allows sorting on any attribute, let's first look at a simple solution that supports sorting on a single attribute only. This will demonstrate the basic behaviors of `Comparator` and from this we'll be able to glean the improvements we wish to make. This solution utilizes the more common use of the `Comparator` interface. There are two classes required to implement this.

### EmployeeTO – Simple Version

First, our `EmployeeTO` can be made sortable by implementing the `Comparator` interface as shown in Listing 1. In addition to the getters and setters for ID, last name, and salary, `EmployeeTO` must implement the `compare()` and `equals()` methods in order to meet the `Comparator`'s requirements. These methods define how `EmployeeTO` is to be sorted. The `compare()` method takes two parameters, both of type `Object`. Note that `compare()` returns an `int`. This return value tells the sort engine the collating sequence equality of two attribute values from each of the object parameters passed to `compare()`, respectively. We need to write the code that performs this evaluation. The first step in `compare()` is to cast the two parameters' objects to `EmployeeTO` objects and extract the employee IDs by calling the `getId()` method on each object in turn. Now we can compare the values. There are really only three possible outcomes that can result from this comparison. Table 1 describes the results based on these outcomes.

The other method we must code is the `equals()` method. Although this method is not used for sorting, it must be implemented in order to meet the contract of the `Comparator` interface.

### Sorting the Collection – Simple Version

The `SimpleTest` class in Listing 2 adds three `EmployeeTO` objects to a `List`, then performs a sort on that `Collection`. (Listings 2–5 can be downloaded from [www.sys-con.com/java/sourcec.cfm](http://www.sys-con.com/java/sourcec.cfm).) Line 30 of `SimpleTest` calls the static `sort()` method of the `Collections` class to actually perform the sort as follows.

```
Collections.sort(elements, new
EmployeeTO());
```

The first parameter passed to `sort()` is the collection object we wish to have sorted. The second parameter is an object of type `Comparator` that contains the customized sorting logic – in this case `EmployeeTO`, which implements the `Comparator` interface. We certainly could have passed any instance of `EmployeeTO` as the second parameter to the above method call. However, instead of reusing one of the three values initially added to the collection, I chose to pass a new instance of the class for purposes of clarity.

A quick note on encapsulation and responsibility assigning seems to be in order here. In this case, it makes sense to encapsulate the specific `compare()` method sorting logic within `EmployeeTO`. With the information we have thus far, `EmployeeTO` is the only class that requires the knowledge of how it should be sorted. Later, as a dynamic sorting solution is provided, we'll see this sorting logic moved out of the `Transfer Object` class as the sorting logic becomes less specific to any one particular `Transfer Object` implementation.

This implementation will run but falls short when it comes to meeting the user's requirements. Remember, we need to be able to sort on any one of the three fields in `EmployeeTO` based on a user's choice. And let's not forget about the ability to control the sort order.

### Enhancement Options

Let's think about the options that are available to improve what we have and meet the requirements. One option is to code nested `if/else` statements in our `compare()` method to allow for sorting on any field in the object based on some field name parameter passed to `EmployeeTO`. The problem with this solution is that it's difficult to maintain

Comparison	Return Value
ID from Object 1's collation sequence equal to ID from Object 2's collation sequence	0
ID from Object 1's collation sequence less than ID from Object 2's collation sequence	-1
ID from Object 1's collation sequence greater than ID from Object 2's collation sequence	1

Table 1 Return values from `compare()` method



York Davis is a senior managing consultant at Software Architects, Inc. With more than 12 years in the software development field, York has been using Java for more than five years and has extensive experience building enterprise application architectures.

[ydavis@sark.com](mailto:ydavis@sark.com)



and the code could get rather lengthy as well. If new fields are added to EmployeeTO, we must update compare() appropriately.

What we would really like to do is invoke any given getter method of our Transfer Object at runtime without having to specifically hard code each possible method call in the compare() method. If we could do this, we could use the results of those method calls to dynamically determine equality. In addition, it would be desirable if this dynamic sorting could be reused for any Transfer Object. The good news is that this functionality can be achieved by leveraging the reflection API and the flexibility of the Collections.sort() method.

### Reflection

The java.lang.reflect.Method class provides the ability to invoke a method of a given object based on the value of a string. For example, using a string containing the value "getId", the method getId() of EmployeeTO can be dynamically invoked at runtime. This string value can then be changed as we wish to cause any of the methods of EmployeeTO to be called.

The reflection API provides a variety of interesting features including the ability to pass parameters to methods and the ability to determine method return types. (For a complete list of these capabilities, consult the Java API Javadoc.) We will need the latter capability as the three getter methods of EmployeeTO return different types and it will thus be necessary to be aware of which type we have when doing the comparison in the compare() method. For example, we'll need to code a different sort of equality test on an int return value as opposed to a string.

### java.util.Collections.sort()

A dynamic solution will also need to take advantage of the flexibility of the Collections.sort() method. Recall that in our simple sorting solution we passed an instance of an object that implemented the Comparator interface as the second parameter to Collections.sort(). In that case, it was the EmployeeTO object that contained the sorting logic. This worked great for what we needed it to do. Now, however, we want something a bit more sophisticated and flexible. What if we were to create a class that implemented Comparator, which was separate from each Transfer Object? In it we could place our dynamic sorting

code and simply pass an instance of this new class as the second parameter to Collections.sort().

Doing this would cause several desirable results. First, we'll have completely decoupled any sorting logic from our Transfer Objects. This is highly desirable as it decreases not only the size of each Transfer Object by essentially eliminating the need for sorting code, but also removes the need for coding individual field comparison logic. Second, we'll have created a reusable utility class that can be used in many different situations where sorting is required. While reusability is not a specifically stated user requirement for our design, it is certainly desirable.

### Dynamic Solution

Figure 1 is a UML diagram (with attribute and method details omitted) that shows how the components of both the simple and the dynamic sorting solutions fit together. Really, the only portion that has changed since our simple solution is where the Comparator interface gets implemented. In the first example, EmployeeTO implemented Comparator directly. Now DynamicComparator implements Comparator and contains an intelligent implementation of the compare() method that can be used by any class that requires a collection of objects to be sorted.

### DynamicComparator

Listing 3 shows the completed code listing for DynamicComparator. There are a number of interesting things about this class. First, note the class signature. The class implements two interfaces – Comparator and Serializable. Comparator should come as no surprise since that interface is the essence of the sorting capabilities we desire. Also, although not a requirement, the API docs recommend that any class implementing Comparator implement Serializable as well.

Second, look at the static sort() method. Classes wishing to utilize DynamicComparator will call this method rather than Collections.sort(). I've decided to make sort() static to mimic the Collections.sort() method. Although the DynamicComparator.sort() method is called statically, internally DynamicComparator creates an instance of itself. This is necessary in order for it to provide access to the nonstatic compare() and equals() methods of the Comparator interface that it supports.

Next note how this method takes three parameters. The first is the collection object to be sorted. The second is a string that defines the field of each object in the collection on which sorting should be performed. The last parameter is a Boolean specifying the sort order. These three parameters are used as arguments to create a new instance of DynamicComparator that is the second parameter passed to the Collections.sort() method.

Third, take a look at the compare() method. This is where the reflection code really kicks in. One of the first things we need to obtain is a reference to a method object at line 43. We'll use this reference to call methods dynamically. This task is done using the getMethod() helper method, which obtains this value via reflection. Next, we need to determine the return type of the methods we are about to call. Remember that we will need to compare the attribute values of the two objects passed into compare(). If the return type is an int, for instance, we'll certainly have to write different code to do the comparison than if the return type is a string. Once we have the return type, we examine it and, based on its value, dynamically perform the actual method invocation and resulting comparison of the two values.

Note that currently there are three separate "if" test blocks – one each for



Figure 1 UML diagrams for simple and dynamic sorting scenarios

string, int, and double. The implementation requires comparison logic for any method return type we expect to encounter. For the range of return types in our EmployeeTO example, these three are sufficient. However, additional code would need to be added to DynamicComparator if comparisons of other types are required – short, java.util.Date, java.math.BigDecimal, etc. Coding for each specific return type here is unavoidable as there is no way to dynamically cast Java objects. Similarly, Java-supplied nonobject data types like int, long, and double use entirely different comparison operators than do first-class object types like String or Date.

There are some other important points about this code. First, DynamicComparator fully supports null values. If either or both of the arguments passed to compare() are null, this method knows how to handle the situation accordingly. Second, look at each return statement within compare(). Remember the requirement that the user be able to control not only the sort field but also the sort order? This code supports the latter by essentially reversing the default sort order with a call to getSortOrder(). This is done if the user has decided to sort the result in descending order based on the Boolean value passed into the constructor from the sort() method. Third, the constructMethodName() method converts a Transfer Object attribute name string into a method name by prepending a “get” string and

capitalizing the first character of the passed value. For instance, constructMethodName() would convert “salary” to “getSalary”.

Last, the equals() method is needed to complete the interface requirements.

### EmployeeTO – Enhanced Version

Listing 4 shows the enhanced version of EmployeeTO. The most obvious change is that EmployeeTO is now even simpler than before. Now that all of the sorting logic has been moved to DynamicComparator and the class no longer implements Comparator, we don’t need to implement the compare() and equals() methods.

### Sorting the Collection – Dynamic Version

Listing 5 is the code for DynamicTest. The only change between this class and SimpleTest is how the sort is called. Here we pass the three parameters to DynamicComparator.sort() (Collection Object, the decapitalized attribute name, and sort ascending flag) and let the DynamicComparator do the rest.

DynamicTest could just as easily have sorted on last name by passing “lastName” or on employee ID by passing “id” as the second parameter on line 29.

### Solution Discussion

Building a class such as DynamicComparator has many benefits in an application that requires robust sorting capabilities. In this design, we have

created a reusable, loosely coupled API that can be used to sort a collection of objects based on getter methods. The sort field is easily configurable and also allows control over the sort order.

This design, however, is not without trade-offs. Although using the reflection API allows us to do lots of cool things, using reflection can slow performance. This is particularly true in applications using pre-1.4 versions of Java. In addition, it’s possible that applications wishing to sort very large collections may find DynamicComparator too slow.

Other inadequacies of DynamicComparator might become evident as well. Although it allows sorting on any one attribute of a collection of objects in an easily configurable manner, DynamicComparator does not address the potential need to sort by multiple fields – primary and secondary field sorts like that occur automatically with the ORDER BY clause in Structured Query Language (SQL).

### Conclusion

This article introduced a reusable implementation of the Comparator interface that utilizes Java reflection to dynamically sort a collection of objects on any one of any number of fields within that object.

If your application or framework has a need for this specific functionality, perhaps this design will fit your needs. ☺

#### Listing 1

```
package simple;

import java.util.Comparator;
import java.io.Serializable;

public class EmployeeTO implements Comparator, Serializable {
    private int id;
    private String lastName;
    private double salary;

    public int getId() {
        return id;
    }

    public String getLastName() {
        return lastName;
    }

    public double getSalary() {
        return salary;
    }

    public void setLastName(String string) {
        lastName = string;
    }

    public void setSalary(double d) {
        salary = d;
    }
}
```

```
public void setId(int i) {
    id = i;
}

public int compare(Object o1, Object o2) {
    EmployeeTO emp1 = (EmployeeTO) o1;
    EmployeeTO emp2 = (EmployeeTO) o2;

    int id1 = emp1.getId();
    int id2 = emp2.getId();

    if (id1 == id2) return 0;
    if (id1 < id2) return -1;
    if (id1 > id2) return 1;

    return 0;
}

public boolean equals(Object o) {
    return true;
}

public String toString() {
    return "id="+ id +
        " lastname="+lastName+
        " salary="+salary;
}
}
```

Compuware

# OptimalJ<sup>®</sup>



## THE POWER TO Develop, Transform, Reuse

Put your J2EE™ application development into overdrive—with unmatched quality and unprecedented flexibility—using Compuware OptimalJ. Increase developer productivity up to 90%. Seamlessly instill quality into development. Forge existing infrastructures and new technologies into an integrated whole.

Realize business agility with applications that are truly reusable, time and time again.

**THE POWER IS RIGHT HERE.**

THE LEADER IN IT VALUE.

**COMPUWARE**<sup>®</sup>

[www.compuware.com](http://www.compuware.com)



# Calling Java from C

*A framework for easier JNI*

by Michael Havey

Though most Java developers think of the Java Native Interface (JNI) as a framework for developing native libraries that can be called from Java, relatively few know that JNI also supports communication in the reverse direction: it provides native programs written in C with the ability to call Java objects. However, the coding is thorny; logic that can be coded readily in a few lines of Java requires several times more lines of C, thanks to JNI's granular programming model and peculiar approaches to exception handling and garbage collection. This article explores the nature and typical use of the C-to-Java JNI interface and presents the design of a framework that eases the programming effort.

## The JNI Architecture

As Figure 1 illustrates, JNI is actually a pair of APIs:

- “JNI Proper” supports the manipulation of Java objects and classes, such as the ability to call object methods.
- The Invocation API is a smaller C library that enables C programs to create and destroy a Java Virtual Machine (JVM).

A C-to-Java program (that is, a C program that uses Java) calls the Invocation API to create a JVM, and calls JNI Proper to use Java objects. As for the Java-to-C direction (not discussed in this article), Java code calls a native method, which is implemented as a C native library function; the C code uses JNI to interpret its Java input types and build its Java output types.

The JVM is packaged as a shared library (“jvm.dll” in the Sun SDK on Windows platforms and “libjvm.so” on Solaris) and exposes JNI Proper and the Invocation API as public exports. As a runtime entity, a JVM is really just the JVM library linked to an executable C program. Any Java developer can code

such a program, e.g., “myCtoJProgram.exe”, shown in Figure 2. Interestingly, the famed SDK C program known as the “launcher” (java.exe on Windows, Java on Solaris) is written in just the same way (for more, see sidebar **JNI Case Study: Java Launcher**). Completing the picture are native libraries, such as “myNative.dll”, whose functions can be called from Java; these libraries are linked to the runtime process alongside the JVM.

## A C-to-Java JNI Design: The Zip Example

The launcher is the best-known example of a C program that uses the JNI's C-to-Java interface; its purpose is to house a JVM and bootstrap a Java application on the JVM by calling the application's main method. Other, less obvious examples are C programs that require functionality whose best or only implementation is Java objects that must be called “in process.”

Step	Description	JNI Usage
1	Initialize the JVM	JNI_CreateJavaVM(), preceded by several lines of code to set up JVM arguments.
2	Get references to classes ZipFile, ZipEntry and Enumeration	FindClass(). Remember JVM expects package names using slashes instead of dots (e.g., java/util/Enumeration rather than java.util.Enumeration).
3	Get references to several methods: ZipFile constructor, ZipFile.entries(), ZipEntry.getName(), Enumeration.hasMoreElements(), Enumeration.nextElement().	GetMethodID(). Method signatures are expressed in an unusual notation understood by the JVM (e.g., “()Z” means “returns a boolean”).
4	Instantiate ZipFile	NewStringUTF() to convert C string to Java string, NewObject() to instantiate, NewGlobalRef() and DeleteLocalRef() to get global reference to zip file object.
5	Call ZipFile.entries()	CallObjectMethod() to call the method, NewGlobalRef() and DeleteLocalRef() to get global reference to enumeration object.
6	In loop, exit when Enumeration.hasMoreElements() returns false.	CallBooleanMethod()
7	Get next element, expecting ZipEntry object	CallObjectMethod()
8	Call ZipEntry.getName()	CallObjectMethod() to call the method, GetStringUTFChars() and ReleaseStringUTFChars() to convert Java string to C string.
9	Cleanup	FreeGlobalRef() to release ZipFile and ZipEntry objects, DestroyCurrentThread() and DestroyJavaVM() to destroy JVM connection.

**Table 1** Summary of C code to list contents of a zip file



**Michael Havey**

is a BEA consultant with nine years of industry experience, mostly with application integration.

mhavey@bea.com

Examples include programs that:

- *Create, extract, or list the contents of zip files.* The Java SDK `java.util.zip` package is the most suitable API available.
- *Transform XML to XML, HTML, or PDF.* Though C XML APIs exist, Java's support for XML is vastly superior. Launching a JVM to do XML with Java is a plausible strategy for a C program.
- *Call an Enterprise JavaBean (EJB).* The C program uses JNI to execute standard EJB client-side logic.

The example of listing the contents of a zip file highlights the coding challenge of C-to-Java JNI. (Source code for this article can be downloaded from [www.sys-con.com/java/sourcec.cfm](http://www.sys-con.com/java/sourcec.cfm).) The Java code to perform this logic, shown in Listing 1, is trivial: line 4 instantiates the class `ZipFile` in `java.util.zip`, passing the zip file name to the constructor; lines 5–9 loop over a `java.util.Enumeration` of `java.util.zip.ZipEntry` objects getting, in line 8, the name of each entry in the zip file.

Developing the equivalent logic in C requires hundreds of lines of code. The main steps are summarized in Table 1.

