

UNLOCKING THE SECRETS OF THE JMF

# JAVA<sup>TM</sup> DEVELOPER'S JOURNAL

*The World's Leading Java Resource*

Volume: 5 Issue: 4, April 2000

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



# Understanding

# EJB Transactions

RETAILERS PLEASE DISPLAY  
UNTIL JUNE 31, 2000  
\$4.99US \$6.99CAN

Exception  
Chaining  
Simplifies  
Debugging

Python  
Programing  
in the JVM

Java  
Servlets:  
Design  
Practices

Unlocking  
the  
Secrets of  
the JMF

# Microsoft

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

# Protoview

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

# Middleware

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

**SYS-CON PUBLICATIONS**

**CONTACT ESSENTIALS**

**SUBSCRIPTION HOTLINE**  
**1 800-513-7111**

International Subscriptions  
& Customer Service Inquiries

**914 735-1900**

or by fax: **914 735-3922**

E-mail: [Subscribe@SYS-CON.com](mailto:Subscribe@SYS-CON.com)

<http://www.SYS-CON.com>

Mail All Subscription Orders or  
Customer Service Inquiries to:

**PowerBuilder Journal**

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

**JAVA DEVELOPERS JOURNAL**

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

**COLD FUSION Developer's Journal**

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

**JBuilder Developer's Journal**

<http://www.JBuilderJournal.com>

**TANGO DEVELOPERS JOURNAL**

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

**XML JOURNAL**

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

**CUSTOMER SERVICE**

Phone: 914 735-1900 • Fax: 914 735-3922

**ADVERTISING & SALES**

Phone: 914 735-0300 • Fax: 914 735-7302

**EDITORIAL DEPT.**

Phone: 914 735-7300 • Fax: 914 735-6547

**PRODUCTION/ART DEPT.**

Phone: 914 735-7300 • Fax: 914 735-6547

**WORLDWIDE DISTRIBUTION by**

Curtis Circulation Company  
739 River Road, New Milford, NJ 07646-3048  
Phone: 201 634-7400

**DISTRIBUTED in the USA by**

International Periodical Distributors  
674 Via De La Valle, Suite 204  
Solana Beach, CA 92075  
Phone: 619 481-5928

[JAVA DEVELOPERS JOURNAL.COM](http://JAVADEVELOPERSJOURNAL.COM)

# JAVA DEVELOPERS JOURNAL

# JDJ CONTENTS

VOLUME: 5 ISSUE: 4, APRIL 2000

FEATURE

## Guide to Using Spreadsheets on the Web

*How to enhance e-business and B2B processes*



COVER STORY

## Understanding EJB Transactions

*Transactional scope is an important part of understanding*

EJB HOME

## What do MTS and EJB Have in Common?

*A stateless component model, which can meet the demand*



FEATURE

## Best Practices for JDBC Programming

*How to further the goals of best practices*

CORBA CORNER

## Improving Reliability and Scalability

*Ease the pain of migrating to a new middleware*



FEATURE

## Basic Object-Oriented Concepts in HTML

*A purpose-built tool for teaching computer science newbies*

FEATURE

## Multimedia Evolution or Revolution? PART 3

*A revealing look at the Java Media Framework 2.0*



FROM THE EDITOR  
**THEY MIGHT BE GIANTS 8**  
by sean rhody

E-JAVA  
**ANATOMY OF A JAVA APPLICATION SERVER 78**  
by ajit sagar

READER FEEDBACK  
**98**

STRAIGHT TALKING  
**THE WHOLE WORLD IS TURNING AMAZON! 20**  
by alan williamson

DISTRIBUTED SOLUTIONS  
**DISTRIBUTED TASKING IN JAVA 86**  
by sam mckenna

BOOK EXCERPT  
**SERVLETS & JDBC 102**  
by alan williamson

PRODUCT REVIEW  
**LINGOGUI 1.1 64**  
by jim milbery

JDJ NEWS  
**96**

IMHO  
**JAVA TECHNOLOGY COMES OF AGE 118**  
by ed lycklama

# TogetherSoft

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

SEAN RHODY, EDITOR-IN-CHIEF



# They Might be Giants

One of the frustrations of editing a monthly magazine, as opposed to a daily newspaper, is that I seldom get to scoop the rest of the press. With our lead times, breaking news is more or less old by the time you hear it from me. So by now you've heard that Corel has merged with Inprise, maker of my favorite IDE, JBuilder.

Corel has been quietly positioning itself as the new challenger to Microsoft's monopoly on the PC desktop. Capitalizing on the open-source revolution that is Linux, Corel has brought forth several different offerings to compete with Microsoft.

There are a good number of Linux vendors, but Corel has a fairly unique position among them. If you looked at it strictly from an operating system standpoint, you might pass on Corel in favor of Red Hat, which is the mindshare leader in the Linux world.

But operating systems are only a part of what makes Microsoft great – or greatly feared, depending on your viewpoint. Strong, integrated software and development tools, such as Office, C++, Visual Basic and SQL Server, round out the story for Microsoft. The ability to one-stop shop is a strong selling point for many IT shops, more because of the reduced finger pointing than overall suitability to task.

Seen from that viewpoint, Corel is building a case to be your one-stop vendor. For productivity purposes they offer WordPerfect Office. Die-hard devotees of WordPerfect can get an operating system with more power and stability than Windows 98, and support from a single source.

The addition of Inprise to the team brings a whole new level to the development aspect of Corel. While public domain tools have long been available on Linux, the addition of JBuilder, C++ Builder and Delphi to the Linux platform will bring the commercial-grade development tools needed to do serious development. Inprise also brings a CORBA ORB, an EJB server and a moderately powerful SQL database server.

Obviously, Corel faces significant technical challenges. Porting JBuilder shouldn't be too difficult, seeing that it already runs on Solaris and is about 80% pure Java. Moving C++ Builder and Delphi may be more difficult, as their IDEs are Windows based. But Corel seems up for the task.

To round out the company and make it truly a competitor of Microsoft, some additional acquisitions are definitely needed. A strong SQL database is necessary to compete with SQL Server. Who better to acquire for the task than the originator of that product – Sybase? Over the past two years I've watched the company as its stock has slowly rebounded from the basement to a more respectable position in the mid-20s (I should have bought at 4). Their new CEO has stemmed the worst of the bleeding and the company is making significant inroads with its mobile database. A Corel purchase or merger, followed by a spin-off of the Powersoft group, would give Corel the database muscle and hard-core UNIX expertise to continue to innovate and integrate their product line.

A strong EJB server would also help make them a key market player. Persistence and BEA may be out of reach, but companies like Secant or possibly Iona might round out the product line nicely. Can you imagine getting any ORB you like, as long as it's from Corel?

Making this all work will be an interesting task for Corel. I like Linux, and run it on one of my machines at home. It's powerful, gets better mileage from the CPU and has finally gotten a modern interface to rival Windows (I use KDE, but I also like Gnome). But Linux is still hard to install. It doesn't recognize a lot of hardware, and it's hard to reconfigure if you add something (just try changing a network card). Ease of installation and use, coupled with strong technical support, is what will make or break Corel's dream of being a giant. In the meantime, I'll be waiting for my copy of JBuilder for Linux. ☺

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

AUTHOR BIO

Sean Rhody is the editor-in-chief of Java Developer's Journal. He is also a principal consultant with Computer Sciences Corporation where he specializes in application architecture – particularly distributed systems.

# JAVA DEVELOPER'S JOURNAL

## EDITORIAL ADVISORY BOARD

TED COOMBS, BILL DUNLAP, DAVID GEE, MICHEL GERIN,  
ARTHUR VAN HOFF, JOHN OLSON, GEORGE PAOLINI,  
KIM POLESE, SEAN RHODY, RICK ROSS,  
AJIT SAGAR, RICHARD SOLEY, ALAN WILLIAMSON

EDITOR-IN-CHIEF: SEAN RHODY  
EXECUTIVE EDITOR: M'LOU PINKHAM  
ART DIRECTOR: ALEX BOTERO  
PRODUCTION EDITOR: CHERYL VAN SISE  
SENIOR EDITOR: JEREMY GEELAN  
ASSOCIATE EDITOR: NANCY VALENTINE  
EDITORIAL CONSULTANT: SCOTT DAVISON  
TECHNICAL EDITOR: BAHADIR KARUV  
PRODUCT REVIEW EDITOR: ED ZEBROWSKI  
INDUSTRY NEWS EDITOR: ALAN WILLIAMSON  
E-COMMERCE EDITOR: AJIT SAGAR

## WRITERS IN THIS ISSUE

DEREK C. ASHMORE, LINDEN DECARMO, ED LYCKLAMA,  
SAM MCKENNA, DANIELA MICUCCI, JIM MILBERY, SEAN RHODY,  
AJIT SAGAR, TODD SCALLAN, MARK SPENCER, ANDREA TRENTINI,  
SAMEER TYAGI, JASON WESTRA, ALAN WILLIAMSON

## SUBSCRIPTIONS

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

SUBSCRIPTION HOTLINE: 800 513-7111

COVER PRICE: \$4.99/ISSUE

DOMESTIC: \$49/YR. (12 ISSUES) CANADA/MEXICO: \$69/YR.  
OVERSEAS: BASIC SUBSCRIPTION PRICE PLUS AIRMAIL POSTAGE  
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: \$12 EACH

PUBLISHER, PRESIDENT AND CEO: FUJAT A. KIRCAALI  
VICE PRESIDENT, PRODUCTION: JIM MORGAN  
VICE PRESIDENT, MARKETING: CARMEN GONZALEZ  
ACCOUNTING MANAGER: ELI HOROWITZ  
CIRCULATION MANAGER: MARY ANN MCBRIDE  
ADVERTISING ACCOUNT MANAGERS: ROBYN FORMA  
MEGAN RING

JDISTORE.COM: JACLYN REDMOND  
ADVERTISING ASSISTANT: CHRISTINE RUSSELL  
GRAPHIC DESIGNERS: JASON KREMKAU  
ABRAHAM ADDO

GRAPHIC DESIGN INTERN: AARATHI VENKATARAMAN  
WEBMASTER: ROBERT DIAMOND

WEB SERVICES CONSULTANT: BRUNO Y. DECAUDIN  
WEB SERVICES INTERN: DICANT B. DAVE

CUSTOMER SERVICE MANAGER: CAROL KILDUFF  
CUSTOMER SERVICE: ANN MARIE MILLILLO  
ONLINE CUSTOMER SERVICE: AMANDA MOSKOWITZ

## EDITORIAL OFFICES

SYS-CON PUBLICATIONS, INC.

39 E. CENTRAL AVE., PEARL RIVER, NY 10965  
TELEPHONE: 914 735-7300 FAX: 914 735-6547  
SUBSCRIBE@SYS-CON.COM

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944)  
is published monthly (12 times a year) for \$49.00 by  
SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306.

Periodicals Postage rates are paid at  
Pearl River, NY 10965 and additional mailing offices.

POSTMASTER: Send address changes to:  
JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,  
39 E. Central Ave., Pearl River, NY 10965-2306.

## © COPYRIGHT

Copyright © 2000 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, SYS-CON Publications, Inc., reserves the right to revise, republish and authorize its readers to use the articles submitted for publication.

## WORLDWIDE DISTRIBUTION BY CURTIS CIRCULATION COMPANY

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

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.

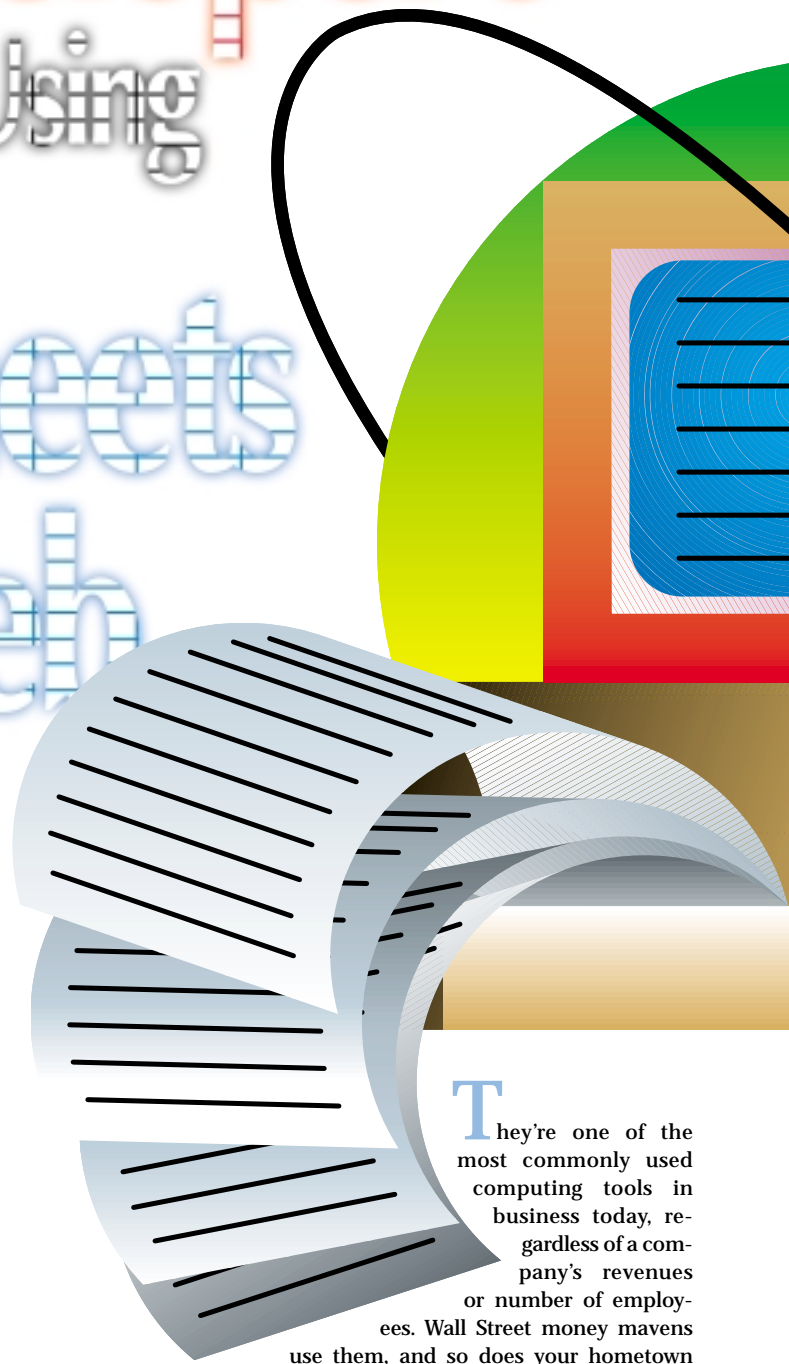


SYS-CON  
PUBLICATIONS

# Java Developer's Guide to Using Spreadsheets on the Web

WRITTEN BY MARK SPENCER

How to enhance e-business and B2B processes by leveraging Excel-like power in multitier applications



**T**hey're one of the most commonly used computing tools in business today, regardless of a company's revenues or number of employees. Wall Street money mavens use them, and so does your hometown accountant...spreadsheets.

Did you ever tweak spreadsheet formulas to play "what-if" games with your stock portfolio? Have you ever received an e-mail with a spreadsheet attachment full of sales figures or forecasts? And by the way, how did your friend send you the results of this week's fantasy football league? Thanks largely to the widespread distribution of Microsoft Office and Microsoft Excel, the spreadsheet is perhaps the top data analysis and reporting tool in the industry.



Distinguished by a rows-and-columns interface that even the casual computer user can understand, spreadsheets provide a wide range of functionality that many types of businesses can leverage. They present data in an understandable format, provide intuitive interfaces for data collection, deliver fast calculations, and report and analyze data from databases and other sources.

Given their popularity and effectiveness, it's logical to use spreadsheets for data analysis when building Java-based e-business and B2B applications. As you'll see later, they're especially effective when

managers establish their own business rules within the application's spreadsheets to notify them when fund prices meet certain levels. When the fund managers need to alter the rules to account for changing market conditions, no one on State Street's development team needs to get involved. Instead the fund managers change the rules in the spreadsheets by themselves. The same scenario is entirely possible using Java spreadsheets on the server as templates for incoming data. When the rules of a business change, a developer or end user can simply provide a new spreadsheet template that contains the new business logic, thereby reducing application development time and costs.

Another use of spreadsheets within Java development is to leverage their inherent analytical capabilities. Especially in applications that call for specialized computations, the formulas built into spreadsheets, together with their engineering, statistical and financial functions, can save developers substantial time. And because most spreadsheets also provide a charting component for additional data display options, developers no longer need to worry about providing that functionality from scratch either.

## Where Spreadsheets Are Used

Many corporations are currently using spreadsheets in multitier architectures in vertical industries such as finance, banking, energy, insurance, retail, securities, software, and many others. Spreadsheets can be used to deliver various types of applications including analysis and reporting, billing and invoicing, cost estimating, expense tracking, fund processing, risk management, sales forecasting, and others.

Some current examples of where and how businesses are using spreadsheets in multitier architectures include:

- An investor services company locates spreadsheets on its server to calculate a constant stream of stock data from databases and other sources. Users can access the data with their browsers at any time of day for up-to-date portfolio valuations.
- A financial firm locates spreadsheets on its server to compile data from different sources into an understandable, easy-to-use file format. The reports are compiled from the previous business day's activity and e-mailed to managers every morning.
- An insurance company customizes spreadsheet interfaces that mimic paper forms. The spreadsheets are deployed in browsers where users can get instant calculations and feedback on the data they input.

There are many more examples but as a general rule any situation that calls for intuitive data-display and -entry, fast calculations, and robust data reports and analysis is one in which spreadsheets can be used to advantage.

## When to Use Spreadsheets

Spreadsheets are at their best when calculating and analyzing a large amount of data quickly. This was, after all, their original purpose; their architecture was developed for performing this exact task.

A developer can determine whether or not an application's requirements truly constitute a large amount of data. For an application with simple calculations involving a small number of variables, a spreadsheet calculation engine obviously might not be necessary. But for volumes of

used to build servlets and Java Server Pages. In this capacity spreadsheets can play the vital role of constructing business logic and rules for data analysis in the middle of Web-centric, three-tier architectures (see Figure 1).

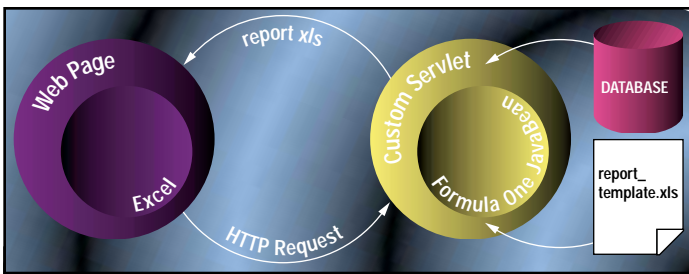
## Why Use Spreadsheets?

The great advantage to using spreadsheets as a component of a larger application is that they're perhaps the single most widely utilized and understood productivity tool in the business world today. Most users, especially those in financial institutions, already know enough about them to construct business rules with them, even if they don't have higher-level programming skills such as knowing how to use SQL or stored procedures. Spreadsheets can ease the complexity and lower the maintenance costs of an application by delegating the construction of business rules to the user.

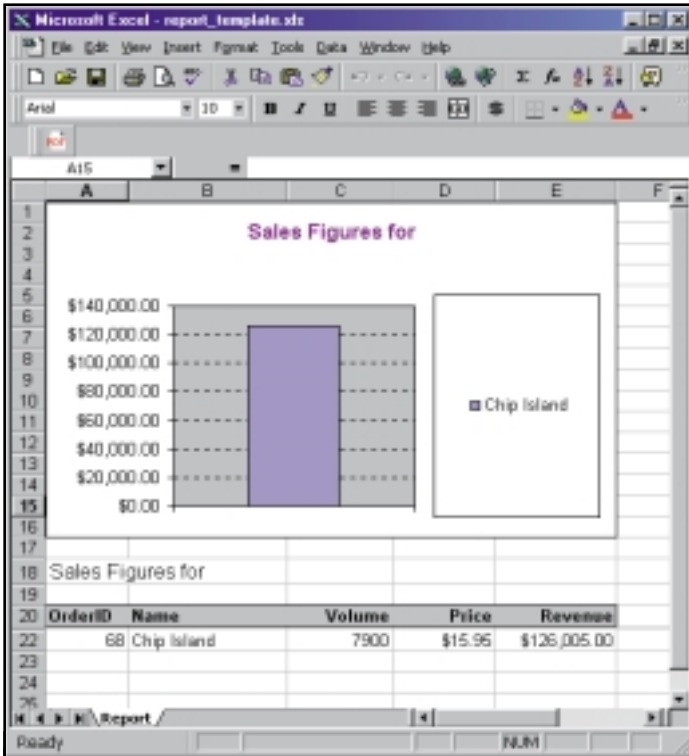
State Street Corporation in Boston, for example, makes extensive use of spreadsheets in an application called NAVAlert. State Street's fund



FIGURE 1 Spreadsheets provide an effective means for building business logic in three-tier architectures.



**FIGURE 2** In this example, a servlet receives input from a Web form, dynamically generates an Excel spreadsheet from a database, and delivers it through a browser to Excel on the client.



**FIGURE 3** This is the report\_template.xls that is used to generate the reports in these examples.

data covering a long time period and typically collected from a database, a spreadsheet provides an extremely manageable paradigm for calculations and a useful grid interface for presenting and manipulating data.

Using spreadsheets for calculations is especially effective in server-side applications. For example, data from several sources might be poured into a spreadsheet for analysis before being passed to clients or stored in yet another database. Numerous financial services companies use spreadsheets on the back end for just this purpose.

In a nutshell, one of the strongest facets of spreadsheets is that they can increase efficiency by performing calculations and data analysis on the server. This is especially true in distributed, three-tier computing environments that rely on Java technology.

## Options for Using Spreadsheets in Java Development

Java developers have several options for implementing spreadsheet functionality in their projects. Several factors affect which option to choose. Most noteworthy is probably the presence (or absence) of Excel in a distributed environment.

Since familiarity with using Excel is widespread, it makes sense to leverage the Excel application, the Excel file format and the expertise of end users whenever possible. But Excel has limitations for Java developers who deploy applications widely on the Web outside firewalls. These

limitations include a dependence on Windows, a limited API, the inability to be embedded and deployed within an application and the inability to leverage advances in Web, application and database servers. These issues noted, developers can choose one of the following options:

- Java developers who aren't concerned about pure platform-independent solutions can use Microsoft's Visual J++ to access Excel; however, this provides a limited API to work with and will require execution of the Java application/servlet in the Microsoft JVM. This also requires that Excel be present on the desktop, as Excel is unable to be embedded as part of an application.
- A cross-platform solution can be built using the Java Native Interface (JNI) to communicate through a C++ implemented COM wrapper to Excel. While this technique is independent of JVM, it still requires native code and thus is tied to a specific hardware configuration. And again, this requires Excel to be present and provides a limited API to customize functionality.
- A third option is to use a JavaBean component that delivers Excel-compatible spreadsheet functionality, such as Formula One 7.0 from Tidestone Technologies. Formula One has been certified as 100% Pure Java, which means that you can use Excel files on non-Windows platforms. It has a defined and documented role as a JavaBean component and can be embedded in distributed applications. Formula One's API is easily accessible with Java code and allows you to embed a spreadsheet engine in any tier. It can leverage advances in Java's use with Web, application and database servers – and it doesn't rely on Excel to be present on the desktop to function.

Formula One is especially effective in the middle layer of three-tier applications where it can be used to construct business rules, perform analysis and distribute data at the core of Java applications, servlets, applets and JavaServer Pages. In these architectures it can be used to access databases through JDBC, perform calculations and analysis, and distribute the results between any client/server environment: HTML for thin clients, Excel for Excel clients or live spreadsheet-powered applets for "heads-down" users who require interfaces more robust than static HTML.

With these abilities in mind, the examples in the remainder of this article are intended to show how to build Java spreadsheet solutions in environments with Formula One – with and without Windows and Excel being present.

## Leveraging Excel on the Desktop With Server-Side Java and MIME Types

The first example shows how to dynamically create a spreadsheet on a server and deliver its contents to the client through a browser connection. By setting a MIME type in the servlet code, Excel will be launched within the browser, giving users the ability to perform further analysis on the spreadsheet using Excel on their desktops. The architecture of this example is shown in Figure 2.

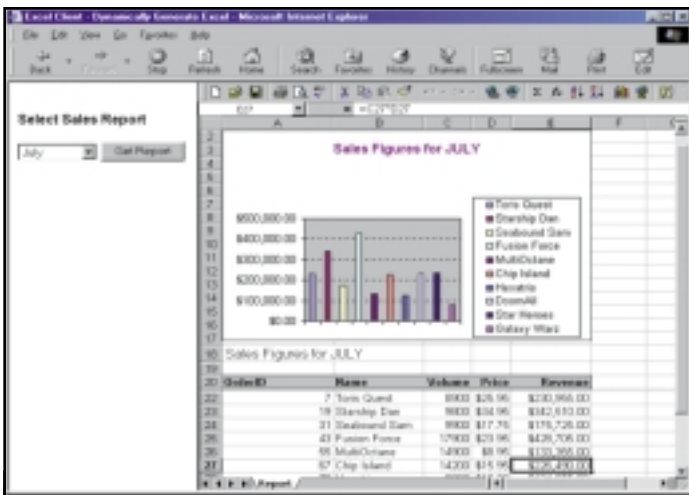
These examples use an Oracle8i database with 12 months of sales figures for a fictitious company. It's assumed a Web page has been built with a form allowing users to select the month of sales figures they'd like to receive in an Excel file. The form has a drop-down box with its NAME attribute set as "month" and its action pointing to the servlet, ExcelServlet.

The ExcelServlet reads in a prebuilt Excel file to initialize the in-memory spreadsheet – report\_template.xls, see Figure 3 – connects to the database to populate selected spreadsheet cells from data returned from a JDBC query, and then writes the spreadsheet to the servlet output stream with an appropriate MIME type. This MIME type, "application/vnd.ms-excel", forces the browser to load the Excel plug-in and display the spreadsheet.

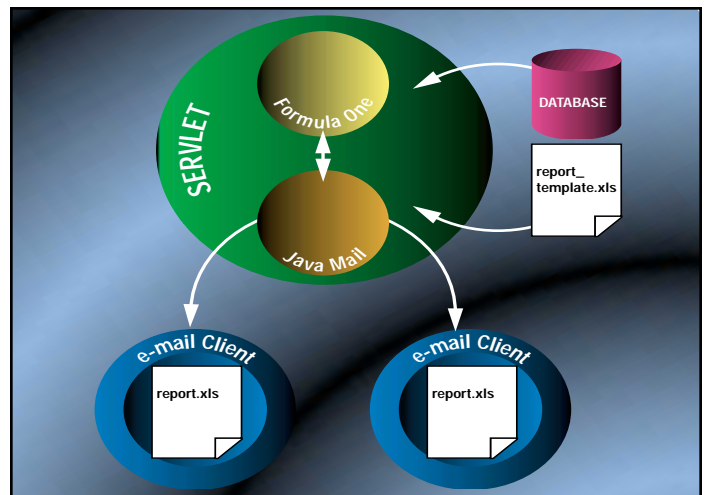
The database-related procedures have been modularized in the dbManipulations.java class (see Listing 1) and the servlet specific code is shown in ExcelServlet.java (see Listing 2). The class dbManipulations.java connects to the database via JDBC and populates rows in the report\_template.xls spreadsheet based on the user input from the Web page. The ExcelServlet.java class provides the servlet "plumbing" to write the Excel data to the browser after the model has been dynamically created.

# Computer Job Store

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



**FIGURE 4** By using Java, it's possible to leverage Excel on users' desktops, even if there isn't a Windows-based server involved on the back end.



**FIGURE 5** In this example, Formula One collects data from a database through JDBC, creates an Excel file and mails the report as an attachment with the JavaMail API to Excel clients.

Invoking the ExcelServlet from a browser with month and report-Template parameters should show a Web page like the one depicted in Figure 4. This demonstrates one way developers can leverage spreadsheets in the middle tier and Excel on clients' desktops, even if the application's architecture doesn't include a Windows-based server.

### Leveraging Excel on the Desktop with Server-Side Java and JavaMail

Another option to consider is e-mailing clients the Excel file generated by Formula One on the server using the JavaMail API. This example of Formula One is ideal for developers who have a large number of desktops with Excel and a large number of Excel-savvy users who require time-sensitive data on a recurring basis. For instance, perhaps reports from the previous day's business activity could be e-mailed to managers every morning or portfolio reports could be sent to investors at the close of trading each day. While this particular example requires a Web server, it would be easy to convert this example into a stand-alone application, which could be scheduled to run in the background on a daily basis.

The addition of JavaMail notwithstanding, this implementation is similar to the prior example and is illustrated in Figure 5.

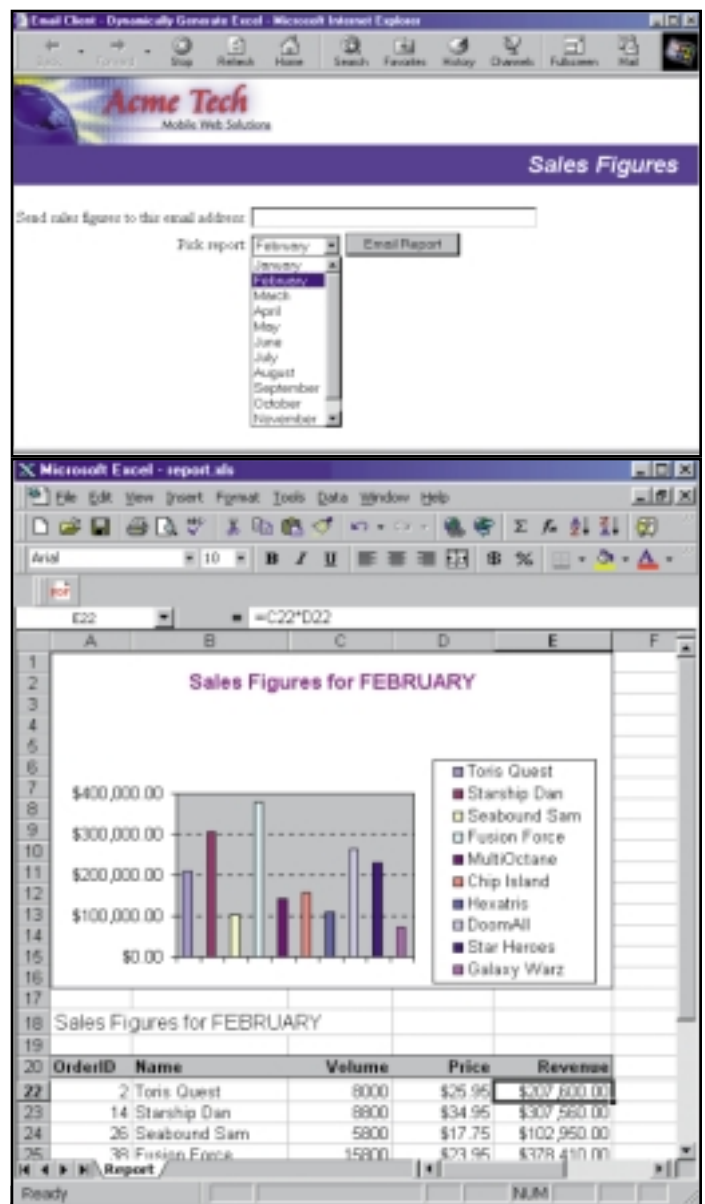
First, it's assumed there's a Web page with a form that asks for an e-mail address where the report will be sent and a month of sales figures for Formula One to generate. The page and resulting spreadsheet that is sent by e-mail could look like those in Figures 6 and 7.

(Note: This demo would also work by accessing the servlet with a name=value pair that includes the parameters &to=name@domain.com&month=SelectedMonth.)

Embedded into a servlet, Formula One uses a spreadsheet created in Excel as a template – for this example we'll use the template in Figure 3 from the prior demo – then connects to the database and populates specified cells with the data values returned from the JDBC call. From there, it performs a recalculation of the new data, writes the spreadsheet as an Excel file and e-mails it to the submitted e-mail account using the standard JavaMail API. The code required to perform these actions can be found in Listing 3, WebMail.java. As with the prior example, the dbManipulations.java class of Listing 1 includes the modularized database code and will be needed to compile this servlet.

### No Excel? How to Deliver Spreadsheets to Thin and Non-Windows Clients

In Web-based computing developers often don't have control over which platform their solutions might eventually operate on. In this situation a cross-platform applet with Formula One could enable the delivery of live spreadsheets to the browser. While this is a plausible option, slow connections and low bandwidths are sometimes a concern, so



**FIGURES 6 & 7** This is the Web form (top) and spreadsheet (bottom) that is sent to the designated recipient in this example.