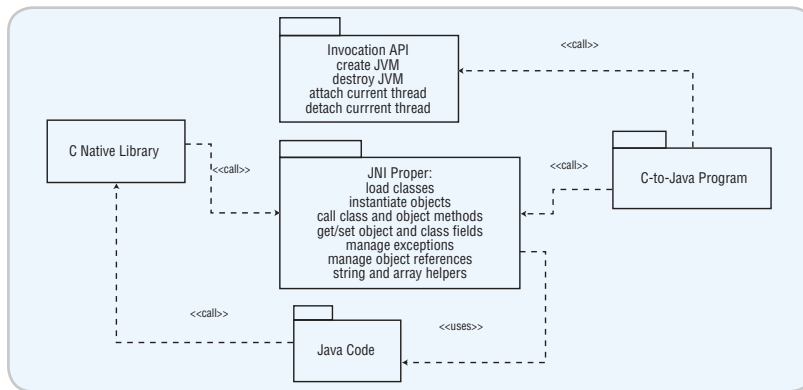


Figure 1 C to Java, Java to C with JNI

Listing 2 shows the code for steps 7 and 8; the JNI calls in lines 5, 10, 15, and 21 are followed by calls in lines 7, 11, 16, and 22 to the `checkException()` function (implementation not shown), which in turn calls the JNI exception handling functions `ExceptionCheck()`, `ExceptionDescribe()`, and `ExceptionClear()` to swallow and report Java exceptions triggered by the JNI calls.

The C code in the zip example can be made easier and less cumbersome by using two design patterns:

1. **Java Proxy:** Put the hard code where it belongs, on the Java side. Develop a Java object, called a proxy, that per-

forms complex logic on behalf of the C code. The C code need only call the proxy.

2. **C Façade:** Hide JNI's peculiar programming model in a C façade library. Have the C program call the façade rather than JNI directly. In addition, build proxy support into the façade; expose façade functions to call the proxy.

The proxy and façade constitute an abstract framework for use in any program resembling the zip program. Java proxies implement the interface shown in Listing 3; the `execute()`



## Went to find himself.

Left you 300K lines of Java linked to 225K lines of C++ using macros to look like Pascal, and a Linux box you can't boot.

[www.scitools.com](http://www.scitools.com)

Tools that help you understand and maintain impossibly large bodies of source code.

method is defined generically as accepting an input, performing some action or set of actions, and returning an output. Listing 4 shows the proxy implementation for the zip example. The execute() method expects as input a string specifying the name of the zip file (line 26); it implements logic similar to that in Listing 1 to enumerate the entries in the zip file (lines 26–33) and returns a string containing the name of the entries in a pipe-delimited list (see lines 30–32 and 35). The method could also have returned an array or Java collection type, but the calling C program is likely happier parsing a string than contending with JNI array or collection class iteration logic.

If the C zip program were to call the proxy using JNI directly, its length would be shorter but the complexity of JNI would remain. Using the façade reduces the length even further and shields the code from JNI oddities. A design for the façade is depicted in Figure 3. The façade consists of a set of data types, modeled as C structures, and a set of functions. The data types represent entities such as JVM (cjJVM\_t), class (cjClass\_t), method (cjMethod\_t), and object (cjObject\_t). The functions are operations performed on the entities (e.g., cjJVMConnect() and cjJVMDisconnect() performed on the JVM), and a special set of proxy opera-

Method	JNI Usage	Description
CjJVMConnect	JNI_CreateJavaVM	Creates a JVM based on options specified by the caller.
CjJVMDisconnect	DetachCurrentThread, DestroyJavaVM	Destroys the JVM.
CjClassCreate	FindClass, GetMethodID	Gets a reference to a given Java class and references to each of the methods specified by the caller.
CjClassDestroy		Cleans up resources created in cjClassCreate()
CjProxyClassCreate	See cjClassCreate	Gets a reference to a Java class that implements the JavaProxy interface. The implementation calls cjClassCreate() passing the name of the class that implements the interface and the names and signatures of the init, shutdown and execute methods of JavaProxy.
CjProxyCreate	NewObject, NewGlobalRef, DeleteLocalRef	Instantiates a JavaProxy class and acquires a global reference to it.
CjProxyDestroy	See cjFreeObject	Releases global reference to the proxy created in cjProxyCreate().
CjProxyExec	CallObjectMethod, NewGlobalRef, DeleteLocalRef	Invokes the execute() method of the JavaProxy object created in cjProxyCreate(). Acquires a global reference to the object returned by the execute() method.
CjFreeObject	DeleteGlobalRef	Releases the global reference to an object.
CjProxyExecString	NewStringUTF, cjProxyExec, GetStringUTFChars, ReleaseStringUTFChars	Calls the proxy's execute() method, passing a Java string, converted from the C string passed by the caller. The execute() method returns a Java string, which the function converts to a C string and returns to the caller.

Table 2 CJ C API Functional Description

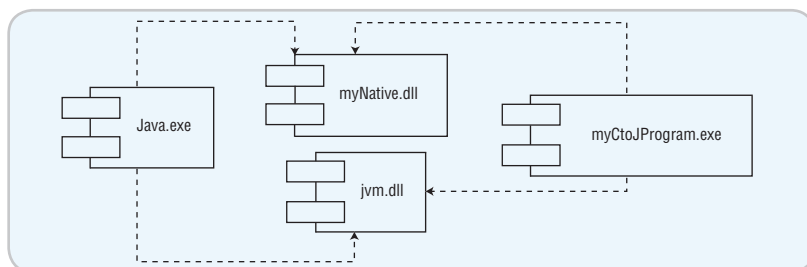


Figure 2 JNI Components: JVM, native libraries, programs

Listing 1: Java code to list contents of a zip file

```
1 import java.util.zip.*;
2 import java.util.Enumeration;
3
4 ZipFile zf = new ZipFile(zipFileName);
5 Enumeration entries = zf.entries();
6 while(entries.hasMoreElements())
7 {
8     String entry = ((ZipEntry)(entries.nextElement()).getName());
9 }
```

Listing 2: Excerpt of C code to list contents of zip file

```
1 int zipEntry(char *csEntryName)
2 {
3     jboolean isException;
4     jstring jsEntryName;
5     jobject oZipEntry = (*jni)-
6 >CallObjectMethod(jni, oZipEntries,
7     midEnumNextElement);
8     isException = checkException();
9     if (isException || oZipEntry == NULL) return
10 0;
11     jsEntryName = (*jni)->CallObjectMethod(jni,
12 oZipEntry, midZipGetName);
13     isException = checkException();
14     if (isException || jsEntryName == NULL)
15 return 0;
16     else
17 {
18     const char *tempData = (*jni)-
19 >GetStringUTFChars(jni, jsEntryName, 0);
20     isException = checkException();
21     if (tempData == NULL || isException)
22 return 0;
23     // copy to caller's buffer and release the
24 UTF
25     strcpy(csEntryName, (char*)tempData);
26     (*jni)->ReleaseStringUTFChars(jni, jsEn-
27 tryName, tempData);
28     isException = checkException();
29     return (!isException);
30 }
31 }
```

Listing 3: CJProxy

```
1 package cj;
2
3 public interface CJProxy
4 {
5     public Object execute(Object args) throws
6     Exception;
7 }
```

Listing 4: Java zip proxy

```
1 package cj.example;
2
3 import cj.CJProxy;
4 import java.io.*;
5 import java.util.zip.*;
6 import java.util.Enumeration;
7
8 /**
9  * CJZipList is a CJProxy that lists the entries
10  * in a zip file.
11  */
12 public class CJZipList implements CJProxy
13 {
14     /**
15     * Proxy execute takes a string with the
16     * name of the zip file.
17     * It returns a pipe-delimited string with
18     * the list of entries
19     * in the zip file.
20     */
21     public Object execute(Object args) throws
22     Exception
23     {
24         if (!(args instanceof String))
25         {
26             throw new RuntimeException("Invalid
27             type for execute");
28         }
29         StringBuffer retbuf = new
30         StringBuffer("");
31         ZipFile zf = new ZipFile((String)args);
32         Enumeration entries = zf.entries();
33         while(entries.hasMoreElements())
34         {
35             String entry = ((ZipEntry)(entries.
36             nextElement()).getName());
37             retbuf.append(entry);
38             retbuf.append("|");
39         }
40         return retbuf.toString();
41     }
42 }
```

tions. Table 2 describes and lists the JNI usage of each of the C functions.

Listing 5 is the complete source code of the C zip program that uses the proxy and façade. (Listing 5 can be downloaded from [www.sys-con.com/java/sourcecef.cfm](http://www.sys-con.com/java/sourcecef.cfm).) The program launches a JVM (line 26), gets a reference to the proxy class (line 29), instantiates it (line 33), and calls the execute() method (line 36). The remaining code (lines 40–50) cleans up the proxy object and class and destroys the JVM.

### Conclusion

Hosting the hard logic in a Java proxy and wrapping JNI calls to the proxy in a C façade reduces the complexity of C-to-Java programming with JNI. ☺

### References

- “Java Native Interface Specification,” Sun Microsystems: <http://java.sun.com/j2se/1.5.0/guide/jni/spec/jni-TOC.html>
- “Tutorial on JNI,” Sun Microsystems: <http://java.sun.com/docs/books/tutorial/native1.1/concepts/index.html>

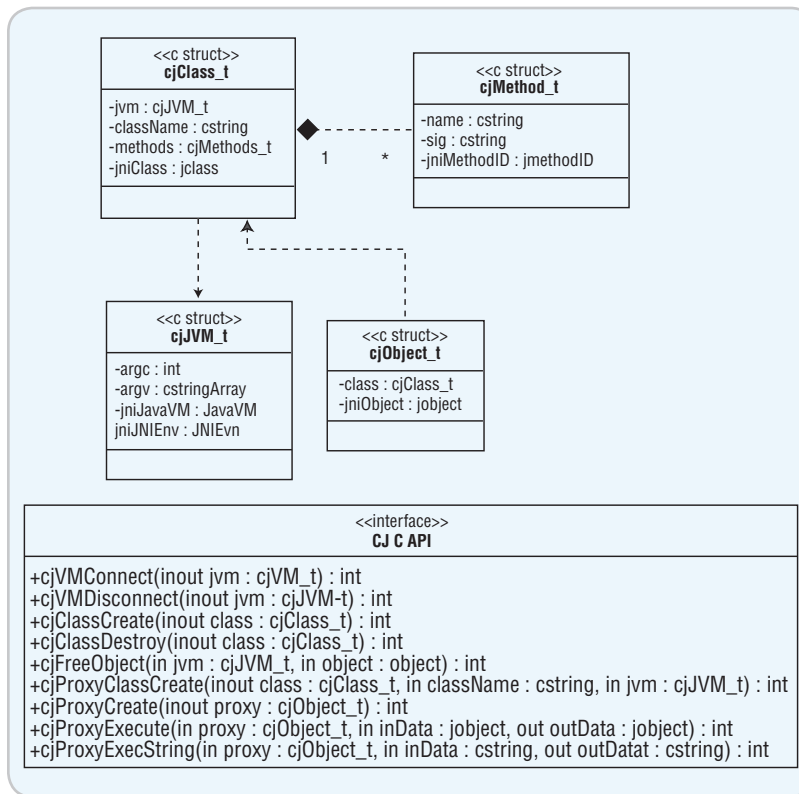


Figure 3 C to Java, Java to C with JNI

**Web Services Edge 2005 East**  
International Web Services Conference & Expo

Leading, 21st Conference & Expo | Wednesday, 21st: Conferences & Expo | Thursday, 22nd: Conferences & Expo

The Largest i-Technology Event of the Year!

February 15-17, 2005, Hynes Convention Center, Boston, MA

3,000 Delegates

Join us in delivering the latest, freshest, and most proven Web service solutions at the Fifth Annual Web Services Edge 2005 East-International Conference & Expo as we bring together IT professionals, developers, policy makers, industry leaders and academics to share information and exchange ideas on technology trends and best practices in secure Web services and related topics including:

- Transitioning Successfully to SOA
- Federated Web services
- ebXML
- Orchestration
- Discovery
- The Business Case for SOA
- Interop & Standards
- Web Services Management
- Messaging Buses and SOA
- Enterprise Service Buses
- SBEAs (Service-Oriented Business Apps)
- Delivering ROI with SOA
- Java Web Services
- XML Web Services
- Security
- Professional Open Source
- Systems Integration
- Sarbanes-Oxley
- Grid Computing
- Business Process Management
- Web Services Choreography

**Announcing**  
**Web Services Edge 2005**  
**Call for Papers**  
**Now Open!**

Submit your proposal by August 15, 2004

**Interested in exhibiting, sponsoring or advertising?**

Becoming a Web Services Edge exhibitor or sponsor offers you a unique opportunity to best position your company's message and visibility to a targeted audience of Web services professionals. Through a wide variety of pre-show, on-site, and post-show marketing programs and opportunities, Web Services Edge 2005 East will serve industry stalwarts as well as exciting start-up companies in reaching the world's most mature internet-based regional market, the US Northeast Region, at Boston's best-known IT business address, the Hynes Convention Center. For Exhibit & Sponsorship information please contact: Grisha Davida, 201-802-3004, e-mail: [grisha@sys-con.com](mailto:grisha@sys-con.com)

Sponsored by:

Contact for Conference Information: Lin Goetz, 201-802-3045, [lin@sys-con.com](mailto:lin@sys-con.com)

**www.sys-con.com/edge**

## JNI Case Study: Java Launcher

Every Java developer who uses the Sun SDK is grateful for the C program known as the launcher (its executable name is “java”), which uses the JNI Invocation API to create a JVM, load a Java class into the JVM, and call its main() method, thereby launching a Java application on behalf of the caller. If the launcher did not exist, a good C developer with JNI knowledge could write an equivalent program in less than a week.

The launcher’s source code is available from Sun and is packaged with the SDK (version 1.4.2\_04, which can be downloaded from <http://java.sun.com/j2se/1.4.2/download.html>). In the base directory of the installed SDK is a file called src.zip. If you extract that file, the exploded “launcher” directory contains the four source files that constitute the launcher program: java.h, java.c, java\_md.h, and java\_md.c. The launcher’s source code is unmistakably C: murky, idiomatic, and circuitous. On the other hand, the end result is a functional program that has been run successfully innumerable times by innumerable users. Understanding how it works is a case study in the use of JNI.

Suppose there is a class called Hi in the package com.mike that has a public main() method and thus can be started as a Java application through the launcher. The following is the source code:

```
1 package com.mike;
2
3 public class Hi
4 {
5     public static void main(String args[])
6     {
7         --
8     }
9 }
```

The Hi application is started under SDK 1.4.2\_04 on Windows 2000 with the following commands:

```
1 set JAVA_HOME=c:\j2sdk1.4.2_04
2 set _JAVA_LAUNCHER_DEBUG=true
3 %JAVA_HOME%\bin\java -classpath src -Xms32m -Xmx64m -Dmy.property=1 com.mike.Hi arg1 arg2
```

Line 3 calls the launcher executable Java in the bin directory of my SDK (which, as line 1 indicates, is c:\j2sdk1.4.2\_04). The arguments passed to the launcher are:

- **-classpath src:** Look for my “Hi” class in the directory src.
- **-Xms32m:** Set its minimum heap size to 32MB.
- **-Xmx64m:** Set the maximum heap size to 64MB.
- **-Dmy.property=1:** Make a property available to the application with the key my.property and value 1.
- **com.mike.Hi:** Run the application in this class.
- **arg1 arg2:** Pass arguments “arg1” and “arg2” to the application’s main method.

Line 2 sets an environment variable called \_JAVA\_LAUNCHER\_DEBUG, which causes the launcher to generate debugging output to the console at runtime:

```
1 ----_JAVA_LAUNCHER_DEBUG----
2 JRE path is c:\j2sdk1.4.2_04\jre
3 jvm.cfg[0] = ->-client<-
4 jvm.cfg[1] = ->-server<-
5 jvm.cfg[2] = ->-hotspot<-
6 jvm.cfg[3] = ->-classic<-
7 jvm.cfg[4] = ->-native<-
8 jvm.cfg[5] = ->-green<-
9 1306 micro seconds to parse jvm.cfg
10 JVM path is c:\j2sdk1.4.2_04\jre\bin\client\jvm.dll
```

```
11 5571 micro seconds to LoadJavaVM
12 JavaVM args:
13 version 0x00010002, ignoreUnrecognized is
JNI_FALSE, nOptions is 6
14 option[ 0] = '-Djava.class.path=.'
15 option[ 1] = '-Djava.class.path=src'
16 option[ 2] = '-Xms32m'
17 option[ 3] = '-Xmx64m'
18 option[ 4] = '-Dmy.property=1'
19 option[ 5] = '-Dsun.java.command=com.mike.Hi arg1 arg2'
20 125113 micro seconds to InitializeJVM
21 Main-Class is 'com.mike.Hi'
22 Apps' argc is 2
23 argv[ 0] = 'arg1'
24 argv[ 1] = 'arg2'
25 32937 micro seconds to load main class
26 ----_JAVA_LAUNCHER_DEBUG----
```

The launcher begins by finding the JRE (line 2) and the right JVM (lines 3–10), and then loads the JVM dynamically (line 11); as we’ll see, this logic is platform dependent. In this case, because we didn’t name a specific JVM when calling the launcher, the launcher defaults to “client” (more on this below). Lines 12–19 show the arguments that the launcher will pass to the JVM; these correspond to the arguments passed to the launcher. In line 20, the launcher starts the JVM. The class whose main method that launcher will call is given on line 21; the arguments passed to it are shown in lines 22–24.

The launcher’s logical design (as of version 1.4.2\_04) is depicted in Figure 4.

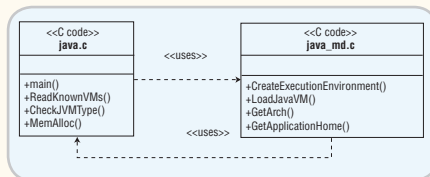


Figure 4 Launcher design

The launcher consists of two modules: java.c, which contains the main function of the launcher program as well as platform-independent helper functions, and java\_md.c, which houses platform-specific functions. The modules share a bidirectional dependency: functions in java.c call java\_md.c and vice versa; for example, main() in java.c calls CreateExecutionEnvironment() in java\_md.c, which in turn calls ReadKnownVMs() in java.c.

The source code in java\_md.c is different for each SDK platform release. For example, the Windows SDK has the Windows version of java\_md.c but does not have the Solaris version. If you want to see both (as I did as I was writing this article), you must download both releases.

The main steps in the launcher’s processing are the following:

1. **Create the execution environment:** The CreateExecutionEnvironment() function, implemented in java\_md.c, is a platform-specific search for the JRE path, JVM path, and JVM type for use by the launcher. The Windows version looks up the JRE path in the registry (on my machine, the registry key HKEY\_LOCAL\_MACHINE\Software\JavaSoft\Java Runtime Edition\1.4\JavaHome is C:\Program Files\Java\jre1.4.2\_04), and then checks whether the J

VM type selected by the caller (corresponding to “-client” or “-server” launcher command-line options) is one of the allowable types listed in the file JREPath\lib\Arch\jvm.cfg (on my machine “Arch” is “i386”). If the caller did not specify a JVM type, the launcher defaults to the first type listed in jvm.cfg (on my machine it’s “client”). The JVM path on Windows is JREPath\bin\JVMType\jvm.dll (e.g., JREPath\bin\client\jvm.dll for the client JVM).

2. **Load the JVM dynamically:** Whereas most programs let the operating system implicitly link shared libraries to their processes, the launcher, which allows the user to specify at runtime which version of the JVM library to use (the “-client” or “-server” command-line arguments to the launcher), explicitly loads the JVM library using a platform-specific interface. The logic resides in java\_md.c’s LoadJavaVM() function. On Windows, this function calls the Win32 LoadLibrary() to load the JVM DLL and link it to the launcher process, and then calls the Win32 GetProcAddress() function to get a pointer to the invocation API function JNI\_CreateJavaVM() used in step 4.
3. **Prepare JVM runtime options based on command-line options passed to the launcher:** Command-line options such as -D, -X, and -classpath are assembled into an array to be passed to the JVM. The launcher adds an additional property for use by the JVM of the form -Djava.sun.command=class arg1 arg2 ..., where class is the fully qualified name of the target class and “arg1 arg2 ...” is the list of command-line arguments to be passed to its main method.
4. **Create the JVM:** The launcher calls the JVM’s JNI\_CreateJavaVM() function, passing the options prepared above.
5. **Load the target class into the JVM by calling the JNI FindClass() method:** The launcher first replaces dots with slashes in the class name (e.g., it converts com.mike.Hi to com/mike/Hi) because the JVM expects slashes instead of dots.
6. **Call the main method of the target class:** First, the launcher gets a reference to the main() method by calling the JNI GetStaticMethodID() function, passing the class reference acquired in step 5, the method name (“main”), and the signature (“([Ljava/lang/String;)V”, the JVM’s peculiar representation of a void method that accepts an array of java.lang.String objects). Second, the launcher calls the method via CallStaticVoidMethod(). The launcher prepares the string array method input using some of the JNI’s array functions (NewObjectArray(), SetObjectArrayElement()), and handles exceptions in the main method using JNI’s ExceptionOccurred(), ExceptionDescribe(), and ExceptionClear().
7. **Shutdown the JVM:** This is done by calling the JNI DetachCurrentThread() and DestroyJavaVM() functions.

A mystery to many Java developers, the launcher is nothing more than a little C program that uses the JNI to initiate a Java application.





<http://www.webappcabaret.com/jdj.jsp>  
1.866.256.7973

J2EE Web Hosting

Point-and-Click Complex J2EE Hosting...

# <JAVA J2EE HOSTING AND OUTSOURCING>

JAVA Developers: Design and Architect with Struts. Develop with Eclipse. Build with Ant. Deploy to WebAppCabaret. It is that easy with a hosting company that understands the Java J2EE standard. Web Services - a buzz word or reality. How many web hosts provide the facilities to deploy and run applications written for Web Services? We Do. At WebAppCabaret, we lead the way in providing comprehensive Java J2EE and PHP/Perl Web Hosting solutions.

Easy Application Deployment is a reality at WebAppCabaret. Each Account's J2EE Application Server is installed with its own instance and default settings so you have complete control. Our Web Control Panel makes it a breeze for JVM restarts. Such is an example of our Advanced Web Hosting Infrastructure and Tools. If you are a consultant, do you have a complex web hosting requirement for your client? Web Host or Reseller, are you looking to provide services without the headaches of managing your own network infrastructure? Look no more.

For JAVA J2EE we offer the latest versions of Tomcat, JBoss, and Jetty Application Servers. For Scripting programming we offer the latest PHP and Perl. We also offer the latest MySql and PostgreSQL Databases. Commercial software, including Resin, JRun, and Oracle, is also available for an extra charge. In addition we provide valuable tools such as Bugzilla Bug Tracking, CVS Version Control plus a whole lot more. Does your service provider offer ENCRYPTED ACCESS for Email (POP3S/SMTPTS/IMAPS), Shell, and FTP? WE DO. All of this is backed by our Tier 1 Data Center with 100% Network Uptime Guarantee. Deploy your web application on one of our Shared, Managed Dedicated, or Clustered plans.

For more details please log on to <http://www.webappcabaret.com/jdj.jsp> or give us a call at 1(866)-256-7973.

## Shared Hosting

The Enterprise plan is our most popular shared solution. Starting at \$39 per month it is loaded with features including: Private JVM which you can restart at anytime via the Point-and-Click NGASI Hosting Manager. Selection of various versions of popular open-source application servers (Tomcat, JBoss, and Jetty) up to the latest. MySql and PostgreSQL are also included. Plus much more.

## Commercial Software

The following are some of the Commercial Software options provided as affordable priced add-ons to our standard Open-Source offering. Resin-considered to be the fastest Servlet Engine. Oracle-the worlds most popular database. JRun-fully certified J2EE and commercial grade quality server. ColdFusion-build web applications with fewer lines of code than ASP, PHP, and JSP.

## Managed Dedicated

Our J2EE-Ready dedicated servers make deploying complex applications a breeze. Along with the popular point-and-click NGASI Web Application Hosting Manager, you can easily deploy a single web site or configure multiple web sites and applications. Each Managed Dedicated server is fully firewalled. Our self-healing monitoring system ensures your server is up and running at all times.

## Load-Balancing and Clustering

Centrally deploy your application to multiple clusters. Our Advanced Cluster Management Tool works for most J2EE application server environments. Combined with a number of software and hardware based Load-Balancing options and you get an affordable High Availability solution.

## VPN

Need to host a site outside your corporate network with secure access? Our VPN service is the solution for peace of mind. Based on CISCO's purpose-built security appliances, our VPN service deliver unprecedented levels of security, performance and reliability.

## Reseller

Run your own Web Hosting service without the headaches of building or managing your network and servers. No system administration required. With the Web Hoster turnkey solution you get your own branded hosting service. Easily setup web accounts with a click of a mouse. Charge your own price.



**Joe Winchester**  
Desktop Java Editor



# Swing Low, Swing High, Sweet Desktop

**S**un has made two significant announcements recently in the Java desktop space: Java Desktop Integration Components (JDIC) (<https://jdic.dev.java.net>) and Java Desktop Network Components (JDNC) (<https://jdnc.dev.java.net>), both of which are open sourced under an LGPL.

## JDIC

JDIC is essentially about allowing Swing access to more native platform resources, such as embedding the operating system's Web browser in a GUI, or enabling more control over taskbar support. I think the goal of elevating the function point of the end-user's experience, so that a Java application looks and behaves no differently for other desktop programs, is wonderful. I won't belabor the obvious point, but for me JDIC is a missed opportunity for bringing Swing and SWT toolkits closer together, as the latter already provides native embedded browser and taskbar support. Both toolkits have farther to travel along this road, as users demand more and more platform fidelity from their programs. All I sincerely hope, for Java's sake, is that the decision to forgo the chance to use the CPL open sourced SWT as the basis for JDIC was made for sound business reasons, not bruised egos.

## JDNC

Java Desktop Network Components is a project that has always promised to be successful because its roots lie in trying to simplify the programming model of writing GUIs that connect to back-end databases or services. By implementing JDNC, developers have helpfully adopted a layered approach.

## Swing Extensions

The extensions include new UI classes such as JXTable, JXTree, JXEditor, or JTreeTable. These extend the basic Swing toolkit to provide a set of controls that are designed out of the

box to work with data. The JForm class, for example, is a nicely thought-out control that helps you easily create a data-bound set of components. What's nice about the Swing extensions is that they can be used without the rest of JDNC and, with the sorting and filtering enhancements, represent a nice turn of the crank for Swing that any GUI developer will hopefully benefit from. Another welcome feature is an overhaul of the way actions work, allowing more flexibility. I believe there might be plans to roll the Swing extension packages into a future release of J2SE. Such a move would be great for the general Swing developer and would provide a welcome set of base enhancements.

## JDNC Components

These mirror the Swing extensions with the prefix JN, so for JXTable there is JNTable, JEditor has JNTable, and so on. These are standard JavaBeans so they should be easy to integrate into GUI builders and intuitive to Java programmers; however, rather than subclass their visual peers, they wrap them instead. I think this is a great decision, and is rooted in the desire to simplify the programming model that currently includes a lot of low-level properties for Swing controls. By using delegation, the API that's surfaced for the developer is one designed around the data binding and access capabilities of the control. There are some nice new listener interfaces as well, such as `org.jdesktop.swing.event.ProgressSource` that allows a `DataSource` to signal the progress and completion of a long-running task. The whole experience using the JNComponents shows that they have been well thought-out and well implemented.

## JDNC Markup Language

The purpose of this layer is to allow the nonprogrammer to easily customize JDNC components using XML. One of the main benefits I can envisage with the XML configuration of components

is not that the unskilled developer will use this in preference to cutting Java source code. Presumably, said person is going to be using a GUI builder-like tool that should be able to hide XML or Java implementations equally well. XML, however, offers the advantage of being easy to create and manipulate at runtime, so GUI behavior can be manipulated dynamically (perhaps the XML prepared by a servlet as the result of a user query) and less hard coding need occur in the actual client layer. One of the goals of any large business GUI application must be to capture as much as possible in rules and rely less on hard-coded, bespoke client screen logic. Having XML prepared by a rule engine that consumes a model definition of the application and serves this up to JDNC to implement the wiring logic might well be the answer. This could become a very powerful usage scenario.

## Open Source

JDNC has been released as an open source project, which is interesting because it's outside the usual JSR/JCP process. I'm very encouraged by this, as I hope it will benefit from the rapid progress that other open source projects have enjoyed, and because the end users of JDNC are programmers trying to build business applications using Swing. It's an invitation for everyone who has done this for the last few years to bring the benefit of their knowledge, gripes, suggestions, and, ultimately, code to the table. If you've had to build your own data access framework on top of Swing, or wished one was there, I strongly encourage you to visit the JDNC homepage to become involved in the project and give the developers the benefit of your experience and ideas. I believe JDNC is one of the most exciting things to come along in the GUI space in any language for many years, and I think it has the potential to take Java client programming to a whole new level. ☺

Joe Winchester is a software developer working on WebSphere development tools for IBM in Hursley, UK.

[joewinchester@sys-con.com](mailto:joewinchester@sys-con.com)

# Yeehaw?



## Some reporting solutions are pretending to be something they're not.

When you look under the fancy packaging, there's only one Java reporting solution with years of proven, real-world experience and thousands of successful deployments worldwide – JReport.

JReport is a comprehensive, 100% J2EE reporting solution that is intuitive in nature and easy to integrate, eliminating the need for costly professional services and training. From departmental applications to enterprise-wide deployments, JReport scales to meet your reporting needs, ensuring uninterrupted access to critical business information, so you get the answers you need, when you need them.

Whether you need to deploy sophisticated reports via the Web or enable users to create ad hoc reports on demand, JReport delivers on the promise of self-serve reporting. And, with JReport's *one-click analysis*<sup>™</sup>, any user can slice-and-dice, pivot, and drill-down on a report, from any Web browser.

Visit [www.jinfonet.com/yeehaw](http://www.jinfonet.com/yeehaw) or call 301-838-5560 today, and discover why, when it comes to Java reporting, JReport is the real McCoy.



# Java Design Patterns for Long Lists

Providing fast performance

by Heman Robinson

In the late 1990s, a GUI design pattern emerged for choosing multiple objects from long lists. In GUI Design Essentials, Susan Weinschenk, Pamela Jamar, and Sarah

Ye called this the Selection Summary pattern. In "A Dual Listbox Selection Manager" by Steve Aube, it's also known as the Dual Listbox Selection interface. In The Java Look and Feel Guidelines, Advanced Topics, it is called the Add-and-Remove idiom.

The Add-and-Remove design pattern (shown in Figure 1) has many variations. One common enhancement is to provide "Move Up" and "Move Down" buttons to reorder the chosen list (see Figure 2). Sometimes the chosen list is displayed as a table to show additional information.

My previous article ("GUI Design Patterns," *JDJ*, Vol. 9, issue 7) showed how to optimize usability for this common GUI design pattern. This article shows how to optimize performance for this design pattern and other GUIs that display long lists.



**Heman Robinson** is a senior developer with SAS Institute in Cary, NC.

He holds a BS in mathematics from the University of North Carolina and an MS in computer science from the University of Southern California.

He has specialized in GUI design and development for 15 years and has been a Java developer since 1996.

hemanrobinson@yahoo.com

## Java Performance Patterns

For short lists, neither the `JList` or the `JTable` are likely to give performance problems; Java and Swing have been optimized many times over the years. If your lists contain thousands of objects though, there are some standard Java design patterns that optimize performance.

The key to all these performance patterns is to realize that the default list and table implementations are general purpose. To optimize performance, you want to bypass this general-purpose code by informing the `JList` or `JTable` of your application's specific needs.

## Fix the Cell Size

For example, `JLists` by default assume that the objects they contain may be of varying sizes. If your application's objects are all the same size, you can inform the `JList` of this fact. It will then bypass its general-purpose, size-checking code.

To fix the cell size, invoke either the `setFixedHeight()` and `setFixedWidth()` methods, or the `setPrototypeCellValue()` method. In "Advanced `JList` Programming," Hans Muller notes that `setFixedHeight()` and `setFixedWidth()` are useful to align a `JList` with another component. Otherwise, it's generally more convenient to use `setPrototypeCellValue()`.

For the prototype cell value, use the value that is largest visually. If the maximum width is known, a prototype value can be assigned without looping over all the values, thus saving initialization time. Alternatively, if the application uses a monospace font, a fast loop can be written to check the string lengths of all values. Otherwise, to allow proportional fonts, anti-aliasing, and other issues, check the `FontMetrics` as in the code below.

```
double width = 0;
String prototype = "";
FontMetrics fm = jList.getFontMetrics( jList.getFont());
Graphics g = jList.getGraphics();
for( int i = 0; ( i < values.length ); i++ )
{ String s = values[ i ].toString();
  if( width < fm.getStringBounds( s, g ).getWidth())
  { width = fm.getStringBounds( s, g ).getWidth();
    prototype = s;
  }
}
jList.setPrototypeCellValue( prototype );
```

## Write a Custom Model

For many applications, fixing the cell size may provide all the performance boost you need. If it doesn't, the next step is to take advantage of Swing's flexible architecture.

Figures 3 and 4 show portions of the `JList` and `JTable` architectures. Both lists and tables allow you to replace their default models with custom models of your own. The only requirement is that your custom model implement the `ListModel` or `TableModel` interface.

The simplest way to do this is to extend `AbstractListModel` or `AbstractTableModel`. These classes provide management of listeners and events. Technically, to support the `ListModel` interface it is necessary to override only two methods in `AbstractListModel`:

```
public Object getElementAt( int i )
public int getSize()
```

Similarly, to support the TableModel interface it's necessary to override only three methods in AbstractTableModel:

```
public Object getValueAt( int row, int column )
public int getRowCount()
public int getColumnCount()
```

These methods support a ListModel or TableModel that is immutable: its contents can't be changed. For an application such as the Add-and-Remove pattern, the contents must be mutable. This requires extending the AbstractListModel with methods to add and remove values from the list:

```
public void addElement( Object o )
public void removeElement( int i )
```

Similarly, the AbstractTableModel can be extended with methods to add and remove rows from the table.

```
public void addRow( Object[] row )
public void removeRow( int i )
```

Writing a custom model informs the Swing GUI control of your application's specific needs, causing it to bypass its general-purpose code. For example, both the DefaultListModel and the DefaultTableModel are Vector based. This means their accessor methods are synchronized. If your application doesn't require synchronization, it can be removed in the custom model, for example, by using an ArrayList instead of a Vector. A custom ListModel based on an ArrayList is shown in Listing 1.

For the Add-and-Remove pattern, a further performance boost can be obtained by realizing that the total number of objects never changes. Both Original and Chosen lists have a fixed maximum size, which is the sum of the number of objects in each. This means that a custom model does not need expandable storage, such as a java.util.Collection, so a more efficient array can be used instead. A custom TableModel based on an array is shown in Listing 2.