# Gemstone

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

developers must be sure their applications minimize download times. In cases like this it makes sense to perform spreadsheet calculations and data manipulation on the server and deliver the results to clients through lightweight HTML or images. With Formula One, it's possible to deliver Java-based solutions in this manner.

This example is similar to the others. The same spreadsheet will be used on the server to calculate sales figures for a selected month. However, in this case the finished spreadsheet will be written out as an HTML table through a JSP that utilizes Formula One's HTMLWriter method. The requesting HTML page references our JSP page, report.jsp, rather than the Excel Servlet and passes the requested month as a parameter (see Figure 8). The code for report.jsp is in Listing 4.

This architecture allows users to leverage a spreadsheet's calculations on a server regardless of the operating system or bandwidth situation, as shown in Figure 9. Formula One also offers the ability to deliver spreadsheets and charts as static GIF, JPG or PNG images for thin-client environments where users simply need to view data. For more information on Formula One visit their Web site at [www.tidestone.com](http://www.tidestone.com).

## Summary

Few technologies are as familiar to users and developers as spreadsheets. When used as the data analysis component of a larger application, spreadsheets offer many benefits: developers can leverage their built-in features such as spreadsheet rules and formulas to increase efficiency and lower maintenance costs, while users benefit from the low learning curve.

Add Java's server-side strengths and it becomes clear that a spreadsheet can be extremely useful behind the scenes of an application, particularly in the middle layer of three-tier architectures when a large amount of data needs to be computed and analyzed. ☘

### AUTHOR BIO

Mark Spencer is the marketing manager of Tidestone Technologies, Inc. Prior to Tidestone he worked for three years for Visual Components, which later was acquired by Sybase. He has written numerous articles and white papers regarding component spreadsheet use in different programming languages.

[m Spencer@tidestone.com](mailto:m Spencer@tidestone.com)

#### Listing 1: dbManipulations.java

```
import java.sql.*;
import com.flj.util.*;
import com.flj.ss.*;

public class dbManipulations {
    private java.sql.Connection m_sqlCon = null;
    private java.sql.Statement m_stmt = null;

    public void retrieveAndPopulateDataFromDB(String strMonth,
        com.flj.ss.Book book, int iSheet) throws Exception {

        String strQuery = "Select orderid, name, volume, price
        from orders, products ";
        String strQryCount = "Select count(*) from orders, products ";
        // Following are parameters specific to Oracle8i and the
        // machine it is used on
        // Change to reflect your configurations
        String strDriver = "oracle.jdbc.driver.OracleDriver";
        String strUrl =
            "jdbc:oracle:thin:@webtogo.domain.com:1521:domain";
        String strUser = "scott";
        String strPassword = "tiger";
        int iSrcStartRow = 21;
        int iRowCount = 0;

        //connect to the database
        createConnection(strDriver, strUrl, strUser, strPassword);

        if (strMonth == null) // default month name
            strMonth = "JANUARY";
        else
            strMonth=strMonth.toUpperCase();
```

```
// Build and execute query string(s)
String strBufWhere = "where upper(orders.month) = '"+strMonth+
    "' and products.productid =
    orders.productid";

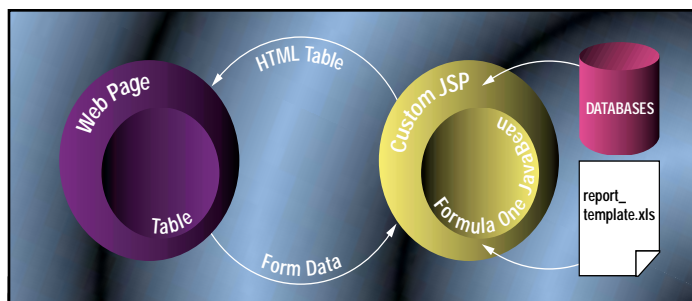
    java.sql.ResultSet rs = queryRecords(strQryCount + str-
    BufWhere);
    if (rs != null && rs.next()) {
        iRowCount = rs.getInt(1);
        rs.close();
    }
    rs = queryRecords(strQuery + strBufWhere);

    try {
        // Populate from ResultSet rs to
        // Spreadsheet
        if (book != null) {
            Sheet sheet = book.getSheet(iSheet);
            com.flj.jdbc.JDBC m_gridJDBC = new
            com.flj.jdbc.JDBC(sheet);
            com.flj.jdbc.JDBCQueryObj m_jdbcQryObj = new
            com.flj.jdbc.JDBCQueryObj();

            setFlagsJDBCQueryObject(iSrcStartRow, 0, m_jdbcQryObj);
            m_gridJDBC.populateGrid(rs, m_jdbcQryObj);

            // Add total calculations to the bottom of the data
            // and format
            int iTotalRow = iRowCount + iSrcStartRow;

            sheet.setText(iTotalRow, 1, "TOTAL:");
            sheet.setFormula(iTotalRow, 2, "SUM(C"+iSrc-
            StartRow+":C"+iTotalRow+")");
            sheet.setFormula(iTotalRow, 4, "SUM(E"+iSrc-
            StartRow+":E"+iTotalRow+)");
```



**FIGURE 8** This example shows how to leverage spreadsheets in thin-client environments by performing calculations on the server and exporting the data as HTML.



**FIGURE 9** Users who don't have Excel or are in bandwidth-restricted areas can still leverage spreadsheets by locating the analysis on a server and exporting the data as HTML or images.

# The Object People

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

```

sheet.copyRange(iTotalRow, 0, iTotalRow, 4,
               sheet, iSrcStartRow-2, 0, iSrc-
               StartRow-2, 4,
               com.flj.ss.Constants.eCopyFormats);
// format totals row

// Add Revenue formula column to all retrieved rows
int iSrcCol = 4; //revenue column
sheet.copyRange(iSrcStartRow+1, iSrcCol, iTotalRow-1,
iSrcCol,
               sheet, iSrcStartRow, iSrcCol, iSrc-
               StartRow, iSrcCol,
               com.flj.ss.Constants.eCopyAll);

// Change Spreadsheet "Title" to correspond to
// requested month
String strTitle = sheet.getText(17, 0) + strMonth;
sheet.setText(17, 0, strTitle);

// Change Chart range to correspond to the # of
// records returned
// The Chart takes its data from the defined names
// "chartData", "chartLegend"
// So we will redefine them to reflect the amount of
// data retrieved
GRChart chart = (GRChart)book.getSheet(iSheet).get
GRObjekt(3);
chart.setTitle(strTitle);

String sheetName = book.getSheet(iSheet).getName();
book.setDefinedName("chartData",
                    sheetName+"!$E$"+(iSrc-
                    StartRow+1)+":$E$"+iTotalRow,
                    0, 0);
book.setDefinedName("chartLegend",
                    sheetName+"!$B$"+(iSrc-
                    StartRow+1)+":$B$"+iTotalRow,
                    0,0);
}
}
finally {
//close the database connections
if (rs != null) rs.close();
closeAll();
}
}

private void setFlagsJDBCQueryObject (int iStartRow, int
iStartCol,
com.flj.jdbc.JDBC-
QueryObj jdbcQryObj) {
jdbcQryObj.setAutoColNames(false); // don't return
// field name as col hdrs
jdbcQryObj.setAutoColFormats(false); // format data
// according to type
jdbcQryObj.setAutoColWidths(true); // autosize columns
jdbcQryObj.setAutoMaxRC(false); // don't change
// max/min on spreadsheet
jdbcQryObj.setStartRow(iStartRow); // start row for
// populating
jdbcQryObj.setStartCol(iStartCol); // start col for
// populating
jdbcQryObj.setColNamesInRow(iStartRow); // put fields
// names in row
}

private void createConnection (String strDriverName,
String strDatasource, String strUsername, String strPass-
word) throws Exception {

Driver d=(Driver)Class.forName(strDriverName).newInstance();
DriverManager.registerDriver(d);
m_sqlCon=DriverManager.getConnection(strDatasource,
strUsername, strPassword);
m_stmt=m_sqlCon.createStatement();
}

// Queries the database using the sqlStatement passed to it.
// It returns the resultset.

private ResultSet queryRecords(String strSqlStmnt) throws
Exception {
if (strSqlStmnt != null)
return m_stmt.executeQuery(strSqlStmnt);
}

```

```

else
return (ResultSet)null;
}

private void closeAll() throws Exception {
if (m_stmt != null)
m_stmt.close();
if (m_sqlCon != null)
m_sqlCon.close();
}
}

Listing 2: ExcelServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.flj.swing.*;

public class Excel97Servlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, java.io.IOException {

ServletOutputStream out = response.getOutputStream();
response.setContentType("application/vnd.ms-excel");

// create a new Formula One workbook and lock it down.
com.flj.swing.JBook jbook = new com.flj.swing.JBook();
jbook.getLock();

try {
// read in the excel file we are using as a template
// for this report
jbook.read(getInitParameter("reportTemplate"));

// Populate data from database into spreadsheet
dbManipulations db = new dbManipulations();
db.retrieveAndPopulateDataFromDB(request.getParame-
ter("month"),
                                jbook.getBook(), 0);
// since we change the contents of the book we force a
// recalc before writing the model.
jbook.recalc();
WriteExcel(out, jbook);
out.close();
}
catch(Throwable e) {
System.out.println(e.getMessage());
}
finally {
jbook.releaseLock();
}
}

// Formatting Excel data requires access to a "seekable" stream.
// Since OutputStream is not seekable, we create a temporary
// file in excel format, then copy the data to the output stream.

private void WriteExcel(OutputStream out,
com.flj.swing.JBook jbook)
throws Exception {
java.util.Date tempFileName = new java.util.Date();
String tempFilePath = System.getProperty("user.dir") +
java.io.File.pathSeparator +
tempFileName.getTime();

// write the book to a temporary file
jbook.write(tempFilePath, jbook.eFileExcel97);

File tempFile = new File(tempFilePath);
FileInputStream tempfis = new FileInputStream(tempFile);

byte buffer[] = new byte[1024];
long totalBytesRead = 0;
int bytesRead = 0;

while (totalBytesRead < tempFile.length()) {
bytesRead = tempfis.read(buffer);
totalBytesRead = totalBytesRead + bytesRead;
out.write(buffer, 0, bytesRead);
}
tempfis.close();
tempFile.delete();
}
}

```



# Insignia

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

### Listing 3: WebMail.java

```
// You will need the activation.jar and mail.jar standard
// java extensions to compile this code.

import javax.mail.*;
import javax.mail.internet.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;

public class WebMail extends HttpServlet {
    private Session m_session;
    private String m_strFile;
    private String m_strTempFileName = "report.xls";

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        ServletContext ctx = getServletContext();
        m_strFile = config.getInitParameter("report_template");
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, java.io.IOException {

        res.setContentType("text/html");
        java.io.PrintWriter writer = res.getWriter();

        //start the session
        // change the postoffice domain reference
        // throughout this code to match your system
        java.util.Properties properties = System.getProperties();
        properties.put("mail.smtp.host",
            "postoffice.domain.com");

        //Connect to the store
        try {
            m_session = Session.getInstance(properties, null);
            Store store = m_session.getStore("imap");
            store.connect("postoffice.domain.com", "demo", "demo");
            sendMessage(req, res, writer);
        }
        catch (Exception e) {
            writer.println("Unable to connect to email account
                specified");
        }
    }

    private void sendMessage(HttpServletRequest req,
        HttpServletResponse res,
        java.io.PrintWriter writer)
        throws ServletException, java.io.IOException {

        String strFrom = "demo@domain.com";
        String strTo = req.getParameter("to");
        String strMonth = req.getParameter("month").toUpperCase();
        String strSubject = "Sales Figures for the Month of " + strMonth;
        com.flj.ss.Book book = new com.flj.ss.Book();

        String strTempDir = null;
        try {
            strTempDir = createTemporaryDir();
            String strTempFile = strTempDir+java.io.File.separator+m_strTempFileName;

            if (strTempFile != null) {
                // Load worksheet template, retrieve data from data-
                // base and write to a temporary file.
                book.getLock();
                book.read(new java.io.FileInputStream(m_strFile));
                dbManipulations m_db = new dbManipulations();
                m_db.retrieveAndPopulateDataFromDB(strMonth, book, 0);
                book.write(book.getSheet(0), strTempFile, book.eFile-
                    Excel97);
                book.releaseLock();

                // build 2-part mail message and send it.
                MimeMessage message = new MimeMessage(m_session);
                Multipart mp = new MimeMultipart();
                MimeBodyPart mbp1 = new MimeBodyPart();
                MimeBodyPart mbp2 = new MimeBodyPart();
```

```
message.setFrom(new InternetAddress(strFrom));
message.setRecipients(Message.RecipientType.TO,

        InternetAddress.parse(strTo));
message.setSubject(strSubject);
message.setContent(mp);

        mbp1.setText("Report successfully sent");

        // create the file attachment part of the message
        mbp2.setDataHandler(new javax.activation.DataHandler(
            new javax.activation.FileDataSource(strTemp File)));
        mbp2.setFileName(m_strTempFileName);

        mp.addBodyPart(mbp1);
        mp.addBodyPart(mbp2);

        //send the message
        Transport.send(message);
        writer.println("<p> Sales report was sent to: " +
            strTo + " </p>");
    }
    deleteTemporaryDir(strTempDir);
}
catch (Exception e) {
    writer.println("<p> " + e.getMessage() + " </p>");
}
}

private String createTemporaryDir() {
    String strNewDir = System.getProperty("user.dir") +
        java.io.File.separator + (new java.util.Date()).getTime();
    java.io.File dir = new java.io.File(strNewDir);
    dir.mkdir();
    return strNewDir;
}

private synchronized void deleteTemporaryDir(String strTempDir) {
    java.io.File dir = new java.io.File(strTempDir);
    if (dir.exists()) {
        java.io.File file = new java.io.File(strTempDir +
            java.io.File.separator + m_strTempFileName);
        file.delete();
        dir.delete();
    }
}
}
```

### Listing 4: report.jsp

```
<%@ page import="dbManipulations" %>
<%
// create a new formula one workbook
com.flj.swing.JBook jbook = new com.flj.swing.JBook();

jbook.getLock();
try {
    java.io.File me = new java.io.File(request.getPathTranslated());

jbook.read(me.getParent()+java.io.File.separator+"report_tem-
plate.xls");

        dbManipulations db = new dbManipulations();

db.retrieveAndPopulateDataFromDB(request.getParameter("month")
,
                                jbook.getBook(), 0);

        jbook.recalc();
        com.flj.ss.HTMLWriter htmlWriter = new com.flj.ss.HTMLWriter();
        htmlWriter.write(jbook.getBook(), 0, 17, 0, 0, 31, 4, out);
    } catch (Throwable e) { System.out.println("Error:
        "+e.getMessage()); }
    finally {
        jbook.releaseLock();
    }
}
%>
```



# KL Group

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

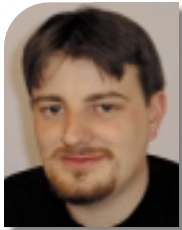
# Segue S

[www.seg](http://www.seg)

# oftware

gue.com

# The Whole World Is Turning AMAZON!



WRITTEN BY  
ALAN WILLIAMSON

Now that you're reading this, I wonder if you have a mental picture of me and what I may be doing as I write this. This isn't an ego question, merely an observational point. I have a mental picture of you. You, the typical reader, sitting down, with maybe a cup of coffee, looking for a 10-minute break from the drudgery of the day. I imagine myself talking to you, on a one-to-one basis. Of course, it's a one-sided conversation as I have to wait approximately six weeks for your response!

Speaking of responses, I'd like to take this opportunity to redress the balance here. Over the past couple of months I've received some very abusive e-mail from disgruntled readers, shouting the odds about a number of things. First of all, I have no problem receiving e-mail from readers regarding anything in this column. In fact, I welcome it. However, if you expect a reply, then make it polite, reasonable and – most of all – constructive. If you don't get a reply from me, you can safely assume that you failed one of the criteria above. The **JDJ** writers aren't here to listen to abuse, but to provide a service. To that end we want to serve you as best we can. Thus positive/negative constructive feedback is always welcome. Trust me, we get some really wacky e-mail. One that was particularly scary was from a gentleman who wanted Linus Torvald to be jailed. 'Nuff said, methinks.

Second, let me explain what this column is all about. It isn't a technical column, so if you're expecting it to be, **stop!** Turn back now and save yourself the disappointment. This is a thinking column. It's designed to make you think about some of the burning issues that face us all as Java developers. You may not agree with me, and that's good. I want to provoke you. I will on occasion offer some rather controversial notions. Whether or not I believe them doesn't matter. What matters is that I made you take some time to think about something from another angle. The active traffic on the **Straight Talking** mailing list proves this to be the case. So when you read this column, keep an open mind and allow me the chance to entertain you...if not provoke you. If we all agreed on everything, the world would be a very dull place indeed.

Last, I'd like to highlight the purpose of the **Straight Talking** mailing list. I write a piece on it every month, but some still seem to skip past that section. I've been accused of many things – most of it I ignore as nonsense. One I refuse to accept is the accusation that **JDJ** writers hide behind their articles and don't answer or address feedback. This is complete nonsense. For starters, the e-mail address printed with each article allows you direct access to the author.



Second, we have the mailing list. This is where we invite you to join and have your say. Being a **JDJ** editorial board member allows me to collate any concerns and relay them to the board to be addressed. The **Straight Talking** mailing list has had much impact on the overall structure and content of **JDJ**, and with your help we aim to keep **JDJ** the best-read Java magazine on the planet.

Let me apologize if this has come over a bit strong, but I want to be clear about the purpose of this column and the mechanisms that are available to you to influence the way **JDJ** operates. Contrary to popular belief, you, the reader, are the most important person to us and we want to make your time spent



with **JDJ** as productive as possible. So help us help you.

With that out of my system, let's get on with this month's **Straight Talking** column.

## Growing Pains

As CEO of my wee empire I've been faced with many new challenges and this month have met some wonderful people. My company, n-ary, is steadily growing, so much so that we're seriously pushing for space. We need more bodies, but our present building has simply stopped taking, so the time has come to look for new premises. Which is a shame, as I was kinda liking this place. But I can't stand in the way of progress, and my team is a credit to me, so with this I've put on my real estate hat, looking out for that perfect location for us all.

Those of you who've had dealings with us know that we're not the sort of team that wants to work in conventional office-space dwellings. That's simply not us. We're all a little unconventional in our own special way, and as the old saying goes, "You can't put a square peg in a round hole." At present we're using an old converted farmhouse as our headquarters, and this has proved to be most successful, just not quite big enough for a dot.com company striding into the millennium (Now *that* sounded like some sort of nonsense a marketing type of person would come up with! I surprise myself at times.)

Our strength is our team, so before embarking on this search I asked the troops what sort of offices they'd like to see themselves in. Darren, who some of you know as "The Riddler," came up with the most interesting answer. He requested an Ally McBeal-type environment. I wasn't too sure what to make of

# Hit Software

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

that. Did he mean the large, open-plan layout, complete with unisex toilets? I queried him on this and discovered he was in fact referring to the quantity of good-looking available women. Bless him.

All jokes aside, we have only two women working with us – one is a Java developer and the other is my PA. I'd have to say that we are a male-dominated company. That said, we do have a woman on the board of directors. When we get CV's in, the ones with a Sex=Female are few and far between. I'm not sure why this is, but there are a number of possible reasons for this and many are controversial.

Do you think there's a serious lack of women in this industry? I guess this is always going to be a controversial topic, so let's discuss it on the mailing list as I'd love to hear your thoughts. I'm not so stupid as to alienate my female readers here and now! Knowing how seriously some of you take this column, I'm going to body-swerve that particular mine-field.

But speaking of unisex toilets...do they really exist? I don't know of any company on this side of the Atlantic that has them. But since it's an American idea, do any of our U.S. readers know of companies that openly advocate them?

I'll let you know next month how our new-office search is going. We're looking at an old farm where we would convert the barns and stables into modern offices, I just need to persuade the present owners to accept the amount of money I'm offering as opposed to the amount they want. I guess this is what they call *bartering*.

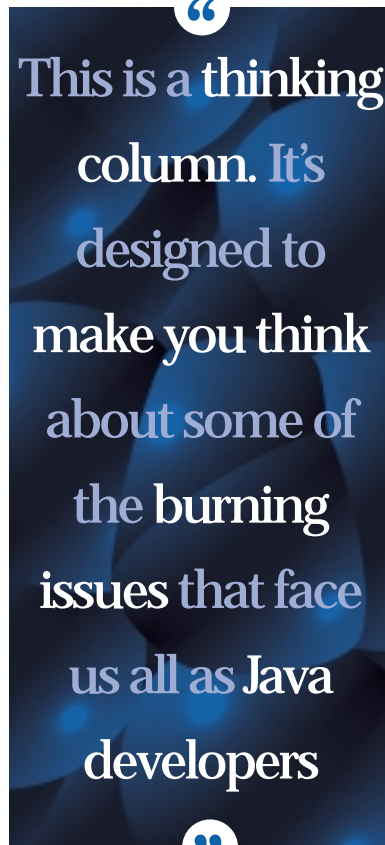
### Support Award of the Month: Hall of Shame

This is the second in our new award section. Last month I chronicled the joys of ordering from Dell and how much fun that can be. If you recall, a staff member told me that Dell didn't use Internet-enabled e-mail internally, which I suspected was a bit of a fib. I recently discovered that this was indeed the case – the support team member who called me up the day after told me that e-mail is used a lot and that I could be cc'd into all relevant communication. What a sorry tale this was. If Dell ever responds to me, I'll let you know the dialog.

I've had some interesting feedback from readers with similar tales. It appears that if you're a small company, or a one-man band, the service from Dell is pretty poor on the whole. Truly a shame, as I have no problems at all with the kit itself.

This month's award is on hold as I'm currently researching a number of queries I've had from readers regarding two big boys. Oracle and Silver-Stream are in the limelight and when I have more information I'll report back.

As a side note, anyone been to the store at Oracle.com lately? Here's another example of a company with absolutely no imagination. Another Amazon.com clone. Oracle isn't the only company to succumb to this look and feel copying; BOL.com, Toysrus.com, HomeDepot.com and Comp-USA.com are just some of the big household names. Come on, you big corporate



chaps...you spend millions a year on your marketing departments and still can't come up with something that makes you stand apart from the competition? I bet Amazon.com is kicking themselves for not trademarking that look and feel...the money gleaned from those lawsuits would have more than made up the operating loss.

If you feel you're getting a raw deal from any company, let me know and I'll check it out and see if something more sinister is going on.

### Mailing List

The mailing list is still growing, with representatives of many different facets of industry joining in on the fun. I'm thor-

oughly enjoying getting to know you a lot better, and with this I'm looking to arrange a sort of **Straight Talking** party for JavaOne this year. So stay tuned for that. We talk about anything that comes into the mind of our developers. So stop by and join in the fun. To sign up, or even just to stop by and have a look at what's being posted, head on over to [http://listserv.n-ary.com/mailman/listinfo/straight\\_talking](http://listserv.n-ary.com/mailman/listinfo/straight_talking).

Our radio show is gaining in popularity (<http://radio.sys-con.com/>). I and my cohost, Keith Douglas, present a daily 15–20 minute **Straight Talking** show. We play music, talk Java and talk mailing list. And with The Riddler offering prizes for anyone that correctly answers his riddles, what have you got to lose?

### Salute of the Month

The award this month goes to the active group of correspondents on our mailing list. You bring a much welcome break from the norm with refreshing views and takes on a number of issues. I've learned a lot from you all, and I hope we can meet up very soon. As you know, JavaOne is looming, and I'm hoping to be able to put faces to the names at the party mentioned above. Let me know if you intend to come to JavaOne.

Now that I end this column, I end it in the same state as I end the month – with a smile. I've been on the prowl for a new car. Over the last 12 months I've been driving a wonderful wee Toyota MR-2, and I have to say I loved it. However, as you know, picking up certain Sun dignitaries from the airport did present a certain logistical problem due to the lack of boot (sorry...trunk) space. Besides, I was also getting a little bored with not having a backseat. That said, I recently took delivery of a brand spanking new n-ary blue Subaru Impreza. For those of you that don't know of this particular motor, it's the one leading the world rally championships. I'm very pleased with it as I know I can take passengers, which is a real novelty.

See you next month. ☺

### AUTHOR BIO

Alan Williamson is CEO of n-ary consulting Ltd, the first pure Java company in the United Kingdom. The firm, which specializes solely in Java at the server side, has offices in Scotland, England and Australia. Alan is the author of two Java servlet books, and contributed to the Servlet API. He has a Web site at [www.n-ary.com](http://www.n-ary.com).

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



# Prosyst

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

# Understanding EJB Transactions

WRITTEN BY SAMEER TYAGI

**E**JB servers are transactional servers that allow developers to concentrate on business logic. The EJB model implements two-phase commits, transaction context propagation and distributed transaction, although it's up to the vendors to decide which technique to use.

A transaction is formally defined as an "ACID" (atomic, consistent, isolated, durable) unit of work.

- **Atomic** transactions are "all or nothing." They either work or they don't – they're never left incomplete.
- **Consistent** transactions always leave the system in a consistent state.
- **Isolated** transactions execute in a safe manner – they won't fail if other transactions running on the same server are failing.
- **Durable** transactions can survive system failures once they're completed and committed.

For example, an online sale may involve the following steps:

1. *Begin transaction.*  
Charge card sale amount.  
Update sale database.  
Update shipping database.
2. *Commit transaction.*

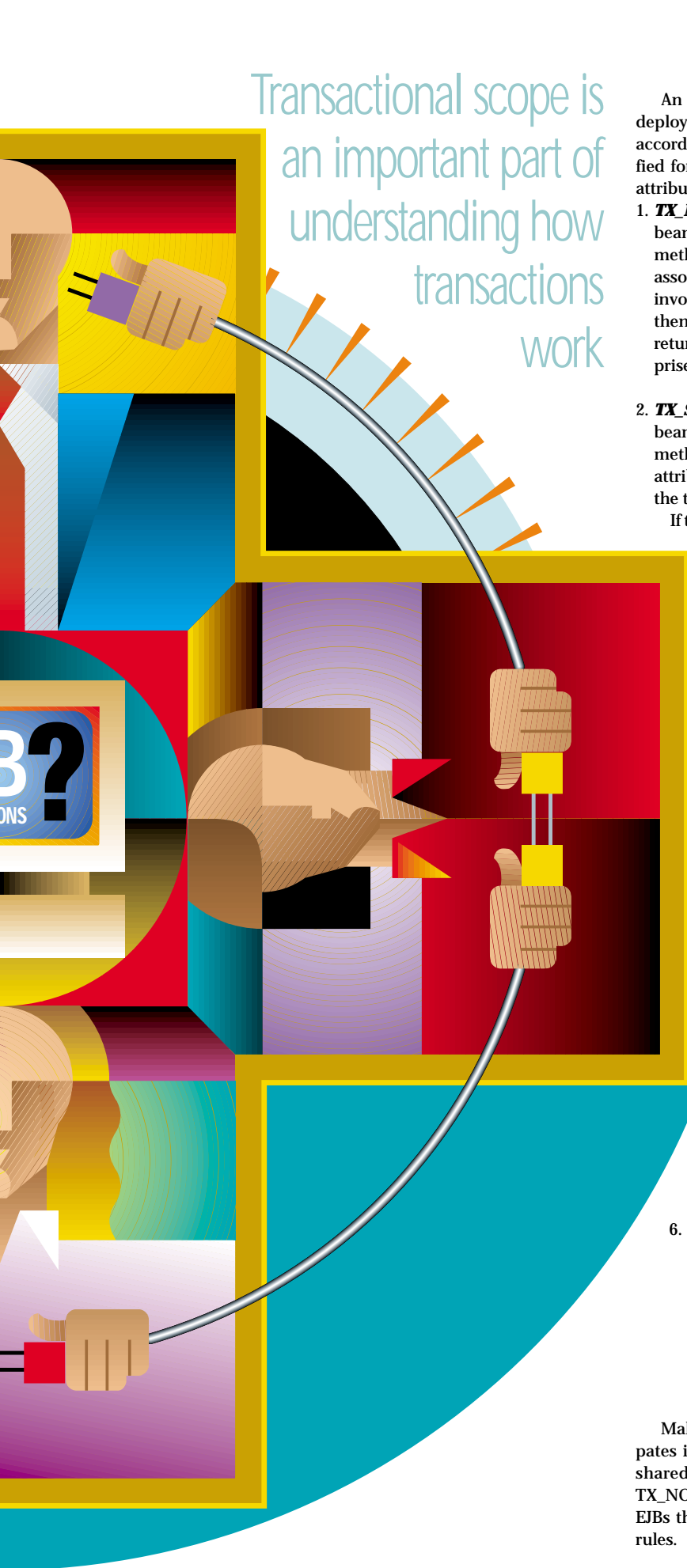
The transaction can end in one of two ways: (1) in a commit and everything is saved; or (2) if any step within the transaction fails, the effects of all preceding steps are rolled back or undone. For example, if the shipping database can't be updated, the charge isn't made and the sale database isn't updated.

In a distributed environment handling transactions involves coordinating the various databases that participate in the transaction. In the EJB framework the bean developer can simply define the transaction policy for the bean during the deployment process, using declarative statements, and let the container handle all distributed transactions (called *bean-demarcated* transactions). Alternatively, the developer can take explicit control of transactions (called *client-demarcated* transactions).

## Transactional Scope and Attributes

Transactional scope is an important part of understanding how transactions work. In EJBs the scope of a transaction includes every bean that participates in a unit of work – the bean method. The scope of a transaction can be traced by looking at the thread of execution. The transaction is propagated to a bean when that bean is invoked and included in the scope of the transaction. Of course, the thread of execution is not the only determining factor in transactional propagation; transactional attributes is the other.





Transactional scope is an important part of understanding how transactions work

An enterprise bean can take one of the following six attributes in the deployment descriptor, and the container manages the transactions according to the specified attribute. Transaction attributes can be specified for the entire bean or the bean can be fine-tuned by specifying the attributes for individual methods.

1. ***TX\_NOT\_SUPPORTED*** (see Figure 1): This tells the container to invoke bean methods without a transaction context. If a client invokes a bean method from within a transaction context, the container suspends the association between the transaction and the current thread before invoking the method on the enterprise bean instance. The container then resumes the suspended association when the method invocation returns. The suspended transaction context isn't passed to any enterprise bean objects or resources that are used by this bean method.

2. ***TX\_SUPPORTS*** (see Figure 2): This tells the container to include the bean or method within the transaction scope in which it is invoked. If a method is part of a transactional scope and it invokes any bean with this attribute, the invoked bean and everything it accesses become a part of the transaction.

If the client invokes the bean method without a transaction context, the container invokes the bean method without a transaction context.

3. ***TX\_REQUIRED*** (see Figure 3): This tells the container that the bean method must be invoked within a transaction scope. If a client invokes a bean method from within a transaction context, the container invokes the bean method within the client transaction context. If a client invokes a bean method without a transaction context, the container creates a new transaction context for the invoked bean. The transaction context is then passed to any beans that are used by this bean method.

4. ***TX\_REQUIRES\_NEW*** (see Figure 4): This tells the container to always invoke the bean method within a new transaction context, regardless of whether the client invokes the method within or without a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

5. ***TX\_MANDATORY*** (see Figure 5): This directs the container to always invoke the bean method within the transaction context associated with the client. The difference between this and the ***TX\_REQUIRED*** attribute is that if the client attempts to invoke the bean method without a transaction context, the container throws the `javax.transaction.TransactionRequiredException` exception. The transaction context is passed to any beans that are used by the invoked bean method.

6. ***TX\_BEAN\_MANAGED*** (see Figure 6): This tells the container that the bean class doesn't have its transactional context managed by the server but it uses JTA, more specifically the `javax.transaction.UserTransaction`, to explicitly manage transaction boundaries.

Using this attribute, however, imposes a restriction that the attributes of different methods cannot be mixed; if even one method has this attribute, then all methods must manage transaction on their own.

Making a bean transactional is expensive at runtime; since it participates in a transaction and conforms to ACID rules, its services can't be shared during the life of a transaction. Declaring a bean to be ***TX\_NOT\_SUPPORTED*** improves performance and may be desirable for EJBs that provide stateless service as they need to conform to the ACID rules.