Finally, significant performance can be gained by adding methods to the custom model to process multiple objects. The custom ListModel of Listing 1 provides three methods to handle multiple objects:

```
public void addAll( Object[] objects )
public void clear()
public Object[] toArray()
```

In the DefaultListModel, to add 100 objects addElement() must be invoked 100 times. In the custom model, the addAll() method can be invoked only once. If an application operates on multiple objects, performance can almost always be improved by writing a custom model.

### Can More Be Done?

For most applications, a custom model provides sufficient performance improvement. However, Swing provides another option using the same architectural pattern. As shown in Figures 5 and 6, JList and JTable rely on renderers to display their contents.

Just as the custom model replaced the default model, a custom renderer can replace the default renderer. The only requirement is that the custom renderer implement the ListCellRenderer or TableCellRenderer interface.



Figure 1 Add-and-Remove Pattern



Figure 2 Add-and-Remove pattern with "Move" buttons and table

	Varied Cell Size,	Fixed Cell Size,	Fixed Cell Size,	Fixed Cell Size,
Objects	DefaultListModel	DefaultListModel	ArrayList Model	Array Model
1,000	11	2	2	2
2,000	21	2	2	2
5,000	50	2	2	2
10,000	104	2	2	2
20,000	208	2	2	2
50,000	500	2	2	2
100,000	1070	3	3	3

Table 1 JList "Add" Performance, in Milliseconds

	Varied Cell Size,	Fixed Cell Size,	Fixed Cell Size,	Fixed Cell Size,
Objects	DefaultListModel	DefaultListModel	ArrayList Model	Array Model
1,000	7	7	1	1
2,000	15	15	1	1
5,000	35	35	3	2
10,000	76	80	6	2
20,000	157	155	11	5
50,000	380	380	28	11
100,000	770	770	55	22

Table 2 JList "Add All" Performance, in Milliseconds

Objects	DefaultTableModel	ArrayList Model	Array Model
1,000	2	2	2
2,000	2	2	2
5,000	2	2	2
10,000	2	2	2
20,000	2	2	3
50,000	3	3	3
100,000	3	3	3

Table 3 JTable "Add" Performance, in Milliseconds

Objects	DefaultTableModel	ArrayList Model	Array Model
1,000	7	1	1
2,000	15	2	1
5,000	38	3	1
10,000	78	6	2
20,000	160	13	5
50,000	385	35	11
100,000	770	75	22

Table 4 JTable "Add All" Performance, in Milliseconds

The performance boost from a custom renderer is roughly proportional to the number of objects being rendered. If your application displays only a few objects, as in the Add-and-Remove pattern, this performance boost is not significant. A better use of a custom renderer is given by Steve Wilson and Jeff Kesselman in *Java Platform Performance: Strategies and Tactics*. Their example displays a sparse table that derives a significant performance boost because the empty cells don't need to be rendered at all.

Some highly specialized applications require more specialized performance patterns. Hans Muller notes that internally JList uses the toString() method to convert objects to strings. If

the application does not need this generality, the conversion time can be saved by building a custom model around the String class rather than the Object class.

In *Christmas Tree Applications* Scott Violet and Kathy Walrath give a fine and detailed example using a custom renderer and other performance patterns. Their code produces fast performance for frequently updated JTables. Patterns such as these are not usually needed, but they show the performance improvements that become possible when Swing is tailored to a specific application.

### How Much Improvement Can We Expect?

Performance benchmarks for the Add-and-Remove pattern show that fixing the cell size is the most cost-effective performance pattern for JLists. Both JLists and JTables can achieve dramatic improvement from custom models, especially when processing multiple objects.

The benchmarks shown in Figures 7 and 8 and Tables 1–4 were run using JDK 1.4.1 under Mac OS X on a G4 CPU at 450MHz. Your mileage will vary, but these conclusions hold for most applications:

- For a JList, fix the cell size.
- For a long list or table, write a custom model.
- For a specialized application, consider specialized design patterns such as a custom renderer.

### Conclusion

The Add-and-Remove GUI design pattern enables users to choose multiple objects from long lists. Appropriate Java design patterns provide fast performance for this GUI and other applications. ☺

### Resources

- Aube, S. (2000). “A Dual Listbox Selection Manager”: [www.codeguru.com/Cpp/controls/listbox/article.php/c4755](http://www.codeguru.com/Cpp/controls/listbox/article.php/c4755)

- Muller, H. (2000). “Advanced JList Programming”: [http://java.sun.com/products/jfc/tsc/tech\\_topics/jlist\\_1/jlist.html](http://java.sun.com/products/jfc/tsc/tech_topics/jlist_1/jlist.html)
- Sun Microsystems Inc. (2002). *Java Look and Feel Design Guidelines: Advanced Topics*. Addison-Wesley Professional: <http://java.sun.com/products/jlf/at/book/Idioms6.html>
- Violet, S., and Walrath, K. (2002). “Christmas Tree Applications”: <http://java.sun.com/products/jfc/tsc/articles/ChristmasTree/>
- Weinschenk, S., Jamar, P., and Yeo, S. (1997). *GUI Design Essentials*. Wiley & Sons. p. 192, 206–207.
- Wilson, S., and Kesselman, J. (2000). *Java Platform Performance: Strategies and Tactics*. Chapter 10: <http://java.sun.com/developer/Books/performance/>

#### Listing 1

```

/** Custom ArrayList model */
private class CustomArrayListModel extends
AbstractListModel
{
    /** @serial List */
    private ArrayList _list;

    /**
     * Returns specified object.
     * <p>
     * @param i index
     * @return object
     */
    public Object getElementAt( int i )
    { return( _list.get( i ) );
    }

    /**
     * Returns size of list.
     * <p>
     * @return size
     */
    public int getSize()
    { return( _list.size() );
    }

    /**
     * Adds the specified element to the end of the list.
     * <p>
     * @param o Object
     */
    public void addElement( Object o )
    { _list.add( o );
      fireIntervalAdded( this,
        _list.size() - 1, _list.size() - 1 );
    }

    /**
     * Removes the specified element.
     * <p>
     * @param i index
     */
    public void removeElement( int i )
    { _list.remove( i );
      fireIntervalRemoved( this, i, i );
    }

    /**
     * Adds all of the specified elements.
     * <p>
     * @param objects Array of Objects
     */
    public void addAll( Object[] objects )
    { for( int i = 0; ( i < objects.length ); i++ )
      { _list.add( objects[ i ] );
        fireIntervalAdded( this, _list.size() -
          objects.length, _list.size() - 1 );
      }
    }

    /**
     * Removes all objects from the list.
     */
    public void clear()
    { int size = _list.size();
      _list.clear();
      fireIntervalRemoved( this, 0, size - 1 );
    }

    /**
     * Returns list as array.
     * <p>
     * @return list
     */
    public Object[] toArray()
    { return( _list.toArray() );
    }
}

```

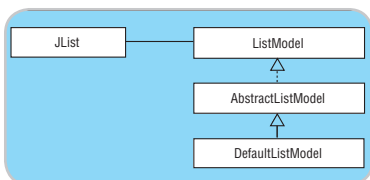


Figure 3 |JList and ListModel

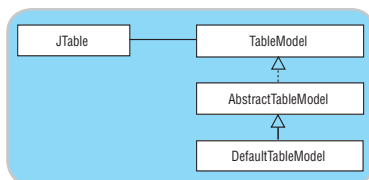


Figure 4 |JTable and TableModel

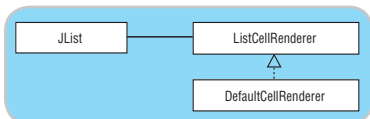


Figure 5 |JList and ListCellRenderer

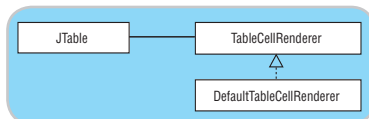


Figure 6 |JTable and TableCellRenderer

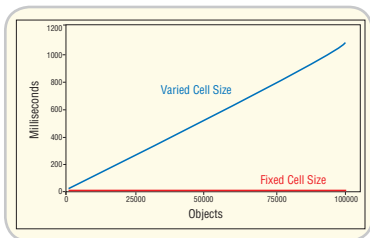


Figure 7 |JList “Add” Performance

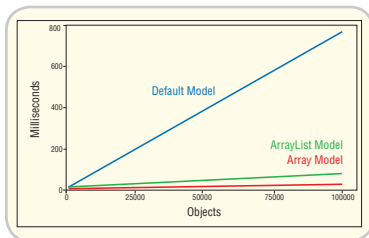


Figure 8 |JTable “Add All” Performance

```

}
/**
 * Constructor.
 * <p>
 * @param size size of list
 */
public CustomArrayListModel( int size )
{ _list = new ArrayList( size );
}
}

```

#### Listing 2

```

/** Custom array model */
private class CustomArrayModel extends AbstractTableModel
{
    /** @serial Data vector */
    private Object[][] _table = new Object[ 0 ][];

    /** @serial Column names */
    private Object[] _columnNames = new Object[ 0 ];

    /** @serial Number of rows */
    private int _rows = 0;

    /**
     * Returns specified value.
     * <p>
     * @param row row index
     * @param column column index
     * @return value
     */
    public Object getValueAt( int row, int column )
    { return( _table[ row ][ column ] );
    }

    /**
     * Returns number of rows.
     * <p>
     * @return number of rows
     */
    public int getRowCount()
    { return( _rows );
    }

    /**
     * Returns number of columns.
     * <p>
     * @return number of columns
     */
    public int getColumnCount()
    { return( _columnNames.length );
    }

    /**
     * Returns row as array.
     * <p>
     * @param i row index
     * @return row
     */
    public Object[] getRow( int i )
    { return( _table[ i ] );
    }

    /**
     * Returns column names.
     * <p>
     * @return column names
     */
    public Object[] getColumnNames()
    { return( _columnNames );
    }

    /**
     * Assigns data and column names.
     * <p>
     * @param objects data vector
     * @param columnNames column names
     */
    public void setDataVector(
        Object[][] objects, Object[] columnNames )
    { System.arraycopy( objects, 0,
        _table, 0, objects.length );
        _rows = objects.length;
        columnNames = columnNames;
        fireTableRowsInserted( 0, _rows - 1 );
    }

    /**
     * Adds the specified row to the end of the table.
     * <p>
     * @param row row
     */
    public void addRow( Object[] row )
    { _table[ _rows ] = row;
        _rows++;
        fireTableRowsInserted( _rows - 1, _rows - 1 );
    }

    /**
     * Removes the specified row.
     * <p>
     * @param i index
     */
    public void removeRow( int i )
    { System.arraycopy( _table, i + 1,
        _table, i, _rows - i - 1 );
        _rows--;
        fireTableRowsDeleted( i, i );
    }
}

```

```

}
/**
 * Adds all of the specified rows.
 * <p>
 * @param objects Array of rows
 */
public void addAll( Object[][] objects )
{ System.arraycopy( objects, 0,
    _table, _rows, objects.length );
    _rows += objects.length;
    fireTableRowsInserted(
        _rows - objects.length, _rows - 1 );
}

/**
 * Removes all rows.
 */
public void clear()
{ int rows = _rows;
    _rows = 0;
    fireTableRowsDeleted( 0, rows - 1 );
}

/**
 * Returns table as array.
 * <p>
 * @return size
 */
public Object[][] toArray()
{ Object[][] objects = new Object[ _rows ][];
    System.arraycopy( _table, 0, objects, 0, _rows );
    return( objects );
}

/**
 * Constructor.
 * <p>
 * @param rows maximum number of rows
 * @param objects data vector
 * @param columnNames column names
 */
public CustomArrayModel( int rows,
    Object[][] objects, Object[] columnNames )
{ _table = new Object[ rows ][];
    System.arraycopy( objects, 0,
        _table, 0, objects.length );
        _rows = objects.length;
        _columnNames = columnNames;
}
}

```

## Google Seeks Expert Computer Scientists

Google, the world leader in large-scale information retrieval, is looking for experienced software engineers with superb design and implementation skills and considerable depth and breadth in the areas of high-performance distributed systems, operating systems, data mining, information retrieval, machine learning, and/or related areas. If you have a proven track record based on cutting-edge research and/or large-scale systems development in these areas, we have plenty of challenging projects for you in Mountain View, Santa Monica and New York.

Are you excited about the idea of writing software to process a significant fraction of the world's information in order to make it easily accessible to a significant fraction of the world's population, using one of the world's largest Linux clusters? If so, see <http://www.google.com/cacm>. EOE.



# A GUI Painter Friendly Table Component

by Gunnar Grim

## *The principle of the column container*

In the early days of Java, GUI forms were written, not drawn. They were created by writing code that instantiated components and added them to containers with various layout constraints. Then the program was run and the result could be admired. This way of working, WYGIWYG (what you get is what you get) was often quite fun, more often frustrating, and never very productive. Today we have a JavaBeans specification and integrated development environments (IDEs) with GUI painters. Some of these are doing really good jobs, considering the difficulties with layout managers and platform portability.

With most components, such as text fields and buttons, the principle of dropping them on the form, setting properties, and adding event listeners is quite sufficient. The JTable though is more problematic. It's just too complex to configure with simple property editors and also so common that you don't want to have to write a lot of code every time you use it.

You can drop a table in a JScrollPane and set a lot of properties on the JTable, but when it comes to adding and customizing columns, the GUI painter can't help you since the columns are not JavaBeans. One solution is for the GUI painter to provide an editor for the table model property, thereby letting you define columns and set a few attributes on them. However, I have never seen an editor that will allow you to customize the columns of the table with the same flexibility you have when you customize text fields on a form.

There is, however, a completely different way to go, which is the one I chose for the table component in our own class library DOI, called the DoiTable.

### Design Time Behavior

The DoiTable doesn't have a table model property editor at all. In fact, when you drop it on the form it doesn't even look like a table. Instead, it behaves like a container during design time, and you fill it with columns by dropping DoiTableColumn components inside it. At runtime, though,

area is an ordinary panel with a flow layout in which column components can be dropped and reordered. The columns must be instances of the DoiTableColumn class. If you accidentally drop some other type of component inside it, the drop area turns red.

A DoiTableColumn is a direct descendant of the DoiTextField class, which is the standard text field in the DOI library, overridden to change the design time appearance and add some properties and behavior that is specific to a table column. As you can see, I've tried to make the

column components look a bit like the columns they will become at runtime. From the GUI painter's point of view, the table is just a container. Therefore, the painter will allow you to set properties on each individual column as if they were ordinary fields on a panel, which is exactly what they are, until you run the application. In Figure 1, one of the columns is selected so

you can see the property sheet for it in the lower right pane. Note also that the column components retain their preferred size even if the table is too narrow to show them all on one line. The fourth column, "Logical", doesn't fit, so it's placed on a new row. This behavior is consistent with any other flow layout panel. Although I could have made them resize themselves to mimic the behavior of a JTable more closely, I decided against it to make the columns easier for the designer to work with.

This is basically how the table component presents itself to the designer. To the user, however, it looks just like



it automatically converts itself to a JTable with all the column properties taken from the design time column components.

Figure 1 shows the design time look of a simple table with four columns. The screenshot is taken from the NetBeans form editor. When the designer drops a table on the form, it appears as a big rectangle. The designer can then give the table a label and activate tools for inserting and deleting rows by setting properties on the table. Note the "Table" label and small tool bar above the rectangle. The rectangle is the drop area for columns. During design time this



**Gunnar Grim** is a programmer, designer, and architect for the consulting firm Know IT ([www.knowit.se](http://www.knowit.se)). He has been in the business for 20 years, programming in everything from Z80 assembly code to SQL Windows. Since early 1996 he has worked almost exclusively with Java, mostly on the server side but also quite a lot with Swing. [gunnar.grim@knowit.se](mailto:gunnar.grim@knowit.se)



a `JTable` in a `JScrollPane`, as shown in Figure 2. I'll shortly go into the details on how this conversion happens, but first a little bit about how the table component communicates with the GUI painter.

## Adjusting the BeanInfo

Every JavaBean component that you can draw on a form must have a supporting `BeanInfo` object, which is an instance of a class that implements the `java.beans.BeanInfo` interface. The `BeanInfo` object is used by the GUI painter to determine which properties and events the bean has. Although it can be created automatically using introspection, it's usually written by the author of the bean. Writing such a class is outside the scope of this article, but there is one important feature that is often forgotten when `BeanInfo` classes are described: the "container delegate" property. At the time of writing it isn't even mentioned in the Java Tutorial. Without this property, all beans must fall into the following two categories:

1. *Component beans* such as `JTextField` or `JButton` – you drop them in containers but you don't drop anything inside them.
2. *Simple container beans* such as `JPanel` – they are initially empty and you can drop components inside them.

The GUI painter can tell them apart by treating empty containers as category 2 and all other beans as category 1. The `DoiTable`, however, falls into a third category. It isn't just a container, but a container that initially has a label, a tool bar, and an inner container for the columns. Without a special "trick" in the `BeanInfo` class, the GUI painter would think that the `DoiTable` is an ordinary component because it isn't empty and won't let you drop anything inside it. This is certainly not the behavior we want, so we must inform the GUI painter that it is a container and that it has a special place for dropping stuff. The following code excerpt from the `DoiTableBeanInfo` class shows how this is done:

```
public BeanDescriptor getBeanDescriptor()
{
    BeanDescriptor bd =
        new BeanDescriptor(itsBeanClass);
    bd.setName("DoiTable");
}
```

```
bd.setValue("isContainer",
    Boolean.TRUE);
bd.setValue("containerDelegate",
    "getColumnContainer");

return bd;
}
```

The method creates a `BeanDescriptor`, which is an object that contains basic properties about the bean. While some of these properties have dedicated methods such as `setName`, others are set using the generic `setValue` method. In the code above, the property `isContainer` is set to `TRUE` to tell the GUI painter that although this bean isn't empty, it is still a container. We also have to tell the GUI painter which method on our bean returns the inner container by setting the property `containerDelegate` to the name of the method. In the `DoiTable` case, the method is called `getColumnContainer`.

## Converting to Runtime Behavior

When the application is run we obviously don't want the table to look like it does in the GUI painter. Instead we want the drop area, a.k.a. the column container, to convert itself to a real `JTable`. This conversion happens in the method `addNotify`, which is called automatically on every component when it is added to a displayable container. This method may be called several times, so we must make sure the table doesn't attempt to convert itself more than once. Also, we don't want it to convert itself at all when we are using the table in the GUI painter. To test for design time or runtime mode, there is a method in the `java.beans.Beans` class called `isDesignTime`. This method returns true when called from a component in a GUI painter, and false otherwise.

The first thing we need to do is implement the `addNotify` method:

```
public void addNotify()
{
    super.addNotify();
    commitColumnContainer();
}
```

The first thing the method does is invoke the same method on the superclass to let it do whatever it needs to do, then it calls the method `commitColumnContainer` to do the real work. This method looks like:

```
public void commitColumnContainer()
{
    commitColumnContainer(false);
}
```

As you can see, it doesn't do much; it just delegates to another method. The reason for this is that the other method has a parameter that allows the caller to force a conversion even if we are in design time. This is useful in certain circumstances, which I'll get back to later. For now we'll look at the first few lines of the "real" `commitColumnContainer` method:

```
public void commitColumnContainer(
    boolean pForce)
{
    if (!pForce && Beans.isDesignTime())
        return;
    if (itsColumnContainer == null)
        return;
```




WHO'S DEVELOPING THE COOLEST WIRELESS APPLICATIONS? YOU ARE!

ENTER TO WIN THE 2005 SIMAGINE DEVELOPERS' CONTEST!

Over \$70,000 will be awarded for innovative SIM card services including a special Cingular award for best submission from North America

Deadline is October 10, 2004!



In association with:










Full contest details at: [www.simagine.axalto.com](http://www.simagine.axalto.com) or call | 888 343 5773  
© Axalto 2004

The method starts by checking if a conversion should happen at all by testing the force parameter and calling the `isDesignTime` method. If these tests are passed, it goes on to check if the table has already been converted. The column container panel is created and added to the table by the constructor and removed when the conversion is completed. This means that if it is null, the table is already converted and the method returns immediately. Now the real conversion can be done. We start off by transferring all column beans from the column container into an internal array:

```
int ccc =
    itsColumnContainer.getComponentCount();

itsColumns = new DoiTableColumn[ccc];
```

```
for (int i = 0; i < ccc; ++i) {
    DoiTableColumn column =
        (DoiTableColumn)itsColumnContainer
            .getComponent(i);
    itsColumns[i] = column;
    column.setTable(this);
}
```

Each column is given a reference back to the table using the `setTable` method of the `DoiTableColumn` class. This reference is used by the column to access various properties on the table that affect its behavior. Now it's time to get rid of the column container and replace it with a scroll pane:

```
remove(itsColumnContainer);
itsColumnContainer = null;
itsScrollPane = new JScrollPane();
add(itsScrollPane, BorderLayout.CENTER);
```

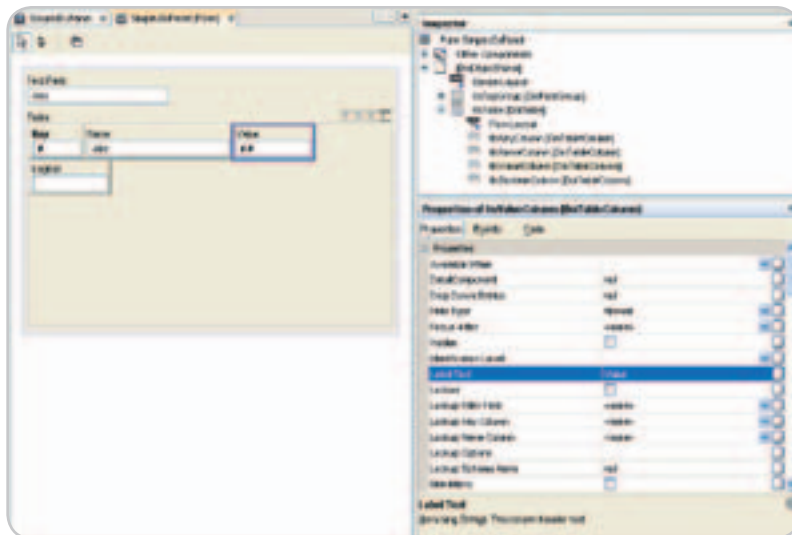


Figure 1 Design Time

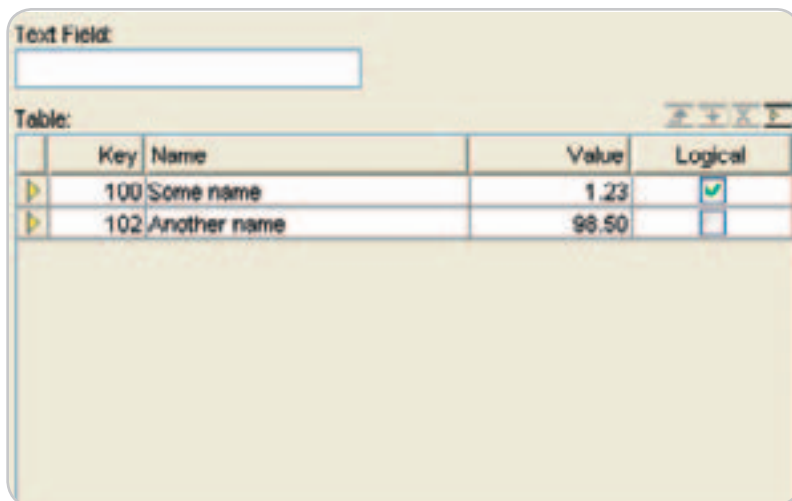


Figure 2 Runtime

The scroll pane will eventually contain a `JTable`, but before we can create it we need a column model, the object used by Swing's `JTable` to represent its columns. A `JTable` can automatically create the column model based on its table model, but we don't want that because the `DoiTableColumn` objects contain much more information about the columns than is contained in an ordinary table model, e.g., preferred width in characters, resizable, label text, etc.

The below code creates a column model that contains column objects of Swing's `TableColumn` class, with relevant properties copied from the corresponding `DoiTableColumn` objects:

```
TableColumnModel colmod =
    new DefaultTableColumnModel();
```

```
for (int i = 0; i < ccc; ++i) {
    // Get the column bean. Skip if hidden.
    DoiTableColumn column = itsColumns[i];
    if (column.isHidden())
        continue;
    // Create a Swing column.
    TableColumn swingColumn =
        new TableColumn();
    // Copy properties.
    swingColumn.setHeaderValue(
        column.getLabelText());
    swingColumn.setResizable(
        column.isResizable());
    // Add to column model.
    colmod.addColumn(swingColumn);
}
```

There is still one little detail before we can create the `JTable`. We need a table model. A `JTable` can't exist without a table model so we need to create one that is initially empty. This is accomplished with the following code:

```
TableModel tm =
    new DefaultTableModel(0, ccc);
```

Now the `JTable` can be created and added to the scroll pane that has replaced the column container. We also tell it not to automatically create a new column model if the table model is replaced later:

```
JTable jt = new JTable(tm, colmod);
jt.setAutoCreateColumnsFromModel(false);
```

```
itsScrollPane.add(jt);
```

That's it. The DoiTable bean now contains a JTable within a JScrollPane instead of a column container panel. The DoiTableColumn beans still exist though, and there is an implicit association between each column bean with the corresponding Swing TableColumn object in the column model. This association will prove very useful for later enhancements, some of which I'll hint at in the next section.

I promised to mention the purpose of the pForce parameter. This parameter can be used by subclasses of the DoiTable that create and add all columns. Let's say you want to create a bean called PhoneNumberTable, with a number type column and a phone number column. This bean would add its columns in the constructor and then call `commitColumnContainer(true)` to force the conversion to a JTable. In this case, the force parameter is necessary since the conversion must happen in design time as well as runtime.

## Enhancements

The purpose of this article is to show you the principle of the column container, not how to write a full-fledged table component. To do that, I'd probably have to fill 10 issues of *JDJ*. For this reason the code examples shown of what really happens inside the DoiTable have been simplified. Still, I'd like to round off with a brief list of some interesting features in the real DOI classes.

### Runtime Propagation of Properties

Many of the DoiTableColumn properties are automatically propagated to the JTable when changed at runtime. This allows runtime code to dynamically change the table by simply setting properties on the DoiTableColumn bean, which is much easier than doing it through the JTable. For example, the column header is updated if the label text of the column is set. This propagation is accomplished through the implicit association between the invisible column bean and the visible table column.

### Runtime Synchronization of Cell Values

The DoiTable has a property called `ContextRowNo` that can be

set programmatically. It is also updated automatically when the user selects a row. I mentioned earlier that the DoiTableColumn class is a subclass of a class called DoiTextField, which is an enhancement of JTextField. This means that a DoiTableColumn bean can have a value. The context row is used to synchronize the value of a column bean and the corresponding cell value. The designer can add a listener on a column bean that's triggered when the user edits the cell. The event handler can then access the cell value through the column bean and set a value on another cell on the same row, also through a column bean. The code for this is easier to write and maintain than using a table model listener.

### Design Time Rendering

As you can see in Figure 1, the text fields in the column beans are not empty. Instead they contain a text value that reflects a few important properties (a feature inherited from the base class DoiTextField): a mandatory column has an exclamation mark suffix, a numeric column is displayed with "#", "##" or "#.#" (depending on if it is an Integer, Long, or Double), an uppercase string column uses "ABC", etc.

### Smart Design Time Checking

In some circumstances, checking for design-time mode is not sufficient. Some IDEs, for example, NetBeans, have a preview function that creates a window with the form inside it where the designer can try it out. The `isDesignTime` method still returns true, however, which causes DoiTable to think that it's still in design mode, and it doesn't convert itself. To get around this, it has its own `isDesignTime` method that first calls the standard method. If it returns false we are in "real" runtime, and no further checking is necessary; if it returns false an extra check for the special preview mode is neces-

sary. This check is IDE dependent, and in the NetBeans case it is done by searching the parent container hierarchy for the innermost frame that has a title starting with "Testing Form[". Other IDEs will most likely need variations of this technique.

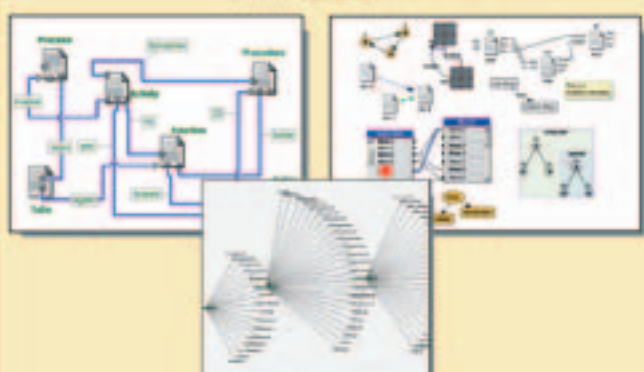
## Conclusion

I hope I've provided you with some ideas that you can use when you write your own beans. The same principle can naturally be applied to very different kinds of widgets, especially complex ones that are easier to design with if they are broken up into parts. The time spent on doing this is earned many times over when the end-user GUIs are designed. ☺

## Resources

- *The Java Tutorial, trail JavaBeans:* <http://java.sun.com/docs/books/tutorial/javabeans/index.html>
- *NetBeans FAQ – GUI Editing:* [www.netbeans.org/kb/faqs/gui\\_editing.html](http://www.netbeans.org/kb/faqs/gui_editing.html)


### Build Incredible Interactive Diagrams with JGo™



Create custom interactive diagrams, network editors, workflows, flowcharts, and design tools. For web servers or local applications. Designed to be easy to use and very extensible.

- Fully functional evaluation kit
- No runtime fees
- Full source code
- Excellent support

Learn more at:  
[www.nwoods.com/go](http://www.nwoods.com/go)  
800-434-9820



# Unlocking Microsoft Office Documents

## An open source alternative

by Ryan Ackley and Avik Sengupta

If you've ever written software to be used by business managers, you will no doubt have received requests for interoperability with the Microsoft Office Applications. "Get me the report in Excel; HTML doesn't cut it and I need to run my own analysis on it"; "Can you index the zillion word documents I have so that the whole organization can search on them?"; "I have all this data in Excel; do I have to enter it again on this Web page?".... These are things we commonly hear as application developers, which is not surprising given the ubiquity of MS Office.

Does this mean you're forced to tie your application to Windows to interface with the COM APIs of Excel or Word? Apart from the fact that you don't want your language or platform decision to be constrained by a lack of choice, it's also important to note that these APIs can be unstable because they're automating a desktop application. Because of this, they are unreliable for any server-side deployment. For the Java developer, however, the power of Jakarta POI is close at hand.

POI is a pure Java application library for reading and writing the Microsoft OLE2 Compound Document Format (OLE2CDF) file formats. This format is used by (among others) various MS Office applications. As the name suggests, this is a format for storing multiple documents (or streams) in one file, for example, storing an embedded spreadsheet along with a presentation. Within this structure are stored the records that contain the application-specific data.

POI is structured along these lines. At its base it has a component known as the POIFS or the POI File System, which is the most complete implementation of the OLE2CDF structure in Java. Layered above this are the components to read

the Excel record structures (HSSF) or the Word record structures (HWPF).

### HSSF

HSSF is the component of POI that allows you to read, write, and manipulate Excel spreadsheets from pure Java applications. It consists of code that understands the Excel record formats, and wraps them up in an easy-to-use API.

How easy does HSSF make reading Excel files? See for yourself!

```
InputStream in = new
FileInputStream("data.xls");
HSSFWorkbook wb = new
HSSFWorkbook(in);
HSSFSheet sheet = wb.getSheetAt(0);
// the 1st sheet
HSSFRow row = sheet.getRow(1);
// get the 2nd row
HSSFCell cell = row.getCell((short)1);
// the 2nd cell of the 2nd row
```

The model of an Excel document in HSSF begins with the HSSFWorkbook object. This object provides access to the sheets (by name or number), which in turn provides access to the rows (HSSFRow) in the sheet. Each row provides access to the individual cells (HSSFCell) it contains.

From the cell object you can retrieve data contained in that cell via accessor methods, depending on the type of data. Listing 1 provides an example.

Given this object model, writing is equally simple. Instead of "get"-ing rows and columns, you "create" them and then "set" the values in the cells as in Listing 2.

Once again, start with the HSSFWorkbook class, whose default constructor provides a new workbook object; then populate the workbook by creating a sheet in which you create rows. In each

row create the cells you need. Finally, populate the cells with the data. As Listing 2 shows, a cell can contain integers, floats, strings, and dates.

### Styles

All that is fine, but plain data is usually not sufficient to keep your users happy. HSSF therefore has a whole range of features designed to let you use a variety of styles and formats that Excel supports.

To start applying styles to cells, first create an instance of an HSSFStyle class:

```
HSSFStyle myStyle = wb.createCellStyle();
// wb is an HSSFWorkbook object
```

The style object will now provide you with methods to set various style parameters, such as foreground and background colors, fonts, borders, and data formats, via conventionally named setters.

### Data Formats

A key component of a cell's style is its data format. This specifies, for example, the number of decimal places in a number, or the format of a date. The data format is set using the setDataFormat method of HSSFStyle. This method takes an integer, which is an index to a format, since Excel keeps a list of indexed built-in formats (and user-defined formats are appended to this list and indexed in a similar fashion).

It's easy to get the index, however. For a built-in format, use the static getBuiltinFormat method in the HSSFDataFormat class. Give it the format string and it will return the correct index, the proper index for you. To set a format:

```
myStyle.setDataFormat(HSSFDataFormat.
getBuiltinFormat("d-mmm-yy");
```



Ryan Ackley

has been an active contributor to the POI project for several years.

sackley@cfl.rr.com



Avik Sengupta

is a domain committer on the Jakarta POI project, and is chief technology officer at Itellix Software Solutions.

avik@apache.org

For a user-defined format, first get an instance of `HSSFDataFormat` from an `HSSFWorkbook` object to ensure that your format is registered with the workbook:

```
HSSFDataFormat df =
wb.createDataFormat();
myStyle.setDataFormat(df.getFormat("ddMM%yyyy"));
```

If you don't want to worry about which formats are user defined (it's documented in the Javadocs for `HSSFDataFormat`), simply use the nonstatic method and it will take care of this issue internally.

When you have defined the style you want, just set it to the cell:

```
cell.setCellStyle(myStyle);
```

Reuse the same style object for cells that are similarly formatted – do not create new style objects for each cell, since Excel has an upper limit on the number of styles that can be referenced in a workbook. For example, you could create one style object for the table headers, one for the body, and one for the footer and use them throughout your spreadsheet.