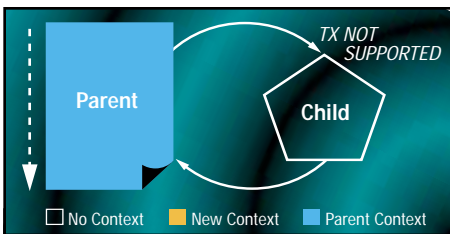


FIGURE 1 Transaction propagation for TX\_NOT\_SUPPORTED

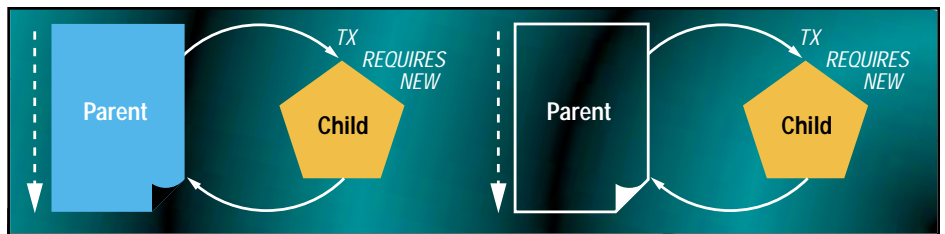


FIGURE 4 Transaction propagation for TX\_REQUIRES\_NEW

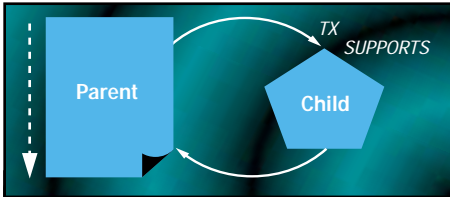


FIGURE 2 Transaction propagation for TX\_SUPPORTS

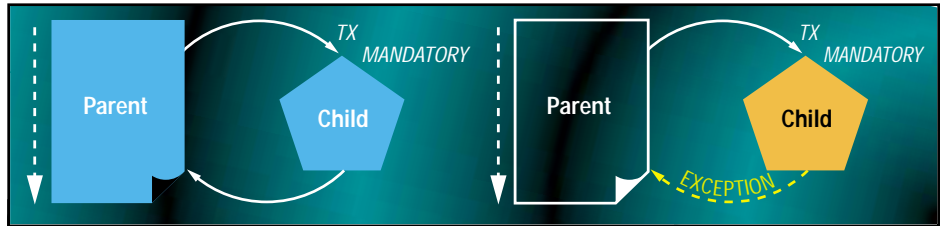


FIGURE 5 Transaction propagation for TX\_MANDATORY

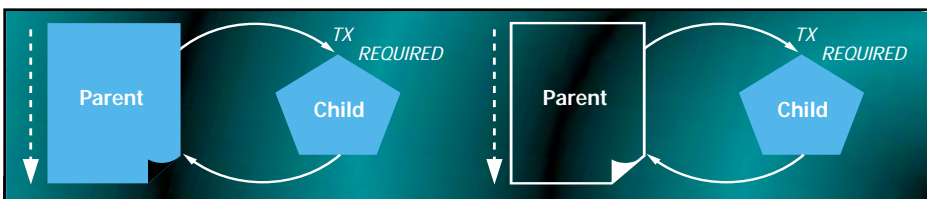


FIGURE 3 Transaction propagation for TX\_REQUIRED

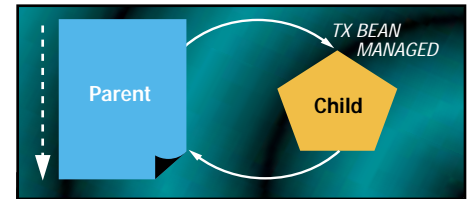


FIGURE 6 Transaction propagation for TX\_BEAN\_MANAGED

## Transaction Isolation Levels

The transaction isolation level determines how isolated one transaction is from another for read purposes only. These isolation levels are defined in terms of three phenomena (defined in the ANSI/ISO SQL standard -SQL92) that must be prevented between concurrently executing transactions.

- **Dirty reads:** A transaction reads data written by another transaction that hasn't been committed yet. In other words, a transaction reads a database row containing uncommitted changes from a second transaction.
- **Nonrepeatable reads:** A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data. In other words, one transaction reads a row in a table, a second transaction changes the same row and the first transaction rereads the row and gets a different value.
- **Phantom reads:** A transaction reexecutes a query, returning a set of rows that satisfies a search condition, and finds that another committed transaction has inserted additional rows that satisfy the condition. In other words, one transaction reads all rows that satisfy a SQL WHERE condition and a second transaction inserts a row that also satisfies the WHERE condition. The first transaction applies the same WHERE condition and gets the row inserted by the second transaction.

Isolation levels aren't new to EJBs; EJB defines these levels based on the ANSI-SQL92 standards. They're mapped in JDBC to the static variables defined in the java.sql.Connection interface. Isolation level, like attributes, can be fine-tuned by specifying them at the method level for EJBs; however, all methods invoked in the same transaction must have the same isolation level.

1. **TRANSACTION\_READ\_UNCOMMITTED:** The transaction can read uncommitted data (data changed by another transaction still in progress).
2. **TRANSACTION\_READ\_COMMITTED:** The transaction can't read uncommitted data.
3. **TRANSACTION\_REPEATABLE\_READ:** The transaction can't change

data that's being read by another transaction. Methods with this isolation level, besides having the same behavior as TRANSACTION\_READ\_COMMITTED, can only execute repeatable reads.

4. **TRANSACTION\_SERIALIZABLE:** The transaction has exclusive read and update privileges to data by locking it; other transactions can neither write nor read the same data. (This does to transaction what the synchronized keyword does to methods.) It is the most restrictive transaction.

Make no mistake – although the TX\_SERIALIZABLE attribute guarantees the highest level of data integrity, it is offset by a performance slag because even simple reads must wait in line. EJBs that need to handle a large number of concurrent transactions should avoid this level. By understanding the level of reads that will occur on the database and how the database handles locking and choosing the correct isolation level, the EJB can be fine-tuned to peak performance (see Table 1).

| ISOLATION LEVEL     | DIRTY READ   | NONREPEATABLE READ | PHANTOM READ |
|---------------------|--------------|--------------------|--------------|
| TX_READ_UNCOMMITTED | Possible     | Possible           | Possible     |
| TX_READ_COMMITTED   | Not possible | Possible           | Possible     |
| TX_REPEATABLE_READ  | Not possible | Not possible       | Possible     |
| TX_SERIALIZABLE     | Not possible | Not possible       | Not possible |

TABLE 1 Isolation levels and reads

## Client-Demarcated Transactions

JTS is a specification based on the CORBA OTS 1.1 for implementing a Java transaction manager that serves as an intermediary between an application and one or more transaction-capable resource managers, such as database servers and messaging systems. The JTS specification includes the JTA API that is used by application programmers to group operations into one or more logical transactions. The Java mapping of the OMG OTS 1.1 specification is specified in two packages: org.omg.CosTransactions and org.omg.CosTSPortability. JTA actually provides three types of services:

# Flashline

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

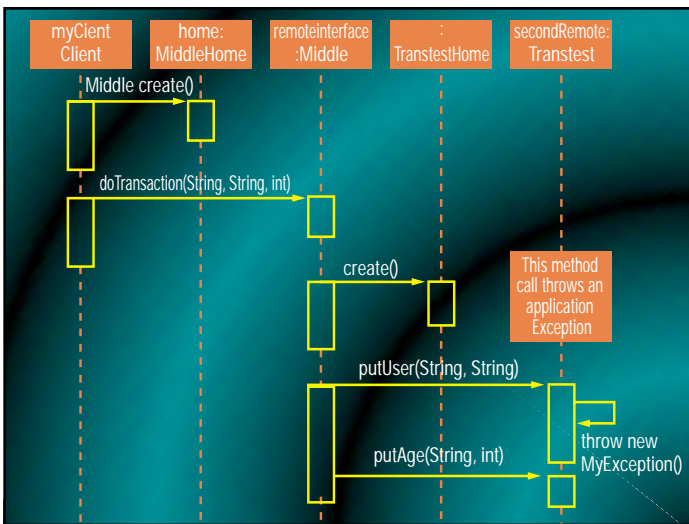


FIGURE 7 Sequence diagram for the exception-handling example

```
public void doTransaction(String customerName, String password, int
age) throws MyException{
try{
UserTransaction trans=sessionctx. ctx.getUserTransaction().begin();
Context ctx = new InitialContext();
TranstestHome home = (TranstestHome)
ctx.lookup("ejb.TranstestHome");          Transtest
bean = home.create();
bean.putUser("Sameer","word");
bean.putAge("Sameer",10);
trans.commit();
}catch(Exception e){
trans.rollback();
throw new MyException("an exception occurred" +e);
}
}
```

This UserTransaction also defines two methods: setRollbackOnly() and getRollbackOnly(). The first method allows the bean to veto a transaction explicitly. Once invoked, the transaction can't be committed by anyone, including the container. The second method remains true if the transaction has been so marked and can be used to avoid further unnecessary work in the method.

JTA allows the UserTransaction object to be exposed via JNDI. EJBs shouldn't use this approach as it compromises the "middleware portability"; that is, other EJB servers might not support that approach.

```
Context ctx = new InitialContext();
UserTransaction utx = (UserTransaction)ctx.lookup("ajndiname");
utx.begin();
// do work
utx.commit();
```

With entity beans and stateless session beans, a transaction managed with the UserTransaction must start and end in the same method. The reason is that entity and stateless session bean instances are shared across many clients by instance pooling and instance swapping on the server.

Stateful session beans allow the UserTransaction object to span multiple method calls because there's always one instance associated with a client, and it maintains conversational state. This bean state (and state of the transaction) is consistent even when the container makes it undergo an internal activation-passivation cycle to conserve server resources.

```
public class MyStatefulBean implements SessionBean {
public SessionContext ctx;

public void setSessionContext(SessionContext ctx){
this.ctx=ctx;
}

public void method1(){
ctx.getUserTransaction().begin();
// do some work
}

public void method2(){
// do some more work
}

public void method3(){
// do yet some more work
// and finally commit
ctx.getUserTransaction().commit();
}
```

Repeated calls to getUserTransaction() in a stateful session bean return a reference to the same UserTransaction object. Its state can be checked using a UserTransaction.getStatus() call.

- Transactional operations in client applications
- Transactional operations in application servers performed on behalf of clients
- Global transactional management in a Java transaction manager, coordinating multiple transaction-capable resource managers such as database servers and messaging systems

EJBs use the high-level transaction manager interface provided by JTA, while the EJB server uses the JTA high-level transaction manager interface and a standard Java mapping of the X/Open XA protocol to handle transactions (javax.transaction.xa package). EJBs use the simple javax.transaction.UserTransaction interface to communicate with the transaction manager and control transaction boundaries programmatically. (The EJB specification doesn't stipulate any specific transaction service or protocol but requires that the javax.transaction.UserTransaction interface of the JTS be exposed to enterprise beans.) Important methods in UserTransaction interface are:

```
public void begin()
public void commit()
public int getStatus()
public void rollback()
public void setRollbackOnly()
public void setTransactionTimeout()
```

The UserTransaction.begin() method starts a global transaction and associates a javax.transaction.Transaction with the execution thread. It throws the NotSupportedException when the calling thread is already associated with a transaction and the transaction manager implementation doesn't support nested transactions.

The UserTransaction.commit() method completes the transaction. If at this point the transaction needs to be rolled back instead of being committed, the transaction manager does so and throws a RollbackException to indicate it.

The UserTransaction.rollback() method undoes any changes made since the start of the transaction and removes the association between the Transaction and the execution thread.

#### GETTING TRANSACTION ACCESS

As mentioned earlier, the ability of the bean to access the transaction service can't be selectively applied to only certain methods of the bean. All methods must have the TX\_BEAN\_MANAGED attribute. That said, bean methods can get access to the transaction service through the getUserTransaction() method of the javax.ejb.EJBContext interface (the superinterface for SessionContext and EntityContext).

# Object Switch

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

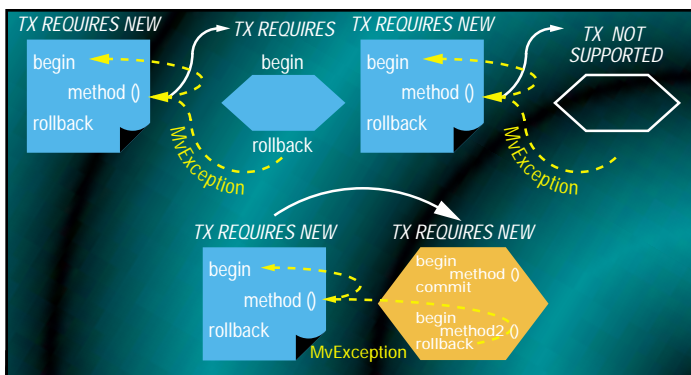


FIGURE 8 The three scenarios that occur when a deliberately planted exception is thrown

| PARENT (MIDDLEBEAN) | CHILD (TRANSTESTBEAN) | TRANSACTION EXPLANATION   |
|---------------------|-----------------------|---|
| TX_REQUIRES_NEW     | TX_REQUIRES           | Rollback because of second bean transaction context   |
| TX_REQUIRES_NEW     | TX_NOT_SUPPORTED      | No rollback in second bean because it is not in context.  |
| TX_REQUIRES_NEW     | TX_NOT_REQUIRES_NEW   | One exception-throwing method is rolled back, the other is not, because each is in a new context. |

TABLE 2 The three scenarios that occur when this deliberately planted exception is thrown

Stateful session beans involve conversational state between method calls. Sometimes it may be desirable to cache this transactional state and postpone database updates. The `javax.ejb.SessionSynchronization` allows the server to inform a stateful session bean of the various stages in the transaction by invoking callback methods such as:

- **afterBegin():** Notifies the bean that a new transaction has started – called before the EJB delegates the business methods to the instance
- **afterCompletion():** Notifies the bean that the transaction has completed – can be used to reset instance variables since it will always be invoked
- **beforeCompletion():** Notifies the bean that the transaction is about to be committed

Client-demarcated transactions in stateful session beans across methods should be avoided since they tend to be overtly complex and any improper method invocation locks up resources. Once a stateful session bean is a part of a transaction, there's no way for it to be accessed by any other transaction context (e.g., a method with `TX_REQUIRES_NEW` is invoked) and the bean can't be removed. In the example above, resource locking will happen when the client doesn't invoke `method3()`.

## Exception Handling and Transactions

What happens when exceptions occur depends on the type of exception (checked or unchecked), isolation level and the transactional attribute of the bean method.

The first rule, which may sound strange, is that any exception thrown outside the transaction scope causes the transaction to roll back.

Consider an example of a client invoking a method called `doTransaction()` on a stateless session bean called `MiddleBean`. The bean internally then invokes methods on another stateless session bean called `TranstestBean`. The required code for these beans and the client can be seen in Listings 1 to 4. The sequence of events that occur as a result of the client call is shown in Figure 7.

The two beans are deployed with the transaction attributes shown in Table 2 and what happens as a result of these attributes is summarized in Figure 8.

In the first case the transaction context is propagated to the `TranstestBean` bean.

When the exception is thrown in the second bean, it falls within the transaction context; it propagates up and the container traps it in the first bean and rolls back the transaction.

In case two the second bean doesn't participate in the transaction and the transaction context is not propagated to it. The exception thrown falls outside the transaction context; the container detects this and rolls back the transaction. There is no way, however, to undo any changes made in the second bean.

In the third case the second bean has a new transaction context for each of the methods. The transaction for the exception-throwing method is rolled back, the exception moves up and the container rolls back the initial transaction. The other method executes successfully in its own transaction.

In general, methods should be logically atomic – all or nothing. If an exception is intended to indicate that the method can't complete successfully, it *shouldn't* be caught. If it is caught, the method should try to correct the problem and continue. The method must throw the exception and propagate out of the method for the transaction to be rolled back.

Application exceptions won't cause a rollback if they're thrown and caught within the transactional scope. Runtime exceptions or unchecked exceptions, on the other hand, always cause a transaction to roll back, regardless of the transaction attribute or transactional scope.

## Unilateral Decisions

The transaction manager allows certain heuristic or speculative decisions to be made based on the state of all participating resources in a transaction and the underlying two-phase commit protocol. A heuristic decision occurs when one of the resources in the transaction unilaterally decides to commit or roll back the transaction without permission from the transaction manager. This breaks the atomicity of the transaction and is captured by the manager in the following exceptions:

- **javax.transaction.HeuristicCommitException** is thrown when a rollback is requested but a heuristic decision was made and all updates were committed.
- **javax.transaction.HeuristicMixedException** is thrown when a heuristic decision was made and some updates have been committed and others were rolled back.
- **javax.transaction.HeuristicRollbackException** is thrown when a commit is requested but a heuristic decision was made and all relevant updates were rolled back. ☹

## Resources

1. A complete definition of two-phase commits by SEI: [www.sei.cmu.edu/activities/str/descriptions/dtpc\\_body.html](http://www.sei.cmu.edu/activities/str/descriptions/dtpc_body.html)
2. *Datamation Magazine*. "What a two-phase commit is and how it works": [www.datamation.com/datab/twophase.html](http://www.datamation.com/datab/twophase.html)
3. SQL 92 standards: [www.ansi.org](http://www.ansi.org)
4. Java Transaction API: <http://java.sun.com/jta>
5. EJB home page at SUN: <http://java.sun.com/products/ejb>
6. The Oracle technical network: <http://technet.oracle.com/>
7. The org.omg.CosTSPortability and other similar packages: <http://java.sun.com/products/jts/javadoc/org/omg/CosTransactions/package-summary.html>
8. Complete source code, console dumps, SQL test queries and UML diagrams: [www.JavaDevelopersJournal.com](http://www.JavaDevelopersJournal.com)

### AUTHOR BIO

Sameer Tyagi has almost four years' experience in n-tier Internet and intranet application development. He currently architects server-side and enterprise Java applications and writes regularly for online and print publications.

[bytecode@crosswinds.net](mailto:bytecode@crosswinds.net)



# StarBase

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

**Listing 1A: The remote interface of the stateless session bean MiddleBean.java**

```
package sameer.ejb;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface Middle extends EJBObject {
    public void doTransaction (String customerName,String pass-
word,int age) throws RemoteException;
}
```

**Listing 1B: The home interface of the stateless session bean MiddleBean.java**

```
package sameer.ejb;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface MiddleHome extends EJBHome {
    Middle create() throws CreateException, RemoteException;
}
```

**Listing 1C: The stateless session bean MiddleBean.java**

```
package sameer.ejb;
import javax.ejb.*;
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.naming.*;
import java.util.*;
import java.sql.*;

public class MiddleBean implements SessionBean {

    public void ejbActivate() {}
    public void ejbRemove() {}
    public void ejbPassivate(){}
    public void setSessionContext(SessionContext ctx) {}
    public void ejbCreate () throws CreateException {}

    public void doTransaction(String customerName, String pass-
word,int age)
throws MyException{
    try{
        Context ctx = new InitialContext();
        TranstestHome home = (TranstestHome)
        ctx.lookup("ejb.TranstestHome");
        Transtest bean = home.create();
        bean.putUser ("Sameer", "word");
        bean.putAge ("Sameer",10);

    }catch(Exception e){
        throw new MyException("an exception occured" +e);
    }
}
}
```

**Listing 2A: The remote interface of the stateless session bean TranstestBean**

```
package sameer.ejb;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface Transtest extends EJBObject {

public void putUser (String customerName,String password)
throws RemoteException,MyException;
    public void putAge (String customerName,int age) throws
RemoteException,MyException;
}
```

**Listing 2B: The home interface of the stateless session bean TranstestBean**

```
package sameer.ejb;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface TranstestHome extends EJBHome {
    Transtest create() throws CreateException, RemoteException;
}
```

**Listing 2C: The stateless session bean TranstestBean**

```
package sameer.ejb;
import javax.ejb.*;
import java.io.Serializable;
import java.rmi.RemoteException;
import java.util.*;
import java.sql.*;

public class TranstestBean implements SessionBean {

    public void ejbActivate() {}
    public void ejbRemove() {}
    public void ejbPassivate(){}
    public void setSessionContext(SessionContext ctx) {}
}
```

```
public void ejbCreate () throws CreateException {}

    public void putUser(String customerName, String password)
throws MyException{
    try{
        String str = "INSERT INTO USERTABLE (name, pwd) VALUES (' ' +
customerName + "',' ' +
        password +'')";
        System.out.println ("Executing stmt: " + str);
        new weblogic.jdbc.jts.Driver();
        Connection
        conn=DriverManager.getConnection("jdbc:weblogic:jts:demoPool" ) ;
        Statement stmt=conn.createStatement();
        int rs=stmt.executeUpdate(str);
        System.out.println(">>>>Username/Pwd insert succeeded
for "+ customerName +
        and password "+password);

        throw new MyException(); // deliberately throw an applica-
tion exception

    }catch(SQLException se){
        throw new MyException("There was an exceptin "+se);
    }
}

    public void putAge(String customerName,int age) throws
MyException {
    try{
        String str ="INSERT INTO AGETABLE (name, age) VALUES (' ' +
customerName + "',' ' +
        + age +'')";
        System.out.println ("Executing stmt: " + str);
        // Class.forName("weblogic.jdbc.jts.Driver");
        new weblogic.jdbc.jts.Driver();
        Connection conn=DriverManager.getConnection("jdbc:weblog-
ic:jts:demoPool");
        Statement stmt=conn.createStatement();
        int rs=stmt.executeUpdate(str);
        System.out.println(">>>>Username/Age insert succeeded for
"+ customerName + "
        and age "+age);
    }catch(SQLException se){
        throw new MyException("There was an exceptin "+se);
    }
}
}
```

**Listing 3: The application exception class that is deliberately thrown**

```
package sameer.ejb;

public class MyException extends Exception {
    public MyException() {}
    public MyException(String message) {
        super(message);
    }
}
```

**Listing 4: The client for the sessionbean MiddleBean**

```
package sameer.ejb;
import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;
import java.util.*;

public class Client {
    static String url = "t3://localhost:7001";
    static String user= null;
    static String password= null;

    public static void main(String[] args) throws Exception {
        Context ctx = getInitialContext();
        MiddleHome home = (MiddleHome) ctx.lookup("ejb.MiddleHome");
        Middle bean = home.create();
        bean.doTransaction("Sameer", "word", 26);
    }

    public static Context getInitialContext() throws Exception
{
        Hashtable h = new Hashtable();

        h.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLIni-
tialContextFactory");
        h.put (Context.PROVIDER_URL, url);
        return new InitialContext(h);
    }
}
```



# Sybase

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

# What do MTS and EJB Have in Common?

A stateless component model, which can meet the demand of your high-volume distributed systems

WRITTEN BY  
JASON WESTRA



As much as I hate to admit it, Microsoft was a pioneer in server-side component architectures. Its COM/DCOM (Distributed Component Object Model) server-side component model for building and deploying components in the Microsoft Transaction Server (MTS) environment already had applications in production by the time the Enterprise JavaBeans specification was publicly released in March 1998.

MTS's approach toward declarative, container-managed components is reflected in the specification's notions of EJB servers and containers, and declarative, transactional properties of enterprise beans through deployment descriptors.

Also, Microsoft's distributed component architecture, DCOM, is based on remote procedure calls to stateless components running within the MTS. This model takes advantage of resource pooling – such as connection pools to the database and instance pooling – to provide a highly scalable distributed component environment. Microsoft's core argument for its stateless component architecture is that stateful servers can't scale. A stateful architecture maintains information between calls from a client, thereby holding resources within the server that could be used for other purposes or freed altogether. An application is scalable when it maintains a certain level of performance even as the number of users and complex transactions per second grows. Thus, from Microsoft's point of view, scalability is achieved by stateless business components that don't hold precious resources from others in need of their services.

The initial release of the EJB specification required session bean component support for 1.0 compliance. However, unlike Microsoft's server-side component model, the 1.0 specification offered two types of components: stateful and stateless session beans. With support for stateful server architectures, EJB split from its Microsoft lineage. In this month's *EJB Home*, however, I discuss the common genes between the two component models, focusing strictly on stateless components.

## Stateless Session Beans

Unlike your neighborhood coffee beans, session beans come in only two

flavors: stateful and stateless. A stateful session bean holds conversational state between method invocations in the form of instance variables from a single client while a stateless session bean maintains no state between method invocations.

Stateful and stateless session Beans have some commonalities:

1. They are relatively short-lived, non-persistent components.
2. They generally do not survive an EJB server crash.
3. They are transaction aware (restrictions in stateless session beans will be covered shortly).
4. They are single-threaded, servicing a single client at a time.

In other words, all session beans rely on their container to manage thread access to the bean, ensuring it has only one thread manipulating the bean at a time. This restriction (container-managed threading) is actually enforced on all enterprise beans, not just session beans. Spawning threads in your enterprise beans isn't a good practice!

Despite their common features, there are some major differences. The lack of conversational state inherent in stateless session beans binds them closely to Microsoft's COM/DCOM. Also, while stateful session beans remain uniquely identifiable within their container, all instances of stateless session beans are equally capable of handling a client request. When a client compares two EJB objects of the same stateless session bean, they will always be equal. The following code provides an example of this comparison:

```
CustomerMgrHome home = (Customer-
MgrHome)
ctx.lookup("java:comp/env/ejb/homelo-
cation");
CustomerMgr mgr1 = home.create();
```

```
CustomerMgr mgr2 = home.create();
return mgr1.isIdentical (mgr2) ; //
return value will be true!
```

Because each stateless session bean of the same class is equal, any method-ready instance is equally qualified to service a request from a client. This allows for a highly scalable middle-tier environment in which the server can instantiate and destroy beans at will to support extreme fluctuations in user volume. Speaking of creating and destroying bean instances, let's take a look at the life cycle of a stateless session bean for a clear understanding of just how simple it is for the server to support stateless components.

## Life Cycle of a Stateless Session Bean

A stateless session bean basically has two states: (1) nonexistent, and (2) method-ready. The EJB container does three steps to bring your stateless bean into a position where it is ready to satisfy a client request. First, the container allocates a new instance of the bean into memory. Next, the `setSessionContext(javax.ejb.SessionContext ctx)` method is called to set the bean's SessionContext. Last, the container calls the `bean.ejbCreate()` method, allowing the bean to perform any initialization code.

Enterprise JavaBeans uses "passivation" to manage large numbers of enterprise beans. In EJB, passivation is the process of storing a bean's state to a persistent store to release resources in the server. During the life cycle of an enterprise bean, passivation may occur if the bean hasn't been accessed by the client for some period of time; however, because stateless session beans have no state to store, they aren't passivated by their container. Instead, `ejbRemove()` is called on the instance of the bean to

# Hot Dispatch

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

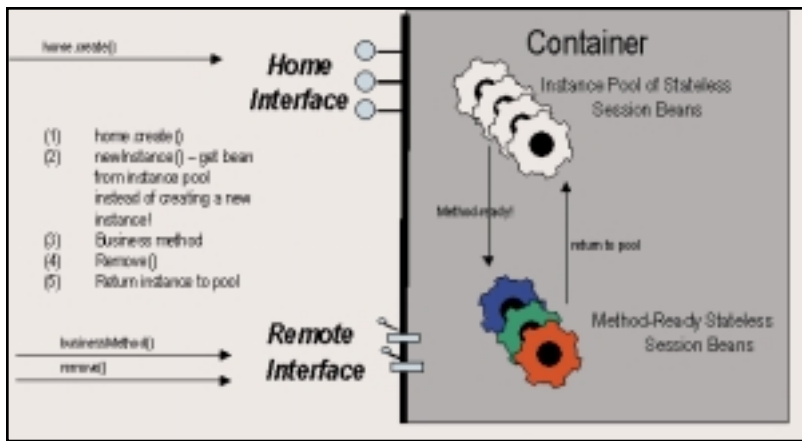


FIGURE 1 Instance-pooling stateless session beans

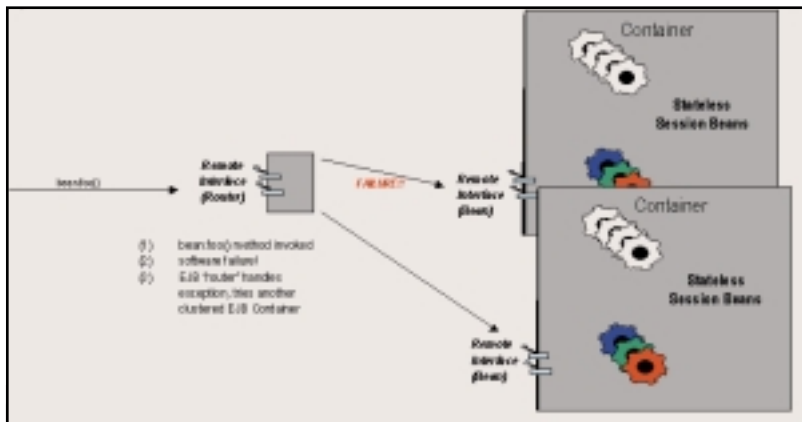


FIGURE 2 Automatic failover for client request

allow it to release any resource(s) it may have opened, and the instance is simply destroyed (or pooled).

## Scalability

While stateful server architectures are necessary in some lines of business, such as real-time quote systems or military command-and-control centers, higher scalability is generally achieved through stateless server components or session beans.

Stateless session beans are more scalable than stateful beans due to less resource allocation and management, and the clustering capabilities of stateless session beans.

With stateless session beans there's no need to hold resources or state longer than a method call. This way, the server isn't overburdened by maintaining information for an extended period of time. While passivation relieves the server of unused resources, the process requires resources to monitor the bean activity and to store the bean state to a pseudopersistent storage until the bean's services are needed again. Stateless beans aren't passivated; thus they demand no maintenance overhead.

The problem with stateful, distributed components occurs when the component's client either disconnects or fails to disconnect from the component. In some cases your server will meet a gradual degradation in performance until the server is recycled, thereby releasing the locked resources. For example, let's say the stateful session bean you're coding connects to a database in its `ejbActivate()` method and holds on to the connection to handle subsequent calls from its client. When `ejbRemove()` is invoked on the bean, it releases its connection. However, if the client forgets to call `remove()` on the bean, the database connection will be tied up until the bean is passivated. Worst of all, if a stateful bean is within a transaction, the passivation service isn't allowed to store the bean. This scenario may prevent other clients from performing their work efficiently if the bean's transaction locked rows in the database.

Instance pooling is another technique used by stateless architectures to lower the demands placed on the server and provide faster allocation of stateless components. Instance pools contain "waiting" components that are pulled from the pool on demand, then placed back into the pool when they're finished

servicing the request (see Figure 1). EJB containers in the market today usually provide some form of instance pooling to achieve scalability while decreasing the amount of garbage collection needed to clean up destroyed bean instances. Recall the life cycle of a stateless session bean. The bean is either nonexistent or method-ready. When moving from a state of nonexistent to method-ready, the container allocates memory for the bean. The container may create a new instance each time; however, a more efficient way to obtain a bean instance would be to grab preallocated memory from an instance pool of empty beans. Instance pooling is purely implementation and not mandatory in the EJB specification.

Last, stateless components scale well because they're easy to deploy in clustered EJB servers. Clustering allows components to be replicated across multiple servers on multiple machines, providing fault tolerance and load balancing. Clustering typically makes it harder for the EJB server to send requests back to the same instance over multiple calls from a client. Because stateless beans don't manage conversational state, it doesn't matter which instance services the client's next request.

Stateless beans scale better than stateful beans in a clustered environment in which 24/7 support is a necessity. If you need your EJB application to provide such support, failover is a main concern for your architecture team. In stateless server architectures any component can service a request.

Figure 2 shows how a client request on a clustered session bean can be transparently serviced during a software failure. When the client calls `foo()` on the session bean's remote interface, a smart object implementing the bean's remote interface intercepts the call and routes it to a clustered bean. During the method invocation, a software failure occurs; however, because every stateless session bean in the cluster is equally capable of servicing the client, the router simply calls the real remote interface of a functional bean in another EJB container.

Transparent failover and clustering in stateful architectures isn't so easy. While any stateless bean can service a client request in the event of a failure, in 24/7 stateful server architecture the server must perform a secondary, "hot" backup of the primary stateful bean in case the primary fails. Tracking deltas and passing changes from a primary bean to a secondary bean is time consuming and results in unnecessary chatter between the two components. I rec-

# Compuware NuMega

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

commend stateless business components over stateful components in a clustered, 24/7 environment.

### There's Always a Downside

There are a few downsides to building a distributed application on the foundation of stateless session beans. Typically, there's a need to hold conversational state across method invocations, especially in today's distributed applications, which may hold session data in an HttpSession, a stateful session bean, a singleton RMI server, a servlet or JNDI, to name a few possibilities. State must be maintained either in the client and passed everywhere it may be referenced, or in a persistent store and retrieved on an as-needed basis. Understanding how best to handle state between calls when using stateless session beans can absorb a significant amount of time in the design phases of your application.

The programming model of stateless components isn't appealing to object-oriented fanatics, who'd rather maintain references to objects that are locally accessible everywhere the objects are passed. This approach was fine in the client/server world, where the majority of processing occurred in a single executable. In distributed systems, howev-

er, it's critical to keep the amount of information you're passing between components to a minimum. Likewise, it's critical to keep the number of distributed references to other components to a minimum. Now take into account the design considerations necessary to support a stateless server architecture and you're looking at a tough paradigm shift for true OO practitioners!

Another restriction I've seen throw a wrench into many designs is the inability for stateless session beans to partake in a transaction beyond a method call. While these beans are transaction aware, the transaction participation for the session bean begins with the method invocation and ends with the return of the method. Depending on when the transaction is eventually committed or rolled back, this bean has no option to synchronize with the results. For instance, stateless session beans can't implement the SessionSynchronization interface, which allows a stateful session bean to receive notification of a coordinated transaction's results. Since each call to a stateless bean could be serviced by a different instance, and the bean's container can destroy an instance at the end of its life cycle, the transaction has no way to reliably communicate to a particular instance.

### Conclusion

While stateful server architectures have their niche, many high-volume distributed systems demand architectures built around stateless business components. These beans, available since the EJB specification 1.0, are capable of meeting this demand. They offer you the ability to build declarative, server-side components in Java that can scale to meet the needs of your enterprise application.

Microsoft has bet the farm on stateless components, believing enterprise-wide scalability can be met by taking advantage of features such as instance pooling, connection pooling and server clustering. Thankfully, the architects behind the EJB specification have included stateless session bean support in their design as well. If I had a farm to bet, I'm sure I'd gamble on building my high-volume, 24/7 application on these beans and win big!

For more information on comparisons of MTS versus EJB, check out Sun's portal to comparison documents at [web2.java.sun.com/products/ejb/ejb-vscom.html](http://web2.java.sun.com/products/ejb/ejb-vscom.html).

[jwestra@vergecorp.com](mailto:jwestra@vergecorp.com)

#### AUTHOR BIO

Jason Westra is the CTO of Verge Technologies Group, Inc. ([www.vergecorp.com](http://www.vergecorp.com)). Verge is a Boulder, Colorado, based firm specializing in e-business solutions with Enterprise JavaBeans.

Geek  
Cruises  
p/u

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

Develop-  
mentor  
p/u

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



# Persistence

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

# Best Practices for JDBC Programming

HOW TO FURTHER THE GOALS OF BEST PRACTICES  
BY IMPROVING THE MAINTAINABILITY AND  
GENERAL QUALITY OF CODE WITHIN AN APPLICATION

**A**s a consultant, developer and database administrator, I've often been asked to provide coding guidelines and tuning assistance for Java code that utilizes JDBC. Over time, I've been introduced to or developed standard coding practices that make JDBC code faster and less error-prone, and easier to read, understand and use. This article documents some of the more important "best practices" for using JDBC libraries to perform database access. As most of my clients are using Oracle database technologies, I've included several practices that are Oracle-specific.

WRITTEN BY DEREK C. ASHMORE

For the purposes of this article the goals of best practices for JDBC programming are maintainability, portability and performance.

- *Maintainability* refers to the ease with which developers can understand, debug and modify JDBC code that they didn't write.
- *Portability* refers to the ease with which JDBC code can be used with multiple databases. It turns out that JDBC doesn't make database programming as platform independent as I'd like. In addition, I consider portability a noble goal even if you have no current plans to support multiple databases. Who knows how long your code will be around and what kinds of changes will have to be made to it?
- *Performance* refers to optimizing the speed and/or memory needed to run JDBC code.

While I've labeled my recommendations best practices, these recommendations change as technology changes and as I discover even better coding practices. In addition, I'm always annoyed by articles that make recommendations and then don't explain the rationale for making them. I'll try not to make that mistake here.

## Best Practices for JDBC Programming

The most common recommendations I make to Java programmers using JDBC are the following (discussed individually later):

- Use host variables for literals – avoid hard-coding them (Oracle specific).
- Always close statements, prepared statements and connections.
- Consolidate formation of SQL statement strings.
- Use the delegate model for database connection.
- Use Date, Time and Timestamp objects as host variables for temporal fields (avoid using strings).
- Limit use of column functions.
- Always specify a column list with an select statement (avoid "select \*").
- Always specify a column list with an insert statement.

### USE HOST VARIABLES FOR LITERALS IN SQL STATEMENTS (ORACLE SPECIFIC)

I recommend that developers use host variables in SQL statements instead of hard-coding literals in SQL strings. As a convenience, many developers embed literals in SQL statements instead. I've provided an example of embedding literals in the following code. While the performance benefits of using host variables greatly improve Oracle performance, it won't hurt performance for other database platforms that I'm aware of. Note that this example places a user ID directly in the SQL statement. (As an aside, note that this example uses the "+" operator for string concatenation. While this is convenient, using `StringBuffers` and the `StringBuffer.append()` method is a faster way to concatenate strings.)

# Career Central

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

```

Statement stmt;
ResultSet rst;
Connection dbconnection;
...
stmt = dbconnection.createStatement();
rst = stmt.executeQuery("select count(*) from portfolio_info where
USER_ID = " + userID);
if(rst.next()){
count = rst.getInt(1);
}

```

To get the benefit of Oracle's optimizations, we need to use Prepared-Statements instead of statements for SQL that will be executed multiple times. Furthermore, we need to use host variables instead of literals for literals that will change between executions. In the code above the SQL statement for User id 1 will be different than for User Id 2 ("where USER\_ID = 1" is different from "where USER\_ID = 2"). A better way to approach this SQL statement is the following:

```

ResultSet rst;
PreparedStatement pstmt;
Connection dbconnection;
...
pstmt = dbconnection.prepareStatement("select count(*) from portfo-
lio_info where USER_ID = ? ");
pstmt.setDouble(1,userID);
rst = pstmt.executeQuery();
if(rst.next()){
count = rst.getInt(1);
}

```

In this code, because we're using host variables instead of literals, the SQL statement is identical no matter what the qualifying user ID is. Furthermore, we used a PreparedStatement instead of a statement. So that we can better understand the source of the performance benefit, let's walk through how SQL statements are processed by the Oracle optimizer. When SQL statements are executed, Oracle will execute (roughly speaking) the following steps:

1. Look up the statement in the shared pool to see if it has already been parsed or interpreted. If yes, Oracle will go directly to step 4.
2. Parse (or interpret) the statement.
3. Figure out how it will get the data you want; record that information in a portion of memory called the shared pool.
4. Get your data.

A flowchart of this decision process can be found in Figure 1.

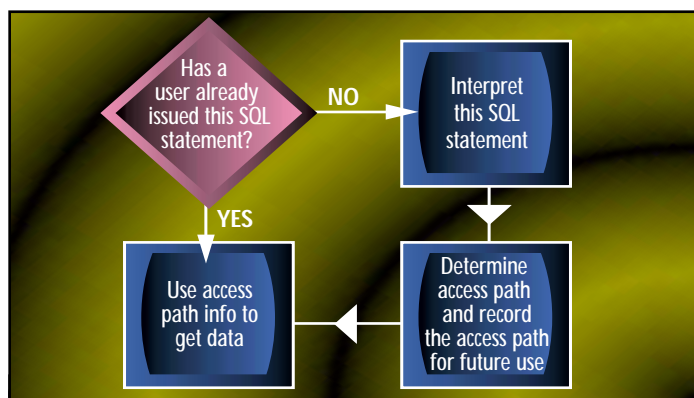


FIGURE 1 Decision process flowchart

When an Oracle user looks up a SQL statement to see if it's already been executed (step 1), he or she attempts a character-by-character match of the SQL statement. If the user finds a match, he or she can use the parse information already in the shared pool and doesn't have to do steps 2 and 3 above because the work has already been done. If you hard-

code literals in your SQL statements, the probability of finding a match is very low ("where USER\_ID = 1" isn't the same as "where USER\_ID = 2"). This means that Oracle will have to reparse the second code example for each portfolio selected. Had the code used host variables, that statement (which would look something like "where USER\_ID = :1" in the shared pool) would have been parsed once and only once.

I've experienced anywhere from a 5% to a 25% performance increase by writing SQL statements that are reusable (results vary with transaction volume, number of users, network latency and many other things). More information on this can be found in the *Oracle Tuning* manual. Within this manual look at the "Writing Identical SQL Statements" sub-heading within the "Tuning the Shared Pool" section.

While this best practice is Oracle-specific, many database platforms optimize preparing and reusing similar SQL statements. Most database platforms do this by optimizing reuse of PreparedStatement objects. Some databases, such as Cloudscape, optionally will store prepared statements in the database so they can be reused and shared by many users. Following this practice won't hurt performance with any database platform I'm aware of.

#### ALWAYS CLOSE STATEMENTS, PREPARED STATEMENTS AND CONNECTIONS

Many databases allocate resources to servicing statements, prepared statements and connections. Many database platforms continue to allocate those resources for a period of time if these objects aren't closed after use. With Oracle databases it's possible to get a "max cursors exceeded" error message when you don't close statements or prepared statements. In addition, with Oracle databases, the connections stay around on the server. This practice improves time and resources spent on maintenance to keep errors from happening.

An example can be found in Listing 1. Note that I use a "finally" block to close the PreparedStatement. I don't close the connection in the example method as it is used elsewhere in the application. Note also that I call a utility to close the PreparedStatement for me. The code for this utility can be found in Listing 2. I use a utility to do the close so I don't have to replicate the exception-catching code everywhere.

### Consolidate Formation of SQL Statement Strings

As a database administrator, a substantial portion of my time is spent reading the code of others and suggesting ways to improve performance. As you might expect, looking at the SQL statements being issued is of particular interest to me. It's hard to follow SQL statements that are constructed by string manipulation scattered over several methods. Developers who maintain this kind of code must have the same problem. It greatly enhances readability if you consolidate the logic that forms the SQL statement in one place.

Listing 2 is a good example of this point. The string manipulation to form the SQL statement is located in one place, and the SQL statement logic is in a separate static block instead of within the method itself. This is done to reduce the number of times this string concatenation happens. Also note that StringBuffer's are used for the string manipulation, not Strings. StringBuffer's are more efficient at string concatenation than Strings are. In a project I recently completed the development team adopted this convention of consolidating SQL statements in static blocks directly above the method in which they were used. We found this practice quite readable and maintainable.

#### USE DELEGATE MODEL FOR DATABASE CONNECTION

I recently had the task of making the same application runnable on Oracle 8i, Cloudscape and Oracle Lite with as few modifications to existing code as possible. The development team wanted to avoid making JDBC-related classes platform-aware. In addition, the team wanted to take advantage of some platform-specific features, such as array processing and write batching in Oracle 8i, in special cases.

I was able to port the application to multiple environments largely through manipulation of one class responsible for managing our database connection. We had the foresight to create a delegate class for the java.sql.connection that manages needed connection functions and

# Elixir

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

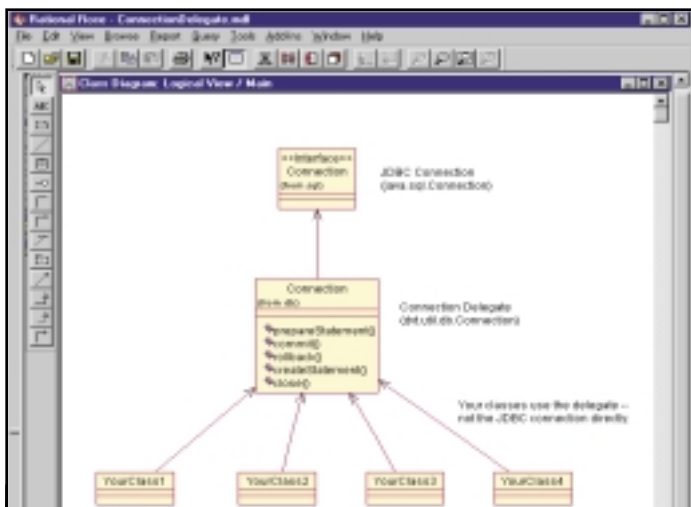


FIGURE 2 Code using the delegate

allows us to take advantage of platform-specific performance-tuning enhancements. All of our code used the delegate, not a native JDBC connection, as illustrated in Figure 2. While the specific class used for the project is proprietary, I've created another delegate, `dvt.util.db.Connection`, that illustrates the concept for the purposes of this article. The source for this delegate can be found in Listing 3.

Note that `dvt.util.db.Connection` determines that the database platform is being used. If the platform is Oracle 8i, I establish array processing by setting the default row prefetch size (available with Oracle database connections) to improve the performance of our "select" statements. I also establish write batching to improve performance of update, insert and delete statements.

Since I consolidate the platform-specific code in my connection object delegate, classes that use my connection delegate don't need to be platform specific. In case they do, however, developers can use `getPlatform()` to get information about the database platform being used. Furthermore, I can add support for additional database platforms (e.g., Cloudscape and Sybase) largely by changing this class. The connection delegate won't solve all portability issues, but it will solve a good percentage of them.

I recommend using a connection delegate even for projects that current supporting only one database platform. As we saw from recent Y2K efforts, you may find that your code is used for longer than you think, and used in other applications down the road.

#### USE DATE, TIME AND TIMESTAMP OBJECTS AS HOST VARIABLES FOR TEMPORAL FIELDS (AVOID USING STRINGS)

For convenience, I've seen many developers use strings as host variables to represent dates, times and timestamps. I think they consider `java.sql.Date`, `Time` and `Timestamp` awkward. I agree with from a coding perspective. Unfortunately, using strings as host variables for temporal fields can affect data access performance.

The following code snippet contains a SQL statement meant for an Oracle platform that uses a string variable to represent a DATE field. Without an understanding of how the database optimizers work, this appears to be an acceptable coding technique. For the small inconvenience of using a "to\_char" function in the SQL statement, we avoid the Java work of converting a `java.sql.Date` or `Timestamp` into a more easily displayable data type elsewhere in the code.

```

Select sum(sale_price)
From order_sales
Where to_char(sale_dt,'YYYY-MM-DD') >= ?
  
```

Unfortunately, Oracle and most database optimizers can't use an index to speed up performance of the query in this snippet. Developers will have to read all rows of the `order_sales` table and convert the `sale_dt` of all rows to a string before they can do the comparison to see which rows satisfy the where clause of the query.

If we rewrite the query in the snippet to use a `java.sql.Timestamp` host-variable, Oracle (and most of the common database platforms) will use an index and significantly improve performance in most cases, as follows:

```

Select sum(sale_price)
From order_sales
Where sale_dt >= ?
  
```

For applications that use Oracle exclusively, I recommend using `java.sql.Timestamp` exclusively. Oracle's DATE data type actually contains time information (hours, minutes, seconds) as well as date information. Most other database platforms would call this type of field a `TIMESTAMP`. Oracle has no direct counterpart for a DATE (which has year, month and day only) and TIME data type offered by other platforms.

#### LIMIT USE OF COLUMN FUNCTIONS

I generally recommend that developers limit use of column functions to the select lists of select statements. Moreover, I tend to stick to aggregate functions (e.g., count, sum, average) needed for select statements that use a "group by" clause. I make this recommendation for two reasons: performance and portability. Limiting function use to select lists (and keeping it out of where clauses) means that the use of a function won't block the use of an index. In the same way that the use of the "to\_char" function prohibited the database from using an index in the earlier code snippet, column functions in where clauses likely prohibit the database from using an index.

In addition, many of the operations for which developers use SQL column functions (data type conversion, value formatting, etc.) are faster in Java than if the database did them. I've had between a 5% and a 20% performance improvement in many applications by opting to avoid some column functions and implementing the logic in Java instead. Another way to look at it is that column functions aren't tunable as we don't control the source code. Implementing that logic in Java makes it code that we can tune if need be.

Moreover, using non-ANSI-standard column functions can also cause portability problems. There are large differences in which column functions are implemented by the database vendors. For instance, one of my favorite Oracle column functions, "decode", which allows you to translate one set of values into another, isn't implemented in many of the other major database platforms. In general, column function use such as the use of "decode" has the potential to become a portability issue.

#### ALWAYS SPECIFY A COLUMN LIST WITH A SELECT STATEMENT (AVOID "SELECT \*")

A common shortcut for developers is to use the "\*" in select statements to avoid having to type out a column list. The line below illustrates this shortcut while the snippet immediate following illustrates the alternative where desired columns are explicitly listed.

```

Select * from customer

Select last_nm, first_nm, address, city, state, customer_nbr from
customer
  
```

I recommend that developers explicitly list columns in select statements as illustrated above. The reason is that if the columns in any of the tables in the select are reordered or new columns are added, the results obtained with the select-asterisk shortcut will change and the class will have to be modified. For example, suppose a database administrator changes the order of the columns and puts column `customer_nbr` first (there are valid reasons why a DBA could reorder columns). In addition, suppose the DBA adds a column called `country`. The developer who used the shortcut `select * from customer` will have to change code. All the offset references used in processing the Resultset will change. The developer who explicitly listed all columns can be oblivious to the change because the code will still work.

Explicitly listing columns in a select statement is a best practice because it prevents the need for maintenance in some cases.

# SlangSoft

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

# Go Online and Subscribe Today!

[www.SYS-CON.com](http://www.SYS-CON.com)



or

Call 1-800-513-7111

Subscribe to the Finest Technical  
Journals in the Industry!

GET YOUR OWN!  
SUBSCRIBE NOW!

## ALWAYS SPECIFY A COLUMN LIST WITH AN INSERT STATEMENT

A common shortcut for developers is to omit the column list in insert statements to avoid having to type out a column list. By default, the column order is the same as physically defined in the table. The first snippet below illustrates this shortcut while the next one illustrates the alternative where desired columns are explicitly listed.

Insert into customer

```
Values ('Ashmore', 'Derek', '3023 N. Clark', 'Chicago', 'IL', 555555)
```

Insert into customer

```
(last_nm, first_nm, address, city, state, customer_nbr)
```

```
Values (?, ?, ?, ?, ?, ?)
```

I recommend that developers explicitly list columns in insert statements as illustrated in the second snippet above. The reason is the same as why we should explicitly list columns in select statements. If the columns in any of the tables in the select are reordered or new columns are added, the insert could generate an exception and insert in class will have to be modified. For example, suppose a DBA, as in the previous example, changes the order of the columns, puts column customer\_nbr first and adds a column called country. The developer who used the first shortcut above will have to change code. The developer who explicitly listed all columns may be oblivious to the change because the code may still work. In addition, note that the version in second snippet above uses host variables so the same PreparedStatement can be used for all inserts if there are multiple inserts.

Explicitly listing columns in an insert statement is a best practice because it prevents the need for maintenance in many cases.

## Recommendations for Stored Procedure Usage

Stored procedure programming languages (such as Oracle's PL/SQL) are handy and in many cases very convenient. I use them often for utility scripts and data-cleansing activities. I'm often asked about recommendations for stored procedure use in applications, but as their capabilities differ greatly among the major database platforms, I can't give platform-independent advice on the subject. I can, however, provide some thoughts on stored procedure use as it relates to portability and performance.

As these languages differ so greatly, their use within applications causes portability issues. For instance, some stored procedure languages allow procedures to return result sets, some do not. Some stored procedure languages allow temporary tables (usable within the current session only), some do not. We could find many more differences, but I think the point is clear. If portability is a concern, I recommend avoiding use of stored procedures except for database triggers.

Performance is a tougher issue because it differs radically between database vendors. Stored procedure use for some database platforms enhances performance; in others it degrades it. For Oracle platforms I advocate stored procedures within Java applications for database triggers only. For most other situations their use provides no benefit. If you want a more detailed discussion on when and how to use stored procedures, functions and packages within Oracle databases, see my article in JDJ December 1999 (Vol. 4, issue 12).

## Summary

This article has discussed several ways to make JDBC code more performance-, maintenance- and portability-friendly on an individual basis. I always recommend team code reviews and documented coding standards as ways to develop more best practices and consistently apply existing practices. Furthermore, team code reviews help further the goals of best practices by improving the maintainability and general quality of code within an application. ☛

## AUTHOR BIO

Derek Ashmore is the senior vice president of development for Delta Vortex Technologies, a Chicago-based consulting firm. He has designed, implemented and managed Oracle-based projects of many different types and sizes.

[dashmore@dvt.com](mailto:dashmore@dvt.com)

JAVA DEVELOPERS JOURNAL.COM



# TideStone

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

### Listing 1

```
private static Connection dbConnection;
private static HashMap companyMap = new HashMap();
private static StringBuffer tempBuffer;

private static final String insertPortfolioCompaniesSQL;
static{
    tempBuffer = new StringBuffer("INSERT INTO Portfolio_Companies ");
    tempBuffer.append("(portfolio_id,business_entity_id, nbr_shares_held) ");
    tempBuffer.append("VALUES(?,?,?)");
    insertPortfolioCompaniesSQL = tempBuffer.toString();
}
private void insertPortfolioCompanies() throws SQLException
{
    PreparedStatement pstmt = null;
    try {
        pstmt = dbConnection.prepareStatement(insertPortfolioCompaniesSQL);
        if(companyMap.isEmpty()){
            Collection companyCollection = companyMap.values();
            for(Iterator i = companyCollection.iterator(); i.hasNext(); ){
                PortfolioConstituent pc = (PortfolioConstituent)i.next();
                pstmt.setDouble(1,pc.getPortfolioID());
                pstmt.setDouble(2,pc.getCompanyID());
                pstmt.setDouble(3,pc.getShares());

                pstmt.execute();
            }
            pstmt.close();
            dbConnection.commit();
        } finally { JDBCUtilities.close(pstmt); }
    }
}
```

### Listing 2

```
static public void close (ResultSet rs) {
    try { if (rs!=null) rs.close(); } catch (Exception e) {}
}

// Works for PreparedStatement also since it extends
// Statement.
static public void close (Statement stmt) {
    try { if (stmt!=null) stmt.close(); } catch (Exception e) {}
}

static public void close (java.sql.Connection conn) {
    try { if (conn!=null) conn.close(); } catch (Exception e) {}
}

static public void close (dvt.util.db.Connection conn) {
    try { if (conn!=null) conn.close(); } catch (Exception e) {}
}
```

### Listing 3

```
package dvt.util.db;

import java.lang.*;
import java.sql.*;
import oracle.jdbc.driver.*;

/**
 * Connection represents a generic database connection.
 *
 * @author Derek C. Ashmore
 * @version 1.0
 */

public class Connection {

    public static final String ORACLE_8I = "8I";
    public static final String CLOUDSCAPE = "CLOUDSCAPE";
    public static final String ORACLE_LITE = "ORACLE_LITE";
    public static final String GENERIC = "GENERIC";
```

```
public static final String CLOUDSCAPE_DRIVER =
"COM.cloudscape.core.JDBCdriver";
public static final String ORACLE_LITE_DRIVER =
"oracle.lite.poljdbc.POLJDBCdriver";
public static final String ORACLE_8I_DRIVER =
"oracle.jdbc.driver.OracleDriver";

/**
 * Registers the database driver and obtains the speci-
 * fied database connection.
 *
 * @param JDBCdriver
 * @param connectString
 * @param localUserId
 * @param localPassword
 */
public Connection( String jdbcDriverName,
String connectString,
String localUserId,
String localPassword) {

    setUserId(localUserId);
    setPassword(localPassword);

    if (jdbcDriverName.equals(ORACLE_LITE_DRIVER))
        platform = ORACLE_LITE;
    else if (jdbcDriverName.equals(CLOUDSCAPE_DRIVER))
        platform = CLOUDSCAPE;
    else if (jdbcDriverName.equals(ORACLE_8I_DRIVER))
        platform = ORACLE_8i;
    else platform = GENERIC;

    registerDBDriver(jdbcDriverName);
    currentConnection = getConnection(connectString, connectString);
}

/**
 * Registers the database drivers.
 */

private void registerDBDriver(String jdbcDriverName) {
    try {
        Class.forName( jdbcDriverName );
        this.setDriverRegistered( true );
    }
    catch (Exception DBError) {
        System.out.println(DBError.getMessage());
        DBError.printStackTrace();
    }
}

/**
 * Prepares a SQL statement.
 *
 * @param localSQLString
 * @exception java.sql.SQLException
 */

public PreparedStatement prepareStatement(String localSQLString) throws SQLException {
    return currentConnection.prepareStatement(localSQLString);
}

/**
 * Returns the current Oracle database connection.
 */

private java.sql.Connection getConnection(String jdbcDriverName, String connectString) {

    if (isConnected()) return currentConnection;
    if (! isDriverRegistered()) registerDBDriver(jdbcDriverName);

    try {
        currentConnection = DriverManager.getConnection(connectString,
        localUserId, password);
    }
```

# 4th Pass

[www.4thpass.com](http://www.4thpass.com)

```

        if (platform.equals(ORACLE_8I)) {
            OracleConnection oConnect = (OracleConnec-
            tion) currentConnection;
            oConnect.setDefaultRowPrefetch( default
            PrefetchSize );
            oConnect.setDefaultExecuteBatch( default
            WriteBatchSize );
            Statement alterDateFormat = currentConnec-
            tion.createStatement();
            alterDateFormat.execute("alter session set
            NLS_DATE_FORMAT = 'YYYYMMDDHHMISS'");
        }
        currentConnection.setAutoCommit( defaultAutoCom-
        mitSetting );

        this.setConnected(true);
    }
    catch (SQLException DBError) {
        System.out.println(DBError.getMessage());
        DBError.printStackTrace();
    }
    return currentConnection;
}

public java.sql.Connection getConnection() { return
currentConnection; }

/**
 * Provides the user id associated with the current
 * database connection.
 */

public String getUserId() {
    return userId;
}

/**
 * Sets the user id used to obtain the database connection.
 *
 * @param localUserId
 */

private void setUserId(String localUserId) {
    userId = localUserId;
}

/**
 * Provides the password used to obtain the database
 * connection.
 */

public String getPassword() {
    return password;
}

/**
 * Sets the password used to obtain the database connection.
 *
 * @param localPassword
 */

private void setPassword(String localPassword) {
    password = localPassword;
}

/**
 * Provides information as to whether or not database
 * driver registration has occurred.
 */

private boolean isDriverRegistered() {
    return driverRegistered;
}

/**
 * Issues a database commit to save all pending changes
 * to the database.
 */

public void commit()    throws SQLException {

```

```

        currentConnection.commit();
    }

    /**
     * Issues a database rollback to abort all pending
     * changes to the database.
     */

    public void rollback()    throws SQLException {
        currentConnection.rollback();
    }

    /**
     * Issues a database disconnect.
     */

    public void close()    throws SQLException {
        currentConnection.close();
    }

    /**
     * Issues a database disconnect and closes a given
     * statement
     * (provided for programatic convenience -- doesn't logically
     * belong here).
     *
     * @param PreparedStatement
     */

    public void close(PreparedStatement preparedStatement)
    throws SQLException {
        try {
            preparedStatement.close();
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        this.close();
    }

    /**
     * Issues a database disconnect and closes a given
     * statement and result set (provided for programatic
     * convenience -- doesn't logically belong here).
     *
     * @param PreparedStatement
     * @param ResultSet
     */

    public void close(PreparedStatement preparedStatement,
    ResultSet resultSet)    throws SQLException {
        try {
            resultSet.close();
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        this.close(preparedStatement);
    }

    /**
     * Records registration status of database drivers.
     *
     * @param LocalDriverRegistered
     */

    private void setDriverRegistered(boolean LocalDriverReg-
    istered) {
        driverRegistered = LocalDriverRegistered;
    }

    /**
     * Provides information about current database connection
     * status.
     */

    public boolean isConnected() {
        return connected;
    }
}

```

# Evergreen

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

```

/**
 * Sets database connection status information.
 *
 * @param LocalConnected
 */

private void setConnected(boolean LocalConnected) {
    connected = LocalConnected;
}

/**
 * Sets the array size used for select statements.
 *
 * @param arraySize
 */

public void setPrefetchSize(int arraySize) throws SQLException {
    if (platform.equals(ORACLE_8i)) {
        OracleConnection oConnection = (OracleConnection)
            currentConnection;
        oConnection.setDefaultRowPrefetch( arraySize );
    }
}

/**
 * Provides the array size used for select statements.
 *
 */

public int getPrefetchSize() throws SQLException {
    if (platform.equals(ORACLE_8i)) {
        OracleConnection oConnection = (OracleConnection)
            currentConnection;
        return oConnection.getDefaultRowPrefetch();
    }
    else return -1;
}

/**
 * Sets the array size used for update, insert, and
 * delete statements.
 *
 * @param arraySize
 */

public void setWriteBatchSize(int arraySize) throws
SQLException {
    if (platform.equals(ORACLE_8i)) {
        OracleConnection oConnection = (OracleConnection)
            currentConnection;
        oConnection.setDefaultExecuteBatch( arraySize );
    }
}

/**
 * Provides the array size used for update, insert, and
 * delete statements.
 *
 */

public int getWriteBatchSize() throws SQLException {
    if (platform.equals(ORACLE_8i)) {
        OracleConnection oConnection = (OracleConnection)
            currentConnection;
        return oConnection.getDefaultExecuteBatch();
    }
    else return -1;
}

/**
 * Sets the AutoCommit specification for the connection.
 * Set to true to
 * have commits automatically issued. Set to false to
 * handle commits
 * and rollbacks manually.
 *
 * @param autoCommitInd
 */

```

```

public void setAutoCommitSetting(boolean autoCommitInd)
throws SQLException {
    currentConnection.setAutoCommit( autoCommitInd );
}

/**
 * Provides the AutoCommit specification for the connec-
 * tion. True means that commits are automatically
 * issued. False means that commits and rollbacks are
 * handled manually.
 *
 */

public boolean getAutoCommitSetting() throws SQLException {
    return currentConnection.getAutoCommit();
}

/**
 * Provides database platform used for the connection.
 */
public String getPlatform() { return platform; }

/**
 * User ID used to obtain database connection.
 */
private String userId;

/**
 * Password used to obtain database connection.
 */
private String password;

/**
 * Prefetch size (select array processing batch size)
 * for database connection.
 */
private int defaultPrefetchSize = 100;

/**
 * Default update Batch size (number of write DML state
 * ments to queue) for database connection.
 */
private int defaultWriteBatchSize = 20;

/**
 * Auto Commit mode for database connection.
 */

private boolean defaultAutoCommitSetting = false;

/**
 * Indicates database driver registration status.
 */
private boolean driverRegistered;

/**
 * Indicates last known database connection status.
 */
private boolean connected;

/**
 * Contains JDBC connection information.
 */
private java.sql.Connection currentConnection = null;

/**
 * Contains JDBC connection information.
 */
private String platform = null;
}

```



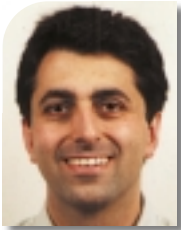
# Modis

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

# Improving Reliability and Scalability of Middle-Tier Servers

## Tips on how to ease the pain of migrating to a new middleware implementation

WRITTEN BY  
TODD SCALLAN



Many three-tier applications built using various middleware products ultimately fail in production due to a lack of scalability, flexibility or reliability. This can trigger a need to migrate an application from one middleware product to another. In this article we'll discuss a process for porting servers between CORBA and EJB middleware implementations.

Object request brokers and application servers are popular middleware technologies you can readily find in the middle tier of distributed applications. These technologies impact the reliability and scalability of the applications they support since middleware introduces a higher degree of sophistication – and therefore greater complexity.

The arrival of distributed object technology into the mainstream of software development has led to the emergence of CORBA and RMI as standard object communication mechanisms. Moreover, thanks to the growing number of e-business applications being developed for the Web, application servers have gained recent prominence within software projects

### Cross-Server Interchangeability

As a way to address component interoperability and interchangeability across servers, the Enterprise JavaBeans specification defines an execution and services framework for server-side Java components. EJB relies on the underlying communication mechanism – typi-

cally CORBA or RMI – for exposing a component's public interface. Figure 1 illustrates how EJB components can interoperate with CORBA and Java objects.

### Why Migrate?

The primary advantages of technologies like CORBA, RMI and EJB are interoperability and portability. As the use of these middleware standards continues to grow, requirements for functional richness and scalability help drive decisions about which implementation to choose. Changes in business and technical requirements, as well as vendor offerings, often result in the need to migrate between ORBs or application servers. More to the point, applications built using various middleware products all too often fail due to a lack of scalability, flexibility or reliability. When faced with this situation, the best course of action may be to migrate your application software to an alternative middleware product.

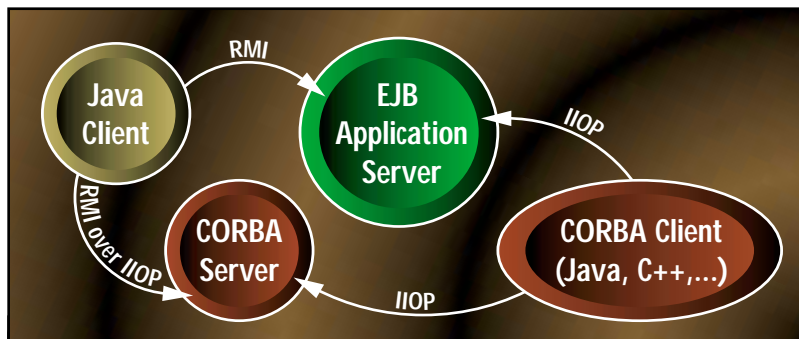
Migration usually involves porting your application from one vendor's mid-

dleware implementation to another, or even between implementations provided by the same vendor. Some vendors offer different application servers depending on whether you require robust EJB support for faster development versus CORBA underpinnings for enterprise scalability. In any event, migration raises a number of potential issues for the application components being ported and possibly also raises interoperability issues for components remaining on dissimilar implementations. (We'll deal with some of these issues shortly.)