### Formulas

Probably one of the most important features of HSSF is the ability to populate cells with formulas. This allows you to create dynamic spreadsheets and facilitate the user's ability to change the data and perform her own analysis (which is indeed the power of spreadsheets, and the number one reason why you would want to output Excel files).

Formulas are created using the `setCellFormula` method of an `HSSFCell` object. The input to this method is a string containing the formula you want at that cell. It should be in the same format that you would type into the edit box in Excel (without a leading "=",) thus:

```
cell.setCellFormula("A1+A2*2");
```

You could use any built-in VBA function, or even a user-defined function, in the formulas:

```
Cell.setCellFormula("average(A1:B1)");
cell.setCellFormula("mySpecialFunction(A1/A2)");
```

If you need to provide your users with the ability to copy-paste or drag an Excel

formula in the resultant sheet correctly, you might want to use absolute references instead of relative. If formulas with relative cell references (the default, e.g., A1) are copied from one cell and pasted to another, the cell references in the formulas change relative to the destination cell.

```
cell.setCellFormula("$A1/$A$25");
```

However, if the formula contains references that are absolute, they stay the same irrespective of the destination cell. Absolute references are specified by adding a \$ symbol to the reference, viz. `$A$1`. Note that the row and the column can be individually addressed while specifying absolute references, viz. `A$1` vs `$A1`.

You can also reference other sheets in the same workbook in the formula. HSSF does not yet support the ability to write formulas referencing external workbook files.

```
Cell.setCellFormula("SUM(Sheet1!A1-Sheet1!A2)"); // formula in cell A1 of Sheet2.
```

Note, however, that the formula results are *not* calculated by HSSF, which is really a file format reader and writer, not a functional replacement for a spreadsheet application. The formula is merely written into the file in the proper format and evaluated when the file is opened in Excel.

### Finally

Among other advanced features, HSSF allows you to create merged cell regions. You can also set headers and footers for sheets, as well as set print areas, to ensure the data prints well. You can create split and freeze panes, set zoom options, or enable sheet protection. Additionally, you can create and manipulate named ranges. Later versions (see sidebar – **A Guide to POI Versions**) also let you programmatically create drawings in sheets.

However, there are always features of an Excel file that POI does not yet support. In such scenarios, templates are invaluable. The idea is to create an empty Excel spreadsheet populated with the attributes that POI doesn't support. You could, for example, create a chart in the spreadsheet referencing named ranges, or create a pivot table in a certain area. At runtime, in Java code, you could read the workbook in with POI and fill in the cells with data from your application. Now when the user opens the workbook in Excel, it comes

loaded with data, charts, and pivot tables. Listing 3 provides an example.

Hopefully this overview of HSSF has convinced you that HSSF has almost all it takes to create professionally produced Excel spreadsheets that'll be a joy to your users, and leave them asking for more.

## Word Documents with HWPFP

The HWPFP (Horrible Word Processing Format) component of POI is a Java library for reading and writing Word documents. It's still in early beta but is relatively stable and it is the only open source Java solution we know of for programmatically accessing and/or creating a Word document.

I am going to give a short introduction to the high-level structure of a Word document. These are basic concepts that can be applied to most styled document formats and they will make later sections of this article easier to digest.

A Word document can be modeled as a tree-like structure. Figure 1 illustrates this. The document has sections, a section has paragraphs, and a paragraph has character runs. Each instance of these is associated with a range of text.

- A *section* can be correlated with a chapter in a book. A section contains obscure properties like the page border and the number of columns.

## A Guide to POI Versions

As an open source project, POI's development is carried out in a public repository by a group of volunteers. As a result, the code is quite dynamic, and this guide will help you navigate the multiple versions you'll find in the wild. In general, note that releases with beta, dev, or RC attached to their names are flagged as development releases, while releases without these postfixes are flagged as production releases.

The 1.5.1 version released early 2002 was the preferred production version for a long time. But after a long series of new features, followed by a longer period of bugfixes and stabilization, the 2.0 version was released in January 2004.

Subsequently, the 2.5 version was released in late February 2004 to incorporate a major new piece of functionality – the ability to create drawings in Excel sheets via what is known as the Escher Layer.

Meanwhile, development had been ongoing in an experimental branch to enable the reading and writing of Word documents (HWPFP). Unfortunately, it's necessary to download this piece of POI directly from CVS and compile it yourself. There are many excellent and free client applications for accessing CVS repositories such as WinCVS and jCVS.

## Getting Started

Getting started with POI couldn't be easier. Download the version you want from [www.apache.org/dyn/closer.cgi/jakarta/poi/](http://www.apache.org/dyn/closer.cgi/jakarta/poi/) as a zip or tar.gz archive. From the archive extract `poi-<version>-<date>.jar`. Add this file to your classpath and you should be set. POI has an optional dependency on `log4j`, but that's needed only if you turn on logging (which is disabled by default).

- A *paragraph* follows the traditional definition of a paragraph. It contains more familiar properties that most Microsoft Word users know. The justification (left, center, right) and the indent setting are good examples.
- A *character run* is a consecutive run of characters that share the same formatting. These contain the most common and visible properties. Some examples are font family, font size, bold, italic, and underline.

This provides you with enough information to use Java to read and manipulate this model. To get started, we have to create an `HWPFDocument` object from a physical Word file.

```
1 FileInputStream in =
2   new FileInputStream("C:\\test.doc");
3 HWPFDocument doc =
4   new HWPFDocument(in);
```

The `Section`, `Paragraph`, and `CharacterRun` classes represent the document tree that I explained earlier. I walk that tree in Listing 4.

First, I get the `Range` object for the entire document. This is the entry point to the object model. The `Range` class is an important piece of the HWPFF API. It represents an arbitrary range of text in the document, with one to many sections, paragraphs, and character runs. The `Section`, `Paragraph`, and `CharacterRun` classes extend the `Range` class.

The methods `numSections()`, `num-`

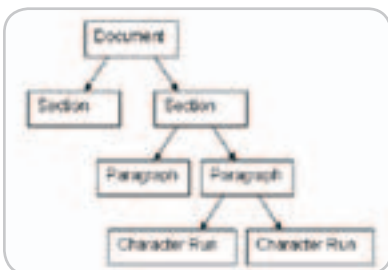


Figure 1 Word doc modeled as a tree-like structure



Figure 2 Word document

`Paragraphs()`, and `numCharacterRuns()` and the correlating getters are actually implemented in the `Range` class. Of course, if you call `numSections()` on a `Paragraph` object, it will return one. That would be the parent `Section` of that `Paragraph` object.

Another important method in the `Range` class is `text()`. This can be used to get the plain text for a particular range. To get the text for a document, use the following code:

```
String plainText =
    doc.getRange().text();
```

Once we have an instance of a `Section`, `Paragraph`, or `CharacterRun` object, we can read its properties by calling its various getters.

```
//Check the number of columns
//for this section
Section sect = r.getSection(x);
sect.getNumColumns();

//See if a paragraph is set to
//have a page break before it.
Paragraph par = sect.getParagraph(y);
boolean breakBefore = par.pageBreakBefore();

//Get the font name of a
//character run
CharacterRun run =
    par.getCharacterRun(z);
String font = run.getFontName();
```

These are quick examples. There are dozens of settings and there isn't enough space to cover them all. I encourage you to read the Javadoc to see what is possible.

### Tables

Behind the scenes, tables are just a group of paragraphs with certain flags set. HWPFF attempts to hide the juicy details but it still needs a little help (see Listing 5).

Listing 5 touches every paragraph in the document, looking for one with the table flag set. When it finds one, it passes it to the `getTable` method on line 8. Notice on line 12 that it's necessary to increment `x` so that the paragraphs that were part of the table aren't processed again.

Tables have `TableRows`, which in turn have `TableCells`. All these classes extend `Range` so you can use all the methods that I've already talked about for getting the contents of these entities.

### Lists

Unlike tables, lists don't have a beginning and an end, because entries in a list can be inserted anywhere in the document and the list numbering can pick up wherever it left off. The `ListEntry` class is used to represent an entry, and it extends the `Paragraph` class. Look at how I get a list entry in the following example:

```
1 for (int x = 0; x < numPars; x++)
2 {
3   Paragraph par =
4     range.getParagraph(x);
5
6   if (par instanceof ListEntry)
7   {
8     ListEntry entry = (ListEntry)par;
9
10    //do something with the entry..
11  }
12 }
```

### Adding New Content

There may be a time when you want to generate new Word documents or modify an existing document using Java. My first word of advice is to make sure that this is absolutely necessary. In most cases, a nonproprietary file format such as PDF, RTF, or HTML is the better choice. There are free libraries available for all of these. In the cases of RTF and HTML, the standard JDK provides the `javax.swing.text` package to manipulate the file formats. A rule of thumb for creating Word documents is: Will the eventual recipients of these documents want to edit them? If not, the PDF or HTML format is a better choice. If they do wish to edit them, consider using RTF instead of the Word file format.

The writing functionality of HWPFF is somewhat experimental so expect some bugs and limited features. Modifying an existing document or creating a new Word document from scratch starts the same way – simply create a new `HWPFDocument` as shown in an earlier example. The only difference is that if you want to create one from scratch, you start with a blank document. The POI distribution comes with one called “blank.doc.”

To commit any changes to a physical file and see what they do, you must write out the modified document. The following code writes out a Word document that contains any changes made to the original object model.

```
FileOutputStream docOut =
    new FileOutputStream(
        "C:\\testout.doc");
doc.write(docOut);
```

To be safe, I wouldn't recommend overwriting the original document. HWPf attempts to keep things that it doesn't directly support in the file, but this doesn't guarantee that they will be there when it writes the file out again.

The Section, Paragraph, and CharacterRun classes define setters that allow the various properties of existing content to be changed. The Range class defines the following methods for adding text and paragraphs to a document.

- **insertBefore(String text):** Inserts a string into the document at the beginning of the Range. Assumes the properties of the character run at the beginning of this range.
- **insertAfter(String text):** Inserts a string into the document at the end of the Range. Assumes the properties of the character run at the end of this range.
- **insertBefore(String text, CharacterProperties props):** Inserts a string into the beginning of the Range with the properties given by props.
- **insertAfter(String text, CharacterProperties props):** Inserts a string into the end of the Range with the properties given by props.
- **insertBefore(ParagraphProperties props, int styleIndex):** Inserts a new empty paragraph at the beginning of this Range. Based on the style at index styleIndex in the stylesheet.
- **insertAfter(ParagraphProperties props, int styleIndex):** Inserts a new empty paragraph at the beginning of this Range. Based on the style at index styleIndex in the stylesheet.

All of the insert methods return the Range that the insertion is now a part of. For example, when inserting a paragraph using insertAfter(ParagraphProperties props, int styleIndex), a Paragraph object is returned. Since Paragraph extends Range, all of the above methods can be used to fill this paragraph with text. The ParagraphProperties and CharacterProperties are similar to the Paragraph and CharacterRun classes. The difference is that classes ending with "Properties" are not associated with a location in a document. There are also SectionProperties and TableProperties.

The methods that insert a paragraph require a style index. Paragraphs and character runs store their settings as del-

tas from a style stored in the stylesheet. Styles provide a convenient way to maintain a consistent look and feel in a document. They also help a person creating a Word document through a user interface to be more efficient. To a programmer this may not matter. No matter what the style is, whatever properties are set for a particular Paragraph or CharacterRun object will appear in the document. I recommend just using the number 0 for a style index. This will always refer to the "Normal" style in the stylesheet.

#### Editing Tables

Because of the complexity, the range class does not currently define methods for inserting tables. However, the TableCell class extends Range, so all of the insert methods defined in Range can be used to add content to the individual table cells of an existing Table.

#### Adding Lists

Adding a list is a little tricky. Unlike most objects in the document, a list is not associated with a range of text. There are paragraphs that are associated with a list and these paragraphs are actual entries in a list. Before an entry can be added to a document, a list must be created. The following code creates a list.

```
1 HWPfList list = new HWPfList(true,
2   doc.getStyleSheet());
3
4 int listID = doc.registerList(list);
```

The HWPfList constructor takes two arguments. The first one is a boolean determining whether the list should be bulleted (if the argument is false, the list will be numbered), and the second is the stylesheet of the document to which the list will belong. The registerList method that I call on the method on line 4 is defined in HWPfDocument. It returns a unique ID that's needed when adding a list entry to the document.

The Range class defines more insert methods for adding list entries.

- insertBefore(ParagraphProperties props, int listID, int level, int styleIndex)
- insertAfter(ParagraphProperties props, int listID, int level, int styleIndex)

What is different from the normal paragraph insert is that both of the above methods require the list ID and the level. The level argument refers to the indent level of the list. At this point, the level argument is ignored because

HWPf only supports writing simple, one-level lists. Figure 2 shows a screenshot of the Word document created using the code in Listing 6.

## Summary

POI has its weaknesses. The biggest by far is the memory consumption in the Excel component (HSSF). The POI team has recognized this problem and is trying to address it in a coming release. The Word component's (HWPf's) biggest problem is that it isn't very mature. Right now it only provides very limited functionality. Even the Excel side of POI could use improvement on its support of some key Excel features, such as charting and images

If POI doesn't cut it, there is a wide selection of commercial libraries for working with Excel, such as SoftArtisans OfficeWriter. SoftArtisans ([www.softartisans.com](http://www.softartisans.com)) is the only vendor I could find that also offers a product that can create Word documents in pure Java. OfficeWriter also supports every feature of Word and Excel.

With the new agreement between Sun and Microsoft, we may one day see the opening of the Microsoft file formats. While you wait for this day to come, POI provides a free open source alternative. ☺

## References

- Apache POI: <http://jakarta.apache.org/poi>
- WinCVS: [www.wincvs.org](http://www.wincvs.org)
- SoftArtisans OfficeWriter: <http://officewriter.softartisans.com/office-writer-240.aspx>

#### Listing 1

```
String value;
switch (cell.getCellType())
{
    case HSSFCell.CELL_TYPE_FORMULA :
        value = "FORMULA "+ cell.getCellFormula();
        break;
    case HSSFCell.CELL_TYPE_NUMERIC :
        value = "NUMERIC value="
            + String.valueOf(cell.getNumericCellValue());
        break;
    case HSSFCell.CELL_TYPE_BOOLEAN :
        value = "Boolean value="
            + String.valueOf(cell.getBooleanCellValue());
        break;
    case HSSFCell.CELL_TYPE_STRING :
        value = "STRING value="
            + cell.getStringCellValue();
        break;
    case HSSFCell.CELL_TYPE_DATE :
        value = "DATE value="
            + cell.getDateCellValue().toString();
        break;
    default :
}
```

**Listing 2**

```
HSSFWorkbook wb = new HSSFWorkbook();
HSSFSheet sheet = wb.createSheet();
HSSFRow row = sheet.createRow((short)0);
HSSFCell cell = row.createCell((short)0);
cell.setCellValue(1); // cell A1
row.createCell((short)1).setCellValue(1.2); //cell A2
row.createCell((short)2).setCellFormula("A1+A2"); //cell
A3 is 2.2
row.createCell((short)3).setCellValue("The next cell is a
boolean, and then a date");
row.createCell((short)4).setCellValue(true); //cell A5
row.createCell((short)4).setCellValue(new Date()); //cell
A6 contains todays date
FileOutputStream out = new FileOutputStream("data1.xls");
wb.write(out);
out.close();
```

**Listing 3**

```
InputStream in = new FileInputStream("data.xls");
HSSFWorkbook wb = new HSSFWorkbook(in); // read in
existing workbook
HSSFSheet sheet = wb.getSheetAt(0);
HSSFRow row = sheet.getRow(0);
if (row == null) row = sheet.createRow(0); // check if
row already exists
HSSFCell cell = row.getCell(0);
if (cell == null) row.createCell(0); // check if
cell already exists
cell.setCellValue(2.5); // update
cell value

FileOutputStream out = new FileOutputStream("data2.xls");
wb.write(out); // and
write it back out.
in.close(); out.close();
```

**Listing 4**

```
5 Range r = doc.getRange();
6
7 int numSections = r.numSections();
8 for(int x = 0; x < numSections; x++)
9 {
10 Section sect = r.getSection(x);
11 int numPars = sect.numParagraphs();
12 for (int y = 0; y < numPars; y++)
13 {
14 Paragraph par = sect.getParagraph(y);
15 int numRuns = par.numCharacterRuns();
16 for(int z = 0; z < numRuns; z++)
17 {
18 CharacterRun run =
19 par.getCharacterRun(z);
20 }
21 }
22}
```

**Listing 5**

```
1 for (int x = 0; x < numPars; x++)
2 {
3 Paragraph par =
4 range.getParagraph(x);
5
6 if (par.isInTable())
7 {
8 Table t = range.getTable(par);
9
10 //do something with the table...
11
12 x += (t.numParagraphs() - 1);
13 }
14}
```