### Migration Process

Now let's explore a process for migrating applications between standard middleware implementations. The aims of this process are to capture baseline test cases and performance metrics, then port the application code – and finally to validate that the port was successful. To quantify the success of a migration effort it's essential to capture test cases and metrics methodically – for example, has the migration indeed resulted in increased scalability? Test automation helps streamline this process, providing a cost-effective way to successfully complete a migration effort and quantify the results.

The migration of an application involves more than just porting source code – it entails careful planning, analysis and validation of results. While CORBA and EJB define interoperability and interchangeability standards, they don't prescribe how your middleware provider may have implemented the underlying infrastructure or how you should architect your application components. So you might very well find that migration involves rearchitecting



**FIGURE 1** Java programs can communicate with EJBs using either RMI or RMI over IIOOP – the CORBA wire-level protocol. CORBA clients can be implemented in any programming language and use IIOOP to talk to EJB or CORBA servers.



# New Atlanta

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

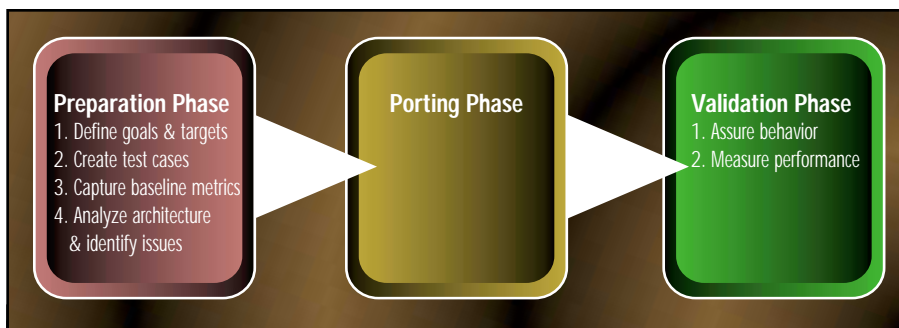


FIGURE 2 Migration process

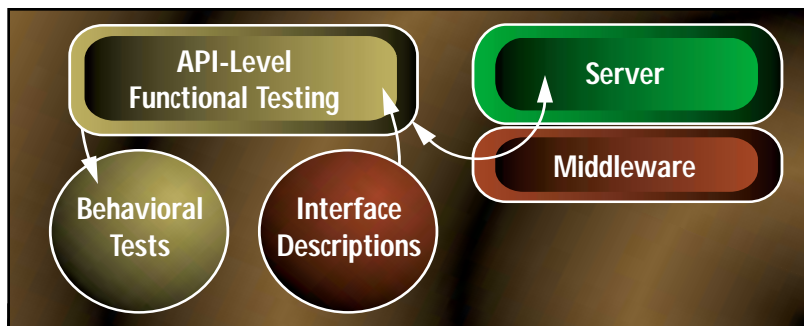


FIGURE 3 Behavioral test cases are created by manually interacting with the server through its public interface and the underlying middleware implementation.

portions of your application to compensate for differences between the current middleware implementation and the new one.

The migration process consists of three phases: preparation, porting and validation. These phases, illustrated in Figure 2, are described in the next sections.

## Preparation Phase

In preparation for a port, you must first create behavioral test cases and capture performance metrics. These tests and metrics should be set in the context of specific goals and targets for the ported application. For example, a goal may be to introduce automatic load balancing. Or a target may be to improve an application's performance by an order of magnitude.

The steps for the preparation phase are:

1. *Define goals and targets.*
2. *Create behavioral test cases.*
3. *Capture baseline performance metrics.*
4. *Analyze the application architecture and identify potential issues.*

### 1. DEFINE GOALS AND TARGETS.

First we need to know where we're going. Typically, the major reason for migrating is to improve the reliability of an application that's targeted for production use. Increasingly such applications are e-business engines that will

experience significant demands in terms of client requests, resource availability, and so on. By defining concrete goals and targets it should be possible to align your application's usage requirements with the capabilities of the target middleware implementation. For instance, does the middleware provide automatic failover or will you have to build this into your application?

### 2. CREATE BEHAVIORAL TEST CASES.

Test cases must be created so that the behavior of the ported application

can be validated. Since middle-tier server applications can consist of multiple processes that interact through published APIs, this isn't a traditional regression testing exercise. Rather, test cases must be defined for each of an application's components (see Figure 3). The challenge becomes how to accomplish this without having to write static test drivers for each component's API.

The following observations can be made about a middle-tier server:

- Public interfaces are typically specified in IDL or Java.
- Objects can be accessed dynamically via standard mechanisms such as CORBA's DII or Java's reflection API.
- A client program can invoke methods and manipulate attributes of server objects irrespective of the implementation language and physical location of the server.

Test cases are most effectively created using an automated functional testing tool that can interact with object-based servers. (Segue's SilkPilot is such a tool for CORBA and EJB servers.) The general approach is to exploit dynamic invocation facilities, allowing you to connect to one or more servers, view information about live objects within the servers, invoke methods with parameters, and view or modify attributes.

Testing should be done within the context of the application's usage model. A banking application, for example, requires an account to be created before funds can be deposited. When an interactive test cycle is completed, a corresponding test case should be generated and saved. Test cases are run later during the validation phase of the

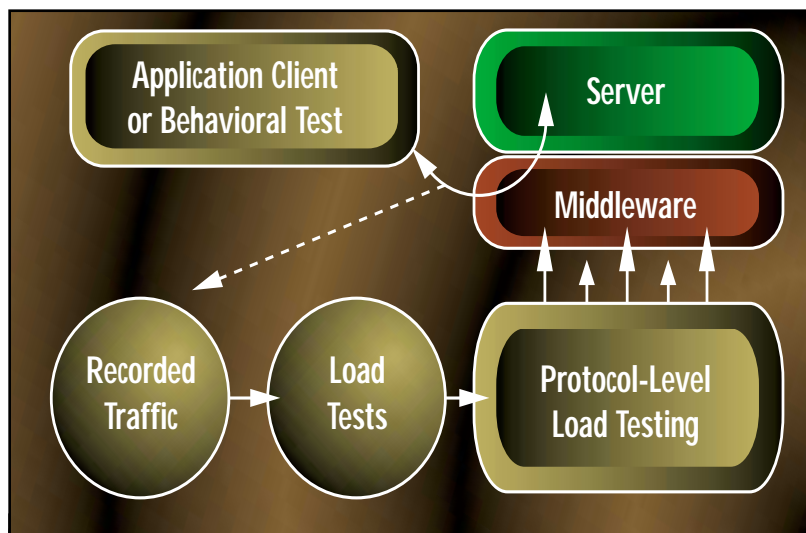


FIGURE 4 Performance metrics can be obtained by recording message traffic for a representative set of interactions with the server, then creating and executing load tests based on expected usage models.

# Intuitive

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

migration effort to ensure that the newly ported application components are functioning properly.

### 3. CAPTURE BASELINE PERFORMANCE METRICS.

You need to quantify the performance of the original application for comparative analysis during the validation phase. Measurement of an application's performance involves simulating usage models under various loads. It's highly advisable to use an automated load-testing tool – such as Segue's SilkPerformer – to accurately simulate message traffic and measure the capacity and scalability of your server applications (see Figure 4).

The first step in capturing performance metrics is to record message traffic for a typical set of interactions with the server. You can intercept and record IIOP communication used by ORBs and highly scalable application servers, for instance. Then you can create a load test by scaling up the recorded traffic to represent the anticipated usage volume, such as a thousand banking clients making deposits rather than just the one representative case used to generate the initial traffic.

Data values captured within the recorded traffic should be replaced with randomized values to create a realistic simulation. Each of the thousand simulated banking clients would thus have a unique account number and deposit amount, for instance. Workloads can then be defined in terms of machines in the network generating the workload, number of concurrent clients executed, transaction frequencies and duration of the simulation. Scalability measurements become extremely useful when obtained under various workloads, such as starting a simulation with 20 clients and then adding 10 more every 30 seconds up to a thousand concurrent clients. Performance measurements include the throughput of an application component and the response time as perceived by client applications.

### 4. ANALYZE THE APPLICATION ARCHITECTURE AND IDENTIFY POTENTIAL ISSUES.

The architecture of the existing application must be analyzed and reconciled against the goals and targets for the application's performance when migrated to the new middleware. Decisions in the initial architecture were often influenced by limitations of the middleware originally used. So the migration may include some rearchitecting of the application to remove certain design concessions or workarounds that aren't necessary any longer.

Issues may also arise from the absence of particular features in the target middleware, or differences introduced by an alternate approach to implementing the underlying infrastructure; for example, multithreaded servers versus single-threaded/multiprocess. All potential issues should be identified as early in the migration process as possible.

Some specific architectural issues to consider include:

- **How clients connect to a server:** For example, does your middleware implementation provide an API for binding directly to an object? Are you required to use a naming service, factory or other facility?
- **Mechanisms for creating and exposing objects within servers:** Are you required to use either a basic object adapter (BOA) or portable object adapter (POA)?
- **Object management:** What activation modes are available? How is object lifecycle managed? Does the middleware implement a dedicated connection between each client and your object or does it pool connections?
- **Load balancing:** Does the middleware implement a transaction-processing framework? Does it provide some other functionality such as object groups?
- **Threading:** Can servers be safely implemented using threads, or does the middleware require single-threaded/multiprocess servers?
- **Nonstandard features:** Does your application take advantage of vendor-specific features such as interceptors, locators, loaders or client-side caching? Are you using special features to accomplish "nonstandard" tasks like piggybacking extra data onto a message?
- **Fault resilience:** Are you depending on middleware-dependent features such as activation modes, automatic connection reestablishment or a transaction-processing framework?
- **Transaction support:** Does your application assume the middleware will handle transaction starts, commits and rollbacks?
- **Process colocation:** Are you using special features to colocate clients and servers within a common address space?
- **Callbacks:** Do your clients expect callbacks from your servers? How does the middleware support this?

## Porting Phase

Having completed the preparation phase, the application source code can

now be ported to the target middleware. Based on the conclusions drawn from your analysis during the preparation phase, it might be necessary to make changes to source code. This is quite likely if your migration effort includes taking advantage of features that are available only in the new middleware.

The steps for the porting phase are the same as for any porting effort: modify source code as necessary, recompile on the target middleware and platform, test that the application functions properly and repeat as required. When porting is completed, it's time to move on to the validation phase.

## Validation Phase

After the application is operational on the new middleware, you must validate the achievement of your goals and targets. The two steps for this phase are:

1. *Make sure that the application behaves properly.*
2. *Measure the application's performance.*

### 1. MAKE SURE THAT THE APPLICATION BEHAVES PROPERLY.

The test cases created during the preparation phase can be used to verify the behavior of the newly ported application. This is an API-level regression test. Each test case should be executed and the results reviewed to make sure that the new server components are responding properly.

### 2. MEASURE THE APPLICATION'S PERFORMANCE.

The load tests created during the preparation phase can be rerun to generate new performance metrics. These measurements can be compared to the initial baseline metrics and performance targets to provide quantifiable evidence that the application's performance has indeed improved.

## Final Thoughts

Middleware standards like CORBA and EJB provide a marvelous basis upon which to design and build three-tier applications. However, choosing the right product for the lifetime of an application is made difficult – even perhaps unlikely – by multiple middleware implementations, each with their own unique design approaches, vendor-specific features and inevitable limitations. If improving the reliability of your CORBA or EJB application requires migrating to a new middleware implementation, the process outlined in this article should help ease the pain. ☛

### AUTHOR BIO

Todd Scallan is the director of project management for Segue Software's distributed computing products. He holds a BS in electrical engineering from Lehigh University and an MS in computer engineering from Syracuse University.

tscallan@segue.com

# Fiorano

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

# Sterling

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

# Software

software.com

# LingoGUI 1.1

by Slangsoft

Use this toolkit to harness your Java apps to support international languages, and LingoGUI as part of your development environment

REVIEWED BY JIM MILBERY



AUTHOR BIO

Jim Milbery is a software consultant with Kuromaku Partners LLC ([www.kuromaku.com](http://www.kuromaku.com)), based in Easton, Pennsylvania. He has over 15 years of experience in application development and relational databases.

[jmilbery@kuromaku.com](mailto:jmilbery@kuromaku.com)



LingoGUI toolkit: Slangsoft  
 Web: [www.slangsoft.com](http://www.slangsoft.com)  
 Phone: 972-2-648-2424  
 Fax: 972-2-648-2425  
 Address: Slangsoft  
 Technology Park Building 1B  
 Manhat, Jerusalem 91847, Israel

Test Environment  
 Client: Sony Vaio Ultrathin, 128MB RAM, 7GB  
 disk drive, Windows 98 Release 2

Pricing  
 Up to 100 users: \$900, up to 1000: \$2,700  
 Unlimited - \$3,600

Desktop applications, handheld devices, telephones – Internet applications can be delivered from a variety of sources and appliances. They can originate in one country and be delivered to another in the blink of an eye. As a result, no matter whether you're constructing an informational Web site or an e-commerce solution, you won't be able to service all possible users unless you can "speak their language."

The folks at Jerusalem-based Slangsoft, one of a crop of high-technology companies to emerge from Israel's version of Silicon Valley, consider it their mission to assist developers with the process of harnessing their applications to support the needs of international communities of users. Slangsoft was kind enough to let me take a look at the release candidate for the newest version of their LingoGUI toolkit for delivering multilingual Web applications.

## LingoGUI Basics

LingoGUI is a development package that consists of a series of lightweight Java graphical-user-interface components, associated fonts and virtual keyboards for 42 different national languages. (LingoGUI provides a framework for managing international languages, but it doesn't actually translate text for you.) The LingoGUI toolkit comes in two different formats, a regular edition (Java class files) and a LingoGUI beans edition that can be used with any of the popular Java Interactive Development Environments such as Oracle JDeveloper and VisualCafé. LingoGUI is meant as a replacement UI for a set of the most popular AWT widgets as listed in Table 1.

Slangsoft provides the necessary fonts and virtual keyboards so you can use LingoGUI no matter what platform the application will run on. You can deliver your program as a Java application, in which case the fonts and virtual keyboards can be stored locally, or as a Java applet – in which case the fonts and keyboards can be stored on your Web server and delivered over the Internet. (You can access Slangsoft's demo applications over the Internet from within your browser and see it for yourself.)

## Working with LingoGUI

I downloaded the release 1.1 candidate from the Slangsoft Web site and installed the software quickly and easily. I got started by working through the sample HTML pages in the examples subdirectory. These pages have links to working Java applets, which gave me a basic understanding of how to work with the LingoGUI. The main demo application shows off all the major components of the product, as you can see in Figure 1.

This same applet can be accessed from Slangsoft's Web site, so you don't even need to download the product to get a handle on how it works. The demo applet doesn't model a real application; it's merely a sample form that shows off all of the LingoGUI widgets using a single panel. By setting the language listbox to "Russian," I was able to enter a string into the text field and then use the push button to propagate the text to the other widgets. The LingoGUI widgets work like any other AWT component, and you can set the same attributes for them including bold, italic, colors and image backgrounds.

Harnessing your applications to support multiple languages requires more than just visual widgets; you also need fonts to display international text and a means to enter text into fields using a different language. Slangsoft provides a Unicode font with LingoGUI called *EmlDefault*, which supports a range of sizes from eight to 72 points. Furthermore, the *EmlDefault* font supports the standard set of style options including plain, bold, italic and underlining. The *EmlDefault* font is packaged inside the small LingoGUI JAR files, which means that it can be downloaded over the Web without difficulty at application-execution-time. If you prefer to use your own fonts with the LingoGUI widgets, you're free to do so, but this defeats the purpose of building your application with LingoGUI. Should you need to make use of your own font, Slangsoft has developed a technique for converting TrueType fonts into compressed Java classes. You can access this service by contacting Slangsoft directly.

Programmers (and end users) can test their multilingual applications by using the virtual keyboard as shown in Figure 2.

I selected "Russian" in order to display the keyboard shown in the figure – the keyboard is composed of Cyrillic characters instead of the

| LINGOGUI COMPONENT | DESCRIPTION  |
|--------------------|--|
| SLabel             | Text label, used as a label for other widgets or for displaying text strings on a form |
| SButton            | Command button   |
| SToggleButton      | Command button that operates like an on/off switch                                     |
| SCheckBox          | Individual checkbox  |
| SCheckBoxGroup     | Group of checkboxes (allows checkboxes to act as radio buttons)                        |
| SList              | List box   |
| STextArea          | Text area (multiline)  |
| STextField         | Simple text field  |

TABLE 1 LingoGUI's AWT widgets



# InetSoft

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

usual North American QWERTY. Using this applet, I was able to test out my text fields and text areas by entering Cyrillic characters. The keyboard applet runs as a separate window but it communicates with the primary applet window as a nonmodal application would. You can move back and forth between either panel and LingoGUI will load the “typed” text into whichever widget on the main panel has “focus.” Closing down the main applet window causes the virtual keyboard to disappear as well, which is exactly what you’d expect it to do. This clever feature allows you to test out multiple languages without having to actually install the language onto your hardware platform.

The really powerful thing about the virtual keyboard is that it automatically maps the keys on your computer’s keyboard as well. Once I had the virtual keyboard displayed in Russian, I was able to enter Cyrillic characters directly on my PC keyboard. (Although I stuck with the Russian language for most of my testing, the Asian languages – Chinese, Japanese and Korean – are impressive as well.)

Once your application has been built using LingoGUI, you can use the Resource Bundle File Builder to manage the various languages that you’ll use within your applications (see Figure 3).

Basically, the Resource Builder allows you to associate resource keys using English along with text strings using one of the supported international languages. You can enter Unicode characters directly into LingoGUI widgets, but you’ll find it easier to use the Resource Bundle File Builder to manage your text strings. The virtual keyboards are available inside the Resource Bundle File Builder as well, and I was able to add some Russian text for the various resource keys in the sample resource files. (Since I don’t speak a word of Russian beyond “Stolichnaya,” I’ve no idea just what I entered, but it looked cool just the same.) The output of the Resource Bundle File Builder is a series of resource file classes with Unicode characters bundled in. I found this utility useful, but it’s a very basic interface. (For example, the only way to open a new resource file is to restart the Resource Bundle File Builder.)

While Slangsoft covers the basic set of AWT widgets, most of the popular Java development tools provide for an extended set of AWT-compatible widgets. VisualCafé, for example, provides a formatted text field that can be used for masking date fields (and they also offer some nice features such as “data-bound” AWT widgets as well). In the meantime, nothing precludes you from mixing these other AWT widgets with the LingoGUI widgets so long as you’re willing to live with “partial” internationalization. I’d expect Slangsoft to add support for some of these additional AWT widgets in the future. (They mentioned they’re already working on a Swing version of LingoGUI that will work with JDK 1.3, which you can read about on their Web site.)

## Summary

Slangsoft provides a nice toolkit for harnessing your Java applications to support international languages, and I’d recommend that you consider LingoGUI as part of your development environment. You can easily evaluate the software yourself by downloading it from their Web site [www.slangsoft.com](http://www.slangsoft.com).

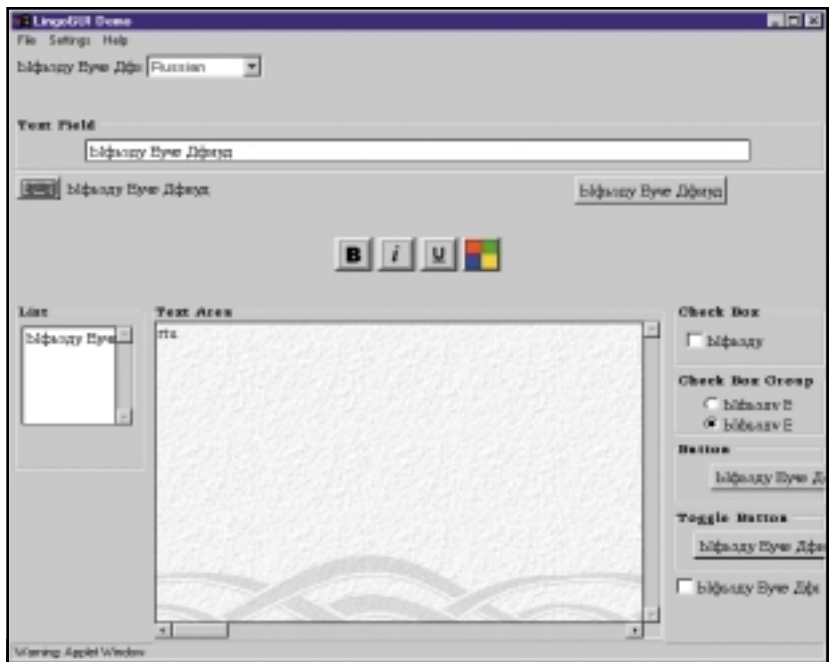


FIGURE 1 LingoGUI demo applet

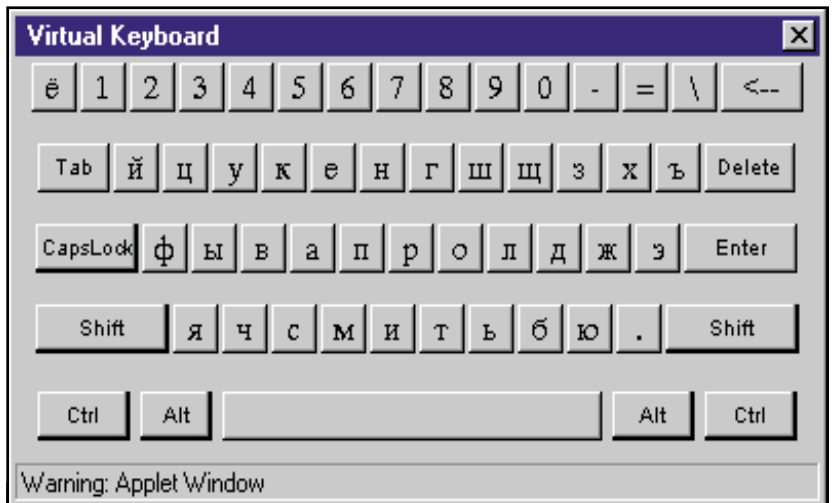


FIGURE 2 Virtual keyboard

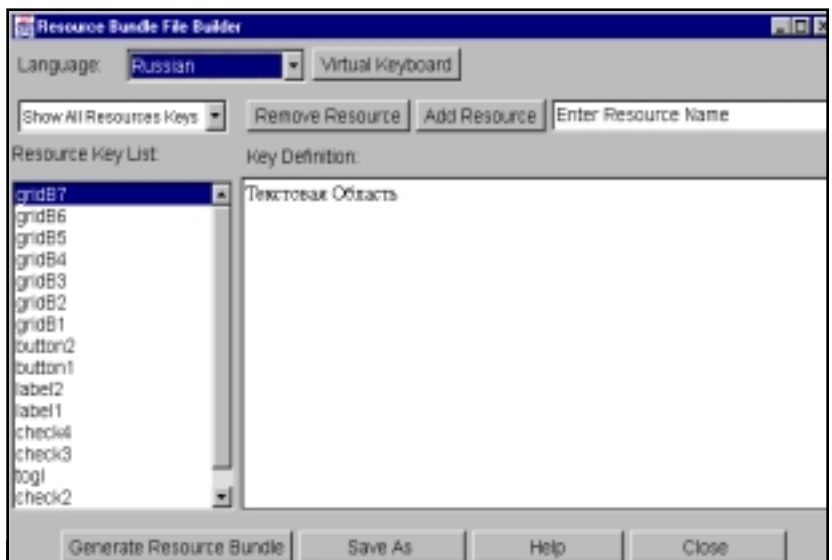


FIGURE 3 Resource Bundle File Builder

# Oops! 2000

[www.oops!2000.com](http://www.oops!2000.com)

# Teaching Basic Object-Oriented

1010011 01 0 1  
 01001 01 0 10  
 01001 01 0 1 0  
 101001 01 0 1 0  
 01001 01 0 1 0  
 101001 01 0 1 0  
 101001 01 0 1  
 01001 01 0 1 0

# Concepts Using HTML

HTMLStream  
 – a purpose-built  
 tool for teaching  
 Java and general  
 OO concepts  
 to computer  
 science newbies

*In this article we're going to describe a tool that we've created to help OO newcomers understand the class/instance relationship, inheritance between classes and linking between objects...by automatically converting an object graph into HTML. The tool we've created is based on the "circlegram" idea used by almost every object-oriented teacher during conventional "chalk and blackboard" lessons.*

*Our work at the computer science department of Milano-Bicocca University in Italy is a mix of software design, programming and teaching. We use Unified Modeling Language (UML) for design, Java for programming and blackboards for teaching. When we teach, we also sometimes use overheads/slides. "So what?" you may say, "every teacher in the world does the same..."*

WRITTEN BY DANIELA MICUCCI & ANDREA TRENTINI

# IAM

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

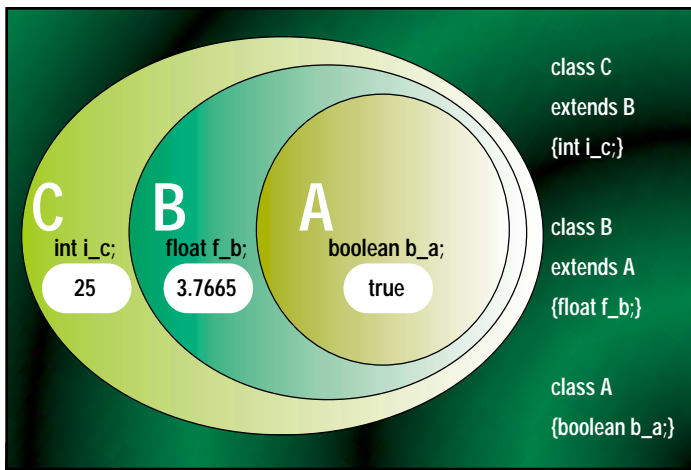


FIGURE 1 The circlegram concept

Right, but we have an added problem: we have to teach Java and general OO (object-oriented) concepts to students at their very first course at the very start of their computer science career. So these students are often a complete tabula rasa – they’ve had no prior programming experience whatsoever.

This happens because in Italy we don’t have screening or filtering at entrance: every high school student can enter almost any university, and major in whatever he or she likes. There are no formal restrictions or requirements, so a student from a high school where they specialize in Greek and Latin can enroll to study software engineering or physics. (Students can then abandon or change studies as and when they realize their mistake, along the lines of the DOR army mechanism – “Dismissed On Request”.)

Given this situation, you can just imagine how our Java programming course attracts an eclectic group of students: we have students with no prior experience in using a PC, let alone programming!

The business world isn’t that different. We dare say this because we also teach short Java courses for programmers from a wide variety of companies. They often have prior experience in languages such as COBOL, RPG, C, Visual Basic, and so on. Sometimes (very rarely) we also meet a C++ programmer. But prior knowledge isn’t always an advantage: it can be misleading when you try to map a procedural language into an object-oriented one.

In both cases (university and business), we face a major problem when teaching Java and OO concepts: we call it the “class/instance problem.” Newcomers and experienced procedural programmers alike find it difficult to fully grasp the meaning and the difference between a class and its instances (except of course in the case of a good C++ programmer).

Now of course we haven’t discovered anything that wasn’t already known; it’s a problem that’s been addressed before – see for example *Top-Down Teaching: Object-Oriented Programming in CSI* by R. Decker and S. Hirsfield (ACM SIGCSE 1993, pp. 270–73). Indeed it’s one of the reasons people debate whether or not choosing an OO language as the first language is the right thing to do.

Nonetheless, the class/instance problem remains the first and probably the biggest hurdle you face when teaching (and, seen from the student’s point of view, when learning) your first OO language. We’ve seen students struggle with “can’t make static reference to a nonstatic attribute” errors. We’ve seen Java programs with classes full of static methods and no instantiation at all. And we’ve seen many techniques described in the famous “How to write unmaintainable code” Web site (<http://mindprod.com/unmain.html>) actually being used!

Alas, the class/instance problem isn’t the only one. When you code in Java you’re forced to divide your product into many files. This policy is good for experienced programmers – they can organize code better, in separate files and directories. But while the experienced programmer

can keep track of the whole product, the Java newcomer isn’t always able to see a bunch of classes as a complete program. When novices are editing a piece of code they have to remember every relationship existing between the current Java file and every other one. They must learn (and, we might add, learn very fast) that the class they’re creating has attributes and methods not present in the current opened file. And we must help them as best we can.

To do so (and to help ourselves in teaching), we tried to address these problems by “porting” a teaching technique taken from traditional classroom lessons to a software tool.

## What We Needed, What We Produced

We started from something we knew very well: classic “chalk and blackboard” teaching. During traditional lessons, when we need to explain the class concept and the instantiation mechanism, we usually try to explain the coupling between class and instance by speaking and drawing on the blackboard. We use phrases like “any class you define is a template...,” “when you instantiate an object you’re in fact allocating memory...,” “attributes are allocated at their respective class level...,” and other more metaphorical ones that we won’t bother to report here. To represent a runtime situation, we usually draw “circlegrams” on the blackboard (see Figure 1).

The example shown in Figure 1 is a common form of circlegram: it represents an instance of a hypothetical class C with its attributes, both native and inherited, correctly placed (every attribute has an example value). To mimic the classroom situation we created a package, called *HTMLStream*, to generate an HTML representation of a circlegram. Remember that we want to map inheritance with a circlegram, but we also want to show the linking between objects (reference attributes).

In HTML there are hyperlinks – a perfect solution for mapping object links. So we had to create only the inheritance representation. We chose the HTML TABLE to represent an object instance, with a recursive table-in-table to show inherited attributes. The current format is shown in Table 1.

This representation should also be familiar to someone accustomed to UML class symbols, except that in our case inherited attributes are embedded inside the object instead of having an arrow pointing to the extended class. The table-in-table format converts an object like the one shown in Figure 2 into the HTML visualization shown in Figure 3.

When converting a complex object graph, our tool will generate an HTML page (it may be a long page) with links and internal anchors. An object graph (a bunch of interlinked instances in memory) is “linearized” in one single HTML page. Every reference between instances is mapped with an internal link to an anchor on the same page; in fact, every single object is converted into a piece of HTML text that contains links to other “objects,” and is also a target (HTML anchor) to be pointed to. Graphically, every object having reference attributes will look like the one shown in Figure 4.

At runtime, in memory, the same object will be allocated as a graph (see Figure 5).

Our generator will create an HTML page similar to the one rendered in Figure 6.

|  |  |
|--|--|
| <hidden NAME tag> <hashcode of object> <class>   |  |
| <name of field>  | <value of field>   |
| <name of field>  | <value of field>   |
| <name of field>  | <if field is pointer, then this is a link (HREF) to another object-anchor> |
| <if this objects extends another one, here you find a “table in table” with the same format> |  |

TABLE 1

# MetaMata

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

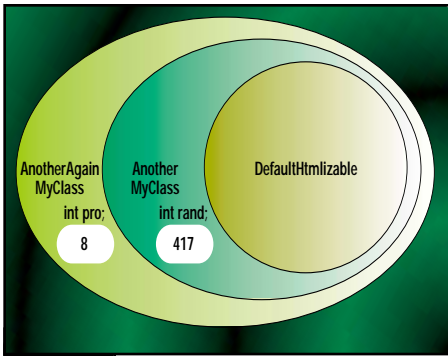


FIGURE 2 Specimen circlegram

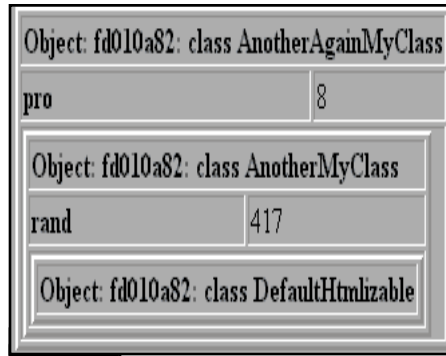


FIGURE 3 Single object "HTML-ized"

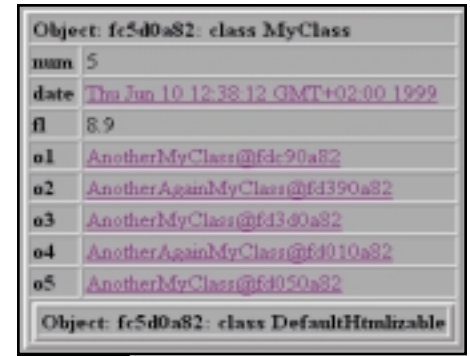


FIGURE 4 Object with reference attributes

The HTMLStream tool can be used during a lesson to explain inheritance and reference attributes, but of course it can also (and should also) be used by all students to experiment with their own source code. Students feel at home browsing an HTML page; they'll explore objects the same way they explore the Internet.

The whole process of generating an HTML page from instances in memory can be described with this checklist:

1. **Design time** Describe your classes using UML – or other formal design languages (see Figure 7).
2. **Coding time** Write your Java source code, adding special printing instruction (see "How to Use HTMLStream" section later in the article). Below is the example source for class MyClass (methods removed) that has been converted and shown in Figure 6:

```
public class MyClass
{
    int num;
    Date date;
    float fl;
    AnotherMyClass o1,o2,o3,o4,o5;
}
```

3. **Runtime** Run the code redirecting output to a file
4. **Browsing time** Examine the result – again see Figure 6.

## Other Products Already Available

Of course, before developing this product we searched the Internet to see if there were any similar tools already in existence. We found there are two main categories of software visualization: source visualization and runtime visualization.

*Source visualization* is used by developers to keep track of what they (or other developers) have done. With source visualizers, you can produce, for example, call graphs, statistical reports and tree representations. *Runtime visualization* is subcategorized into pre- and post-mortem visualization, "pre" meaning "running" and "post" meaning "after completion."

Information about software visualization is readily available starting at [www.cc.gatech.edu](http://www.cc.gatech.edu) (the research area of professor Stasko and colleagues). There's also a journal article to be aware of, "An Effective Web-based Software Visualization Learning Environment" by J. Domingue and P. Mulholland, in the *Journal of Visual Languages and Computing*, 9 (5), 1998, pp. 485–508. And there's a book chapter too, "Using Software to Teach Computer Programming: Past, Present and Future" by P. Mulholland and M. Eisenstadt, in *Software Visualization: Programming as a Multimedia Experience* (Cambridge, MA: MIT Press, 1998).

In its present version, HTMLStream is mainly a post-mortem visualizer, even though it produces output throughout the execution of the analyzed program. At the end of execution, the user can browse through the HTML page and view what's just happened.

After our resource search we decided not to use any of the preexisting tools for the following reasons:

- Most of the products already created are targeted at runtime visualization, that is, they generally don't save any state for later viewing (apart from "core dumps").
- Very few of them are developed for the Java language.
- Most of the ones already usable for Java weren't free.
- The most relevant candidate among those that were free, BlueJ (BlueJ – Teaching-Oriented Java IDE – see [www.sd.monash.edu.au/bluej](http://www.sd.monash.edu.au/bluej)) is an interesting product developed for teaching OO that has an object inspector but doesn't show attribute inheritance. (BlueJ is also an IDE, whereas for teaching purposes we needed nothing more than the standard JDK. We didn't want to confuse students with a GUI product; we believe they must learn the language by writing code character by character.)

## How to Use HTMLStream

First, here's a four-step checklist for the impatient:

1. Download the package available from the HTMLStream home page ([www.sal.disco.unimib.it/~atrent/htmlstream.htm](http://www.sal.disco.unimib.it/~atrent/htmlstream.htm)).
2. Compile everything ("javac \*.java").
3. Run MyClass ("java MyClass" or "java MyClass > file.html" if you want to capture output in a file for later viewing).
4. Watch the screen, and you will be shown a JFrame containing an example HTML-ization (see Figure 8).

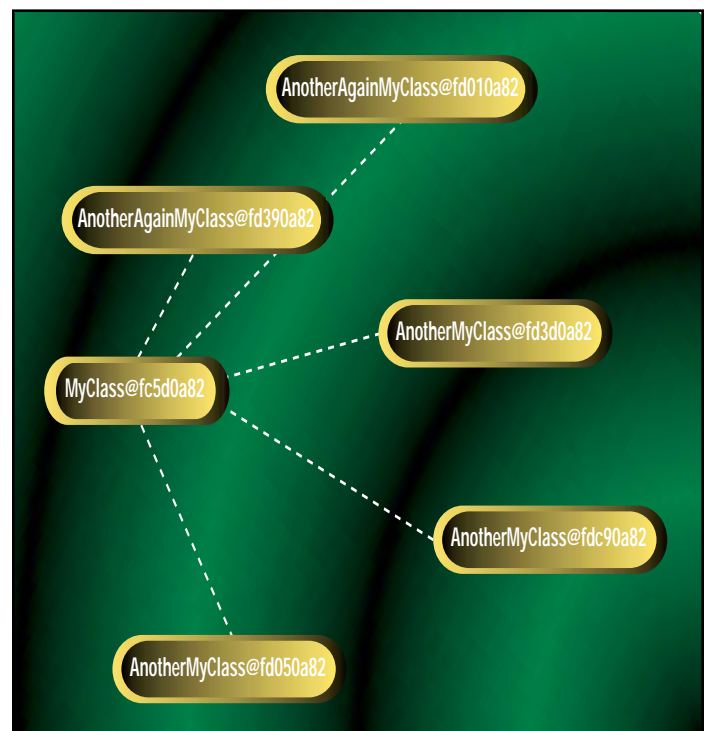


FIGURE 5 Object graph in memory



# VisiComp

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

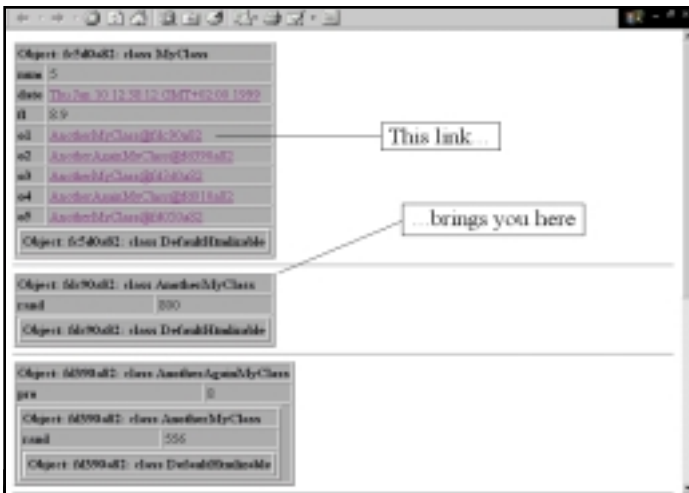


FIGURE 6 An object graph converted to html

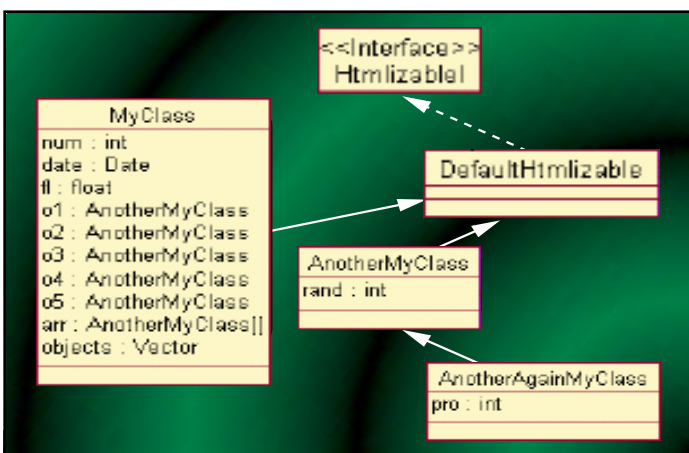


FIGURE 7 UML fragment

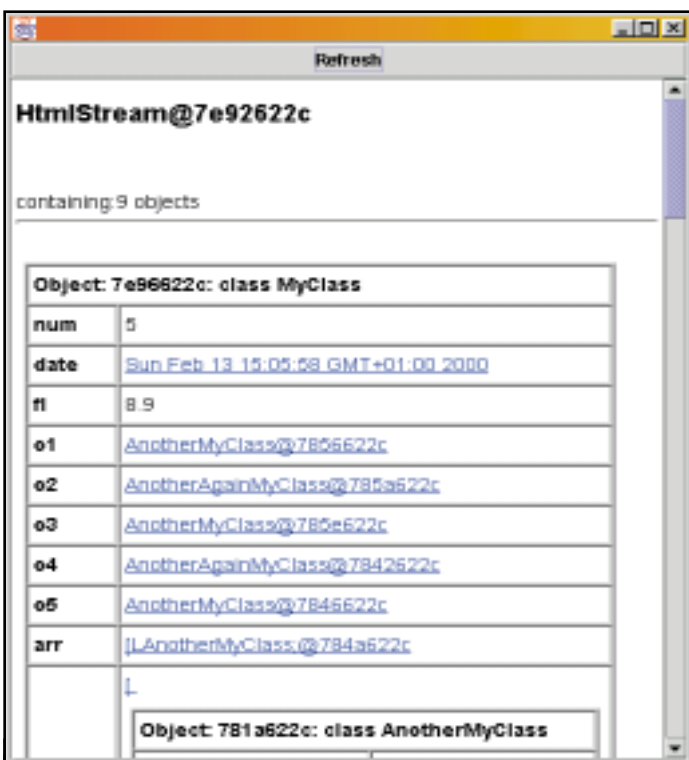


FIGURE 8 JFrame containing htmlization

Next, instructions for everyone else. When you need to “print” (convert to HTML) an object you must:

1. Create an instance from the HtmlStream class.
2. Pass the reference of the printable object to the HTMLStream.
3. “Print” the HTMLStream object, in the usual Java way (System.out.println).

This means that you can use HTMLStream only by inserting special printing statements in your code. Another thing to bear in mind is that not every object is “printable”: you can print only certain types of objects, namely the ones that implement the Htmlizable interface. Let’s see our first example:

```
// MUST be HtmlizableI
MyClass toBePrinted = new MyClass();

// "the printer"
HtmlStream hStream = new HtmlStream(toBePrinted);

// print the HTML representation
System.out.println(hStream);
```

The HTMLStream is also useful if you want to add (or “queue”) other objects at a later time – assuming that we still have hStream in scope:

```
// adding new objects
hStream.grow(new AnotherHtmlizableI());
hStream.grow(new AgainAnotherHtmlizableI());

/* print the HTML representation,
   this time includes the new objects */
System.out.println(hStream);
```

This last example is one reason for the existence of the HTMLStream class: it’s useful when you need to queue more than one reference for printing. It’s very similar in usage to the ObjectOutputStream; in fact, the HTMLStream works more or less the same way (i.e., conceptually). A mental aid when using HTMLStream is the “printing queue” metaphor: you submit objects to the stream when you’re satisfied you can start the actual printing of the stream itself.

## How Can I Make My Objects Printable?

As we mentioned before, an object must be HTML-printable to be passed to the HTMLStream. To make an object fulfill this criterion you have two basic choices:

- Your object implements the Htmlizable interface. This way (and it’s the hard one), you must implement every method in that interface, but at least you spare the Java single-inheritance for something else.
- Your object inherits from DefaultHtmlizable class (provided in the package). This way, you don’t need to implement anything; you just extend a particular class and that’s it: your object inherits HTML printability. By contrast, you can’t inherit anything else (at least in Java).

A problem that affects this implementation (and will probably be circumvented in future releases) is that standard JDK objects are not already HTML-printable. To print a standard JDK object you either have to build a wrapper class that translates it into an Htmlizable object, or you must “explode” it if it’s some kind of container. An example of the latter case can be found inside the DefaultHtmlizable class source: we explode a Vector or a Hashtable; look at the method isComplexUsable(Object o) in the file DefaultHtmlizable.java.

# Software AG

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

## How We Did It — Implementation Notes

(*Authors' Disclaimer:* Please note that we designed this tool for teaching purposes only, so our implementation is somewhat “quick and dirty” and can certainly be improved upon.)

To implement this htmlization mechanism, we took inspiration from the serialization feature already present in Java. This “pattern” is very similar to the Externalizable mechanism: the serializable object must “do something” to be serialized; it's not completely passive.

Remember that we had to render the three aspects mentioned above: inheritance, attributes and links (references to other objects). Inheritance is represented by the recursive table-in-table (every attribute is printed coupled with its value). Any link (attribute that is a reference) will be a hyperlink to an anchor somewhere on the same page.

The main problem in performing this conversion is, for example, when an object points to another object, which in turn points to another one, which in turn points to the first one – that is to say, a circular list. This is a common situation in a running program. In fact, we call them object “graphs,” not trees. If we want to convert everything automatically, meaning that every reference inside the root object (initial object) is automatically followed, we need a way to avoid infinite loops when converting circular graphs or even the simple multiple referentiation.

Loop avoidance is the primary rationale for the existence of the HTMLStream class. An HTMLStream is a “container” that can be “grown” by adding HtmlizableI objects. An “add” operation won't succeed if an object is already present in the stream.

We must solve another problem: automatic reference following. If we want to “print” an object through this stream, we want to avoid passing every single reference to be printed; we just want to pass the graph “root” (quoted since it is a graph and there is no formal root) and let the stream do the rest. This is the role of any HtmlizableI object (implemented in the DefaultHtmlizable class): instances from this class can scan their attributes and treat them “correctly,” automatically adding to the stream every referenced object (have a look at the method grow() in DefaultHtmlizable.java). The pseudocode for the scan&grow algorithm is:

```
for every attribute of this instance
  if attribute is NOT primitive
    if attribute is HtmlizableI
      stream.grow(attribute)
  else
    wrap it (if possible) and grow stream
```

Actually, this procedure is executed at each class “level.” We use reflection to extract attribute details, and we do this for each class level up to but excluding Object, the top class in Java. For every class level scanned we open a new TABLE tag. At the end we close everything, thus drawing our desired “html circlegram.”

## Ideas for Past, Present and Future Development

Last year, at the beginning of our work, we thought about using XML for conversion and viewing (see [www.xml.org](http://www.xml.org), [www.xml.com](http://www.xml.com) and [www.w3.org](http://www.w3.org)). Back then, we abandoned the idea because of the prevailing lack of XML tools, but the situation is evolving rapidly. Since it's now gaining respect, even in the Java world, XML could be a viable option now.

There are two categories of ideas for development: minor (e.g., aesthetic or “easy” to implement) and major (e.g., semantic) improvements.

Minor ones could be:

- **Adding access indicators:** Add public, private and protected access indicators to the attributes printed.
- **Resetting the stream:** If you need to add a previously added object to the stream you can't; at present you must instantiate the HtmlStream. We need a method similar to the ObjectOutputStream.reset().
- **Adding a “link count” feature to the HtmlStream:** Incrementing a counter instead of simply discarding an object already present in the

stream). This feature could roughly evaluate the “spaghettization” level of a student's code.

- **Generating a Web site:** Instead of generating a single text stream, it would be useful to generate a Web site: every instance could be represented as a single page with links to other pages instead of internal links (anchors). This way you could graph your site with existing tools (FrontPage, for example, generates a graphical map of a site). The graph would represent a visual glimpse of your runtime situation and you could also navigate every printed instance with the omnipresent Web browser! To implement this new format we need to rearrange the classes a little bit (e.g., an object must present itself in at least two HTML forms: the single-page mode and the sitemode). In sitemode every page should be named after the object itself; in Java this could be based on the toString() form of an object – something like MyClass@fdc90a82.html (<classname>@<hashcode>.html).
- **Adding a historicization mechanism:** The object htmlization as seen here is a somewhat static mechanism: at some point in code you print an object, overwriting the previous state. A historicization mechanism would be very useful, but it would have to be carefully planned. To be honest, it's probably a major change in source code. Also, great care must be taken in designing the htmlization: how an object state change could be rendered in HTML. We need an effective way of simultaneously representing objects, links and their modification in time.

As for major improvements, there are only two:

- The current mechanism (extending the DefaultHtmlizable class or implementing “ex novo” a HtmlizableI object) is good enough for teaching purposes. But in real-world applications it would be better if an object could be printed anyway simply by encapsulating it into another object. In Java you can do this by using the serialization protocol and converting a serialized stream in an HTMLStream. This way, the only constraint (at least in Java) is that your printable object must also be declared java.io.Serializable (an interface – this way you don't miss inheritance with other classes). Implementing a serialization wrapper needs more work than for the simple “extends DefaultHtmlizable” solution.
- Using VRML as output format would also be very interesting, but we don't have the resources at the moment to experiment in the VRML field.

## Conclusion

Is HTMLStream good for teaching? Is HTMLStream good for first-year students without programming experience? We believe that, yes, this tool *is* good for teaching: we've used it in some short courses (on students with some prior programming experience) and it greatly helped us with “bootstrapping” people. But we still have to test it on students as their first-ever experience.

A tool like HTMLStream – and not only HTMLStream, but *any* tool – must be introduced with care. Novice users need to be introduced first to general programming principles, then to the joys of editing – we had students editing source files who didn't know where they'd saved them! – and to the problems associated with using a PC (for example, if they're very young they often ignore basic concepts such as command prompt and redirection). We'd be very happy to receive some feedback from **JDJ** readers. The complete source code for this article can be downloaded from the **JDJ** Web site, [JavaDevelopersJournal.com](http://JavaDevelopersJournal.com). 🍌

### AUTHOR BIOS

Both Daniela Micucci and Andrea Trentini have worked since 1997 as researchers at Dipartimento di Informatica, Sistemistica e Comunicazione (DISCO), the Computer Science Department of Milano-Bicocca University in Italy. Since receiving their computer science degrees, they've both been teaching Java and OO design to companies, in schools and at the university level. Andrea's research field involves studying and designing software architectures for educational management; Daniela's involves designing traffic control and monitoring software.

[micucci@disco.unimib.it](mailto:micucci@disco.unimib.it) / [trentini@disco.unimib.it](mailto:trentini@disco.unimib.it)

# SIC Corp

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

# Anatomy of a Java Application Server

A back-to-basics attempt to define the relationship between Web application servers from application servers

WRITTEN BY  
AJIT SAGAR



For the past few months, I've been focusing the discussions in this column specifically on Web and Java application servers and on application servers that integrate with Java technologies. Recently several readers e-mailed me asking about different aspects of application servers in general, so this month I thought I'd go back to the basics and talk in more general terms about Web servers and application servers.

Several folks in the computing industry think of 1999 as having been the "Year of the Application Server." But while the term *application server* itself may be a fairly recent addition to the software computing vocabulary, the application server market has already become one of the fastest-growing markets in *n*-tier computing. Business analysts estimate its value as being likely to reach the multibillion-dollar level in 2001.

In today's distributed computing environment, the term application server is associated with state-of-the-art technology. Perhaps that's why several vendors sell their products under the category of application servers, regardless of whether the product actually offers the features of a basic application server or not. For example, there are site builders, Web page designers, integrated development environments, Web development tools and enterprise-level development environments. The definition of what comprises an application server is more often than not open for interpretation.

## What's an Application Server?

An application server is, by definition, "a computer server that serves applications." More precisely, an application server serves up application services. The main purpose of an application server is to reduce the workload of applications by taking over the responsibility of mundane activities involved in executing the application and making the application's services available to external modules in a reliable manner.

I'd like to take a stab at defining an application server as follows: an application server is a computer program that resides on a server in a distributed

network and whose main function is to provide the business logic for an application program; an application server provides a customizable and flexible execution environment for hosting business logic components, thus providing distributed services and integrity for application execution.

Traditionally, the application server has been associated with three-tier applications. To recap, the components of a three-tier application are:

1. **Front-end client:** Typically a graphical user interface on a personal computer, laptop or a workstation
2. **Middle-tier application:** Typically a business logic application on a LAN or the intranet server
3. **Back-end application:** Typically a database/transaction server that provides access to legacy or back-office data

An application server provides an execution environment that decouples front-end clients from back-end data access. The execution environment is supported by an infrastructure that enables integration among different applications. Application servers enable this integration by offering software components that can be used to create business logic for an enterprise application. The supporting infrastructure may include architectural frameworks such as messaging systems, transactional managers and database accessors.

## Benefits of Application Servers

The Internet being the most powerful phenomenon driving application development and deployment today, application servers came into existence because of a need in the market to offer flexible, robust, extensible and stan-

dards-based enterprise applications that are developed at Internet speeds.

Internet applications are typically shared between multitudes of parties participating in e-business transactions. This necessitates standard architectures and frameworks that allow application hosting. Application servers provide the execution environment for Internet applications. Vendors in the application server market add value to the equation by taking over the burden of application hosting and offering commoditized products that enable organizations to concentrate their resources on building the applications themselves.

In some ways, the application server market is moving toward the space currently occupied by operating systems. Operating systems are developed and maintained by third-party vendors. Companies use operating system services to develop applications in their business niche. In the same way, application server vendors can provide third-party services for use by distributed application developers.

The benefits to an application development vendor of using a third-party application server are:

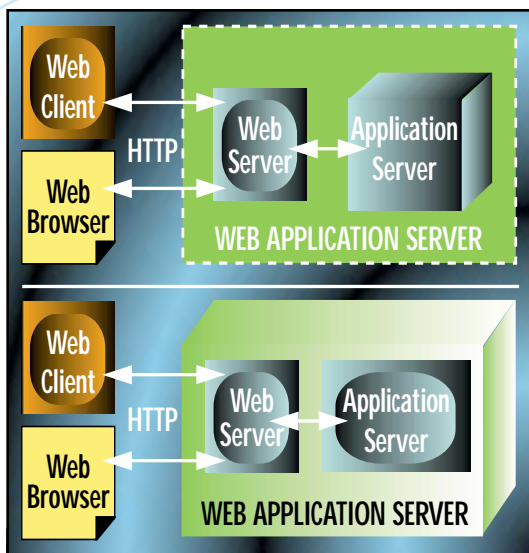
- *Better product focus*
- *Abstraction*
- *Indirection*
- *Application interoperability*
- *Better resource utilization*
- *External support and maintenance*

## "Pre-Web" Application Servers

The first generation of application servers came into existence before the Web became popular. These "pre-Web" application servers may be categorized based on the types of services they offer. Each category of application server decouples the client from the actual

# VSI

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



**FIGURE 1** A Web application server is a Web server with an application server.

source of the service. In some respects, the application server acts as a broker: the client requests a service; the application server then makes a request to the appropriate service provider and passes the results of the request back to the client.

We could categorize application servers based on the services they offer. Some pre-Web application servers are:

- *TP monitors*
- *Data servers*
- *Document servers*
- *Distributed object servers*

## Back to Web Application Servers

Web application servers have emerged as a result of developments in the computing industry. Principal among these are:

1. The acceptance of the Internet as the ubiquitous medium for data interchange
2. The emergence of HTML as the ubiquitous format for data presentation
3. The evolution of standard distributed component models
4. The evolution of standard protocols for data transport

### AUTHOR BIO

Ajit Sagar is a senior solutions architect in a firm specializing in B2B market places. A Sun-certified Java programmer with nine years of programming experience, including three in Java, Ajit holds an MS in computer science and a BS in electrical engineering.

The Web is another environment that supports *n*-tier applications. Consequently, the Web application server is a natural by-product of the evolution of *n*-tier distributed application development.

Web application servers are a new type of Internet software. They're a result of combining HTTP servers with distributed component frameworks (which are also the basis for the distributed object servers mentioned above). Currently, most application server ven-

dors offer a combination of object application servers and Web application servers. Each of the other categories of application servers mentioned above has also evolved into specific categories of Web application servers.

When you add the Web to an application server, you have, literally, a Web application server. But what does this actually mean? Let's go back to the basics. The Web, in its simplest definition, is a group of computers that communicate via the HTTP protocol – the communication medium being the Internet. Hence, a Web server is basically an HTTP server. Consequently, a Web application server is a server that makes an application's service available to the client over HTTP.

The relationship between Web application servers and Web servers is illustrated in Figure 1. The top half of the figure shows an application server with an external Web server. The bottom half of the figure shows an application server with the Web server bundled in. Note that the only difference is in the packaging. In the latter case, the Web application server vendor bundles a Web server with his or her product. Such products should still be able to substitute their proprietary Web server with a third-party Web server.

## Recipe for a Web Application Server

Let's take a look at what it takes to cook up a typical Web application server. As we've seen earlier in this article, a Web application server needs a Web server and an application execution environment. The application execution environment hosts business objects. The functionality of these business objects is exposed outside the application server in the form of programming interfaces. The execution environment itself comprises the following frameworks and services:

- *Object containers*
- *Naming/directory services*
- *Messaging/event framework*
- *Communications framework*
- *Security framework*
- *Transaction framework*
- *Data access framework*

Figure 2 illustrates how these frameworks fit in an application server. The Web client accesses the business objects of the Web application server via a Web browser. The request is made to the Web server, which forwards the request to the object container using the Naming Framework to find the business object. The Internet

client uses a distributed communications protocol such as RMI, IIOP or DCOM (depending on the computing environment) to get to the application server. It also finds the desired object via the Naming Framework. The business objects exist in the context of an object container. Object containers expose the interfaces of the objects to the external world. The Communications Framework is used by the object containers for interobject communications. The application server interacts with other remote systems via the messaging, transaction and security frameworks. Data from the back-end tier is accessed via a Data Access Framework.

## Application Server Computing Platforms

In some ways, operating systems can be considered as the "0th" generation of application servers. When life was simpler, the operating system hosted the application and handed computing resources to applications when required. The application developer had a certain degree of control over the execution environment. However, application development required knowledge of system-level programming. This knowledge was not portable – that is to say, application developers had to learn about completely new systems if they changed the computing platform. Application servers abstract application developers from the gory details of the operating system's execution environment. They do this by managing the interaction with the operating system for using system resources and offering these resources as application server services. Desired features of the runtime environment such as load balancing, fault tolerance, persistence and so on are offered by application servers for optimally running the application. Since application servers abstract the platform or operating system, one logical way to classify them is on the basis of the computing platform in which they execute.

Currently, there are two camps in the application server industry – the Windows (Microsoft) camp and the Java camp. Note that here we refer to the Java software platform, not Java the programming language, and so the comparison is between software computing platforms. The Windows platform enforces the operating system and corresponding hardware. On the other hand, the Java platform is a virtual platform that enforces the programming language but can run on a variety of operating systems and hardware platforms. In



# YouCentric

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

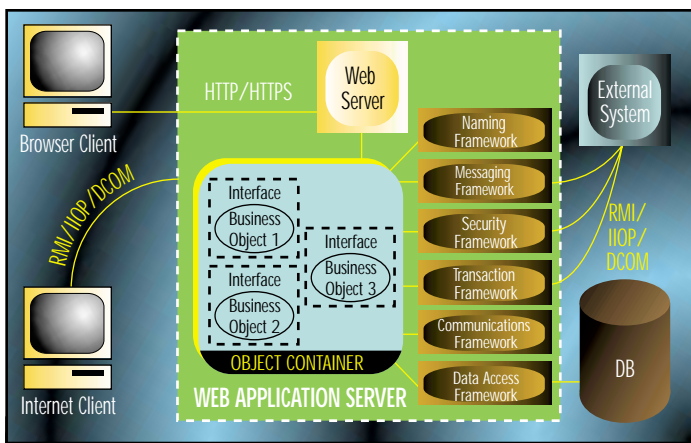


FIGURE 2 Components of a basic Web application server

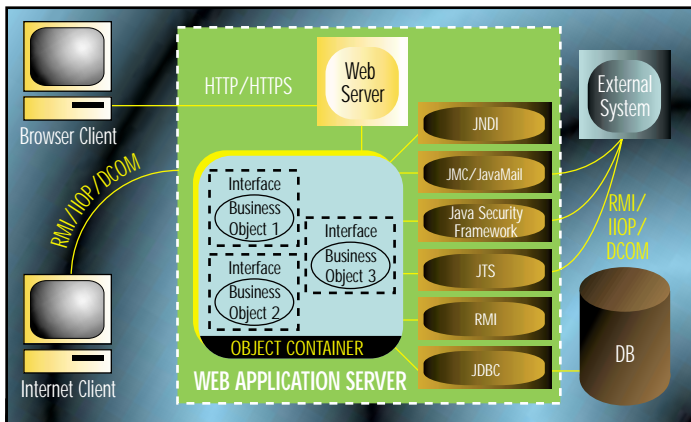


FIGURE 3 Components of a basic Java application server

# Cimmetry Systems, Inc.

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

both cases, the application servers offered by these platforms meet the criteria of abstracting the application hosting details from the application providers.

## Web and Java Application Servers

As mentioned earlier in this article, the class of application servers that comprises a component of Web architectures comes under the umbrella of Web application servers. Simply put, Web application servers serve application services over HTTP (Hypertext Transfer Protocol). A Web application server is always associated with a corresponding Web server. Today's distributed applications that leverage the features of the Sun Microsystems Java platform often base their architecture on one or more Web application servers. Similarly, when someone mentions application servers today, it's more than likely that they're talking about application servers that support the Java platform. Such application servers are often called "Java Application Servers." The application servers discussed in previous articles are all either based on the Java platform or integrate with the Java platform.

## Java Platform Application Servers

Java application servers are a by-product of Java's increasing presence in server-side middleware and the definition of Java Enterprise APIs by Sun Microsystems in collaboration with its industry partners. Java Enterprise APIs define enterprise-level services for server-side deployments. As described earlier in this article, the appearance of application servers in the market dates back to when the concept of multitiered computing became popular. These application servers provided a hosting environment for middleware components. Before the stabilization of the Java Enterprise APIs, however, the definition of middleware components for a ubiquitous software platform was not uniform across operating systems. So the application servers were operation-system-specific.

Consequently, each of these application servers provided middleware services in a proprietary way, making portability and reuse of the components difficult. The emergence of the Java Enterprise APIs has enabled the definition of a standard architecture for middleware components composed of business objects. This architecture clearly defines well-formed interfaces between the application server's object containers and the objects or components themselves. In Java this is made possible by:

- A standards object model (EJB) for designing business objects
- Uniform APIs for accessing the business objects (remote interfaces)
- Container APIs for interacting with the vendor's mechanisms for accessing system resources (home interfaces)
- APIs for finding the business objects (JNDI)
- Standard means of accessing these components through a distributed protocol (Java servlets, RMI)
- Standard APIs for connecting to back-office data sources

The above elements form the components for a basic Java application server. This is illustrated in Figure 3. Note that this is a version of Figure 2 with Java-specific technologies. (Not all of the technologies that play a part in Java application servers are shown in the figure.)

## Trading Places

The application server market is evolving rapidly. In a couple of years, the survivors of the current "app server wars" will emerge. Java has been instrumental in getting application servers the attention that they're currently receiving – because Java adds hardware platform independence for the implementation and use of application servers, vastly enhancing their scope. As J2EE and the EJB model matures and industrial-strength applications are developed using this model, Java application servers will continue to play a crucial role in the development of enterprise-level distributed applications. ☛

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

# PointBase

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

# JavaCo

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

n 2000

n2000.com

# Distributed Tasking in Java

Increase overall performance by executing database queries in parallel



WRITTEN BY  
SAM MCKENNA

We've all heard about the great benefits of distributed computing, especially in the areas of scalability and performance. With Java, implementing a distributed solution has never been easier or more practical. We're given three distributed object options that work quite naturally with Java: namely, Java RMI, CORBA and EJB. The issue that many Java developers face when pondering distributed archi-

tures is whether or not a distributed solution is appropriate for the problem at hand. Often the problem being attacked is one that has been traditionally solved in a nondistributed fashion. One approach for finding distributed solution potential is to look for basic tasks in your software that can be broken out. Once you identify these tasks, all you need is a framework for distributing execution.

This article offers an approach for building distributed solutions. The fundamental idea is to separate the problem into two pieces, the critical task and the non-critical tasks. The critical task makes decisions about work to be done. In turn, this work is done by noncritical tasks. It's this separation of responsibility that creates increased scalability and performance.

## Distributed Architecture

A distributed solution usually includes both a distributed software design and a distributed network design. An ideal software solution should be, in part, transparently scalable with the addition of new processors to the network. A three-tier approach can facilitate this scalability with the critical task on one tier and the non-critical task "handlers" on another (see Figure 1). The noncritical tasks are simple and atomic. The handlers simply take a non-critical task and execute it. On the middle tier lives a centralized task scheduler that simply accepts new tasks and makes them available to the handlers. Adjusting the number of task handlers independently with regard to the other tiers can increase performance and scalability.

## Task Framework

The defining relationship that constitutes the Task Framework is the division of responsibility between the Task class and the Handler class. The Task class is an abstract class whose subclasses contain the information necessary for remote execution. The Handler class is an abstract class whose subclasses are customized to handle different types of Task objects. It's the Task object that's accessed or transported across the network, depending on which distributed object mechanism you use. In order to keep network traffic at a minimum, Task objects must be as lightweight as possible. They should contain only data attributes and accessor methods. The Handler class is

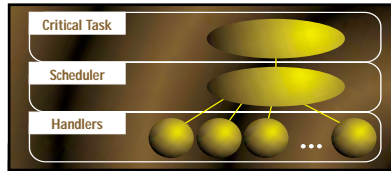


FIGURE 1 Tasking tiers

responsible for accessing the Task objects and performing the appropriate execution. That means that it's the Handler class that maintains connectivity to external resources such as databases, mail servers and network printers. In other words, the Task class is responsible for the information and the Handler class is responsible for the action. Because the Task objects are atomic and simple, the Handler objects are equally basic. This means that you can have any number of tasks and handlers instantiated. Only resources and practicality limit you.