**Listing 6**

```
import java.io.*;

import org.apache.poi.hwpf.*;
import org.apache.poi.hwpf.usermodel.*;

public class Listing1
{
    public Listing1()
    {
    }

    public static void main(String[] args)
    {
        try
        {
```

```
FileInputStream in = new FileInputStream("C:\\blank.doc");
HWPFDocument doc = new HWPFDocument(in);
Range range = doc.getRange();
```

```
CharacterProperties props = new
CharacterProperties();
// Set the font size in half points
Range currentRange = range;

// Slowly increase the font size
for (int x = 8; x <= 64; x += 4)
{
    // Set the half point size of the font
    props.setFontSize(x);
    currentRange = currentRange.insertAfter(" Hello
World!", props);
}
```

```
// Display Bold characters
props.setBold(true);
currentRange = currentRange.insertAfter(" Bold",
props);

// Display Italic characters
props.setItalic(true);
currentRange = currentRange.insertAfter(" Italic",
props);
```

```
// Display charcters with a Double Strikethrough
props.setDoubleStrikeThrough(true);
currentRange = currentRange.insertAfter(" Double
Strikethrough", props);
```

```
// Insert an empty paragraph for readability
currentRange = currentRange.insertAfter(new
ParagraphProperties(), 0);
```

```
// Reset the character properties
props = new CharacterProperties();
props.setFontSize(32);
```

```
// Create a numbered list
HWPFFList list = new HWPFFList(true, doc.get-
StyleSheet());
int listID = doc.registerList(list);
```

```
// Insert a list entry
currentRange = currentRange.insertAfter(new
ParagraphProperties(), listID, 1, 0);
props.setIco24(0xff0000);
currentRange = currentRange.insertAfter(" Blue list
entry", props);
```

```
// Insert another list entry
currentRange = currentRange.insertAfter(new
ParagraphProperties(), listID, 1, 0);
props.setIco24(0xff);
props.setFontSize(38);
props.setCapitalized(true);
currentRange = currentRange.insertAfter(" larger red
capitalized", props);
```

```
//Last list entry
currentRange = currentRange.insertAfter(new
ParagraphProperties(), listID, 1, 0);
props.setIco24(0);
props.setCapitalized(false);
props.setCharacterSpacing(150);
currentRange = currentRange.insertAfter(" Large char-
acter spacing", props);
```

```
// Write out the document
FileOutputStream out = new FileOutputStream("C:\\
hello.doc");
doc.write(out);
out.flush();
out.close();

} catch (Throwable t)
{
    t.printStackTrace();
}
}
```

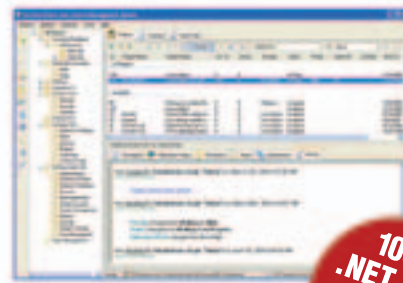


# DEFECTS FIXED. PROBLEMS SOLVED.



## Team-based Defect Tracking and Feature Management Solutions for Windows, Web and as a Hosted Service.

- Management of Defects and Features is performed through an easy-to-use hierarchical view of projects
- Fully customizable Role-Based and Project-Based Security ensures the proper level of access for all users
- Simple and easy-to use workflow allows one-click processing of defects and features from inception to completion
- File and Image attachments can provide users a single source of information to get the whole picture about a defect or feature
- Automatic Audit Trails & History of Changes allows users and administrators the ability to keep track of all modifications
- Extensive filtering capabilities allow users to define and save their own private or public data filters
- PDF Exportable reports allows users to quickly filter and print any list of items based on any public or private filter
- Automatic email notifications for new, changed and deleted items allows users to quickly be informed of their assignments
- Optional Windows Integrated Authentication eliminates the need for users to remember yet another userid and password
- Microsoft SQL Server backend (or MSDE) ensures performance, reliability & scalability as your team grows
- Designed and developed from the ground up using .NET managed code for maximum reliability and performance.
- Optional Web Services SDK allows defect reporting and feature request capabilities to be built into your existing applications



OnTime 2004 Windows Edition



OnTime 2004 Web Edition

100%  
.NET Code  
& SQL  
Server

## Only \$79 Per User

Limited Time: Single-User Version is FREE!  
Download a free copy from [www.axosoft.com](http://www.axosoft.com)

### axosoft

software for software development™

10429 South 51st Street, Suite 205  
Phoenix, AZ 85044, USA

800-653-0024

[www.axosoft.com](http://www.axosoft.com)



CONFERENCE: August 2 – 5, 2004 EXPO: August 3 – 5, 2004

MOSCONE CENTER • SAN FRANCISCO, CA

## LinuxWorld Conference & Expo

is the #1 marketplace for companies that sell, market or promote open source based products, services, applications and solutions. Experiencing astounding growth, it's the largest gathering of open source professionals in the world.

Where  
**OPEN MINDS**  
Meet



CORNERSTONE SPONSOR

**ORACLE**

PLATINUM SPONSORS



**DELL**



**IBM**

**intel**

**Novell**

**Sun**

## Keynote Speakers



### Matthew Smith

Red Hat Chairman,  
Chief Executive Officer  
and President

Tuesday, August 3  
9:15 am - 10:00 am



### Nick Swartz

Senior Vice President,  
Technology and  
Manufacturing, IBM Corp.

Wednesday, August 4  
11:00 am - 11:45 am



### Martin Fliss

Vice President, Linux  
Business, Enterprise Storage  
and Services Business Unit,  
Hewlett-Packard Company

Tuesday, August 3  
11:45 am - 12:30 pm



### Alfred Chang

BEA Systems Founder,  
Chairman and CEO

Wednesday, August 4  
2:30 pm - 3:15 pm



### Michael Steven Burke

Executive Vice President,  
Global Support Services  
& Platform Technologies,  
Oracle Corporation

Tuesday, August 3  
2:45 pm - 3:30 pm



## Attend LinuxWorld and...

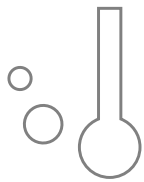
- Evaluate the **latest Linux and open source products and solutions** showcased by the industry's top companies
- Discover **real business solutions** for real business problems
- Network with peers and gain **valuable insight** from open source industry leaders
- Participate in LinuxWorld's world-class educational conference and realize Linux as more than an operating system, but as **a world of applications**
- Learn how Linux can **save your company time and money**



Register Online With Priority Code: **D2001**

[WWW.LINUXWORLDEXPO.COM](http://WWW.LINUXWORLDEXPO.COM)

 IDG  
WORLD EXPO



# VERITAS i<sup>3</sup> for J2EE

Reviewed by  
Rob Halleron

Sometimes as J2EE application developers we feel like we are in a darkened room. We know that something is wrong with our application, but we have no idea where the problem is. Application performance management (APM) tools, such as VERITAS i<sup>3</sup> for J2EE, has helped us “turn on the lights” by enabling us to see exactly where in the application our problem really is. Once we identified the problem, it all flowed from there, as we could look at how the problem affected our application from end to end and make the right decisions on how to fix the problem. VERITAS i<sup>3</sup> APM software is the only solution we found that provides such end-to-end application visibility.

## VERITAS i<sup>3</sup> for J2EE

Application performance management is a continuous process that detects past, current, and future potential application bottlenecks. It finds where the problem resides by drilling down into the application tiers to find the problem's root cause, and it improves application and end-user productivity by helping IT staff to fix problems proactively, before end users are affected. Key parts of the VERITAS i<sup>3</sup> software suite are VERITAS Inform, which provides alerts and reports; VERITAS Insight, which tells you where an application bottleneck is; and VERITAS Indepth, which tells you how to solve the problem.

VERITAS i<sup>3</sup> for J2EE can quickly, efficiently, and unobtrusively capture the metrics necessary to appropriately tune J2EE-based applications. It presents these important metrics in a manner that enables crisp communication, rapid detection, correction, and verification throughout the application's life cycle.

## Installing and Using VERITAS i<sup>3</sup>

VERITAS i<sup>3</sup> for J2EE loads from CDs. We struggled a bit trying to install and configure agents on each tier of the application and then setting up a central “performance warehouse.” Perhaps this was

caused by the fact that our application is hosted by another group within our parent company. However, the rest of our experience was, and remains, fantastic. We began using VERITAS i<sup>3</sup> to test our GENIE holiday (vacation) booking system, and now use it in deployment. The chief thing it does is tell us when we are not meeting

desired service levels – such as the time it takes to serve page content.

VERITAS i<sup>3</sup> for J2EE provides great visibility into application performance problems through a GUI that lets you drill down from an alert to where the problem lies. For example, it understands response time contributions from

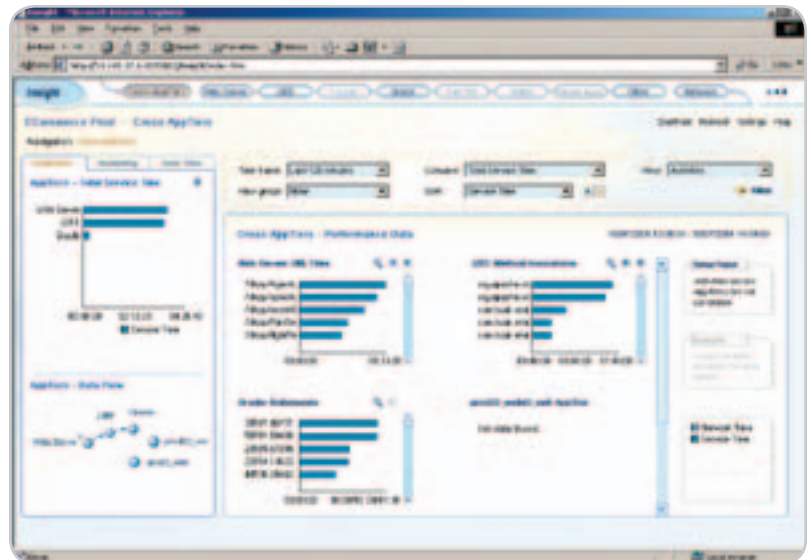


Figure 1 Overview of the Web, application, and database servers

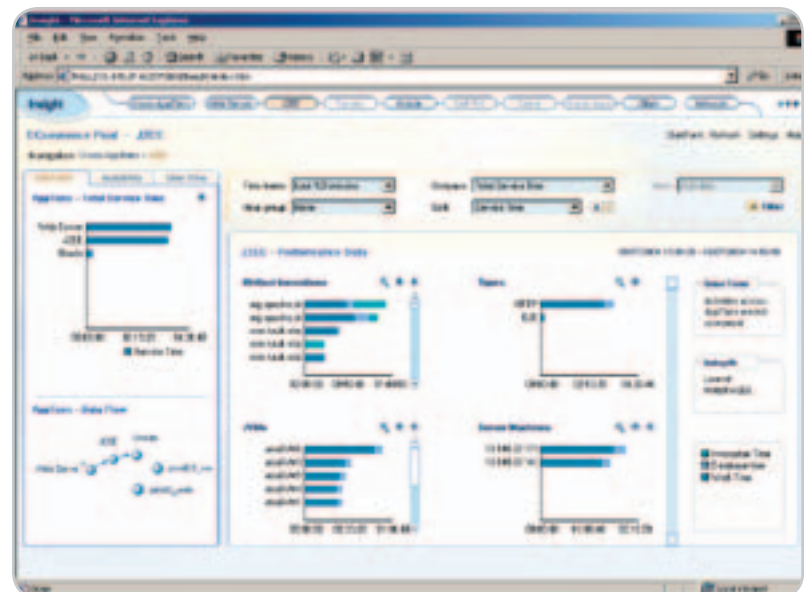


Figure 2 The method invocation graph shows that the logon process could be investigated further



**Rob Halleron** has been a technical architect with Lunn Poly for the past nine years. He was involved with the definition and deployment of its J2EE- and Oracle-based GENIE holiday booking application. Lunn Poly is a leisure travel retailer in the UK with more than 750 retail stores.

rob\_halleron@tui-uk.co.uk



## Unleash the Power of the Application Lifecycle at the 2004 Borland Conference

**The Borland Conference** is a premier event for technical education, focusing on all the technologies impacting software development. With more than 200 technical sessions, you will see how you can facilitate teamwork, enhance productivity, improve quality, reduce costs, cut maintenance time, and accelerate business flexibility and success. Learn how the entire development team can create and deploy better software, faster – with the integrated Borland suite of products for the analyst, architect, developer, tester, deployment group, and manager.

Special discount of up to 50% on select Borland products • Exhibit hall • Free conference proceedings CD • Product Solution tracks covering all Borland products, including JBuilder®, Delphi™, Together®, StarTeam®, CaliberRM™, C#Builder™, C++BuilderX™, Optimizeit™ ServerTrace, Borland® Enterprise Server, Janeva™, and InterBase® • Interest Area tracks, including ALM, Methods, and Processes; Architecture, Models, and Patterns; Microsoft® .NET Framework; J2EE™; SOA; emerging technologies; and more!

September 11-15, 2004 • San Jose, California

For complete conference details and session information: [connect.borland.com/borcon04](http://connect.borland.com/borcon04)

Made in Borland® Copyright © 2004 Borland Software Corporation. All rights reserved. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. • 22127.5

**Borland**<sup>®</sup>  
Excellence Endures™

Java servlets, JSP, EJBs, JMS, JNDI, JDBC, and XML. It correlates activity across Web, multiple JVMs, and DB servers. It also has a SmartTune feature that gives you great advice on how to fix the problem.

For example, a third party wrote part of our application that served up static content about cruise holidays. These pages should have been delivered fast, since they can be stored in cache memory. Using VERITAS i<sup>3</sup> we found the problem was that the application was making a database call for each statement asking for content. The product allowed us to find and fix that problem quickly. In another instance, we were able to identify poorly

performing SQL statements, including one particular query that was running at 0.5 of a second but was occupying one entire processor. We were able to tune this query down to 0.08 of a second.

Starting at the Insight screen in Figure 1, there is an overview of the three layers to the system: Web, application, and database servers. From the graph on the left, most of the time is spent in the J2EE layer. We could investigate that further by choosing the J2EE option on the top menu.

In Figure 2, the method invocation graph appears to show two high usage items but these are part of struts and so will normally be high. However, the third

item is the logon process and should be quick, so this could be investigated further. The JVM etailJVM6 is also more heavily loaded than the others, which may indicate a balance problem.

Clicking on the third item in the list digs into that particular call to reveal these

method invocation graphs gives more details (see Figure 3).

Clicking on the top item in the list digs into that particular call to reveal these

## JDJ Product Snapshot

**Target Audience:** Java application architects/developers and application managers

**Level:** Beginner to advanced

### Pros:

- Understands response time contributions from Java servlets, JSP, EJBs, JMS, JNDI, JDBC, and XML
- Correlates activity across Web, multiple JVMs, and DB servers
- Spans the application cycle (development, testing, deployment)
- Gathers data in real time; stores historical data
- Alerts you in advance of an SLA breach
- Easy-to-use GUI
- Analysis spans entire application, from end user to storage
- Provides advice on how to solve application-performance problems

### Con:

- Difficult installation process needs to be streamlined (I'm told this is remedied in v7, due to ship in Q4 2004)

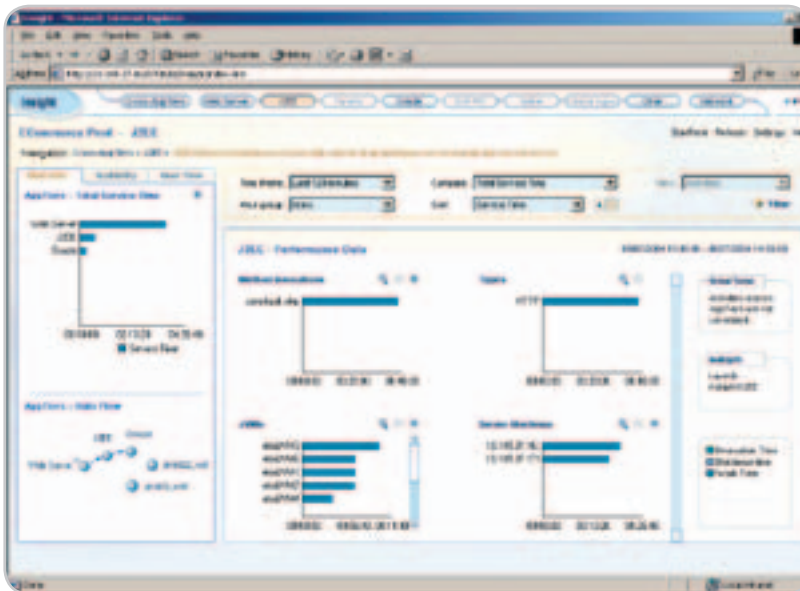


Figure 3 Drilling down gives more detail on the "JVM etailJVM6" logon process

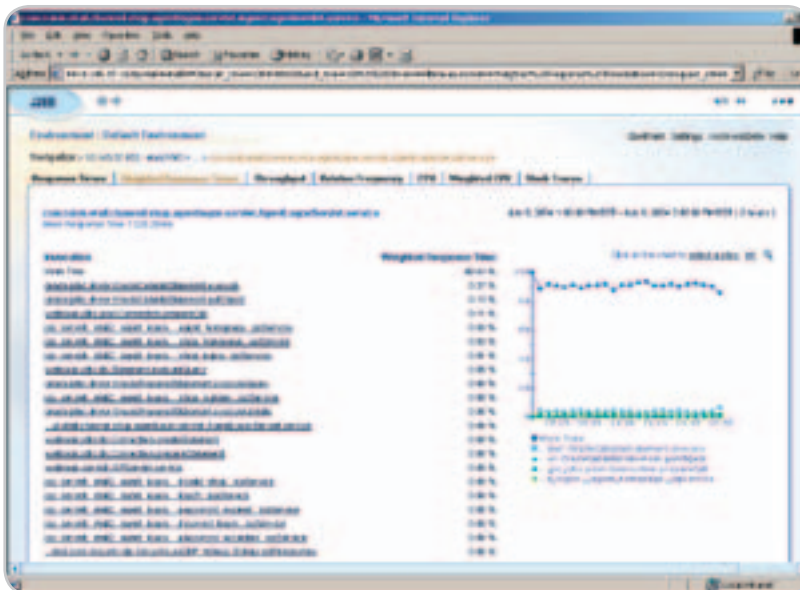


Figure 4 Looking at routines to find out why it is using up the majority of the response time

## VERITAS Software Corporation

350 Ellis Street

Mountain View, CA 94043

Phone: 800 327-2232

650 527-8000 (outside U.S.)

Web: [www.veritas.com](http://www.veritas.com)

## Specifications

**Application Servers:** BEA WebLogic Server 5.1, 6.0, 6.1, 7.0, 8, 8.1; IBM WebSphere 3.5.x, 4.x, 5.x; Oracle 9iAS 9.0.2, 9.0.3; Tomcat 3.x, 4.x; Macromedia JRun 3.x; Sun Java Enterprise System

**Operating Systems:** Sun Solaris 2.6, 7, 8, 9; IBM AIX 4.3.3, 5.1, 5.2; HP-UX 11.0, 11i; Windows NT SP6a, 2000 SP3; Linux Red Hat 7.2, 8 Advanced Server 2.1; SuSE Linux 8.0, Linux S/390

**Pricing:** Based on number of processors and server class.

## Test Environment

Sun Servers (two Web servers: 2 CPU Sun Enterprise 280R; two application servers: 4 CPU Sun V480; database server: F15K 6 CPU domain) running the Solaris 8 operating system

sub calls (see Figure 4). Most of the time is spent local to the routine `com.tuiuk.etail.channel.shop.agentlogon.servlet.AgentLogonServlet.service`. A developer can now investigate why it is using up the majority of the response time.

If we take a step back to Figure 1, we can investigate the top Oracle statement in the graph. Clicking on the top item in the bar graph and then the Oracle tab in the top menu bar takes us to the screen in Figure 5.

We can now launch Indepth for Oracle to determine what the statement is. As you can see in Figure 6, this is a very large INSERT statement that is part of our content-refresh process, so it's not unreasonable for it to take a while to process; nothing to worry about there.

**Summary**

VERITAS i<sup>3</sup> for J2EE is an excellent tool to diagnose and fix J2EE application-performance issues at any point in the application life cycle. Its ability to drill down and find the root cause of your performance issue is superb. If you need end-to-end visibility into your application, this is the ideal solution. ☺

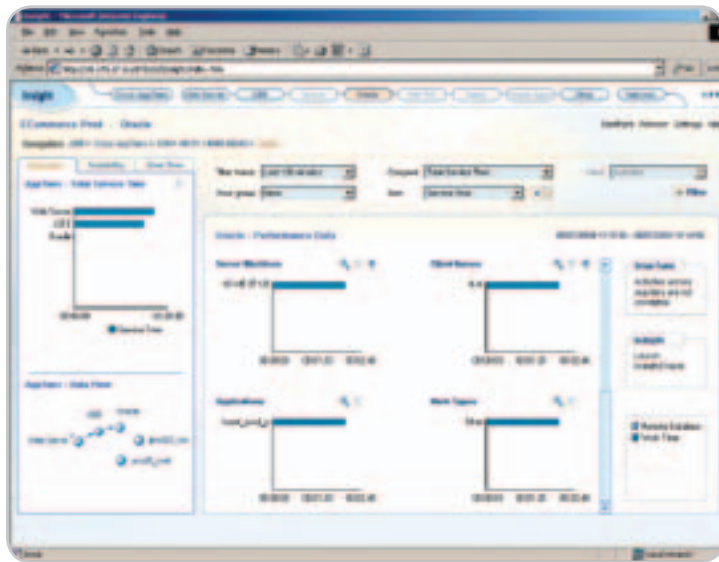


Figure 5 The highest-usage Oracle statement



Figure 6 The INSERT statement takes a while to process

Advertiser	URL	Phone	Page
Altova	www.altova.com	978-816-1600	9
AMD	developer.amd.com	408-749-4000	31
Axalto	www.simagine.axalto.com	888-343-5773	49
Axosoft	www.axosoft.com	800-653-0024	57,65
Borland	www.go.borland.com/j6	831-431-1000	7
Borland Conference 2004	www.connect.borland.com/borcon04		61
ClearNova	www.clearnova.com/thinkcap	770-442-8324	23
Compuware	www.compuware.com		35
Enerjy	www.enerjy.com	866-598-9876	17
Google	www.google.com/cacm	650-623-4000	47
Identify Software	www.identify.com		11
InferData	www.inferdata.com/jdjmag	888-211-3421	25
InterSystems	www.intersystems.com/match6	617-621-0600	4
Jinfontet	www.jinfontet.com/yeehaw	301-838-5560	43
LinuxWorld Conference & Expo	www.linuxworldexpo.com	508-424-4847	58-59
Northwoods Software Corporation	www.nwoods.com/go	800-434-9820	51
Oracle	www.oracle.com/platform	800-633-0753	Cover II
Parasoft Corporation	www.parasoft.com/soapstest	888-305-0041	13
Quest Software, Inc.	http://www.quest.com/jdj	800-663-4723	Cover IV
Rascal Software	www.rascalsoftware.com/java	206-624-7300	15
ReportingEngines	www.reportingengines.com/download/flere.jsp	888-884-8665	19
Scientific Toolworks, Inc.	www.scitools.com		37
Sleepycat Software	www.sleepycat.com/bdbje	510-597-2128	27
Software FX	www.chartfx.com	800-392-4278	Cover III
Tangosol	www.tangosol.com	617-623-5782	21
WebAppCabaret	http://www.webappcabaret.com/jdj.jsp	866-256-7973	41
Web Services Edge 2005 East	www.sys-con.com/edge	201-802-3045	39

**General Conditions:** The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of *Java Developer's Journal*. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in *Java Developer's Journal*. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc.

This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions.

# Industry News

## Quest Manages J2EE Performance Issues with PerformaSure 3.5

(San Francisco) – Quest Software, Inc., a provider of application, database, and Windows management solutions, has announced the release of Quest PerformaSure 3.5, an application-centric diagnostics tool that helps companies tune, diagnose, and resolve performance issues in multitier J2EE applications. Featuring new support for J2EE application servers from Oracle, JBoss, and Apache, and continued support for BEA WebLogic and IBM WebSphere, Quest PerformaSure 3.5 enables companies to utilize its diagnostics capabilities to identify and resolve performance issues in complex J2EE applications, regardless of the application servers they choose.  
[www.quest.com](http://www.quest.com)

## Sun Releases Java Platform Upgrade

(Santa Clara, CA) – Sun Microsystems has announced a significant upgrade to the Java platform and programming language. Known as Project Tiger, the beta release of the Java 2 Platform Standard Edition (J2SE) 5.0 aims to offer easier development, new application monitoring and management features, a dedicated focus on rich client support for the PC desktop, and improved performance.

The J2SE 5.0 software development kit (JDK) includes tools such as compilers and debuggers necessary for developing applets and applications and the Java Runtime Environment (JRE).

Sun also announced that Java Specification Request (JSR) 176 has reached final draft through the Java Community Process (JCP). J2SE 5.0 is based upon JSR 176.  
[www.sun.com](http://www.sun.com)

## DataDirect Technologies to Enhance Sun's Data Connectivity Capabilities

(Bedford, MA) – DataDirect Technologies, a provider of components for connecting software to data, has announced that Sun Microsystems has selected its DataDirect Connect for JDBC suite of drivers to expand the functionality and performance of the Sun Java Studio Creator offering and the Sun Java System Application Server. DataDirect Technologies' JDBC components will enhance Sun's data connectivity capabilities in both development and deployment environments.  
[www.datadirect.com](http://www.datadirect.com)

## ILOG Acquires JLOOX Business from eGENUITY

(Paris) – ILOG, a provider of enterprise-class software components and services, has announced it is acquiring the intellectual property and other selected assets of the JLOOX product line for USD 1.7 million from eGENUITY Technologies Inc., a Montreal, Canada-based maker of software. JLOOX is used for the development of advanced visual applications.

In addition to JLOOX intellectual property, ILOG will acquire the JLOOX customer base and prospects. ILOG also plans to enter into a three-year OEM agreement with eGENUITY that will allow eGENUITY to continue to use ILOG's visualization technology in its STAGE products.  
[www.ilog.com](http://www.ilog.com)

## Borland, eBay, and PayPal to Expand Opportunities for Java Developers

(San Francisco, CA) – eBay and Borland Software Corporation have announced an agreement to provide users of Borland JBuilder with tools and resources that enable them to create Java applications for the eBay and PayPal platforms and communities.

Through the joint distribution agreement, Borland will now make the eBay and PayPal Software Development Kits (SDKs) available to JBuilder developers. The agreement provides the extensive JBuilder developer community with access to code examples and technical resources that are designed to help them build highly available Java applications that tap into the eBay marketplace as well as PayPal's online payment services.

[www.borland.com](http://www.borland.com)  
<http://developer.ebay.com>  
[www.paypal.com/pdn](http://www.paypal.com/pdn)

## Oracle Application Server 10g Enhances Integration with Certification of B2B Standards

(Redwood Shores, CA) – Oracle Application Server 10g is certified to support all leading business-to-business (B2B) standards, enabling organizations to comply with industry mandates required by companies such as Cisco, Intel, Wal-Mart, Home Depot, and Lowe's.

By enhancing support for standards such as EDI over the Internet-AS2 (EDIINT AS2) and RosettaNet, Oracle Application Server 10g enables companies in the high technology, manufacturing, retail, and consumer packaged goods industries to connect to business partners' supply chains using B2B standards. As a result, organizations can meet integration mandates set forth by Cisco, Intel, and Wal-Mart by using Oracle Application Server 10g's pretested connectivity tools and integration features.

[www.oracle.com](http://www.oracle.com)

## Zero G Introduces SolutionArchitect

(San Francisco / Grapevine, TX) – Zero G Software has introduced SolutionArchitect, a software installation and configuration solution for building ready-to-deploy software packages using the new Solution Installation packaging standard. Designed to improve the process of packaging software, SolutionArchitect produces self-configuring and self-healing software packages, and can combine these packaged components together into complete, customized software solutions that can then be deployed using Zero G's multi-platform application deployment solution InstallAnywhere.

[www.zerog.com](http://www.zerog.com)

## Fiorano Is Leading EAI Product for SMB's Report Network Computing Labs

(Los Gatos, CA) – Fiorano Software, Inc., a provider of standards-based integration, business-process management, and enterprise messaging software and solutions, has announced that its Business Integration Suite has been chosen by the Editors of Network Computing (NWC) Labs as the best product for the mid-market EAI segment.

Fiorano's Integration Suite is built on a second-generation Enterprise Service Bus (ESB), in which the logical application design is mapped directly to the physical implementation, making the development process more intuitive and easier than that of conventional integration suites.

[www.fiorano.com](http://www.fiorano.com)

**FIORANO**  
Enabling change at the speed of thought

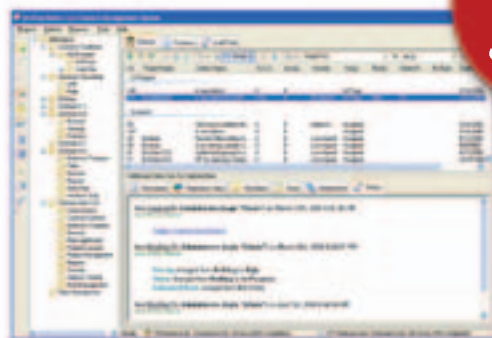




# DEFECTS FIXED. PROBLEMS SOLVED.



**Team-based**  
**Defect Tracking and Feature Management**  
for Windows, Web and as a Hosted Service.



OnTime 2004 Windows Edition

100%  
.NET Code  
& SQL  
Server



OnTime 2004 Web Edition

**FREE SINGLE-USER VERSION**  
Download a free copy from [www.axosoft.com](http://www.axosoft.com)

**No more excuses...**  
**deliver your next project OnTime.**

**axosoft**  
software for software development™

10429 South 51st Street, Suite 205  
Phoenix, AZ 85044, USA

800-653-0024

[www.axosoft.com](http://www.axosoft.com)

# Unified Diversity



Ted Goddard

**T**he network effect is the impetus behind today's software platforms, but a balance must be struck between homogeneous vulnerability and fractured inefficiency. Comparing J2EE to .NET shows clear advantages for J2EE through vendor diversity, portability, standardization community, educational opportunity, language commonality, and security. .NET's attempt to replicate J2EE is shallow, providing technological similarity in a disconnected and proprietary package.

Broadly speaking, the network effect is the growth experienced by networks due to the feedback loop induced by

unified through common standards. This is not only an economic principle; looking at software platforms, we see the network effect on many levels. Let's examine how J2EE and .NET compare.

Everything derives from the human network of education, and both J2EE and .NET technologies are excellent starting points for an education in computer science. The difference is that only J2EE is suitable for a formal curriculum. Unlike the .NET unmaintained prototype in "Shared Source," J2EE code is freely available in its entirety for educational and research purposes. Academic integrity is preserved only when full examination and

integration point that gives developers and deployers choices at every stage. In opposition, .NET holds up a single supplier, eager to collect high taxes and exercise control.

For the long term, the most important network is that of the platform developers. How do people work together to define the standards and technologies that make up a platform? The Java Community Process may not be as fair or as open as some would like, but fundamentally it does provide a way for developers and vendors to reach a consensus and participate in the evolution of J2EE. In contrast, platforms imposed by dictatorship are technology mo-

“ For the long term, the most important network is that of the platform developers. How do people work together to define the standards and technologies that make up a platform? ”

the increasing value of joining a growing network. Consider fax technology. It has been successful because the network of fax machines, connected by the telephone system, communicates reliably – thanks to a common standard. The adoption of fax machines showed runaway growth because, in a sense, the value of a fax machine increased as the size of the entire fax network increased.

Such networks may start slowly, but when they do succeed the effect is dramatic. Clearly, interoperability is crucial to their success. Why not guarantee interoperability by insisting on a single manufacturer, a fax machine monopoly?

The reasons against this have been repeatedly established: an absence of competition leads to lower quality, higher prices, a lack of innovation, and a vulnerable system. The free market demands a diversity of suppliers,

discourse are encouraged.

Isn't .NET better for teaching? With its Common Language Infrastructure, it can be used to teach any language. However, this is a dangerous illusion. .NET reduces the interesting differences of programming languages to a syntactic tower of Babel. Exposure to a variety of languages is an essential part of every education, but they must be seen in their true form to be of value. In contrast, Java makes no such claim of universality; it simply unites developers with a common language, allowing them to effectively share source code and ideas. The network of developers is connected through Java, not through compiled bytecode.

Less abstract is the network of middleware, virtual machine, and operating system suppliers. Once again, the free market demands that this be a diverse collection. J2EE, through its focus on portability, provides an

nopolies, leading to lower quality and lost innovation. Herein lies the tragic flaw of .NET. Publishing a document and declaring it to be a standard omits the peer-refinement process, and the technology becomes little more than a tool of its owner.

Finally, let's turn to the unfortunate reality of our somewhat hostile world. Billions of years of evolution have demonstrated the catastrophes that await homogeneous systems with their replicated single points of failure. Today we see this in the explosive growth of software viruses. Some are merely reckless, some have criminal intent, but all are destructive. J2EE provides a system designed from the ground up for security and is deployable on a substrate that is robust through its diversity. The foundation of the alternative is well traveled by worms. This too serves as a lesson that the network effect must be approached wisely. ☛

**Ted Goddard** is a senior software architect at ICESoft.

Prior to ICESoft, he held positions at Java Software, Sun Microsystems, in device management, and as an XML architect at Wind River Systems.

Ted received his PhD in mathematics in 1996 from Emory University in Atlanta, Georgia.

ted.goddard@icesoft.com

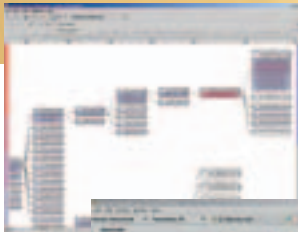
# FREE JAVA CHARTS!

Download the Chart FX for Java Community Edition now.

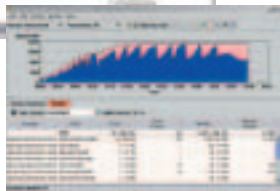


[www.chartfx.com](http://www.chartfx.com)

SPEND LESS TIME PROBLEM SOLVING... AND MORE TIME DEVELOPING APPLICATIONS.



PerformaSure – a system-wide performance diagnostic tool for multi-tiered J2EE applications running in test or production environments.



JProbe – a performance tuning toolkit for Java developers.

**Join The Thousands of Companies Improving Java Application Performance with Quest Software.**

Whether it's a memory leak or other performance issues, Quest Software's award-winning Java products — including JProbe® and PerformaSure™ — help you spend less time troubleshooting and more time on the things that matter. Quest's Java tools will identify and diagnose a problem all the way down to the line of code, so you no longer have to waste time pointing fingers or guessing where the problem lies. Maximize your team's productivity with Quest Software by downloading a free eval today from <http://www.quest.com/jdj>.



© 2004 Quest Software Inc., Irvine, CA 92618 Tel: 949.754.8000 Fax: 949.754.8999