Java RMI, CORBA and EJB each have a lightweight mechanism that you can take advantage of when passing around objects or their interfaces. If you're like me, this makes a lot more sense when you see some code. For Java RMI, the Task class simply needs to implement the Serializable interface from the java.io packages, as shown in Listing 1. Listing 2 shows the CORBA IDL for implementing the Task information as a structure. Using EJB, the Task class would be an entity bean as shown in Listing 3.

The remaining examples will use Java RMI since it's the simplest way to illustrate the approach. Please note that CORBA tends to be a bit faster than Java RMI and an EJB solution may be less resource-intensive by taking advantage of instance pooling. Of course, the underlying remote method invocation protocol that your EJB server uses will affect performance as well.

The Handler extends the Thread class and runs in its own process space (see Listing 4). It periodically polls for tasks that are waiting to be executed. In order to increase performance and reduce network traffic, pending tasks are transported in chunks. A chunk is simply an unordered

group of tasks. Each Task object is passed to the handle() method for execution.

## Task Flavors

While it's important that a task is atomic for performance and simplicity, its overall function need not be atomic. Allowing for different types of task behaviors will give you greater flexibility by allowing tasks to function collectively. Three useful strategies that come to mind are execution scheduling, task dependency and conditional task creation.

In many cases, you may want to assign your task a future execution time. Obvious examples include backing up information, clearing out log files and even rebooting a server. There are two categories of scheduled tasks: tasks that execute once at a specific time and ones that execute more than once on a regular interval. You may be thinking, "Why not use cron?" The answer is that you could, but you'd have to create separate scripts for cron to execute. Additionally, you have the flexibility of allowing your software to adjust the scheduled tasks based on internal state.

Often the appropriate time for execution is dependent on the completion of another task. Suppose you have an e-commerce Web site. You may have a task responsible for charging a credit card and another that sends order confirmation e-mail to the customer. The e-mail task can simply wait for the credit card task to complete before it becomes available for execution.

A task handler may contain conditional logic that triggers the creation of new tasks. Examples of this tend to be more esoteric and often involve workflow issues. Perhaps you're updating a customer database where, if the number of customers reaches a certain threshold, a new set of tasks responsible for updating statistical data elsewhere in the database is triggered.

## Task Scheduling

The Scheduler facilitates communica-

### AUTHOR BIO

Sam McKenna is a software developer and consultant based in Denver, Colorado. He has over 10 years' programming experience using C++, Forté and Java.

tion between the critical task and the task handlers. Think of the Scheduler as a synchronized queue. Any remote process can place a task object in the queue. Execution handlers in turn pick up task objects, perform the appropriate execution and report the result of the task execution. In some cases, the task handler may create new tasks and execute them immediately or place them in the Scheduler. The Scheduler class is responsible for tracking the various stages of execution for all tasks as well as controlling the scheduling needs required by the various task flavors. Tasks exist within the Scheduler in three states. The pending state includes tasks that are waiting to be picked up by a handler. Once a handler has grabbed a task, it moves into the processing state. After that, the tasks move into the completed state. In other words, the Scheduler is responsible for making tasks available at the appropriate time, whether it's the scheduled time or, in the case of a dependent task, when another task has been completed. Listing 5 shows the interface for the Scheduler. Since multiple handlers may be requesting pending tasks at any given time, care must be taken to synchronize access to the tasks.

## Example Application

Maintaining a relational database warehouse is a good application of this approach. Let's say you're designing a process that continually scans a database for current information, crunches some numbers and then updates the database with the results. In this situation, high performance equates to a more accurate and up-to-date data warehouse. I have found that in most cases scanning the database is cheap and updating the database is expensive. Delegating the responsibility of performing the updates will yield a significant increase in overall performance. Relational databases are designed to handle multiple concurrent users. You can take advantage of this database feature by distributing the tasks in order to increase performance. Think of the critical task and the task handlers as database users. The critical task spends its time scanning the database for necessary changes. It creates Task objects that represent database updates and schedules them for immediate execution. The task handlers pick up the scheduled tasks, generate the required SQL and execute the SQL to update the database. At first it may seem counterintuitive to add more users to a database, but you'll find that this design actually increases the overall performance. This solution scales easily by adding more handlers on more

processors. When you deploy this solution, you'll need to do a little experimentation to find out what the best layout is for your particular network. Listing 6 illustrates a Handler class designed to update the warehouse.

Extending this solution beyond updating the database is relatively easy. Simply introduce new Task subclasses and new Handler subclasses into the mix and make the new Task objects available to the Scheduler.

## Conclusion

Distributed tasking can be applied to solve a wide variety of problems. As in the database-warehousing example, using this technique increases overall performance by executing database queries in parallel while freeing up the critical task for faster processing. You can also take advantage of the various task flavors to utilize distributed tasking to control workflow management by having tasks trigger other tasks. Nearly any noncritical task can be delegated to a remote handler. Having a distributed framework in place as a part of your design will encourage future software enhancements to follow a distributed model.

There are a few limitations to discuss. Throughout this article, the Task instances are stored in memory. This design is fine for small-scale systems, but it can prove limiting for large-scale systems where tasks may be queued rapidly and in large quantities. The best solution in this case is to have the Scheduler physically store the Task objects until needed. If you decide to go the EJB route, your EJB server will do this automatically for Entity beans either through serialization or through a relational database mapping. Another area that needs to be addressed is exception handling. Exceptions are likely to occur on the handler tier. Depending on the type of exception, you may want to have your handler retry the execution or pass the exception back to the Scheduler and ultimately the critical task.

It's never been easier to design and implement distributed solutions. Not only are distributed architectures interesting and fun to implement, they're now also accessible to practically every Java developer. Java RMI is free and EJB servers threaten to be omnipresent in the near future. The challenge facing the Java developer is finding new approaches that exploit the features of distributed computing. ☪

[sam@kenatek.com](mailto:sam@kenatek.com)

### Listing 1

```
public abstract class Task implements
java.io.Serializable
{
...
}
```

### Listing 2

```
struct Task
{
// Add information specific to the task
};
```

### Listing 3

```
public abstract class Task implements
javax.ejb.EntityBean
{
...
}
```

### Listing 4

```
import java.rmi.*;
import java.rmi.server.*;
public abstract class Handler extends
Thread
{
```

## Meet JDJ

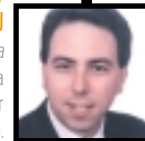
### EDITORS AND COLUMNISTS

Attend the biggest Java developer event of the year and also get a chance to meet JDJ's editors and columnists



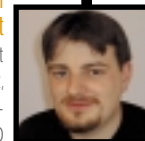
#### Sean Rhody Editor-in-Chief, JDJ

Sean is the editor-in-chief of *Java Developer's Journal*. He is also a principal consultant with Computer Sciences Corporation.



#### Alan Williamson JDJ Straight Talking Columnist

Alan is the "Straight Talking" columnist of *JDJ*, a well-known Java expert, author of two Java books and contributor to the servlet API. Alan is the CEO of n-ary Consulting Ltd., with offices in Scotland, England and Australia.



#### Ajit Sagar Editor-in-Chief, XML-Journal

Ajit is the founding editor of *XML-Journal* and a well-respected expert in Internet technologies. He is a member of the technical staff at i2Technologies in Dallas, Texas, where he focuses on Web-based e-commerce.



#### Jason Wesra EJB Home Columnist

Jason is the "Enterprise JavaBeans" columnist of *JDJ* and a managing partner with Verge Technologies Group, Inc., a Java consulting firm specializing in Enterprise JavaBeans solutions.



MEETING  
September 24-27, 2000  
Santa Clara Convention Center  
Santa Clara, CA



# Wintertree Software

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

# North- woods Software

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

```
protected abstract void handle(Task task) throws Exception;
public void run()
{
    try
    {
        Scheduler scheduler = (Scheduler)
Naming.lookup("rmi://localhost/scheduler");
        while (true)
        {
            Task[] tasks = scheduler.getTasks(10);

            for (int i=0 ; i<tasks.length ; i++)
            {
                try
                {
                    handle(tasks[i]);
                }
                catch (Exception handleException)
                {
                    handleException.printStackTrace();
                }
            }
            // Sleep for five seconds
            try { sleep(5000); } catch (Exception sleepException) { }
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

## Listing 5

```
import java.rmi.*;
public interface Scheduler extends Remote
{
    // Add a Task to the queue
    public void addTasks(Task[] task) throws RemoteException;

    // Get tasks
    public Task[] getTasks(int max) throws RemoteException;

    // Indicate completion
    public void complete(Task[] task) throws RemoteException;
}
```

## Listing 6

```
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;
public class UpdateHandler extends Handler
{
    private Connection connection_ = null;
    public UpdateHandler() throws Exception
    {
        Class.forName(driver);
        connection_ = DriverManager.getConnection(url, user, password);
    }
    protected void handle(Task task) throws Exception
    {
        if (task instanceof UpdateTask)
        {
            Exception exception = null;
            UpdateTask updateTask = (UpdateTask) task;
            Statement stmt = -znull;
            try
            {
                stmt = connection_.createStatement();

                String sql = updateTask.getSQL();

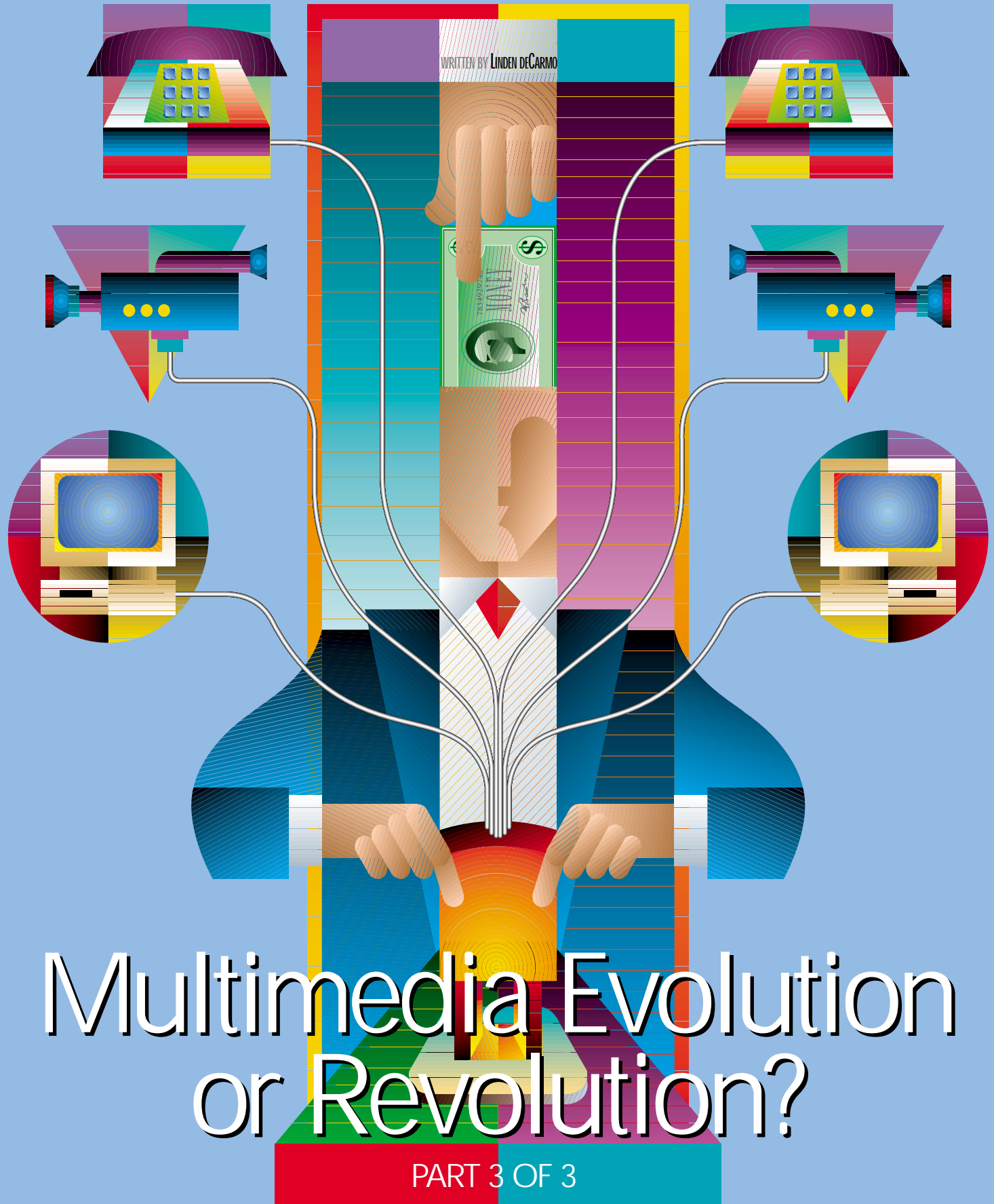
                stmt.executeUpdate(sql);
            }
            catch (Exception e)
            {
                exception = e;
            }
            finally
            {
                try
                {
                    if (stmt != null)
                        stmt.close();
                }
                catch (Exception fe)
                {}
            }
        }
    }
}
```





# IBM

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



WRITTEN BY LINDEN DECARMO

# Multimedia Evolution or Revolution?

PART 3 OF 3

# A revealing look at the Java Media Framework 2.0

*Java programmers have been anxiously awaiting the release of the Java Media Framework 2.0 for more than a year. Not only does JMF 2.0 finally let you capture audio and video content, but it claims to solve the most irritating limitations of the JMF 1.x release. Does JMF 2.0 live up to its hype? This article explores the new features and reveals whether this release was worth the wait.*

Although the JMF 1.x API was a dramatic improvement over Sun's previous multimedia efforts, it is a work in progress. For instance, you can't record (or capture) multimedia content. Furthermore, it's a closed system: it is impossible to modify or examine multimedia content once a player begins streaming. Final-

a processor, which runs an algorithm on the data and streams the result to a destination object. Although writing DSP routines may appear intimidating, you don't need to be an electrical engineer to use them. In fact, any competent Java programmer can create simple DSP routines.

What differentiates a processor from a JMF 1.x player are plug-ins. Earlier JMF players had DSP-like capabilities, but there was no API to access them. By contrast, all DSP operations in processors are performed by well-documented plug-in objects. Sun divides processors' plug-ins into five categories: demultiplexer, effect, CODEC (compression/decompression), multiplexer and renderer (see Figure 1).

Demultiplexer plug-ins receive a single stream of multimedia content and produce one or more tracks (or streams) of data. These types of plug-ins are typically used to separate distinct audiovisual elements in file formats such as QuickTime. For instance, a QuickTime demultiplexer would separate video, audio and text data into three independently modifiable tracks (see Figure 2).

Once a track has been demultiplexed, the processor invokes the preprocessing effect plug-ins (preprocessing effects occur before streams are decompressed). Effect plug-ins tweak content but don't fundamentally change the stream. For instance, common audio effects are gain control and noise removal.

After the preprocessing stage, the processor checks to see if the track contains (or should contain) compressed content. If compression/decompression is required, a CODEC plug-in is executed. Although JMF 2.0 provides

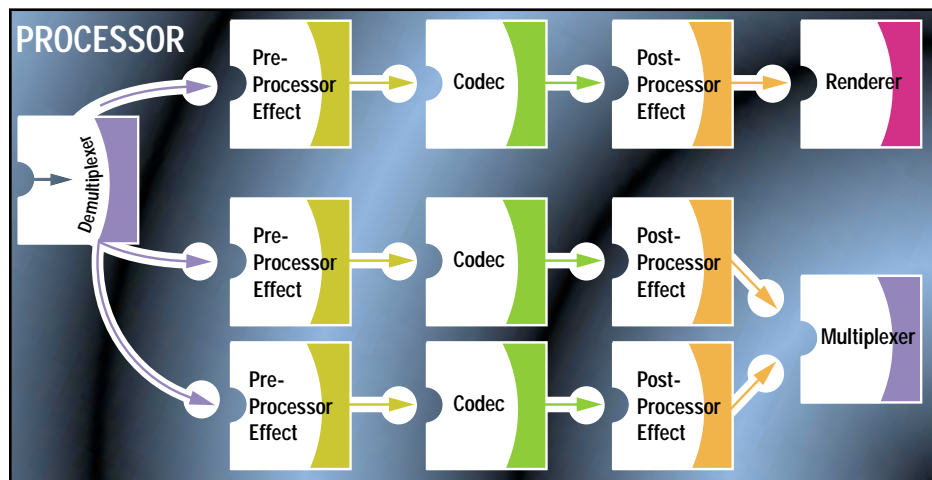


FIGURE 1 Illustration of plug-in stages (demultiplexer, effect, codec, multiplexer and renderer)

ly, the RTP APIs are strangely designed and poorly integrated with JMF.

To circumvent these limitations, JMF 2.0 introduces three types of objects: processors, plug-ins and DataSinks. These objects unleash exciting new features while maintaining backwards-compatibility with your existing JMF 1.x programs.

The improvements in JMF 2.0 are caused by processors, a special type of Player that lets you perform digital signal processing (DSP) operations on multimedia content. Content flows into

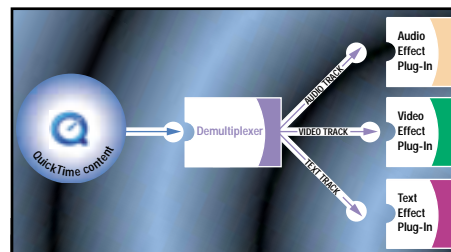


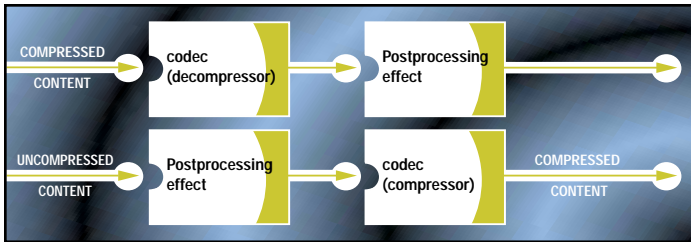
FIGURE 2 Illustration of the use of a demultiplexer

# Embarcadero

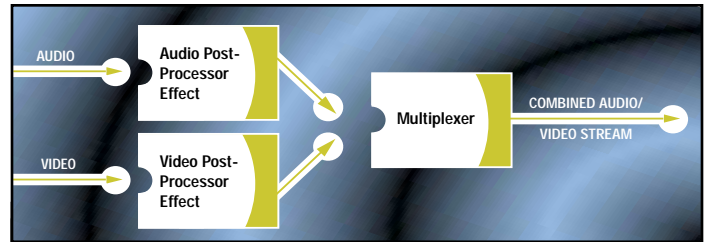
WWW.

embarcadero.

com



**FIGURE 3** Preprocessing effects operate before compression while postprocessing effects run after decompression.



**FIGURE 4** Illustration of how a multiplexer combines two streams into a single stream

CODECs for common formats such as MPEG-audio and Cinepak video, you can insert your own CODEC for proprietary or unsupported formats.

Postprocessing plug-in effects are started when the CODEC plug-in finishes. Normally, both pre- and postprocessing plug-in effects operate on uncompressed content. Conse-

quently, if you're decompressing (or playing) content, you should implement a postprocessing effect. Similarly, when compressing (or recording), effect processing should be done in the preprocessing phase (see Figure 3).

When the postprocessing plug-in completes, the processor can send the resultant tracks to either a renderer plug-in or a multiplexer plug-in. Renderers are terminating objects that transport a track to its final destination device. For example, an audio renderer would stream uncompressed pulse code modulation (PCM) content to its associated audio hardware. Likewise, a video renderer paints bitmaps onto a video device or window.

Multiplexers are the inverse of demultiplexers: they combine two or more tracks into a single track (see Figure 4). Multiplexers are popular in recording scenarios since they let you combine multiple tracks into a single format (e.g., QuickTime or MPEG).

Once the plug-ins have finished, you can stream the output to a DataSource created by the processor. The DataSource can be used to transfer the output of the processor to another player or for recording purposes.

### New States of Mind

Conventional JMF players go through five state changes (unrealized, realizing, realized, prefetching and prefetched) before they can commence playback (see my February *JDJ* article [Vol. 5, issue 2] for more information on states). Processors undergo two additional state transitions before they enter the realizing state: configuring and configured (see Figure 5).

When a processor enters the configuring state, it attempts to demultiplex its input stream and determine the type of content in each track. A processor becomes configured when the input stream has been demultiplexed and essential track information has been retrieved. It alerts you about the new state by reporting a ConfigureCompleteEvent to your application.

Once the processor is configured, TrackControl object(s) can be obtained via the getTrackControls() method. These objects let you determine which plug-ins are active and the specific phase a plug-in should execute.

Most programmers don't care about DSP processing. Rather, they only need to play or

Softwired  
www.softwired.com

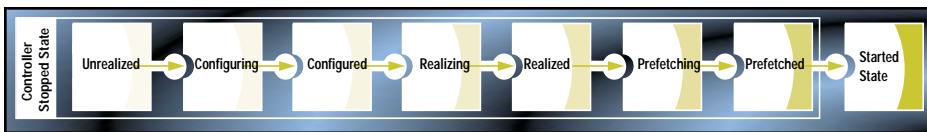


FIGURE 5 Illustration of processor state changes



FIGURE 6 Connecting an MPEG DataSink to an MPEG capture device results in an MPEG file being saved.



FIGURE 7 A broadcast DataSink can output multimedia content to the Internet rather than a file.

record. Consequently, JMF lets you bypass these details via the processor's `start()` method. If you `start()` the processor when it's unrealized, it implicitly goes through the configuring, configured, realizing, realized, prefetching and prefetched states before finally commencing playback or capturing data.

### The Kitchen Sink

Once a processor completes its task, you use a `DataSink` to save content to a specific file format or retransmit the media across a network. Like renderers, `DataSinks` are terminal objects that stream content to an output device. For example, if you had an MPEG processor that captured and compressed MPEG audio streams, you could connect its `DataSource` to an MPEG `DataSink`. The MPEG `DataSink` would in turn save the content to a properly formatted MPEG Layer 3 (or .mp3) file (see Figure 6).

`DataSinks` aren't restricted to saving data to files. In fact, they can communicate with a variety of output devices. For instance, you could create a broadcast `DataSink` whose output device is the Internet. It takes the output of a JMF `DataSource` and transmits it to a well-known multicast IP address (see Figure 7). All computers that are listening to the multicast IP address can receive and decode the stream produced by the broadcast `DataSink`.

Since `DataSinks` must be connected to an input `DataSource`, it can be tricky to construct one. Fortunately, the `Manager` simplifies this process with the `createDataSink()` method. First, obtain a `DataSource` (typically from a processor) and pass it to `createDataSink()`. The manager then constructs a `DataSink` and attaches it to your `DataSource`.

### Capture the Flag

If you want to capture content, you'll need to construct at least one processor and a `DataSink`. First, create and configure the processor (adding CODECs or effects). Then attach the processor's output `DataSource` to the `DataSink` via `createDataSink()`.

Before recording commences, you must `start()` the `DataSink`. This ensures that the

`DataSink` is ready to process the content when it arrives from the processor. The capture process commences when you call the processor's `start()` method.

When you're finished recording, you must flush (or write) all remaining buffers to the file created by the `DataSink`. To ensure no data is lost, you first `close()` the processor to cease recording. Then you `close()` the `DataSink` to ensure that all content is written to the file.

### RTP Cleanup

As we discovered last month, the JMF 1.x RTP architecture is inconsistent and often confusing. Fortunately, Sun has dramatically improved the JMF 2.0 RTP API by leveraging processor plug-ins such as CODECs, demultiplexers and multiplexers.

As we also discovered last month, the `RTPSessionManager` shields you from the RTP programming complexities. The JMF 2.0 version of `RTPSessionManager` is equally simple: you create a `MediaLocator` and construct a player from the `MediaLocator`.

### Spaghetti Architecture

Although RTP player creation is similar between JMF 1.x and 2.0, the objects composing a player are dramatically different. For instance, a JMF 1.x RTP player consists of an `RTPSessionManager` (i.e., an `RTPSocket` and `MediaProxy` objects) and an RTP `MediaHandler`. Alas, neither are pure JMF objects.

The `RTPSocket` is a strange `DataSource` that not only transmits and receives RTP content but also contains another `DataSource` used for communicating Real-Time Control Protocol (or RTCP) information. Since JMF 1.x had no API for transmitting outbound streams, Sun created an RTP-specific outbound `DataSource` – `RTPIODataSource` – and a nonstandard interface – `PushDestStream` – to push (or stream) content to the `RTPIODataSource`.

`RTPSessionManager` also contains a `MediaProxy` that converts the output of the `RTPSocket DataSource` into a format that the RTP `MediaHandler` can understand. It consists of an RTP Protocol Handler, a depacketizer and a depacketized `DataHandler` (see Figure 8).

# Embarcadero

WWW.

embarcadero.

com



FIGURE 8 RTP MediaProxy operation

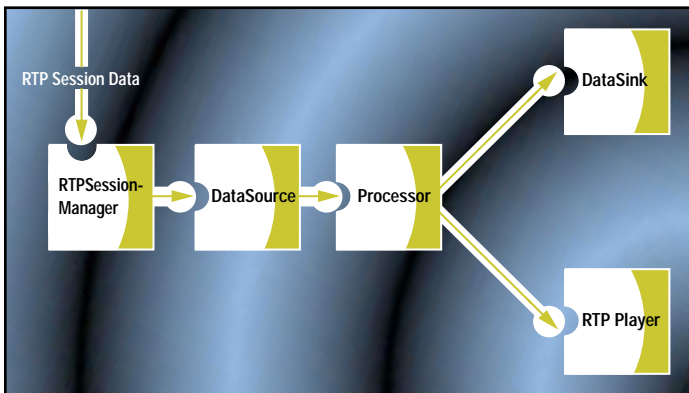


FIGURE 9 Illustration of JMF 2.0 RTP Playback architecture

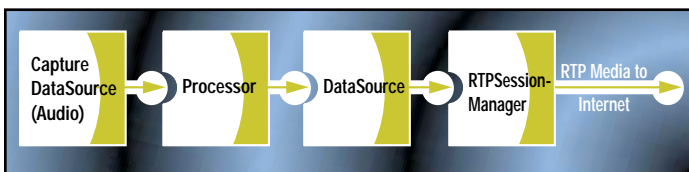


FIGURE 10 RTP transmission components

Although the MediaProxy supports the most popular RTP formats, Sun realized that developers would need to support additional formats (or RTP payloads). Unfortunately, JMF 1.x doesn't define a mechanism to customize MediaProxys. Thus they created the depacketizer interface, a non-JMF interface that lets you decode custom RTP payloads inside an RTP-centric MediaHandler.

### Elegant Design

The JMF 2.0 RTPSessionManager uses a dramatically different approach to create an RTP player. Rather than relying on nonstandard interfaces like the depacketizer and PushDestStream, the 2.0 RTPSessionManager replaces them with standardized objects such as RTPStreams, processors, CODECs and demultiplexers (see Figure 9).

An RTPStream represents the flow of RTP data and this interface is used by the RTPSessionManager to communicate with other parties in the RTP session. There are two types of RTPStreams: ReceiveStream and SendStream. ReceiveStream objects are created when content is received from a remote party. SendStream objects are constructed when content must be sent to a remote party.

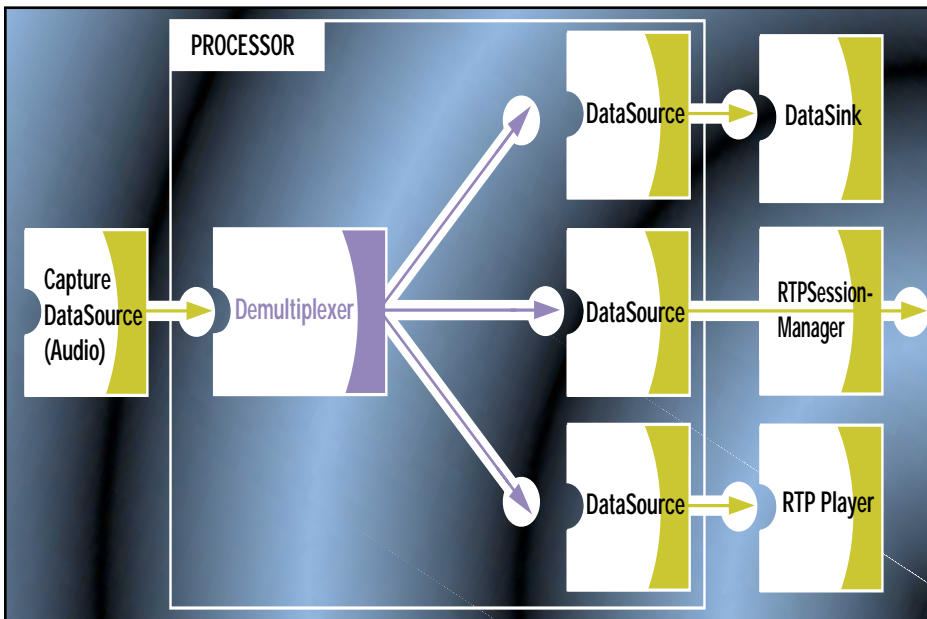
If you're receiving (or playing) RTP content, then the RTPSessionManager will create a ReceiveStream for each stream in the RTP session. The output of the RTPSessionManager is streamed into a standard JMF DataSource, which then funnels its output into a processor where the content can be decompressed and effect plug-ins executed. The processor's output is then sent to a DataSink, renderer or MediaHandler (see Figure 9).

To broadcast an RTP stream, you create a DataSource to retrieve the content. This information is streamed into a processor for compression and then sent to a DataSource connected to the RTPSessionManager. The RTPSessionManager uses the SendStream interface to transmit this data to other parties in the RTP session (see Figure 10).

Although the RTPSessionManager still uses RTPSocket to send and

# QuickStream

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



**FIGURE 11** The demultiplexer splits captured content into three streams: one is transmitted to the RTPSessionManager; another is saved to a file; the last is sent to a player so you can monitor what you're recording.

receive RTP streams, Sun transitioned from the temporary RTPIODataSource and PushDest-Stream to RTPPushDataSource and PushSource-Stream. Since the latter is also used when capturing content, the new RTP architecture is finally consistent with JMF principles.

Alas, RTPSocket retains the concept of an RTCP DataSource contained within the RTP DataSource. From a purely object-oriented standpoint, a cleaner solution would be a single DataSource that could simultaneously handle both RTP and RTCP streams. Since this design has survived the transition from JMF 1.x to 2.0, we must assume that Sun believes this DataSource within a DataSource is the proper design.

Sun has also replaced the RTP-specific depacketizer with processors and CODECs. If you need to implement support for a proprietary RTP payload, you'd insert a plug-in (or CODEC) for that custom RTP payload into the processor. Another benefit to using a processor is the ability to insert a multiplexer or demultiplexer to split or combine RTP streams (see Figure 11).

Although these architectural changes create a more robust environment for RTP development, they aren't backwards-compatible with the JMF 1.x RTP APIs. If you heeded my advice last month and concentrated on creating high-level RTP applications, you should have very few code changes. By contrast, if you're using low-level interfaces such as the depacketizer, you've got a lot of code to rewrite.

## RTSP at Last?

Until JMF 2.0, the only way to play RTSP content was to use RealNetwork's RTSP-based player. Although it isn't well publicized, JMF 2.0 contains early alpha-level code for accessing

RTSP servers. Sun is emphatic that RTSP support is for evaluation only and shouldn't be used in production code. If you need to deploy an RTSP solution, you should write your own RTSP MediaHandler or use RealNetwork's product.

## Conclusion

Although JMF 1.x provides important multimedia features, it can't capture content, is difficult to customize and has a strange RTP architecture. JMF 2.0 solves these problems with processors, plug-ins, DataSinks and a revamped RTP architecture. Processors and plug-ins let you perform digital signal processing operations on multimedia content, while DataSinks can be used to store or display content. By combining processors, plug-ins and DataSinks, you can capture and compress multimedia content.

Sun's new RTP architecture also leverages processors, plug-ins and capture interfaces to create a flexible solution that can handle common RTP payloads. And it can also be easily extended to accommodate custom formats.

Unfortunately, there are some glaring weaknesses in JMF 2.0. The RTP API isn't backwards-compatible and the RTSP support is immature. Despite these flaws, JMF 2.0 is a quantum leap forward for Java multimedia programmers. ☞

### AUTHOR BIO

Linden deCarmo is a senior software engineer at NetSpeak Corporation, where he develops advanced telephony software for IP networks. Linden is also the author of Core Java Media Framework, published by Prentice Hall.

[lindend@mindspring.com](mailto:lindend@mindspring.com)

# Embarcadero

www.

embarcadero.

com

## PointBase and ThinWEB Announce Partnership

(Mountain View, CA, and Ottawa, Ontario) – ThinWEB Technologies Corporation and its subsidiary, NoTime Wireless Corporation, announce an alliance to facilitate advanced mobile and wireless access to corporate information with PointBase, Inc., a market leader in 100% Pure Java embedded database technology.

The specific benefits that this alliance provides to the companies' customers is that they can now Internet-deploy their applications on devices without browser restrictions, firewall considerations, bulky client downloads, or JDBC driver restrictions, and without the need for a continuous network connection. ☪

[www.thinweb.com](http://www.thinweb.com)  
[www.pointbase.com](http://www.pointbase.com)

## The Object People Launch TOPartner Program

(Ottawa, Ontario) – The Object People launches its TOPartner Program, an opportunity for VARs, technology vendors and solution providers to build offerings complementary to The Object People's award-winning TOPLink family of products. Companies eligible are those that have a commercial product ship-



ping (or under development) that's complementary to the TOPLink family of products, or that have one or more service or training offerings in place complementary to the TOPLink family of products. ☪

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

## Verge Technologies Joins Persistence PowerPartner Program

(San Mateo, CA) – Persistence Software announces that Verge Technologies Group, Inc., a leading provider of EJB solutions, has joined the



Persistence PowerPartner Program. The two companies will co-market and deliver e-business solutions and training based on the PowerTier for EJB application server. ☪

[www.vergecorp.com](http://www.vergecorp.com)  
[www.persistence.com](http://www.persistence.com)

## Persistence Software to Acquire 10BaseJ

(San Mateo, CA) – Persistence Software announces that it has entered into a definitive agreement to acquire 10BaseJ. Based in San Diego, California, 10BaseJ is a leading developer of a specialized, high-performance enterprise Java servlet engine, ServletMill. The ServletMill 1.1, a scalable Java servlet container system, was introduced last December and is being integrated into the Q1 2000 release of PowerTier. ☪

[www.10BaseJ.com](http://www.10BaseJ.com)  
[www.persistence.com](http://www.persistence.com)



## Progress SonicMQ Adds Support For Linux

(New York, NY) – Progress Software Corporation announces the Developer Edition of its award-winning Progress SonicMQ Internet messaging server, which will support



the Linux operating system.

The Progress SonicMQ Developer Edition is available at no charge from [www.sonicmq.com](http://www.sonicmq.com). ☪



## TogetherSoft Releases Together Enterprise 3.1

(Raleigh, NC) – TogetherSoft launches Together Enterprise 3.1. New features include forward and reverse engineering for sequence diagrams (Java), visual report designer and custom diagram types and diagramming. ☪

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



## Unify eWave Commerce Delivers E-commerce Solutions

(San Jose, CA) – Unify Corporation introduces Unify eWave Commerce, an advanced Java-based enterprise-wide e-commerce system.

Unify eWave Commerce extends the Unify eWave product family, the first end-to-end e-commerce solution based on an open, component-based, scalable architecture. ☪

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



## eXcelon Corporation and BEA Team Together

(Burlington, MA) – eXcelon Corporation is entering into a

strategic technical and marketing relationship with BEA Systems, Inc. The partnership is designed to promote the widespread integration of BEA WebLogic Server and eXcelon's Javlin EJB data-cache server.

In addition to their technology agreement, eXcelon Corp. and BEA are planning a variety of joint sales and marketing activities, such as joint seminars, trade show appearances and sales calls. ☪

[www.bea.com](http://www.bea.com)  
[www.exceloncorp.com](http://www.exceloncorp.com)



## SAGA To Support Progress SonicMQ

(Bedford, MA and Reston, VA) – Progress Software Corporation and SAGA Software announce that the preferred messaging server for SAGA's Sagavista is the award-winning Progress SonicMQ Internet messaging server.



Sagavista is able to intelligently link front-office Web applications with any and all desired enterprise information, integrating virtually all operating environments within any enterprise. ☪

[www.sagasoftware.com](http://www.sagasoftware.com)  
[www.progress.com](http://www.progress.com)

## Pervasive and SYS-CON Offer Trial Subscriptions of Tango Developer's Journal

(Austin, TX, and Pearl River, NY) – Pervasive Software Inc. and SYS-CON Publications, Inc., jointly announce a free trial subscription program for

### Tango Developer's Journal

(TDJ). The first 500 individuals who purchase Tango 2000 products or upgrades from the Pervasive Online Store or who register Tango 2000 products or upgrades will receive a complimentary one-year subscription to TDJ.

Published quarterly by SYS-CON Publications, *Tango Developer's Journal*



is the industry's only publication devoted to the subject of end-to-end Tango Web application development and deployment.

The free, one time TDJ trial subscription offer is available to Tango 2000 purchasers in the United States only through September 30, 2000, or while the supply of complimentary promotional subscriptions lasts.

You can go to [www.pervasive.com/purchase/](http://www.pervasive.com/purchase/) to purchase Tango Development Studio and Tango Application Server products from the Pervasive Online Store. ☪


[www.pervasive.com](http://www.pervasive.com) [www.sys-con.com](http://www.sys-con.com)



# Pramati

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


## Bluestone Software Introduces Total-E-Business

(Philadelphia, PA) – Bluestone Software, Inc., launches Total-e-Business, a comprehensive, standards-based, e-business platform. Built upon J2EE standards, Total-e-Business utilizes advanced JavaServer Pages and Servlet technology and employs pure Java and XML.   
[www.bluestone.com](http://www.bluestone.com)

## Worldweb.net Partners with IIS

(Alexandria, VA) – Worldweb.net is partnering with Integrated Information Systems (IIS), a full-service provider of integrated Internet solutions. IIS will integrate Worldweb.net's flagship product, Expressroom, into e-commerce applications for leading businesses.   
[www.iisweb.com](http://www.iisweb.com)  
[www.worldweb.net](http://www.worldweb.net)

## Inxight Ships Summary Server

(Palo Alto, CA) – Inxight Software, Inc., announces Inxight Summary Server Java, a software product that creates abstracts of online documents faster and more thoroughly than comparable products. Summary Server Java allows Solaris and other high-end UNIX system users to view summarized versions of documents in subsecond speeds, and lets them customize the content to obtain the information.   
[www.inxight.com](http://www.inxight.com)

## Enhydra 2.3 Available On CD

(Orlando, FL) – Lutris Technologies announces that Enhydra version 2.3 is now available on CD. Enhydra 2.3 includes tight integration with Inprise/Borland's JBuilder Foundation IDE through the inclusion of a new product, the Enhydra Tools for JBuilder.   
[www.enhydra.org](http://www.enhydra.org)

## JDJ Reader Feedback...

Brickbats and bouquets for our ongoing series by regular contributor Rick Hightower

### Hightower Citations Don't Support Claim

I'd like to point out that [www.sys-con.com/java/archives/0502/hightower/index\\_b.html](http://www.sys-con.com/java/archives/0502/hightower/index_b.html) contains a claim not backed up by the studies cited. The statement, "The paper states that a scripting language is 5 to 10 times more productive than a strongly typed language like Java" is patently false. This is shown by the research cited on John Ousterhout's Web page ([www.scriptics.com/people/john.ousterhout/scripting.html](http://www.scriptics.com/people/john.ousterhout/scripting.html)).

Ousterhout cites [www.spr.com/library/0langtbl.html](http://www.spr.com/library/0langtbl.html) – which shows Perl at a language level of 15, Java at 6.

However, the productivity table claims languages at levels 4–8 produce 10–20 function points per programmer per month. Languages at levels 9–15 produce 16–23. This looks to me like a 100% productivity increase at best, a far cry from 500 to 1000%!

I also found it disingenuous that the Hightower article failed to mention that Ousterhout wrote Tcl!

—Doug Schwartz  
 Commercial code crafter  
[dougs@keystroket.net](mailto:dougs@keystroket.net)

### A New Subculture?

Congratulations are in order for Rick Hightower! He does a great job on the column and I wish **JDJ** the best of luck with it!!! It is indeed a change from the traditional Java programming language periodical.

Hightower has kind of created a "sub-magazine" – or potentially a subculture – that deals with the Java Virtual Machine and not the Java programming language itself. That will definitely add substance to **JDJ** and

open the eyes of Java coders to alternate 100% Pure Java (Virtual Machine) development solutions.

Perhaps Rick Hightower should liaise with the editor-in-chief about being allowed to preside over such a subdivision of the magazine. It would be grueling for you to provide rich content over the diverse JVM alternatives unassisted; but orchestrating other articles within your realm would probably be manageable. Just an idea.

—Ron Crawford II  
 Sr. Software Engineer  
[rcrawford@ahtech.com](mailto:rcrawford@ahtech.com)

### Two Omissions

Rick Hightower's article on scripting languages misses two very important entries: *NetRexx* (IBM) – the language known to every IBM platform programmer (and I have never meet anyone who programmed in it and didn't love it) and *Component Pascal* (Oberon-2) – the language that covers both scripting and system programming.

The real point of scripting languages should be to enable nonprogrammers to get into the component game and to enable programmers other than the Unix-C-Java group to be productive without extensive retraining. In this way, no one language would be *the one* – it would be accessible to many forms of programming.

The point should be to set interfaces, typing and exception handling as the programming paradigms to learn. Maybe the important thing is to have these languages be morphable into each other so that language exposure is not an inhibitor to productivity?

In the time I have been in the business, machines have become very fast but software productivity and quality if anything has gone downhill. In my opinion it's time to rethink the underlying value of programming.

—Brian R. Atkins,  
 Chester, VT  
[BrianAtkins@stargazer.com](mailto:BrianAtkins@stargazer.com)

### Programming Languages for the JVM

I am writing in response to Rick Hightower's article in the **JDJ** February 2000 issue [Vol. 5, issue 2] about programming languages for the JVM. He mentions 7 programming languages that can be used within a JVM but doesn't mention NetRexx. If the author honestly has never heard of NetRexx, then that's an honest mistake that anyone can make, but if he knows about NetRexx and didn't mention it, *why* didn't he mention it? Doesn't he realize that it is no service to his readers that he omits from a list of options one option that is very useful and helpful?

If someone were to speak about basketball's best players and not mention Michael Jordan, they would INSTANTLY discredit themselves – as would someone who would talk about baseball's best teams and mention the New York Yankees (and I happen to be a NY Mets fan!).

Mr. Hightower has spoken about basketball's greatest players and has not mentioned Michael Jordan; he has spoken about baseball's greatest teams and has not mentioned the Yankees; he has spoken about useful software tools and not mentioned Rexx or NetRexx.

—Celestino J. Pena  
[cpena02@cs.fiu.edu](mailto:cpena02@cs.fiu.edu)

### Sun's Withdrawal from ECMA

Contrary to Sean Rhody's claim that "Having Sun in charge allows us all to benefit from the quick reaction time of a single entity," having a single entity in charge is a guarantee that the entity concerned will have a chokehold on development of the language and can delay the addition of new feature to suit its own needs. An open standards process would allow features like real-time extensions and enterprise JavaBeans to come out when the technology is ready... and not when Sun is ready.

—Don W. Andrews  
[dwa@us.ibm.com](mailto:dwa@us.ibm.com)

# Concentric

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

# n-ary

[www.n-ary.com](http://www.n-ary.com)

## ADVERTISER INDEX

| ADVERTISER                     | URL                                | PH             | PG      |
|--------------------------------|------------------------------------|----------------|---------|
| 4TH PASS                       | WWW.4THPASS.COM                    | 877.484.7277   | 37      |
| AMERICAN CYBERNETICS           | WWW.MULTIEDIT.COM                  | 800.899.0110   | 35      |
| APPLIED REASONING              | WWW.APPLIEDREASONING.COM           | 800.260.2772   | 71      |
| CAREER CENTRAL                 | WWW.CAREERCENTRAL.COM/JAVA         | 888.946.3822   | 86      |
| CAREER OPPORTUNITY ADVERTISERS |                                    | 800.846.7591   | 102-113 |
| CONCENTRIC NETWORK             | WWW.CONCENTRICHOST.NET             | 800.476.0196   | 89      |
| DEVELOPENTOR                   | WWW.DEVELOP.COM                    | 800.699.1932   | 97      |
| ELIXIR TECHNOLOGY              | WWW.ELIXIRTECH.COM/DOWNLOAD/       | 65 532.4300    | 41      |
| EMBARCADERO                    | WWW.EMBARCADERO.COM/ADMINISTER     |                | 83      |
| EMBARCADERO                    | WWW.EMBARCADERO.COM/DESIGN         |                | 85      |
| EMBARCADERO                    | WWW.EMBARCADERO.COM/DEVELOP        |                | 87      |
| EVERGREEN INTERNET, INC.       | WWW.EVERGREEN.COM                  |                | 43      |
| FIORANO SOFTWARE, INC.         | WWW.FIORANO.COM                    | 408.354.3210   | 45      |
| FLASHLINE                      | WWW.FLASHLINE.COM                  | 216.861.4000   | 39      |
| GEEK CRUISES                   | WWW.GEEKCRUISES.COM                | 650.327.3692   | 97      |
| GENERIC LOGIC, INC.            | WWW.GENLOGIC.COM                   | 413.253.7491   | 10      |
| HOTDISPATCH.COM                | WWW.HOTDISPATCH.COM                |                | 4       |
| IAM CONSULTING                 | WWW.IAMX.COM                       | 212.580.2700   | 61      |
| IBM                            | WWW.IBM.COM/DEVELOPERWORKS         |                | 67      |
| MODIS SOLUTIONS                | WWW.MODISIT.COM                    | 703.821.8809   | 29      |
| INETSOF TECHNOLOGY CORP        | WWW.INETSOFTECH.COM                | 732.235.0137   | 73      |
| JAVACON2000                    | WWW.JAVACON2000.COM                |                | 78-79   |
| JDJ STORE                      | WWW.JDJSTORE.COM                   | 888.303.JAVA   | 100-101 |
| KL GROUP INC.                  | WWW.KLGROUP.COM/DEADLINE           | 888.328.9596   | 116     |
| KL GROUP INC.                  | WWW.KLGROUP.COM/REAL               | 888.328.9596   | 77      |
| KL GROUP INC.                  | WWW.KLGROUP.COM/THREADS            | 888.328.9596   | 2       |
| METAMATA, INC.                 | WWW.METAMATA.COM                   | 510.796.0915   | 55      |
| MICROSOFT                      | MSDN.MICROSOFT.COM/SUBSCRIPTIONS   |                | 11      |
| MICROSOFT                      | MSDN.MICROSOFT.COM/WINDOWSDND      |                | 13      |
| N-ARY                          | WWW.N-ARY.COM                      |                | 95      |
| NEW ATLANTA                    | WWW.NEWATLANTA.COM                 | 678.366.3211   | 53      |
| NUMEGA                         | WWW.COMPUWARE.COM/NUMEGA           | 800.4-NUMEGA   | 31      |
| OBJECT DESIGN                  | WWW.OBJECTDESIGN.COM/JAVLIN        | 800.962.9620   | 58-59   |
| OBJECTSWITCH CORPORATION       | WWW.OBJECTSWITCH.COM/IDC35/        | 415.925.3460   | 51      |
| INTUITIVE SYSTEMS, INC         | WWW.OPTIMIZEIT.COM                 | 408.245.8540   | 33      |
| PERSISTENCE                    | WWW.PERSISTENCE.COM                |                | 17      |
| POINTBASE                      | WWW.POINTBASE.COM/JDJ              | 877.238.8798   | 25      |
| PRAMATI                        | WWW.PRAMATI.COM/J2EE.HTM           | 914.876.3007   | 69      |
| PROTOVIEW                      | WWW.PROTOVIEW.COM                  | 800.231.8588   | 3       |
| QUICKSTREAM SOFTWARE           | WWW.QUICKSTREAM.COM                | 888.769.9898   | 49      |
| RIVERTON SOFTWARE CORPORATION  | WWW.RIVERTON.COM                   | 781.229.0070   | 93      |
| SEGUE SOFTWARE                 | WWW.SEGUE.COM/ADS/CORBA            | 800.287.1329   | 20-21   |
| SIC CORPORATION                | WWW.SIC21.COM                      | 822.227.398801 | 75      |
| SLANGSOFT                      | WWW.SLANGSOFT.COM/CODE/SPIRUS.HTML |                | 19      |
| SOFTWARE AG                    | WWW.SOFTWAREAG.COM/BOLERO          | 925.472.4900   | 47      |
| SYBASE INC.                    | WWW.SYBASE.COM                     | 800.8.SYBASE   | 15      |
| SYS-CON                        | WWW.SYS-CON.COM                    | 800.513.7111   | 34      |
| TIDESTONE TECHNOLOGIES         | WWW.TIDESTONE.COM                  | 800.884.8665   | 27      |
| TOGETHERSOFT LLC               | WWW.TOGETHERSOFT.COM               | 919.772.9350   | 6       |
| VISICOMP, INC.                 | WWW.VISICOMP.COM                   | 831.335.1820   | 57      |
| VISUALIZE INC.                 | WWW.VISUALIZEINC.COM               | 602.861.0999   | 88      |
| VSI                            | WWW.BREEZEXML.COM                  | 800.556.VSI    | 65      |
| YOUCENTRIC                     | WWW.YOUCENTRIC.COM/NOBRAINER       | 888.462.6703   | 63      |

# Applied Reasoning

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

# Servlets & JDBC

## How to use JDBC effectively in a servlet environment

WRITTEN BY ALAN WILLIAMSON

One of the core building blocks of any system – distributed, local or virtual – is a database. At some point in the chain of processing, the ability to store and retrieve data needs to be addressed. The capacity to access a database successfully is a high priority for many projects. Coupled with the onslaught of the Web and the need to place some sort of front-end access to a database, the demand for database connectivity at the server side is at an all-time high.

To this end I'll present a number of solutions for connecting server-side processes to a database. This includes the well-known database pooling contrasted to the single connection.

### Method #1: Connection Per Client

In this example, for each client request that comes in we will open up the connection to the database, perform a query and then output the results back to the client.

#### CONNECTING TO THE DATABASE

Before we can run any queries on the database, we have to get a handle or reference to it. Once we have an instance of `Connection`, we can run as many queries as we want. However, we do not directly create an instance of `Connection`, but ask the `DriverManager` class to supply one for us.

The `DriverManager` will attempt to connect to the given database with the optional username and password. If it is successful, then an instance of `Connection` is returned. Listing 1 illustrates this procedure. (All listings, 1–10, can be accessed on the *JDJ* Web site, [www.JavaDevelopersJournal.com](http://www.JavaDevelopersJournal.com).)

Since we do not have the driver registered in the `jdbc.properties` file, we have to make sure the driver class is available to the virtual

Second in a series of articles adapted from *Java Servlets: By Example* by Alan R. Williamson, reproduced here by permission of Manning Publications.



machine. We do this with a call to the `forName` method from the `Class` class. This will load and link the class name into the virtual machine. In this instance, we are looking to use the standard JDBC-ODBC driver that ships with the JDK. This is controlled by the class `sun.jdbc.odbc.JdbcOdbcDriver`.

Next, we have to call the method `getConnection(...)` from the `DriverManager` class. If all goes well, a newly created `Connection` will be returned. Be careful to get the driver case correct; some drivers are fussy. Another common problem is when the connection refuses to connect. Check to see that no one else has a session open to it. Most databases have a limit to the number of concurrent connections that can be open at any one time (this includes shutting down the file in MS-Access if it is open).

#### RUNNING THE QUERY

Now that we've successfully opened the connection to the database, it is ready for querying. Since this is such a generic example, let's run a query that will return the complete table data: every column and every row. We will print this data back out to the client with a row per line.

Using the JDBC API we don't need to know the name of the columns beforehand (or the data type, for that matter). We can ask the assistance of a helper class. You'll learn more about that later.

The SQL query in our instance will be:

```
SELECT * FROM USER_TABLE
```

It will begin with the creation of a new `Statement` class. We make a call to the `Connection` class with the `createStatement()` method, which will return a new instance of `Statement`. From here we can execute the SQL statement above with a call to `executeQuery(...)`.

This method will return with a `ResultSet` instance, which represents the new result table. Only one `ResultSet` can be active per statement.

Listing 2 shows the code used to execute the query. Since this example has asked for all the columns back, we get a handle to the `ResultSetMetaData` class from the `ResultSet` class. This class handles all the information that describes the data that was returned. In our example we are only interested in the number of columns that were returned. But this class gives access to all the data types of each column and the column name, if it's available.

Running through each row of the result table is done using the `next()` method. This method moves the table cursor on one row. Unless you are using a full JDBC version 2 driver, you will not be able to return to a previous row. If you need to, you will have to rerun the query.

This servlet does nothing fancy except return all the data, one row at a time per line. This is done by building up a temporary string of the column data by making a call to retrieve the data in each column of the result table. The `getString(...)` method returns the column data in the given column index in the form of `String`. The `ResultSet` method provides a `getXXX(...)` method for each of the data types available.

Once the query is complete, the result set, the statement and the connection are closed. It is important to close it down in the correct order, or a `SQLException` will be thrown.

#### PERFORMANCE

From a technical point of view, the servlet in this example is perfect. It opens up a connection to a database, it runs a query and displays the results, and closes the connection down again. It's a textbook example, one might even say.

However, from a practical point of view, it is useless and you would not use it in a real-world example. Why, you ask?

For every client request that comes in, a new database connection is created. This is not a major problem if only one person at a time comes to your Web site. However, this isn't the case. We have to assume that many people will be accessing the servlet at once. Therefore, we could potentially use up all the concurrent slots on a database engine.

The servlet is also very inefficient. The JDBC API tells us we can happily reuse the `Connection` class, with no need to open and close it all the time. Ironically, making the connection to the database can be one of the most time-consuming operations performed. But this servlet does it for every client request.

One potential way around this problem is to have the `Connection` class static, starting off live as a null. When the first client request comes in, it can create the database connection, and each subsequent connection can then reuse that connection. However, you still

# KL Group

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

have to safeguard against multiple hits; implementing the `SingleThreadModel` interface from the servlet API can easily resolve this issue.

Although technically correct, it is still very restrictive. First, only one client thread can run through the `service(...)` at any one time. Second, what if we develop another servlet to operate from the same database? Do we have to create a new connection to the database?

For these reasons, the implementation in the next section is a much better, cleaner solution.

## Method #2: Connection Pool

This section will demonstrate the design and implementation of a class that will be used to manage all the connections for the database.

### OVERVIEW

We want to be able to open up a pool of connections. Every time a class needs to run a query, it will ask for a connection from the pool. If one is available, the connection is temporarily lent to the class on the condition that it is returned after it's used. If one isn't available, then the class can wait for one to become available.

One of the things we don't want to have to do is to carry around a reference to the connection pool. This would make it awkward, as we would have to make sure all classes had a reference to it. Fortunately, with Java we don't have to worry about this.

We will design a class, `dbBroker`, to handle all the connections. It will also be responsible for the distribution of the actual connections. In order not to have to carry an instance to this class around with us, we will make all the public methods static, with the class itself holding the reference to an instance of itself.

Listing 3 shows how to set up this class. Before a method retrieves a connection from the pool, it first must make a call to `dbBroker.getInstance()`. This is a call to verify that an instance has been created and is ready to serve.

To make sure this class isn't created outside of this, we will make the constructor private. The next section will look at what happens in this constructor.

### MANAGING NEW CONNECTIONS

The connection pool will manage the connections to the database, including all the checking and administration of lending out the connections to using classes. To make things a little easier, we will define a wrapper class for each `Connection` and these classes will be used to store all the necessary information associated with the "hire" of the connection.

The class, `dbConnection` (shown in Listing 4), shows all the methods and data for each `Connection`. In addition to the `Connection` object, a flag to indicate its current status will be kept. This flag will be set when a class is using the `Connection`, using the `setActive()` method.

In multiuser systems, it can often be difficult to estimate the number of concurrent connections that are actually needed. In order for this decision to be an easier one to make, we will keep a little statistical information on each `Connection`, including the number of times the `Connection` has been used, the average time for each use and the maximum time a connection has been kept out for. The `dbConnection` handles all this information through the use of the `setActive()` and `setInactive()` methods.

When a class makes a call to `dbBroker.getInstance()`, the constructor shown in Listing 5 is run. This constructor will create the number of necessary connections and make them available for use.

One of the criteria of the connection pool was not to have any intelligence about the opening of the database distributed all over the system. Therefore, this class will open up a special file that will describe the complete connection parameters. We can therefore control all the parameters through a simple text file, and access these parameters with the `java.util.Properties` class.

The database driver, database name, username and password will be stored in the `dbBroker` class. This will allow us to reopen any connections if necessary without the need to reload the file. Another parameter that is read in is the number of connections the pool manager will manage.

Each connection will be stored in a list using the `Vector` class. Knowing the number of connections to be created makes filling up this list a trivial matter. For each connection, a call to the method in Listing 6 is made and the `openConnection()` method attempts to create a new `Connection` instance. If it's successful, then a new instance of `dbConnection` is created and inserted into the list.

You can see that the method for creating the `Connection` instance is no different from the method we used in the servlet in the first section.

### CONTROLLING CONNECTIONS

We will allow classes access to the connection pool through two methods: `pop()` and `push(...)`. The `pop()` method will look through the list of connections for a connection that is not in use. If one is found, then it is flagged as active and the `Connection` is returned.

If one is not available, then this suggests that all the connections are being used. If this is the case, then the method call will be suspended until one does become available. We can do this with a call to `wait()`. When this returns, we will reattempt to get a free connection. The method shown in Listing 7 illustrates this process.

The method called `getFreeConnection()`, which can be seen in the complete source code, simply runs through the `Vector` of `dbConnections` looking for an inactive connection.

Once a class has finished using the connection, it is returned with a call to the `push(...)` method shown in Listing 8. The method looks for the corresponding wrapper class that holds this connection. Once the wrapper class is found, the `Connection` is cleaned up with a call to `commit()` and `clearWarnings()`. This guarantees that no errors or warnings roll over to the next use.

If something goes wrong with this cleanup procedure, an `Exception` will be thrown. In this instance, the `Connection` is closed and a reattempt to open it is made. After the `Connection` has been placed back into the list as inactive, a call to `notifyAll()` notifies any waiting classes that are waiting on a free class.

### VERIFYING CONNECTIONS

It would be useful to print out all the statistical information that is being held every so often. To do this, we can set the `dbBroker` class as a threaded class and have it print out the statistics of each `dbConnection` class once every period.

The method shown in Listing 9 sleeps for 30 minutes before printing out a status report detailing the average use time and maximum time, and the number of times the connection has been accessed.

### USING THE POOL MANAGER

Now that we've created the pool manager, we can use it. We will use the same example we used before and replace the `service(...)` method with a much improved version (see Listing 10).

The complete database creation section has been replaced with a simple call to `dbBroker.getInstance()` and then `dbBroker.pop()`. The `dbBroker` class will do all the necessary loading and connecting to the database, and return a clean `Connection` instance.

After we have finished using it, we return it back to the pool manager with a call to `dbBroker.push(...)`. As you can see, there is no need to hold a separate instance to `dbBroker`, as all the methods are accessed through static calls.

## Summary

This article presented the user with an alternative to the highly inefficient method of database handling. A servlet is not like a normal application where you have a degree of control over the usage patterns. A servlet is called into action when a client makes a request; therefore, the traditional way of handling database connections has to be rethought.

Although it is already highly efficient, the `dbBroker` could be extended to include the ability to handle multiple pools. This would allow connections to different databases to be handled and manipulated at once. This is an essential feature for applications that require a distributed database layer. 🍌



# JDJ Store

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

# XML Dev

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

# Con 2000

[con2000.com](http://con2000.com)



# Java Technology Comes of Age

Java has emerged from its own hype relatively unscathed and is now showing itself capable of matching the lofty predictions made for it. The two main indicators of this rite of passage are standardization and evolving best practices – developments that are bringing corporations much nearer to achieving the productivity gains that Java can deliver.

Java's advantages are well known. In the spotlight from the outset has been the promise of WORA ("write once, run anywhere"). Even in an imperfect state WORA offers corporations bottom-line benefits that accommodate past, present and future: cost-effective integration with legacy investments, relatively painless synchronization of heterogeneous business units and longer-term platform viability that protects against costly future changes in technology infrastructure.

Developing in Java also brings productivity benefits. Because it's object-oriented, Java offers more opportunity for reuse of blocks of code, allowing teams to amortize development costs over more projects and longer time periods. Java is also easier to learn and work in, which allows for faster development and condensed training.

Now that its advantages are becoming recognized, Java is finally approaching its heyday in the enterprise. How do we make sure it behaves itself, though, having got to this point?

While emerging technologies always tend to be accompanied by rosy expectations of cost savings and streamlining of processes, turning such expectations into reality is another matter. While other languages and platforms already have a history long enough to benefit from the emergence of best practices, Java development has remained pretty much a game without rules. But this is changing.

## Standardization Is On Its Way

The trend toward standardization is evidence of this shift. While there's no shortage of companies peddling Java products, the shake-down has begun, driven by the desire of corporate purchasers to secure reputable, stable vendors in an insecure and immature market.

Pressure is growing to standardize on best-of-breed tools and components from preferred vendors. Good candidates for standardization include application servers and IDEs, as well as JavaBeans such as KL Group's JClass JavaBeans or Rogue Wave's StudioI. These reusable GUI components allow developers to build graphical front ends quickly and easily. Acting as ready-made building blocks, JavaBeans provide functionality for requirements that tend to recur from project to project (e.g., graphs, charts and tables). By standardizing on one family of components, teams can ensure high-quality interfaces and protect themselves from inconsistencies in their code base. What's more, by purchasing ready-made components rather than dedicating in-house resources to building them from scratch, companies can save valuable development time and focus on core competencies.

Equally important is a commitment to best-of-breed methodologies such as rigorous and timely code tuning. Again, other languages have a head start on Java. In Java development, early efforts concentrated more on the seemingly glorious capabilities than on the limitations. As Java becomes a serious contender for corporate application development, however, cost-conscious managers are increasingly focused on identifying and overcoming these limitations.

A key concern for Java has been performance. Software companies have risen to the challenge with solutions designed to enhance Java performance that include native compilers, VM improvements, and performance tuning and code analysis tools. Products such as JProbe Suite from KL Group and OptimizeIT! from Intuitive Systems are good examples of this. These tools offer some or all of the following functionality: performance profiling, memory debugging, thread analysis and code coverage.

## Best Practices Are Evolving

How and when should these tools be used? Best practices associated with other programming languages may be useful, even when least expected.

Take memory debugging, for instance. With the zeal typical of early adoption, developers once heralded Java as the language that would rid applications of memory leaks once and for all, thanks to the garbage collector. Yet today developers are beginning to recognize that garbage collection isn't a panacea after all. Java has its own unique brand of memory leaks that – though quite different in nature from C++ leaks – can have an even more devastating effect on performance. Consequently, memory debugging is now recognized as a critical component of the Java development cycle.

Performance tuning is another best practice that can be applied equally well to Java. Donald Knuth once quipped that "premature optimization is the root of all evil." This 25-year-old quote is often taken out of context to imply that all performance tuning should be performed in the QA and acceptance phases; conveniently omitted is the first half of the quote: "We should forget about small efficiencies, say about 97% of the time." Knuth's maxim was written at a point when computing time was several orders of magnitude more expensive than it is now. Thus developers learned to use "coding tricks" to squeeze performance out of programs, often at the expense of maintainability. More often than not, the highly obfuscated code that resulted wasn't a performance bottleneck to begin with.

In the Internet age each new revision must be rolled out within a shorter time frame than ever before. These applications, with a large user base, have performance and scalability requirements that were unheard of 25 years ago. Today's applications are larger too, and more complicated, relying on a great deal of componentized code written by many different authors. Postponing performance tuning until the end of a project can result in problems that are difficult to diagnose and often require massive rework and redesign to address, which threatens the project schedule. No development team can afford to take those kinds of risks in today's environment, where time-to-market is paramount.

When should performance tuning be done? Move tuning efforts too early and developers may run right back into the evils of premature optimization, even in the Java space. Experts recommend a risk management strategy that sees tuning begin immediately after proof of concept is safely in the bag. This strategy avoids profligate optimization efforts on mere prototypes while simultaneously ensuring that performance problems aren't given the opportunity to accumulate prior to deployment. Consequently, thread analysis, memory debugging and performance profiling in Java are increasingly taking place at the functional unit level, typically earlier than in C++ development. This new best practice makes it easier for companies to deploy reliable Java applications on time and within budget.

Talk of Java on the server and in the enterprise is now increasingly underpinned by pragmatic technologies and methodological principles that are turning the promises into reality. Standardization and best practices are a clear indicator of Java's coming of age and the real bottom-line results will soon follow. ☘

## AUTHOR BIO

Ed Lycklama is the chief technology officer and cofounder of KL Group, with primary responsibility for overseeing the company's technology direction and intellectual capital.

[el@klgroup.com](mailto:el@klgroup.com)

# SilverStream

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

# KL Group

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